Lenguaje de Programación
Euphoria
versión 2.4
Lanzamiento Oficial
3 de Julio de 2003

Me tomé la libertad de traducir al español la documentación que acompaña al entorno de desarrollo Euphoria y convertirla en un archivo .pdf, para facilitar la consulta de la misma. Como ocurre en los países de habla hispana, la terminología técnica traducida del inglés sufre ligeras variantes, por lo que traté de mantener una redacción general y fácilmente comprensible.

Usé el editor de HTML/ASP/PHP, HAPedit de freeDaniLab y el conversor HTML a PDF, HTMLDOC 1.8.23 de Easy Software Products, ambos freeware.

Fernando Velo

Tabla general de la documentación de Euphoria

Octubre de 2003

¿Esta es la última versión de Euphoria?
Visite: http://www.RapidEuphoria.com

Para instalar o desinstalar Euphoria, lea install.htm

¿Qué es lo nuevo de esta versión?
Lea relnotes.htm

Lenguaje de Programación Euphoria
versión 2.4
Lanzamiento oficial
3 de Julio de 2003

Bienvenido a Euphoria!

End User Programming with Hierarchical Objects for Robust Interpreted Applications

(Programación del usuario final con objetos jerarquicos, para aplicaciones interpretadas robustas)

Euphoria recorrió un largo camino desde que la versión 1.0 se lanzó en julio de 1993. Hay más de mil usuarios registrados, ubicados en 61 países alrededor del mundo, tanto como muchos miles más no registrados. Existe un **grupo de noticias** de Euphoria, alt.lang.euphoria, así como una **lista de correo** automatizada con más de 400 usuarios suscriptos. El sitio web de Euphoria contiene más de 1100 archivos .zip de contribución, empacados con rutinas de librería y archivos fuente de Euphoria. Decenas de personas tienen sus propios sitios web independientes con contenido relacionado a Euphoria. Se usó Euphoria en una gran variedad de **programas comerciales.** La versión **Windows** se usó para crear numerosas **interfaces gráficas de usuario, utilidades y programas relacionados con Internet.** La versión **DOS** se usó para crear muchos excitantes **juegos de acción de alta velocidad**, con soporte completo de efectos sonoros de **Sound Blaster.** Las versiones **Linux** y **FreeBSD** se usaron para escribir **programas para la interfaz X Windows, CGI** y muchas herramientas y utilitarios.

¿Otro lenguaje de programación?

Euphoria es un lenguaje de muy alto nivel, con varias características que lo destacan de los demás:

- Los programas Euphoria corren en **Windows**, **DOS**, **Linux**, y **FreeBSD**.
- El lenguaje es flexible, poderoso y fácil de aprender.
- No hay que esperar a compilar y enlazar − sólo editar y ejecutar.
- Se pueden crear y distribuir archivos **ejecutables_independientes.exe**, sin tener que pagar regalías.
- **La asignación dinámica de memoria** es fundamental para Euphoria. Las variables crecen o decrecen en tamaño sin que el programador tenga que preocuparse de reservar o liberar espacios de memoria. Los elementos de un array (secuencia en Euphoria) pueden tener una mezcla de distintos tipos y tamaños de datos.
- Euphoria provee una **amplia verificación de errores en tiempo de ejecución** para**:** índices fuera de rango, variables no inicializadas, parámetros erróneos para rutinas de librería, asignación de valores ilegales a las variables, y mucho más. Si algo no está bien, obtendrá un completo mensaje de error, con las llamadas de la pila y un listado de los valores de las variables. Con otros lenguajes, típicamente ocurre una falla de protección y obtiene un incomprensible volvado de datos de registros de la máquina y direcciones.
- El intérprete Euphoria es más de **30 veces más rápido** que el de Perl o Python, y es considerablemente más veloz que cualquier otro lenguaje interpretado, de acuerdo a la prueba "Great Computer Language Shootout" (ver demo\bench\bench.doc). Y si no es suficiente, existe el Traductor Euphoria a C que acelera la velocidad aún más. ¿Por qué perder tiempo depurando código C/C++, cuando los programas Euphoria son mucho más sencillos de desarrollar?
- Los programas Euphoria no están limitados por restricciones de memoria de 640K o 64K, por las cuales el MS−DOS es tristemente famoso. Las versiones **DOS32**, **WIN32**, **Linux** y **FreeBSD** de Euphoria le permiten usar la totalidad de la memoria instalada en su máquina y, en caso de no alcanzarle, un archivo de intercambio le proveerá una memoria virtual adicional.
- Se incluye un **depurador/trazador a nivel de fuente de pantalla completa** integrado y fácil de usar.
- También están disponibles un **analizador de perfiles por conteo de ejecución**, y otro **por tiempo**.
- Hay una gran cantidad de excelentes programas y librerías de terceras partes, la mayoría incluyendo todo su código fuente, que crece día a día.
- RDS desarrolló un sistema de base de datos extremadamente flexible (**EDS**) que corre por igual en todas las plataformas Euphoria.
- La implementación **WIN32** de Euphoria puede acceder a cualquier rutina API de WIN32 API, como también a rutinas de C o Euphoria en archivos .dll. Un equipo de gente desarrolló una librería de interfaz gráfica del usuario para Windows (**Win32Lib**) completa con un Entorno Interactivo de Desarrollo (IDE). Puede diseñar gráficamente una interfaz de usuario, especificar las sentencias Euphoria que se ejecutarán cuando alguien

haga click, y el IDE creará en su lugar, un programa Euphoria completo. Hay librerías de Euphoria Windows para acceso a Internet, juegos 3D, y muchas otras áreas de aplicación.

- La implementación **DOS32** de Euphoria en MS−DOS contiene librerías gráficas propias. Si es necesario, puede acceder a las interrupciones por software de DOS. Puede llamar a rutinas en código de máquina. Inclusive, puede hacer sus propios manejadores de interrupciones de hardware. En Euphoria se desarrollaron completamente muchos juegos de acción de alta velocidad, con efectos de sonido Sound Blaster, sin necesidad de recurrir al código de máquina.
- Las implementaciones de Euphoria para **Linux** y **FreeBSD** le permiten acceder a rutinas y variables de C en librerías compartidas, para tareas que van desde los gráficos, hasta la programación de la interfaz gráfica de usuario de X windows, pasando por la programación CGI para Internet. La buena noticia es que estará programando en Euphoria, no en C.
- Euphoria está escrito en C. El código fuente está disponible por sólo u$s 49. Puede mejorarlo, entregar sus mejoras a RDS, o inclusive vender su versión binaria mejorada. Lea Licencia del código fuente.

¿Quienes se benefician usando Euphoria?

novatos / estudiantes

- Euphoria es uno de los lenguajes más simples y fáciles de aprender.

hobbistas

- Visite nuestro sitio web site y encontrará una extensa variedad de interesantes juegos y programas. La mayoría ha sido escrita por otros hobbistas. Casi todos los programas Euphoria son de fuente abierta, por lo que puede aprender mucho echándoles un vistazo.

profesionales

- Puede desarrollar programas confiables, completamente depurables y de fácil mantenimiento en ***mucho*** menos tiempo en Euphoria que en C/C++.
- Euphoria es ideal para el desarrollo ***rápido y fácil*** de filtros de archivos y otras **utilidades**.
- Puede desarrollar **programas de Internet** y de ***interfaz gráfica***, sin necesitar hacer un curso de 6 semanas.
- Puede distribuir sus programas Euphoria como archivos **.exe,** sin tener que pagar regalías.
- Puede obtener una copia de los archivos fuente del intérprete Euphoria. Esté seguro que sus aplicaciones Euphoria serán mantenidas indefinidamente, pudiendo modificar el intérprete para ajustarlo a sus necesidades.
- Hace 10 años que RDS está en el mercado, y hay cientos de personas en la lista de correo de Euphoria, lo que le representa una fuente de soporte técnico asegurada.

Plataformas y ediciones

Euphoria corre en cuatro plataformas diferentes, **WIN32**, **DOS32**, **Linux**, y **FreeBSD**. El paquete principal es el **Intérprete Euphoria**. También existe el **Traductor Euphoria a C** en nuestro sitio web, que consta de unos pocos archivos adicionales.

El paquete del Intérprete Euphoria viene en dos ediciones diferentes: una es la **Edición de Dominio Público** y la otra es la **Edición Completa**. La Edición Completa (registrada) tiene estas características ***adicionales:***

- Puede **enmascarar** (encriptar) y **enlazar** cualquier programa Euphoria con el Intérprete Euphoria para crear archivos **únicos, independientes, inviolables .exe** de fácil distribución. (Ver **bind.doc**)
- Puede **analizar los perfiles** de cualquier programa Euphoria para determinar los "puntos calientes" del rendimiento y encontrar errores lógicos. Se proveen tanto el análisis de perfiles por conteo de ejecución, como por tiempo (solo **DOS32**).
- Puede usar la utilidad **trazadora** (depurador interactivo de pantalla completa) para depurar programas de cualquier tamaño. La Edición de Dominio Público también provee trazado, solo que limitado a programas de hasta 300 sentencias (las líneas en blanco y los comentarios no se cuentan como sentencias). Con ambas Ediciones toda vez que tenga un error en tiempo de ejecución, obtendrá siempre un informe completo de errores, con los valores de las variables y de la pila de llamadas.

Queremos que disfrute escribiendo algunos buenos programas en Euphoria. Luego, cuando decida que quiere el lenguaje y desea sacar ventaja de las **características mejoradas** de la Edición Completa, esperamos que lo registre. La registración del paquete Intérprete cuesta solamente u$s 29 (DOS32+WIN32+Linux+FreeBSD), y puede reducir o aún eliminar este costo, contribuyendo con código útil para nuestro sitio web. Lea **register\register.doc** para más información.

También existe una versión gratuita del Traductor Euphoria a C que puede descargar de nuestro sitio web. La versión completa del **Traductor Euphoria a C** cuesta solamente u$s 29. Vea el paquete Traductor en nuestro sitio web para más información.

La documentación contenida en este paquete viene tanto en formato de texto plano, como en formato HTML. Los archivos de texto plano (**.doc**) se pueden ver en cualquier editor de texto, tal como el Bloc de Notas de Windows o el WordPad. Los archivos HTML (**.htm**) los puede ver en su navegador de internet. Una herramienta que desarrollamos en Euphoria, nos permite generar automáticamente tanto el texto plano, como los archivos HTML, partiendo de una fuente común. Así, el contenido de cada archivo del subdirectorio **doc** es idéntico al contenido del archivo correspondiente en el subdirectorio **html,** sin contar la falta de vínculos, fuentes y colores. Lea **doc\overview.doc** (o **html\overview.htm**) un resumen de los archivos de documentación.

Puede distribuir libremente la **Edición de Dominio Público**, entera o en parte, por lo tanto cualquiera puede correr un programa Euphoria que Ud haya desarrollado. Tiene completa libertad para distribuir cualquier programa Euphoria que escriba, libre de regalías, aún si no ha registrado el producto.

Para correr la versión **WIN32** de Euphoria, necesitará Windows 95, o cualquier otra versión posterior. En Windows XP corre muy bien.

La versión **DOS32** correrá bajo cualquier versión de Windows y en DOS con cualquier procesador 386 o superior. Contrariamente a la opinión popular, el DOS no ha muerto. Puede ejecutar programas Euphoria DOS en la ventana del símbolo del sistema bajo Windows XP.

Para correr la versión **Linux** de Euphoria, necesitará una distribución Linux razonablemente al día, que tenga libc6 o posterior, además de ncurses. Por ejemplo en Red Hat 5.2 o superior correrá muy bien.

Para correr la versión **FreeBSD** de Euphoria, necesitará una distribución FreeBSD razonablemente al día, que tenga ncurses.

Primeros pasos

1. Después de instalar Euphoria, los archivos de documentación estarán en los directorios **doc** y **html**. **overview.doc** le da una introducción a la documentación. Debería leer primero **refman.htm** (o **refman.doc**). Si quiere buscar más información de algún tópico, escriba **guru**.
2. Diviértase corriendo los programas del directorio **demo**. Modifíquelos libremente, o ejecútelos en el modo **trazado** agregando:

   ```
   with trace
   trace(1)
   ```

   en las dos primeras líneas del archivo **.ex** o **.exw**.
3. Pruebe escribiendo algunas sentencias simples y ejecutándolas. Puede usar cualquier editor de texto. Más tarde podría usar **ed**, el editor de Euphoria, o descargar el editor Euphoria de David Cuny desde el sitio web de Euphoria.
   No tema de probar cosas, Euphoria no lo morderá!
4. Lea más ideas en **what2do.doc**.
5. Visite el sitio web de Euphoria, descargue algunos archivos y suscríbase a la ***lista de correo*** de Euphoria.

Si es novato en programación y encuentra que **refman.htm** es difícil de seguir, descargue el tutorial interctivo de **David Gay** llamado *"A Beginner's Guide To Euphoria"*, que está en la sección Documentación de nuestro Archivo.

Si al instalar tiene algún problema, lea install.htm

Aviso a los Vendedores de Shareware:

Lo animamos a que distribuya esta edición de Dominio Público de Euphoria. Puede cobrar lo que guste por él. La gente puede utilizar Euphoria tanto como guste sin obligación. Hacemos dinero con aquellos que comienzan a desarrollar seriamente aplicaciones, y quieren soporte técnico y ayuda en el trazado, análisis de perfiles y enlazado de programas grandes.

RENUNCIA:

Las Ediciones de Dominio Público y la Completa de Euphoria se proveen "como son" sin garantía de ninguna clase. En ningún caso Rapid Deployment Software será hecho responsable por cualquier daño originado en el uso, o incapacidad para usar, este producto.

Guía para los archivos de Documentación

La documentación de Euphoria está distribuida entre varios archivos de texto plano dentro del directorio **euphoria\doc**. Para cada archivo **.doc** en el directorio **euphoria\doc**, existe un archivo **.htm** correspondiente en el directorio **euphoria\html**. También puede escribir **guru** para buscar rápidamente información relevante en todos los directorios contenidos en **\euphoria**.

Todos deberían comenzar leyendo **euphoria\readme.doc** (o **euphoria\readme.htm**).

Si tiene algún problema instalando Euphoria, lea:

- **install.doc** – Instrucciones para instalar Euphoria

Si quiere aprender a programar en Euphoria, lea primero estos archivos:

- **refman.doc** – El manual del núcleo del lenguaje Euphoria
- **library.doc** – Documentación de todas las rutinas de librería de Euphoria
- **what2do.doc** – Algunas cosas por hacer para empezar con Euphoria
- **platform.doc** – Una discusión entre las cuatro ***plataformas*** en las que Euphoria corre, es decir, **DOS32** (**ex.exe**) vs. **WIN32** (**exw.exe**) vs. **Linux** (**exu**) vs. **FreeBSD** (**exu**). También incluye una discusión de cómo usar programas Euphoria con librerías C.

El resto de los archivos se pueden leer en cualquier momento, según los necesite:

- **database.doc** – El sistema de base de datos de Euphoria (EDS)
- **e2c.doc** – El Traductor Euphoria a C
- **ed.doc** – Documentación para **ed**, el editor estandár de Euphoria
- **trouble.doc** – Lista de los problemas más comunes y sus soluciones
- **bind.doc** – Describe las opciones de **bind.bat** y **shroud.bat** para **enlazado** y **encriptación**
- **perform.doc** – Consejos sobre prestaciones
- **relnotes.doc** – Una descripción acerca de lo nuevo en esta versión y un resumen de las versiones previas de Euphoria
- **c.doc** – Propaganda dirigida a programadores C/C++

La mayoría de los subdirectorios debajo de **\euphoria** tienen un archivo **.doc** que describe los archivos que contiene. El subdirectorio **euphoria\register** contiene el archivo **register.doc**. Léalo cuando quiera registrarse.

Temas dependientes de la plataforma para Euphoria

1. Introducción

Rapid Deployment Software soporta actualmente a Euphoria en cuatro ***plataformas*** distintas. En el futuro se agregarán varias más.

La primera plataforma se llama **DOS32**, que está basado en el sistema operativo DOS, pero con la CPU operando en modo protegido de 32 bits.

La segunda plataforma se llama **WIN32**, en el que el sistema operativo subyacente es Microsoft Windows, particularmente, las versiones de 32 bits de Windows que se usan en Windows 95/98/ME, tanto como NT/2000/XP y sistemas posteriores.

La tercera plataforma es **Linux**. Linux está basado en el sistema operativo UNIX. Recientemente se ha hecho muy popular entre los usuarios de PC. Hay varios distribuidores de Linux como Red Hat, Debian, Caldera, etc. Linux es un sistema operativo de código abierto y se puede obtener comprando un CD de muy bajo precio.

La cuarta plataforma es **FreeBSD**. FreeBSD también está basado en el sistema operativo UNIX, es muy popular como servidor de Internet y, al igual que Linux, también es de código abierto.

Algunos usuarios que compraron el código fuente de Euphoria, portaron Euphoria a otras plataformas tales como el UNIX de HP y el UNIX de Sun.

La instalación de Euphoria para DOS32+WIN32 contiene dos archivos **.exe**. El primero se llama **ex.exe**. Este corre programas de Euphoria en la plataforma DOS32. El segundo es **exw.exe**. Este, en cambio, ejecuta programas de Euphoria en WIN32. Los programas de Euphoria programs que están hechos para correr en WIN32 son archivos de extensión **.exw**, mientras que aquellos hechos para DOS32 tienen extensión **.ex.**

El archivo .tar de Euphoria para Linux contiene solamente **exu**. Corre programas Euphoria en la plataforma Linux. Los programas Euphoria desarrollados para Linux o FreeBSD tienen extensión **.exu**.

Para instalar la versión FreeBSD de Euphoria, primero hay que instalar la versión para Linux, y luego reemplazando exu para Linux, por la versión de exu para FreeBSD.

Muchos programas de Euphoria pueden correr en dos, tres o las cuatro plataformas sin cambios. La extensión debería indicar la plataforma preferida del programa. Cualquier intérprete Euphoria puede intentar ejecutar cualquier archivo Euphoria, pero hay que especificar el nombre completo del archivo, incluida la extensión que cada intérprete busca para identificar su tipo de archivo (.ex, .exw or .exu).

Algunas veces verá que la mayor parte del código es el mismo para todas las plataformas, pero hay pequeñas partes que se escriben especialmente para cada una de ellas. Use la función ***platform()*** para determinar la plataforma en la que se está ejecutando. Observe que ***platform()*** devuelve el mismo valor (3) tanto en Linux como FreeBSD, ya que ambos sistemas son muy similares.

2. La plataforma DOS32

Si es novato en programación, debería comenzar con **ex.exe** en la plataforma DOS32. Debería intentar comprender lo fundamental de Euphoria, antes de meterse en la programación GUI de Windows. Todas las versiones de Windows (aún XP), le permiten abrir la ventana de texto Símbolo del Sistema y ejecutar programas DOS.

Los programas Euphoria corren en modo protegido de 32 bits y tienen acceso a toda la memoria instalada en la máquina. Muchos lenguajes de programación para DOS lo limitan al modo real de 16 bits. Esto hace imposible a acceder a más memoria que 640K. Su máquina puede tener instalados 256 Mb de memoria, pero su programa se quedará sin memoria después de haber agotado los 640K. QBasic es aún peor; lo limita a solamente 160K.

Los programas DOS32 pueden alternar entre el modo de texto y el modo gráfico de píxel, y Euphoria provee rutinas de librería para ambos modos. Raramente necesitará llamar al DOS directamente, pero puede hacerlo usando la rutina ***dos_interrupt()***. También puede acceder a posiciones especiales de memoria, mediante ***peek()*** y ***poke()***, para ejecutar gráficos de alta velocidad o para acceder a detalles de bajo nivel del sistema.

Bajo **DOS32 para Windows 95 y sistemas posteriores**, los archivos Euphoria pueden tener nombres largos, lo mismo que los archivos que estos programas abran para lectura y escritura, pero no podrá crear archivos nuevos con esta característica.

Bajo **DOS puro, fuera de Windows**, no hay un sistema de archivo de intercambio, por lo que el expansor DOS incluido en **ex.exe** creará uno para que lo use su programa. Este archivo se crea cuando inicia su programa Euphoria

bajo DOS, y se borra cuando termina el programa. Comienza como un archivo de 0 bytes y crece solamente cuando es necesario. Se crea en el directorio del disco rígido apuntado por la variable de entorno TEMP o TMP. Si ninguna de esas variables existiera, se crea en el directorio que contiene a **ex.exe** o en el que está su archivo Euphoria **enlazado** (**.exe**). Puede forzar su creación en un directorio en especial, estableciendo la variable de entorno CAUSEWAY del siguiente modo:

```
SET CAUSEWAY=SWAP:path
```

donde **path** es la ruta completa del directorio. Puede evitar la creación del archivo de intercambio con:

```
SET CAUSEWAY=NOVM
```

Mientras se desarrolla el intercambio de información con la memoria virtual en disco, su programa sigue funcionando pero más lentamente. Una mejor aproximación es podría ser liberar más memoria extendida, restándosela al SMARTDRV y otros programas que reservan grandes cantidades de memoria para usos propios.

El archivo de intercambio no se creará si el espacio vacío del disco es menor que la cantidad de memoria RAM instalada en su máquina.

3. La plataforma WIN32

Euphoria para WIN32 (**exw.exe**) tiene mucho en común con Euphoria para DOS32. Con WIN32 también tiene acceso a toda la memoria de su máquina. La mayoría de las rutinas de librería trabajan del mismo modo en cada una de las plataformas. Muchos programas DOS32 de modo de texto se pueden ejecutar usando **exw** sin ningún cambio. Con **exw** puede ejecutar programas desde la línea de comandos y mostrar texto en una ventana DOS estándar (comúnmente de 25 líneas x 80 columnas). En la terminología Windows, la ventana DOS se conoce como ***consola***. Euphoria hace trivial la transición desde la programación de modo de texto de DOS32 a la programación de consola de WIN32.**Puede agregar llamadas a funciones de C en WIN32, y luego si lo desea, crear verdaderas ventanas GUI de Windows.**

Cuando su programa Euphoria escriba algo en pantalla o lea desde el teclado, automáticamente se creará una ventana de consola. También verá una ventana de consola cuando lea desde la entrada estándar o escriba hacia la salida estándar, aún cuando hayan sido redireccionadas a archivos. La consola desaparecerá cuando finalice la ejecución del programa, o mediante una llamada a ***free_console()***. Si hay algo en la consola que quiere que el usuario lea, debería avisarle y esperar su respuesta antes de terminar. Para evitar la desaparición rápida de la consola, podría incluir una sentencia como:

```
if getc(0) then
end if
```

que espera que el usuario ingrese alguna cosa.

Si quiere ejecutar programas Euphoria sin lanzar una nueva ventana de consola, puede hacer una versión de exw.exe que use la ventana de consola actual, como cuando un programa DOS usa ex.exe. Para hacer esta versión alternativa de exw.exe, necesita ejecutar la utilidad **makecon.exw** que está en euphoria\bin. Para ahorrar espacio en el archivo de instalación, no hemos creado este archivo .exe. También hay una utilidad llamada **make40.exw** que creará una versión alternativa de exw.exe que usa una versión más reciente (versión 4.0) de la interfaz gráfica de usuario de Windows.

Bajo WIN32, están completamente soportados los nombres largos de archivos, tamto para lectura, como para escritura.

3.1 Programación WIN32 de alto nivel

Gracias a **David Cuny**, **Derek Parnell**, **Judith Evans** y algunos otros, hay un paquete llamado **Win32Lib** con el que se pueden desarrollar aplicaciones Windows GUI en Euphoria. Es notoriamente sencillo de aprender y de usar, y viene con una buena documentación y algunos pequeños programas de ejemplo. Puede descargar Win32Lib y el entorno de desarrollo de Judith desde el Sitio Web de Euphoria. Recientemente, **Andrea Cini** ha desarrollado otro similar, un paquete algo más pequeño llamado **EuWinGUI**. También está disponible en nuestro sitio.

3.2 Programación WIN32 de bajo nivel

Para permitir el acceso a WIN32 de bajo nivel, Euphoria provee un mecanismo para llamar a cualquier función de C en cualquier archivo .dll del API de WIN32, o cualquier otro archivo .dll de Windows de 32 bits que usted cree o que otro haya creado. También existe un mecanismo de call−back que le permite a Windows llamar a sus rutinas de Euphoria. Las call−backs son necesarias para crear interfaces gráficas de usuario.

Para hacer completo uso de la plataforma WIN32, necesitará documentación para programación de Windows de 32 bits, en especial la Interfaz de Programación de Aplicaciones (API) de WIN32, incluyendo las estructuras de Cdefinidas por la API. Hay un extenso archivo WIN32.HLP © de Microsoft que está disponible con muchas herramientas de programación para Windows. Hay numerosos libros al respecto de la programación WIN32 para C/C++. Mucho de lo que encuentra en esos libros, lo puede adaptar para el mundo de la programación Euphoria para WIN32. Un buen libro es:

Programming Windows
de Charles Petzold
Microsoft Press

Se puede descargar desde el sitio de Borland un archivo de ayuda del API de WIN32 (8 Mb):
ftp://ftp.inprise.com/pub/delphi/techpubs/delphi2/win32.zip

También vea la página Web Archivo − "documentación" de Euphoria.

4. Las plataformas Linux y FreeBSD

Euphoria para Linux, y Euphoria para FreeBSD comparten ciertas características con Euphoria para DOS32, y algunas otras con Euphoria para WIN32.

Como en WIN32 y DOS32, puede escribir texto en una consola, o en la ventana xterm, en varios colores y en cualquier línea o columna.

Como en WIN32, puede llamar rutinas de C en librerías compartidas y código C puede llamar a sus rutinas (call−back).

Euphoria para Linux y FreeBSD no tienen soporte integrado para gráficos de píxel como DOS32, pero Pete Eberlein creó una interfaz Euphoria para **svgalib**.

Actualmente, la programación de interfaz gráfico del usuario Xwindows es más fácil usando la interfaz de Irv Mullin para la librería GTK GUI.

Al portar código desde DOS o Windows a Linux o FreeBSD, notará las siguientes diferencias:

- Alguno de los números asignados a los 16 colores principales en graphics.e son distintos. Si usa las constantes definidas en graphics.e bo tendrá problemas; si en cambio, escribe los números de color literalmente dentro del código, notará que el azul y el rojo habrán cambiado.
- Los códigos para teclas especiales como Inicio, Fin, teclas de flechas, son diferentes y hay algunas diferencias adicionales cuando ejecuta bajo XTERM.
- La tecla Enter tiene el código 10 (avance de línea) en Linux, mientras que en DOS/Windows tiene el 13 (retorno de carro).
- Linux y FreeBSD usan '/' en sus rutas. DOS/Windows usan '\\'.
- Cosas altamente especializadas como ***dos_interrupt()***, obviamente no funcionan en Linux o FreeBSD.
- Las llamadas a ***system()*** y ***system_exec()*** que contienen comandos DOS, cambiarán obviamente a los correspondientes comendos de Linux o FreeBSD. Por ejemplo, "DEL" se convierte en "rm", y "MOVE" en "mv".

5. Interfacing con código C (WIN32, Linux, FreeBSD)

En WIN32, Linux y FreeBSD es posible relacionar código Euphoria con código de C. Su programa Euphoria puede llamar rutinas de C y leer y escribir variables. Las rutinas de C pueden igualmente llamar ("callback") a sus rutinas Euphoria. El código de C tiene que estar en una librería de enlace dinámico de WIN32 (archivo .dll), en una librería compartida de Linux o FreeBSD (archivo .so). Al conectarse con las .dll o las .so, puede acceder completamente a las interfacesde programación de esos sistemas.

Usando el Traductor Euphoria a C, puede traducir las rutinas Euphoria a C, y compilarlas dentro de archivos .dll o .so. Puede pasar átomos y secuencias Euphoria a esas rutinas Euphoria compiladas, y recibir datos Euphoria como resultado. Comúnmente, las rutinas traducidas/compiladas se ejecutan mucho más rápido que las rutinas interpretadas. Para obtener más información, vea ***el Traductor***.

5.1 Llamando Funciones de C

Para llamar una función de C en un archivo **.**dll o **.**so, tiene que seguir los suientes pasos:

1. Abrir el archivo **.**dll o **.**so que tiene la función de C, llamando a ***open_dll()*** en **euphoria\include\dll.e.**

2. Definir la función de C, llamando a ***define_c_func()*** o ***define_c_proc()*** en **dll.e**. Esto le dice a Euphoria la cantidad y tipo de argumentos, tanto como el tipo del valor de retorno.
   Euphoria soporta actualmente todos los tipos enteros y punteros de C, como argumentos y valores de retorno. También soporta para los mismos casos de tipo punto flotante (tipo double de C). Actualmente noes posible pasar estructuras de C por valor y recibir una estrctura como función resultado, aunque puede pasar un puntero a la estructura y obtener un puntero a la estructura como valor de retorno. Pasar estructuras de C por valor es un requerimento raramente solicitado por las llamadas del sistema operativo.

   Euphoria también soporta todas las formas de datos de Euphoria − átomos y secuencias arbitrariamente complejas, como argumentos para rutinas Euphoria traducidas/compiladas.
3. Llamar la función de C, invocando a ***c_func()*** o ***c_proc()***.

Ejemplo:

```
include dll.e

atom user32
integer LoadIcon, icon

user32 = open_dll("user32.dll")

-- el nombre de la rutina en user32.dll es "LoadIconA".
-- Toma como argumentos un puntero y un entero, y devuelve un entero.
LoadIcon = define_c_func(user32, "LoadIconA", {C_POINTER, C_INT}, C_INT)
icon = c_func(LoadIcon, {NULL, IDI_APPLICATION})
```

Ver en **library.doc − Llamando funciones de C** las descripciones de c_func(), c_proc(), define_c_func(), define_c_proc(), open_dll(), etc. Ver los programas de ejemplo en **demo\win32** o **demo\linux**.

En Windows hay más de una convención de llamadas. Las rutinas de la API de Windows usan la convención **__stdcall**. Sin embargo, la mayoría de los compiladores de C tienen **__cdecl** establecida por defecto. ***__cdecl*** permite pasar una cantidad variable de argumentos. Euphoria asume ***__stdcall***, pero si necesita llamar una rutina de C que usa ***__cdecl***, puede poner un signo '+' al comienzo del nombre de la rutina en ***define_c_proc()*** y ***define_c_func()***. En el ejemplo de más arriba, tendría "+LoadIconA", en lugar de "LoadIconA".

Puede examinar un archivo .dll haciendo click derecho sobre él y eligiendo "vista rápida" (si existe en su sistema). Verá la lista de todas las rutinas de C que el .dll exporta.

Ejecute **euphoria\demo\win32\dsearch.exw** para determinar el archivo .dll contiene una función C WIN32 en especial.

5.2 Accediendo a Variables de C

Puede obtener la dirección de una variable de C, usando ***define_c_var()***. Luego, puede usar ***poke()*** y ***peek()*** para acceder al valor de la variable.

5.3 Accediendo a estructuras de C

Muchas rutinas de C necesitan que les pase punteros a estructuras. Puede simular las estructuras de C asignando bloques de memoria. La dirección devuelta por ***allocate()*** se puede pasar como si fuera un puntero de C.

Puede leer y escribir los miembros de las estructuras de C usando ***peek()*** y ***poke()***, o ***peek4u()***, ***peek4s()***, y ***poke4()***. Puede asignar espacio para las estructuras usando ***allocate()***. Tiene que calcular el offset de un miembro de una estructura de C. Normalmente esto es sencillo, porque cualquier cosa en C que necesite 4 bytes, se asignará 4 bytes en la estructura. Así, los "int" de C, los "char", los "unsigned int" , punteros a lo que sea, etc. todos tomarán 4 bytes. Si la declaración C se ve como:

```
// Warning C code ahead!

struct ejemplo {
    int a;          // offset  0
    char *b;        // offset  4
    char c;         // offset  8
    long d;         // offset 12
};
```

Para reservar espacio para "struct ejemplo", necesitará:

```
atom p

p = allocate(16) -- tamaño de "struct ejemplo"
```

La dirección que obtiene de ***allocate()*** es siempre de al menos 4 bytes alineados. Esto es provechoso, ya que las estructuras de WIN32 are supposed to start on a 4−byte boundary. Fields within a C structure that are 4−bytes or more in size must start on a 4−byte boundary in memory. 2−byte fields must start on a 2−byte boundary. Para hacer esto, tiene que dejar pequeños huecos dentro de la estructura. En la práctica no es difícil alinear la mayoría de las estructuras, ya que el 90% de los campos son punteros de 4 bytes o enteros de 4 bytes.

Puede asignar los campos, usando algo como:

```
poke4(p + 0, a)
poke4(p + 4, b)
poke4(p + 8, c)
poke4(p +12, d)
```

Puede leer un campo con algo como:

```
d = peek4(p+12)
```

Consejo:

Por legibilidad, utilice offsets para declarar sus constantes Euphoria. Ver el ejemplo siguiente:

Ejemplo:

```
constant RECT_LEFT = 0,
         RECT_TOP  = 4,
         RECT_RIGHT = 8,
         RECT_BOTTOM = 12

atom rect
rect = allocate(16)

poke4(rect + RECT_LEFT,    10)
poke4(rect + RECT_TOP,     20)
poke4(rect + RECT_RIGHT,   90)
poke4(rect + RECT_BOTTOM, 100)

-- pasa rect como un puntero a una estructura de C
-- hWnd es un "handle" para la ventana
if not c_func(InvalidateRect, {hWnd, rect, 1}) then
    puts(2, "Falló InvalidateRect\n")
end if
```

El código Euphoria que accede a las rutinas de C y las estructuras de datos, puede verse un poco feo, pero normalmente será solamente una pequeña parte de su programa, especialmente si usa Win32Lib, EuWinGUI, o la librería X Windows de Irv Mullin. La mayor parte de su programa se escribirá en Euphoria puro, que le dará una gran ventaja sobre aquellos forzados a codificar en C.

5.4 Call−backs hacia sus rutinas Euphoria

Cuando crea una ventana, el sistema operativo Windows necesitará llamar a su rutina Euphoria. Esto es un concepto extraño para los programadores DOS, quienes han usado llamadas a rutinas del sistema operativo, pero no han tenido llamadas del sistema operativo a ***sus*** rutinas. Para lograrlo, tiene que obtener una dirección de "call−back" de 32 bits para su rutina y dársela a Windows. Por ejemplo (en **demo\win32\window.exw**):

```
integer id
atom WndProcAddress

id = routine_id("WndProc")

WndProcAddress = call_back(id)
```

routine_id() identifica unívocamente a un procedimiento o función Euphoria, devolviendo un valor entero pequeño. Este valor se puede usar más tarde para llamar a la rutina. También puede usarlo como un argumento de la función ***call_back()***.

En el ejemplo de más arriba, la *dirección de call−back* de 32 bits, WndProcAddress, se puede almacenar en una estructura de C y pasarla a Windows mediante la función C de API RegisterClass(). **Esto le da a Windows la capacidad de llamar rutinas Euphoria, WndProc(), toda vez que el usuario realiza una acción en una cierta clase de ventana.** Las acciones incluyen hacer click en el ratón, presionar una tecla, redimensionar la ventana, etc. Lea el programa de ejemplo **window.exw,** para ver la historia completa.

Nota:

Se puede obtener una *dirección de call−back* para ***cualquier*** rutina Euphoria que cumpla las siguientes condiciones:

- ◊ la rutina tiene que ser una función, no un procedimiento.
- ◊ tiene que tener de 0 a 9 parámetros.
- ◊ los parámetros deberían ser todos declarados de tipo átomo (o entero), no como secuencia.
- ◊ el valor de retorno debería ser un valor entero de 32 bits.

Puede crear tantas direcciones de call−back como desee, pero no debería llamar a ***call_back()*** varias veces a la misma rutina Euphoria − cada dirección de each call−back que cree necesita un pequeño bloque de memoria.
Los valores que se pasan a su rutina Euphoria pueden ser cualquier átomo de 32 bits **sin signo**, es decir, no negativo. Su rutina podría elegir interpretar números positivos grandes como negativos, si fuera necesario. Por ejemplo, una rutina de C intenta pasarle −1, aparecería como el número hexadecimal FFFFFFFF. Si se pasa un valor que no se ajusta al tipo elegido para un parámetro, ocurrirá un error de verificación de tipos de Euphoria (dependiendo de **with/without type_check**). No ocurrirá ningún error si declara al parámetro como **átomo.**

Normalmente, como en el caso de más arriba de WndProc(), Windows inicia esas call−backs a sus rutinas. **También es posible que una rutina C en cualquier .dll llame a una de sus rutinas Euphoria.** Solamente tiene que declarar adecuadamente la rutina C y pasarle la dirección de call−back.

Hay un ejemplo de una rutina de WATCOM C que toma su dirección de call−back address como único parámetro y entonces llama a su rutina Euphoria de 3 parámetros:

```
/* rutina C de 1 parámetro que llama desde Euphoria */
unsigned EXPORT APIENTRY test1(
         LRESULT CALLBACK (*eu_callback)(unsigned a,
                                         unsigned b,
                                         unsigned c))
{
    /* Su rutina Euphoria de 3 parámetros se llama aquí
       via el puntero eu_callback */
    return (*eu_callback)(111, 222, 333);
}
```

La sentencia C de más arriba, declara a test1 como una rutina que se puede llamar externamente y que tiene un solo parámetro. Ese parámetro es un puntero a una rutina que tiene 3 parámetros sin signo, es decir, su rutina Euphoria.
En WATCOM C, "CALLBACK" es lo mismo que "__stdcall". Esta es la convención de llamadas que se usa para llamar a las rutinas WIN32 API, y el puntero C a su rutina Euphoria se deberá declarar de esta forma también, u obtendrá un error cuando su rutina Euphoria intente regresar de su .DLL.

Si necesita que su rutina Euphoria se la llame usando la convención __cdecl, tendrá que escribir la llamada a ***call_back()*** como sigue:

```
myroutineaddr = call_back({'+', id})
```

El signo más y las llaves indican la convención __cdecl. El caso simple, sin llaves, es __stdcall.

En el ejemplo de más arriba, su rutina pasará los valores 111, 222 y 333 como argumentos. Su rutina devolverá un valor a test1. Luego, ese valor se devolverá inmediatamente al llamador de test1 (que puede estar en algún otro sitio de su programa Euphoria).

Se puede pasar una dirección de call−back a la función signal() de Linux o FreeBSD para especificar una rutina Euphoria, para manejar varias señales (por ejemplo, SIGTERM). También se le pueden pasar a C rutinas como qsort(), para especificar una función de comparación de Euphoria.

Cómo instalar Euphoria en Windows

Descargar y ejecutar e24setup.exe. Este archivo instalará Euphoria en cualquier versión de Windows, desde Windows 95 en adelante.

Después de instalarlo, ver **doc\what2do.doc** (o **html\what2do.htm**) para obtener ideas de cómo usar este paquete. También debería leer **readme.doc** (o **readme.htm**) si áun no lo ha hecho.

Possibles problemas ...

- Si parece que el ***programa de instalación*** se ejecutó correctamente, pero los programas de ejemplo no funcionan ¿recordó reiniciar la computadora?
- En WinME/98/95 si falla el ***programa de instalación***, editar el archivo **autoexec.bat**, el cual deberá Ud. mismo. Siga el procedimiento que se describe al final de este manual.
- Euphoria se puede ejecutar bajo DOS en los viejo sistemas Win 3.1, pero no existe un programa de instalación para esto. Instale Euphoria en un nuevo sistema y copie el directorio EUPHORIA al sistema viejo. Actualice el archivo AUTOEXEC.BAT manualmente en el sistema viejo en la forma que se describe más abajo.

Cómo editar manualmente el archivo autoexec.bat (WinME/98/95/3.1)

1. − En el archivo **c:\autoexec.bat** agregar **C:\EUPHORIA\BIN** a la lista de directorios del comando **PATH**. Puede usar el comando Edit de MS−DOS, Windows Notepad o cualquier otro editor de texto.
   También puede ir al menu Inicio, seleccionar Ejecutar e ingresar **sysedit** y presionar Enter. El archivo **autoexec.bat** debería aparecer como uno de los archivos del sistema que se pueden editar y guardar.
2. − En el mismo archivo **autoexec.bat** agregar una nueva línea:

   ```
   SET EUDIR=C:\EUPHORIA
   ```

   La variable de entorno **EUDIR** indica la ruta completa del directorio principal de Euphoria.
3. − Reiniciar la máquina. Esto definirá las nuevas variables de entorno **PATH** y **EUDIR**.

En WinNT/2000/XP y posteriores, si por alguna razón **EUDIR** y otras variables de **PATH** no están correctamente establecidas, entonces hay que establecerlas según la manera que el sistema lo permita. Por ejemplo, en Windows XP elegir: Inicio −> Panel de Control −> Performance−> Sistema −> Avanzado y entonces hacer clic en el botón "Variables del Entorno". Hacer clic en el botón "New..." e ingresar **EUDIR** como nombre de variable y **c:\euphoria** (o lo que sea correcto) en el valor, haciendo clic finalmente en el botón OK. Buscar **PATH** en la lista de variables, seleccionarla y hacer clic en "Editar...". Agregar al final **c:\euphoria\bin** y hacer clic en OK.

Algunos sistemas como Windows ME, tienen un archivo **autoexec.bat**, que está oculto para que no se vea en los listados de directorios. No obstante está ahí y puede verse o editarse, si fuera necesario, tipeando: **notepad c:\autoexec.bat** en una ventana DOS.

Si tiene el archivo **autoexec.bat**, pero éste no contiene el comando PATH, deberá crear una entrada para él, incluyendo **C:\EUPHORIA\BIN**.

Cómo desinstalar Euphoria

1. Si desea recuperar su versión anterior de Euphoria, o partes de ella, el subdirectorio "backup" contiene una copia de seguridad de cada uno de los subdirectorios anteriores de Euphoria, más algunos archivos que haya agregado. Sin embrgo, no contiene ningún subdirectorio que se pueda haber añadido.

2. Si no existe ningún archivo que necesite, puede borrar el directorio en que se instaló Euphoria.

3. Borrar la variable de entorno EUDIR, y quitar el directorio EUPHORIA del comando PATH, en el archivo C:\AUTOEXEC.BAT, o en Panel de Control/Sistema/Avanzado.

4. Borrar las referencias a EUPHORIA del menu Inicio.

¿Qué hacer?

Ahora que Euphoria está instalado, hay algunas cosas que podría intentar:

- Ejecutar cada programa de demostración en el directorio **demo**. Solo escriba **ex** o **exw** o **exu** seguido del nombre del archivo **.ex** o **.exw** o **.exu**, por ejemplo:

```
ex buzz
```

  ejecutará el archivo **buzz.ex**. Dependiendo de su tarjeta gráfica, podría tener que editar alguna línea de alguno de los archivos **.ex** para elegir los distintos modos gráficos. Algunos programas de demostración intentan usar los modos **SVGA**, que podría no funcionar con su tarjeta de video. También necesitará soporte para un ratón DOS para ejecutar **mouse.ex** y **ttt.ex**.
  También puede hacer doble clic sobre un archivo **.ex** (**.exw**) desde **Windows**, pero tendrá que "asociar" los archivos **.ex** con **ex.exe** y los archivos **.exw** con **exw.exe**. Algunos pocos de los demos están hechos para ejecutarse desde la línea de comandos, pero la mayoria se verá bien en Windows.
- Use el editor Euphoria, **ed**, para editar un archivo Euphoria. Advierta el uso de los colores. **Puede ajustar estos colores junto con el tamaño del cursor y algunos otros parámetros "modificables por el usuario", editando las declaraciones de constantes en ed.ex.** Use **Esc q** para salir del editor o **Esc h** para obtener ayuda. Hay muchos, e incluso mejores, editores orientados a Euphoria, en el Archivo.
- Crear algunos nuevos ensayos de prestaciones. Ver **demo\bench**. ¿Obtiene las mismas tasas de velocidad que hicimos en comparación a otros lenguajes populares?
- Lea el manual en **doc\refman.doc** o **vea su versión HTML** haciendo doble clic en el vínculo para iniciar su navegador de Internet. The simple expressive power of Euphoria makes this manual much shorter than manuals for other languages. Si tiene una pregunta específica, escriba **guru** seguido de una lista de palabras. El programa **guru** buscará todos los archivos **.doc** tanto como todos los programas de ejemplo y otro archivos, y le presentará una lista *ordenada* de las más relevantes porciones de texto que puedan responder a su pedido.
- Pruebe correr un programa Euphoria con el **trazado** activado. Agregue:

```
with trace
trace(1)
```

  al comienzo de cualquier archivo **.ex** o **.exw**.
- Ejecute algunos de los programas tutoriales en **euphoria\tutorial**.
- Pruebe modificar alguno de los programas de demostración.

  Primero, algunas modificaciones ***simples*** (toma menos de un minuto):

  > ¿Qué pasa si hubieran 100 naves C++ en **Language Wars**? ¿Qué pasa si **sb.ex** tiene que mover 1000 bolas en lugar de 125? Cambie algunos parámetros en **polygon.ex**. ¿Podría hacer que aparezcan imágenes más bonitas? Agregue algunas frases divertidas a **buzz.ex**.

  Entonces, algunas ***ligeramente difíciles*** (toma algunos minutos):

  > Defina una nueva función de x e y en **plot3d.ex**.

  Entonces un ***desafío*** (toma una hora o más):

  > Inicie su propia base de datos personalizada definiendo campos en **mydata.ex**.

  Entonces un proyecto ***mayor*** (varios días o semanas):

  > Escriba un algoritmo *más inteligente* para 3D TicTacToe.
- Intente escribir su propio programa en Euphoria. Un programa puede ser tan simple como:

```
? 2+2
```

Recordar que después de encontrar cualquier error, puede simplemente escribir ed y saltar a la línea del archivo que genera el problema.
Una vez que comience a usarlo, estára desarrollando programas ***mucho*** más rápidos en Euphoria que los que podría hacer en Perl, Java, C/C++ o cualquier otro lenguaje que conozcamos.

Mensaje importante para los programadores C/C++ ...

24 Razones por las que escribirá su próximo programa en Euphoria!

- porque está cansado de tener que reinventar el establecimiento de almacenamiento dinámico por cada programa que escribe
- porque ha desperdiciado demasiadas horas buscando errores alrededor de malloc
- porque alguna vez lo fastidió un molesto error que aparecía y desaparecía y era provocado por una variable no inicializada
- porque no importa cuanto empeño ponga en eliminarlos, siempre que algún problema de almacenamiento
- porque está cansado que su máquina se bloquee, o su programa se cancele sin indicación alguna de error
- porque sabe que la **verificación de índices** le habría evitado horas de depuración
- porque su programa no debería poder escribir aleatoriamente áreas de memoria a través de punteros "traviesos"
- porque sabe lo malo que es sobrecargar la pila, pero no tiene idea de cuán cerca está de ello

- porque una vez tuvo un error misterioso al llamar una función que no debía devolver valor alguno, pero en lugar de ello devolvía basura aleatoriamente
- porque desea que las rutinas de librería deberían detenerse al pasar argumentos incorrectos, en lugar de solamente de establecer "errno" o lo que sea (¿quien revisa errno en cada llamada?)
- porque quisiera "recompilar el mundo" en una fracción de segundo, en lugar de tardar varios minutos −− se trabaja mucho más rápido con un ciclo edición/ejecución que con edición/compilación/enlazado/ejecución
- porque la 3ra edición de *El Lenguaje de Programación C++* de Bjarne Stroustrup es un "servicio de emergencias" demasiado denso, (y no discute temas de programación específicos para las plataformas DOS, Windows, Linux o ninguna otra).
- porque ha estado programando mucho tiempo en C/C++, pero hay muchas características misteriosas del lenguaje que no comprende completamente
- porque la portabilidad no es tan fácil de realizar como se debiera
- porque conoce el rango válido de los valores de las variables, pero no tiene forma de forzarlos en tiempo de ejecución
- porque le gustaría pasar un número variable de argumentos, pero no quiere saber nada con la forma complicada de hacerlo en C
- porque le gustaría una ***forma clara*** de devolver múltiples valores desde una función
- porque quisiera un **depurador a nivel de fuente de pantalla completa** integrado que sea tan fácil de usar que no tenga que estar consultando el manual todo el tiempo, (o rendirse y recompilar con instrucciones printf por doquier)
- porque detesta cuando su programa empieza a trabajar justo cuando agregó instrucciones de impresión para depurarlo o lo compila con la opción de depuración
- porque le gustaría tener un **analizador de perfiles de ejecución a nivel de instrucciones** confiable y preciso para comprender la dinámica interna de su programa y mejorar sus prestaciones
- porque muy pocos de sus programas tienen que exprimir al límite cada ciclo de ejecución. La diferencia de velocidad entre Euphoria y C/C++ no es tan grande, especialmente cuando usa el Traductor Euphoria a C. Pruebe algunos tests de prestaciones. Lo sorprenderemos!
- porque no quiere invadir su disco rígido con archivos **.**obj y **.**exe
- porque en lugar de ejecutar su programa, ha estado navegando entre cientos de páginas de documentación para decidir que opciones del compilador y enlazador necesita
- porque su versión de C/C++ tiene 57 rutinas diferentes de establecimiento de memoria, y 67 rutinas diferentes para manejar cadenas y bloques de memoria. ¿Cuántas de esas rutinas necesita Euphoria? **Respuesta: ninguna. En Euphoria, el establecimiento de memoria ocurre automáticamente y las cadenas se manejan del mismo modo que cualquier otra secuencia.**

El editor Euphoria

uso 1: ed <nombre de archivo>
uso 2: ed

Resumen

Después de cualquier error, sólo escriba **ed**, y se ubicará en el editor, en la línea y columna donde ocurrió el error. El mensaje de error estára en la parte superior de su pantalla.

Los archivos relacionados con Euphoria se muestran en colores. Los otros archivos de texto, en blanco y negro. Sabrá que ha deletreado algo mal cuando el color no cambia como usted espera. Las palabras claves son azules. Los nombres de las rutinas internas del intérprete aparecen en magenta. Las cadenas son verdes, los comentarios son rojos, la mayoría del resto del texto es negro. Los paréntesis equilibrados (en la misma línea) tienen el mismo color. **Puede cambiar esos colores tanto como otros parámetros de ed.** Ver "parámetros modificables por el usuario" en la parte superior de **ed.ex**.

Las teclas de flecha mueven el cursor hacia la izquierda, derecha, arriba o abajo. La mayoría del resto de los caracteres se inserta inmediatamente en el archivo.

En ***Windows***, puede "asociar" varios tipos de archivos con **ed.bat**. Al hacer doble clic sobre ellos (por ejemplo **.e**, **.pro**, **.doc,** etc), serán abiertos por **ed.** Los archivos principales de Euphoria terminados en **.ex** (**.exw**) sería mejor que estén asociados con **ex.exe** (**exw.exe**).

ed es un editor ***DOS*** **multiarchivo/multiventana**. **Esc c** separará su ventana, por lo que puede ver y editar hasta 10 archivos simultáneamente, con cortado y pegado entre ellos. También puede usar varias ventanas de edición para ver y editar distintas partes del mismo archivo.

Si no le gusta **ed**, tiene aún muchas alternativas. El **editor EE** de David Cuny es un editor ***DOS*** para Euphoria que está escrito en Euphoria. Tiene una amigable interfaz basada en el ratón con menúes desplegables, etc. Está disponible en el sitio web de RDS. Hay muchos otros editores orientados a Euphoria que corren en ***DOS***, ***Windows***, Linux and y FreeBSD. Consulte la sección **Editors** de nuestro Archivo. De hecho, se puede usar *cualquier* editor de texto, incluyendo el Edit de ***DOS*** o el Bloc de Notas de ***Windows***.

Teclas especiales

Algunas teclas de la PC no trabajan en la consola de texto de ***Linux*** o ***FreeBSD***, o en Telnet, y algunas teclas tampoco lo hacen en xterm bajo X windows. Se han provisto teclas alternativas. En algunos casos en ***Linux/FreeBSD*** tendría que editar ed.ex para mapear la tecla deseada en la función deseada.

Supr	− Borra el caracter actual sobre el cursor.
Backspace	− Mueve el cursor hacia la izquierda y borra un caracter.
Ctrl+Supr	− Borra la línea actual (**Ctrl+Supr** no está disponible en todos los sistemas).
Ctrl+D	− Borra la línea actual (igual que **Ctrl+Supr**).
Insert	− Reinserta la serie precedente de **Supr** o **Ctrl+Supr** antes del carácter actual o de la línea actual.
Ctrl+flecha izquierda	− Mueve el cursor al inicio de la palabra anterior. ***Linux/FreeBSD*** use **Ctrl+L**.
Ctrl+flecha derecha	− Mueve el cursor al inicio de la palabra siguiente. En ***Linux/FreeBSD*** use **Ctrl+R**.
Inicio	− Mueve el cursor al inicio de la línea actual.
Fin	− Mueve el cursor al final de la línea actual.
Ctrl+Inicio	− Mueve el cursor al inicio del archivo. En ***Linux/FreeBSD*** use **Ctrl+T** (es decir, inicio)
Ctrl+Fin	− Mueve el cursor al final del archivo. En ***Linux/FreeBSD*** use **Ctrl+B**, (es decir, final)
Re Pág	− Se mueve una pantalla arriba. En un xterm de ***Linux/FreeBSD*** use **Ctrl+U**
Av Pág	− Se mueve una pantalla abajo. En un xterm de ***Linux/FreeBSD*** use **Ctrl+P**
F1 ... F10	− Seleciona una nueva ventana actual. Las ventanas están numeradas de arriba hacia abajo, con la ventana superior en pantalla asignada a la tecla **F1**.
F12	− Este es un **comando** especial **personalizable**. Está preparado para insertar un comentario Euphoria "−−" al comienzo de la línea actual. Puede **cambiarlo para realizar cualquier otra pulsación que desee**, simplemente redefiniendo la constante CUSTOM_KEYSTROKES en la parte superior de **ed.ex**.

Comandos de escape

Presione y libere la tecla **Esc**, luego presione una de las siguientes teclas:

h − Obtener ayuda de texto para el editor o para Euphoria. Esta pantalla está separada, por lo tanto puede leer la ayuda y el código al mismo tiempo.

c − "Clona" la ventana actual, es decir, hace una nueva ventana de edición que inicialmente muestra el mismo archivo en la misma posición que la vetana actual. Puede querer usar **Esc l** para obtener más líneas en pantalla. Cada ventana que crea, se puede desplazar independientemente y cada una de ellas tiene su propia barra de menú. Los cambios que haga a un archivo, aparecerán inicialmente sólo en la ventana actual. Al presionar una **tecla de Función** para seleccionar una nueva ventana, cualquier cambio aparecerá allí también. Puede usar **Esc n** para leer un nuevo archivo en cualquier ventana.

q − Cierra (borra) la ventana actual y abandona el editor, si no hay más ventanas. Será advertido si esta es la última ventana usada para editar un archivo modificado. Cualquier ventana restante tendrá más espacio.

s − Guarda el archivo que se está editando en la ventana actual, entonces cierra la ventana actual como lo hace **Esc q.**

w − Guarda el archivo pero no cierra la ventana.

e − Guarda el archivo y entonces lo ejecuta con **ex**, **exw** o **exu**. Cuando termina la ejecución del programa, escuchará un bip. Presione **Enter** para volver al editor. Esta operación puede no funcionar si tiene muy poca memoria extendida. No puede suministrarle al programa ningún **argumento de la línea de comandos.**

d − Ejecuta un comando del sistema operativo. Después del bip, presione **Enter** para volver al editor. También podría usar este comando para editar otro archivo y luego regresar, pero probablemente **Esc c** sea más conveniente.

n − Comienza a editar un nuevo archivo en la ventana actual. Las líneas/caracteres borrados y las cadenas buscadas, están disponibles para usarse en el nuevo archivo. Alternativamente, puede arrastrar el nombre de un archivo desde la ventana del Administrador de Archivos de ***Windows*** a la ventana ***MS−DOS*** de **ed**. Esto escribirá por usted la ruta completa.

f − Busca la siguiente ocurrencia de una cadena en la ventana actual. Al escribir una nueva cadena, existe la opción de "coincidencia de mayúsculas" o no. Presione **y** si necesita que coincidan las mayúsculas/minúsculas. Mantenga presionado **Enter** para encontrar las ocurrencias subsecuentes. Cualquier otra tecla detiene la búsqueda. Para buscar desde el principio, presione ***Ctrl+Inicio*** antes de **Esc f**. La cadena por defecto a buscar, si no escribe nada, se muestra entre comillas dobles.

r − Reemplaza globalmente una cadena por otra. Opera como el comando **Esc f**. Mantenga presionado el **Enter** para continuar reemplazando. Tenga cuidado −− *no hay manera de saltear un posible reemplazo.*

l − Cambia la cantidad de líneas mostradas en pantalla. Están permitidos solamente algunos valores, dependiendo de su tarjeta de video. Muchas tarjetas aceptarán 25, 28, 43 y 50 líneas.
En una consola de texto ***Linux/FreeBSD*** you're stuck con la cantidad de líneas disponibles (generalmente 25). En una ventana xterm de ***Linux/FreeBSD***, **ed** usará la cantidad de líneas que están disponibles inicialmente al arrancar ***ed***. Cambiar el tamaño de la ventana no tendrá efecto hasta que se arranque ***ed.***

m − Muestra las modificaciones hechas hasta ahora. El búfer de edición actual se guarda como **editbuff.tmp**, y se lo compara con el archivo en disco usando el comando ***fc*** del ***DOS***, o el comando ***diff*** de ***Linux/FreeBSD.*** **diff** command. **Esc m** es muy útil cuando quiere cerrar el editor, pero no recuerda que cambios hizo, o si está bien guardarlos. También es útil cuando comete un error de edición y quiere ver como se veía el texto original. Al salir del editor tiene la posibilidad de borrar **editbuff.tmp**.

ddd − Mueve el cursor a la línea número *ddd*. Por ejemplo, **Esc 1023 Enter** mueve el cursor a la línea 1023 del archivo.

CR − **Esc Carriage−Return**, es decir, **Esc Enter**, le dirá el nombre del archivo actual, tanto como la línea y posición del caracter en el que está, y si el archivo fue modificado desde la última grabación. Si presiona **Esc** y cambia de idea, solamente presione **Enter,** así puede ir nuevamente a edición.

Recordar cadenas anteriores

Los comandos **Esc n**, **Esc d**, **Esc r** y **Esc f** le piden que ingrese una cadena. Puede recordar y editar esas cadenas del mismo modo que lo hace en la línea de comandos de ***DOS*** o ***Linux/FreeBSD***. Presione flecha hacia arriba o flecha hacia abajo para ciclar entre las cadenas que previamente ingresó para un dado comando, entonces use las flechas izquierda o derecha y la tecla Supr para editar las cadenas. Presione Enter para aceptar la cadena.

Cortar y pegar

Al aplicar **Ctrl+Supr** (o **Ctrl+D**) a una serie de líneas consecutivas, o **Supr** a una serie de caracteres consecutivos, se crea un "búfer de borrado" conteniendo solamente aquello que borró. Este búfer de borrado se puede reinsertar al mover el cursor y presionar **Insert**.

Cada vez que abandona la tarea y comienza a borrar alguna otra parte, se inicia un nuevo búfer de borrado, perdiéndose el anterior. Por ejemplo, corte una serie de ***líneas*** con **Ctrl+Supr**. Entonces mueva el cursor a donde quiere pegar las líneas y presione **Insert**. Si quiere copiar las líneas sin destruir el texto original, primero presione **Ctrl+Supr** e inmediatamente **Insert** para reinsertarlas. Luego muévase a alguna otra parte donde quiera insertarlas nuevamente y presione **Insert** tantas veces como quiera. También puede presionar **Supr** sobre una serie de ***caracteres*** individuales, mueva el cursor y pegue los caracteres borrados en algúnotro lugar. Inmediatamente presione **Insert** después de borrar si no quiere perder los caracteres originales.

Una vez que tiene el búfer de borrado, puede presionar **Esc n** para leer un nuevo archivo, o puede presionar una **tecla de Función** para seleccionar una nueva ventana de edición. Luego puede insertar su búfer de borrado.

Uso de tabuladores

El ancho estándar del *tabulador* es 8 espacios. El editor asume tab=8 para la mayoría de los archivos. Sin embargo, es más conveniente al editar un programa tener un tabulador igual a la cantidad de espacios que utiliza como sangría. Por lo tanto encontrará que los tabuladores están establecidos a 4 cuando edita archivos Euphoria (o archivos .c, .h o .bas). El editor convierte de tab=8 a tab=4 cuando lee su *programa*, y lo convierte nuevamente a tab=8 al guardarlo. Así su archivo se mantiene compatible con el mundo de tab=8, por ejemplo PRINT, EDIT, etc del ***MS−DOS***. **Si quisiera elegir una cantidad diferente de espacios de sangría,** cambie la línea que dice "constant PROG_INDENT = 4" en la parte superior de **ed.ex**.

Líneas largas

Las líneas que se extienden más allá del borde derecho de la pantalla, se marcan con un caracter de *video inverso* en la 80ª columna. Esto le advierte que hay más texto "allí afuera" que no puede ver. Puede mover el cursor más allá de la 80ª columna. La pantalla se desplazará hacia la izquierda o derecha, por lo que la posición del cursor siempre estará visible.

Tamaño máximo del archivo

Como cualquier otro programa Euphoria, **ed** puedre acceder a toda la memoria de su máquina. Puede editar enormes archivos y, salvo que se use el archivo de intercambio, la mayoría de las operaciones serán muy rápidas.

Archivos que no son de texto

ed está diseñado para editar archivos de texto puro, aunque puede usarlo para ver otros archivos. Cuando **ed** lee un archivo, reemplaza ciertos caracteres no imprimibles (menores que ASCII 14) con ASCII 254 − un cuadrado pequeño. *Si intenta guardar un archivo que no es de texto, será alertado acerca de esto.* (El Edit de ***MS−DOS*** corromperá silenciosamente un archivo que no es de texto − ¡no lo guarde!). Como **ed** abre todos los archivos como "texto", aparecerá un caracter **Ctrl+Z** (26) inserto en un archivo que **ed** interpretará como el *fin del archivo*.

Nombres largos

Aunque **ed** es un editor ***DOS***, puede editar los archivos *existentes* que tengan rutas con nombres largos en ellas y el nombre completo será preservado. Sin embargo, esta versión de **ed** no creará ***nuevos*** archivos con nombres largos. El nombre se truncará según el estándar ***DOS*** 8.3 de longitud.

Terminador de línea

El terminador de fin−de−línea en ***Linux/FreeBSD*** es simplemente **\n**. En ***DOS*** y ***Windows***, las líneas de los archivos de tecto terminan con **\r\n**. Si copia un archivo ***DOS*** o ***Windows*** a ***Linux/FreeBSD*** e intenta modificarlo, **ed** le dará la opción de mantener los terminadores **\r\n**, o guardar el archivo con terminadores **\n**.

Código fuente

El código fuente completo de este editor está en **bin\ed.ex** y **bin\syncolor.e**. Sus mejoras serán bienvenidas. Hay una sección en la parte superior de **ed.ex** conteniendo parámetros de configuración "modificables por el usuario" que puede ajustar. Tal vez sea necesario ajustar los colores y el tamaño del cursor en algunos sistemas operativos.

Plataforma

ed corre mejor con **ex.exe** o **exu**, pero también lo hará con **exw.exe**.

Traductor Euphoria a C

1. Introducción

El Traductor Euphoria a C traducirá cualquier programa Euphoria a su equivalente de código C.

Existen versiones del traductor para **Windows**, **DOS**, **Linux** y **FreeBSD**. Después de traducir un programa Euphoria a C, puede compilarlo y enlazarlo usando uno de los compiladores de C soportados. Esto dará como resultado, un ejecutable que normalmente correrá mucho más rápido que si se utiliza el intérprete Euphoria.

2. Instalación

Se asume que ya tiene instalado el paquete **Intérprete Euphoria 2.4** en su sistema, y que las variables de entorno **EUDIR** y **PATH** están correctamente establecidas.

Para cada compilador de C, en **Windows**, **DOS**, **Linux** o **FreeBSD**, encontrará un archivo .ZIP en el sitio de RDS, conteniendo:

```
1. el traductor (uno por plataforma)

      ec.exe   - DOS
      ecw.exe  - Windows
      ecu      - Linux
      ecu      - FreeBSD

2. una librería de tiempo de ejecución (una por compilador de C)

      ec.lib   - DOS (Watcom)
      ec.a     - DOS (DJGPP)
      ecw.lib  - Windows (Watcom)
      ecwl.lib - Windows (Lcc)
      ecwb.lib - Windows (Borland)
      ecu.a    - Linux (GNU)
      ecu.a    - FreeBSD (GNU)

3. (Watcom solamente) archivos para soportar el expansor DOS CauseWay

      cwc.exe    - compresor de archivos
      le23p.exe  - conversor de formato
      cwstub.exe - el expansor DOS


4. (DJGPP solamente) la librería gráfica Allegro, compilada especialmente para
                el traductor. Descargar liballeg.zip
```

Para instalar el **Traductor Euphoria a C**, poner los archivos .exe y los de librería requeridos en el directorio **euphoria\bin**. El directorio **euphoria\include** ya contiene el archivo **euphoria.h**, un archivo de inclusión de C necesario para traducir todos los programas.
Nota:

> El archivo **liballeg.zip** es especial. Después de instalar el compilador DJGPP de C, debería descomprimir **liballeg.zip** y poner **liballeg.a** en el directorio **DJGPP\LIB**.

3. Compiladores de C Soportados

El **Traductor** trabaja normalmente con GNU C en **Linux** o **FreeBSD**, tanto como con Watcom C o DJGPP C en **DOS**, y con Watcom C, Lcc o Borland 5.5 en **Windows**. Las implementaciones de Watcom y GNU C son 100% compatibles con el **Intérprete Euphoria**. Los otros son 99% compatibles. Recomendamos Borland sobre Lcc. Borland compila más rápido, produce mejor código y tiene muy pocos errores en comparación a Lcc.

Se probó el **Traductor** con GNU C y la librería ncurses disponible en Red Hat Linux 5.2 o posterior, y con FreeBSD 4.5 o posterior.

También se lo probó con Watcom C/C++ 9.5, 10.6 y 11.0. El compilador Watcom 11.0 es gratuito y de código abierto. Búsquelo en: http://www.openwatcom.org

El paquete Watcom DOS32 incluye el expansor DOS y compresor de archivos CauseWay DOS. CauseWay es ahora gratuito y de código abierto. Puede encontrarlo más acerca suyo en: http://www.devoresoftware.com

emake.bat y **objfiles.lnk** enlazarán automáticamente con el expansor CauseWay. Otros expansores DOS, tales como DOS4GW, no trabajan bien con el **Traductor**.

El **Traductor** busca "WATCOM", "LCC", "BORLAND" o "DJGPP" tanto como variables del entorno como directorios en **PATH**. Así, generará un archivo **emake.bat** que invocará al compilador y enlazador adecuados.

Notas:

- A diferencia de Watcom, DJGPP no mapea la memoria baja de DOS en el mismo segmento como otra memoria. Las rutinas de código de máquina escritas para el **Intérprete** o **Traductor Euphoria** basado en Watcom, no trabajarán con DJGPP, y probablemente se abortarán si intenta acceder a memoria baja, tal como memoria de video. Los comandos Euphoria como **peek()**, **poke()**, **mem_copy()**, **mem_set()**, etc. ***trabajarán*** correctamente, si el **Traductor** usa una macro especial de DJGPP para acceder a la memoria baja. Se pueden portar esas rutinas de código de máquina a DJGPP, pero se necesitará consultar la documentación de DJGPP para conocer las formas posibles de acceso a la memoria baja.
- DJGPP soporta completamente los nombres largos para creación, lectura y escritura. Watcom no soporta la creación.
- DJGPP soporta algunos modos de texto más, por ejemplo, el modo de 35 líneas.
- DJGPP le permite al usuario abortar un programa en cualquier momento, presionando Ctrl+C.
- La implementación Lcc ignora **lock_file()** y **unlock_file()**. No hacen nada.
- El **Traductor** no utiliza la opción de optimización −O de Lcc en **emake.bat**. Esta opción actualmente es demasiado poco confiable. Si desea acelerar su velocidad, puede experimentar agregando −O para algunos archivos .c. Esto funcionará en la mayoría de los casos. Si no funciona, por favor envíe un informe de error a los desarrolladores de Lcc, no a RDS.
- Las advertencias se desactivan cuando se compila con **emake.bat**. Si las activa, puede ver algunos mensajes inofensivos acerca de variables declaradas pero sin uso, rótulo declarados pero sin uso, prototipos de funciones no declarados, etc.
- En Windows, el enlazador de Watcom advierte que no puede abrir graph.lib. Puede ignorarla, porque graph.lib no se usa. Parece no haber una forma sencilla de eliminar este mensaje.
- El compilador Microsoft C++ para Windows no está soportado todavía. Sin embargo, probablemente pueda importar los archivos generados por **ecw.exe**, y la librería de tiempo de ejecución de Borland, Lcc o Watcom al proyecto Microsoft, y compilar/enlazar solo con algunos pequeños problemas.

4. Cómo Correr el Traductor

Ejecutar el **Traductor** es similar a ejecutar el **Intérprete**. En DOS debería escribir:

```
ec shell.ex
    o
       ec shell
```

pero en lugar de correr el programa .ex, el **Traductor** creará varios archivos fuente de C. También creará un archivo llamado **emake.bat** que compilará y enlazará los archivos de C por Ud. Escriba solamente:

```
emake
```

Cuando la compilación y el enlazado de C terminen, obtendrá un archivo llamado: **shell.exe**

Al ejecutar shell.exe, debería correr igual que si hubiera escrito: **ex shell** para ejecutarlo con el **Intérprete**, excepto que debería ejecutarse mucho más rápido.

Nota para usuarios de Linux y FreeBSD:

> Los archivos se llamarán **emake** y **shell**, y deberá escribir **./emake** para compilar y enlazar, y **./shell** para ejecutar el programa.

Opciones de la línea de comandos

Si ocurre que tiene más de un compilador de C para una plataforma dada, puede elegir el que desee usar, mediante una opción de la línea de comandos:

-bor
-lcc
-wat
-djg

en la línea de comandos para **ec** o **ecw**, por ejemplo:

```
ecw -bor pretend.exw
```

Para hacer un archivo .dll de **Windows**, o un .so de **Linux** o **FreeBSD**, agregue solamente **-dll** a la línea de comandos, por ejemplo:

```
ecw -bor -dll mylib.ew
```

Nota:

Para Lcc y Borland, no existe una variable de entorno estándar, por lo tanto el **Traductor** buscará en la variable **PATH** el directorio del compilador. La búsqueda se hará en lugares estándar tales como: ..\LCC, ..\BCC.., ..\Borland.. etc. Si se instaló el compilador en otro directorio (no estándar), se lo deberá renombrar.

5. Librerías de Enlace Dinámico (Librerías Compartidas)

Agregando simplemente **-dll** a la línea de comandos, el **Traductor** generará un archivo .dll de **Windows** .dll (.so en **Linux**/**FreeBSD**), en lugar de generar un programa ejecutable.

Puede traducir y compilar un conjunto de rutinas Euphoria útiles, y compartirlas con otras personas, sin necesidad de entregar sus archivos fuente. Además, sus rutinas correrán mucho más rápido al estar traducidas y compiladas. Tanto los programas traducidos/compilados, como los interpretados podrán ser capaces de usar su librería.

Solamente las funciones y procedimientos globales de Euphoria, es decir, aquellos declarados con la palabra clave "global", se exportarán desde el archivo .dll (.so).

Cualquier programa Euphoria, sea traducido/compilado o interpretado, puede vincularse con un .dll (.so) de Euphoria usando el mismo mecanismo que le permite vincularse a un .dll (.so) escrito en C. El programa llama primero a **open_dll()** para abrir el archivo .dll o .so, entonces llama a **define_c_func()** o **define_c_proc()** para cualquier rutina que quiera invocar. Llama a esas rutinas usando **c_func()** y **c_proc()**. Ver **library.doc** por más detalle.

Los nombres de las rutinas exportadas desde .dll de Euphoria .dll variarán según el compilador de C que utilice.

GNU C en Linux o FreeBSD exporta los nombres exactamente como aparecen en el código C producido por el **Traductor**, por ejemplo, una rutina Euphoria:

```
global procedure foo(integer x, integer y)
```

se exportaría como "_0foo" o tal vez como "_1foo", etc. El guión de subrayado y el dígito se agregan para evitar un posible conflicto de nombres. El dígito se refiere al archivo Euphoria en donde está definido el símbolo. El archivo principal se numera como 0. Los archivos include se numeran en el orden en que el compilador los encuentra. Debería verificar el archivo fuente de C para estar seguro.

Lcc exportaría a foo() como "__0foo@8", donde 8 es el número de parámetros (2) cuatro veces. Puede verificar el archivo .def creado por el **Traductor** para ver todos los nombres exportados.

El **Traductor** para Borland, también crea el archivo .def, pero en este archivo se renombran los símbolos exportados con el mismo nombre que se usó en el código fuente de Euphoria, por lo tanto, foo() se exportaría como "foo".

En el caso del compilar Watcom, ocurre lo mismo que con el de Borland, pero en lugar de crear un archivo .def, se agrega un comando EXPORT a **objfiles.lnk** por cada símbolo exportado.

Con Borland y Watcom, se puede editar los archivos .def o **objfiles.lnk**, y volver a ejecutar **emake.bat**, para renombrar los símbolos exportados, o remover aquellos que no desea exportar. Con Lcc, se pueden quitar símbolos, pero no renombrarlos.

No es imperioso tener esmerados nombres exportados, puesto que el nombre solamente necesita aparecer una vez en cada programa Euphoria que use el archivo .dll, es decir, en un solo comando **define_c_func()** o **define_c_proc()**. El autor de un archivo .dll posiblemente les entregue a sus usuarios un archivo de inclusión de Euphoria conteniendo los comandos **define_c_func()** y **define_c_proc()** necesarios, e incluso proveer un conjunto de rutinas Euphoria que

"envuelvan" a las llamadas a las rutinas en el archivo .dll.

Al llamar a **open_dll()**, cualquier instrucción Euphoria de alto nivel en el .dll o .so se ejecutará automáticamente, como en un programa normal. Esto le da a la librería la oportunidad de inicializar sus estructuras de datos antes de la primera llamada a una rutina. Para algunas librerías, no se necesita la inicialización.

Para pasar datos Euphoria (átomos y secuencias) como argumentos, o recibir un objeto Euphoria como un resultado, se necesitará agregar los siguientes nuevos tipos de datos a **euphoria\include\dll.e**:

```
-- Nuevos tipos Euphoria para argumentos y valores de retorno en .dll (.so):
global constant
        E_INTEGER = #06000004,
        E_ATOM    = #07000004,
        E_SEQUENCE= #08000004,
        E_OBJECT  = #09000004
```

Uselos en **define_c_proc()** y **define_c_func()** del mismo modo que usa C_INT, C_UINT etc. para llamar los .dll y los .so de C.
Actualmente, los números de archivo devueltos por **open()**, y el ID de rutina devuelto por **routine_id()**, pueden ser pasados y devueltos, pero la librería y el programa principal tienen por separado, su propia idea de lo que esos números significan. En lugar de pasar el número de archivo a un archivo abierto, podría pasar el nombre del archivo y permitir abrir el .dll (.so). Desafortunadamente no hay una solución simple para el pasaje del ID de rutina. Esto será cambiado en el futuro.

Las .dll (.so) se pueden usar también con programas C, mientras que solo se intercambien valores enteros de 31 bits. Si se tiene que pasar un puntero de 32 bits o un valor entero, y se tiene el fuente del programa en C, se podría pasar el valor en dos argumentos enteros de 16 bits (16 bits superiores y 16 bits inferiores) y combinarlos en una rutina Euphoria para obtener el átomo deseado de 32 bits.

6. Tamaño y Compresión del Ejecutable

En DOS32 con Watcom, si el **Traductor** encuentra los archivos de CauseWay, cwc.exe y le23p.exe en **euphoria\bin**, agregará comandos a **emake.bat** para comprimir su archivo ejecutable. Si no desea la compresión, puede editar **emake.bat**, o quitar o renombrar cwc.exe y/o le23p.exe.

En Linux, FreeBSD, Windows, y DOS32 con DJGPP, **emake** no incluye un comando para comprimir el archivo ejecutable. Si desea comprimir el ejecutable, le sugerimos que lo intente con el compresor gratuito UPX. Puede obtener UPX en: http://upx.sourceforge.net

El **Traductor** borra las rutinas que no se usan, incluyendo aquellas que están en los archivos de inclusión estándar de Euphoria. Después de borrar las rutinas sin uso, busca iterativamente otras rutinas que estén sin uso hasta hallarlas todas. Esto puede hacer una gran diferencia, especialmente con programas basados en Win32Lib, donde se incluye un gran archivo, pero muchas de las rutinas allí incluidas no se usan.

No obstante, su archivo ejecutable compilado será probablemente tan grande como cualquier programa Euphoria enlazado con el **Intérprete**. Esto se debe en cierto modo a que el **Intérprete** es un ejecutable comprimido. También, los comandos Euphoria son extremadamente compactos cuando se almacenan en un archivo enlazado. Necesitan más espacio después de ser traducidos a C y compilados dentro del código de máquina. Las futuras versiones del **Traductor** producirán ejecutables más pequeños y veloces.

7. Intérprete vs. Traductor

Todos los programas Euphoria se pueden traducir a C, y con unas pocas excepciones indicadas más abajo, correrán igual que con el **Intérprete** (pero más rápido).

El **Intérprete** y el **Traductor** comparten el mismo analizador de código, por lo que deberían obtener los mismos errores de sintaxis, errores de variables no declaradas, etc, tanto uno como el otro.

El **Traductor** lee enteramente el programa antes de intentar cualquier traducción. Ocasionalmente puede capturar un error de sintaxis que el **Intérprete** no vea, porque el **Intérprete** comienza ejecutando inmediatamente instrucciones de alto nivel, sin esperar el fin del programa. Esto también significa que si hay instrucciones de alto nivel en el programa que modifican un archivo que se incluye después, se obtendrá un resultado diferente con el **Traductor**. Muy pocos programas usan esta técnica de "inclusión dinámica".

El **Intérprete** expande automáticamente la pila de llamadas (hasta que la memoria se agota), por lo que puede haber una cantidad enorme de llamadas anidadas. La mayoría de los compiladores de C, en la mayoría de los sistemas, tienen un límite preestablecido del tamaño de la pila. Consulte el manual de su compilador o enlazador si quiere incrementar el valor límite, por ejemplo si tiene una rutina recursiva que necesite miles de niveles de recursión. Modifique el comando de enlace en emake.bat. Para Watcom C, use **OPTION STACK=nnnn**, donde nnnn es la

cantidad de bytes de espacio de pila.

Nota:

> El **Traductor** asume que el programa no tiene errores en tiempo de ejecución, los que deberían ser capturados por el **Intérprete**. El **Traductor** no verifica: índices fuera de rango, variables no inicializadas, asignaciones de datos de tipo erróneo a una variable, etc.

Debería **depurar** su programa con el **Intérprete**. Cuando el código de C se cancela, típicamente se obtiene una excepción de máquina muy críptica. Si se aborta el programa .exe traducido, la primera cosa que debería hacer es ejecutarlo en el **Intérprete**, usando las mismas entradas, y preferiblemente con la opción **type_check** activada.

Algunas de la rutinas son capaces de capturar e informar errores en tiempo de ejecución, poniéndolos en el archivo **ex.err**.

8. Restricciones Legales

En cuanto a RDS concierne, cualquier programa ejecutable que se cree con este **Traductor** se puede distribuir sin cobrar regalías. Puede incorporar a su aplicación cualquier archivo Euphoria provisto por RDS.

No se puede distribuir ninguna Edición Completa del Traductor, o ninguna librería en tiempo de ejecución que venga en la Edición Completa de ninguna plataforma.

No se puede distribuir ninguna versión "hackeada" o "crackeada" (ingeniería inversa) de ninguna librería o el Traductor, sin diferenciar que sea de la Edición de Dominio Público o de la Edición Completa.

En enero de 2000, Michael Devore donó al Dominio Público el expansor de DOS CauseWay, entregando su copyright, y alentando a todos a usarlo libremente, incluyendo el uso comercial.

En general, si desea usar código Euphoria escrito por terceros, tendría que hacer honor a cualquier restricción que se aplique. En caso de duda, debería consultar.

En Linux, FreeBSD y DJGPP para DOS32, la licencia de Librerías GNU normalmente no afectará a los programas creados con este **Traductor**. Simplemente por compilar con GNU C no le da a la Fundación de Software Libre (FSF) ninguna jurisdicción sobre su programa. Si enlaza estáticamente librerías de ellos, quedará sujeto a su licencia de Librerías, pero el procedimiento estándar de compilación/enlazado en **emake** no enlaza estáticamente ninguna librería de FSF, por lo que no debería tener ningún problema. La librería ncurses es la única enlazada estáticamente, y aunque FSF ahora mantiene el copyright, la licencia de Librerías GNU no se aplica a ncurses, debido a que fue donada a FSF por autores que no quieren que la licencia GNU se le aplique. Ver el aviso de copyright en ncurses.h.

Renuncia:

> Esto es lo que creemos es la cuestión. No somos abogados. Si usted lo cree importante, debería leer la licencia de Librerías GNU, la licencia de ncurses, los comentarios legales en DJGPP, Lcc y Borland, y el archivo read.me de Michael Devore en el sitio de Euphoria, para forma su propio juicio.

9. La Edición Completa del Traductor

Los programas (incluyendo .dll's y .so's) creados usando el Traductor **libre** de Dominio Público mostrarán un breve mensaje en la pantalla previa a la ejecución.

Al registrar la Edición Completa del Traductor, obtiene:

- La Edición Completa del Traductor para todas las plataformas y compiladores de C soportados.
- Ningún mensaje o demora al comienzo de la ejecución.
- **with trace** y **trace(3)** le permitirán ver los comandos de Euphoria que han sido ejecutados (por el código C) al momento de abortarse el programa. También verá un trazado de los (hasta) 500 comandos Euphoria precedentes. Buscar un archivo llamado **ctrace.out** (producido por el programa principal) o **ctrace−d.out** (producido por un .dll o .so) en el directorio actual. **ctrace.out** se usa como un búfer circular para mantener los comandos trazados. Una vez que se escriben 500 comandos en **ctrace.out**, el puntero del archivo se restablece al comienzo. Esto evita que el archivo se haga demasiado grande. El comando Euphoria que se está ejecutando cuando termina el programa es exactamente el anterior a la línea **"=== THE END ==="**. Si obtiene un mensaje de error en tiempo de ejecución, se le mostrará automáticamente.
- Con la mayoría de los compiladores de C, al presionar Ctrl+C se detiene la ejecución del programa. **ctrace.out** mostrará donde se detuvo el programa que se estaba ejecutando. Esto es una buena manera para diagnosticar ciclos infinitos.
- El programa se ejecutará mucho más despacio mientras se escribe **ctrace.out**. Puede ayudar en esta situación la invocación a **trace(3)** y **trace(0)** para elegir las porciones de su programa que desea trazar. También puede

usar **without trace** para eliminar el trazado en ciertas partes de su código.

- Las sentencias Euphoria se insertan como comentarios en el código fuente en C. De este modo, puede tener una idea más clara de como se implementan las sentencias Euphoria en C.
- Los beneficios comunes de ser un usuario registrado son:
  - ♦ actualizaciones gratuitas por 12 meses
  - ♦ descuentos en futuras actualizaciones
  - ♦ 3 meses gratuitos de soporte técnico de RDS
  - ♦ la posibilidad de votar $3.00 por mes en **Micro−Economy**

10. Respuestas a Preguntas Frecuentes (FAQ's)

P − ***¿Cuánta más velocidad debería esperar?***

R − Todo depende de lo que su programa haga. Los programas que principalmente hacen cálculos con enteros, no llaman frecuentemente rutinas de tiempo de ejecución y no hacen mucho uso de E/S, tendrán la mayor mejoría, comúnmente hasta 5 veces más rápido. Otros programas podrían tener solamente un pequeño porcentaje de mejoría.
Los distintos compiladores de C no tienen la misma capacidad de optimización. Watcom, GNU C y DJGPP producen el código más veloz. Borland es bastante bueno. Lcc queda levemente detrás de los otros, aún cuando se usa la opción −O.

El compilador de Borland es el más rápido, el de Watcom el más lento.

P − ***¿Podría registrar el Traductor sin primero registrar el Intérprete?***

R − Le resultará más fácil la depuración de programas grandes si tiene la Edición Completa del **Intérprete**. Idealmente, debería desarrollar y depurar su código a fondo con el **Intérprete**, y usar el **Traductor** como paso final para obtener velocidad. Sin embargo, puede comprar el Intérprete o el Traductor separadamente y comprar el otro más tarde.

P − ***Si compro el Código Fuente del Intérprete, ¿podré modificar la librería en tiempo de ejecución del Traductor?***

R − No. Sin embargo, podrá ver el código fuente de la mayoría de las rutinas en la librería en tiempo de ejecución del Traductor, y las versiones modificadas de los archivos de inclusión de algunas rutinas de su programa. También podrá construir su propio intérprete.

P − ***¿Qué pasa si quiero cambiar las opciones de compilación o enlazado en emake.bat?***

R − Usted es libre de hacerlo, sin embargo debería copiar **emake.bat** a su propio archivo llamado (digamos) mymake.bat, entonces ejecutar mymake.bat después de ejecutar el Traductor. Ocasionalmente, la cantidad de archivos .c producidos por el Traductor podría cambiar.

P − ***¿Cómo puedo hacer para que mi programa corra aún más rápido?***

R − Es importante declarar las variables como enteras toda vez que sea posible. En general ayuda si elige el tipo más restrictivo cuando declara las variables.
Típicamente, los tipos definidos por el usuario no hacen más lenta la ejecución. Como se supone que su programa está libre de errores de tipos, el Traductor ignorará los tipos, salvo que los llame directamente desde alguna función normal. La única excepción se da cuando la rutina de tipos definidos por el usuario tiene efectos secundarios (es decir, se establece una variable global, ejecuta "pokes" en memoria, E/S, etc). En ese caso, si **with type_check** está activo, el Traductor creará código para llamar a la rutina de tipos e informará de cualquier falla en la verificación de tipos que pueda ocurrir.

En Windows y DOS tuvimos que excluir la optimización de ciclos /ol para wcc386 de Watcom. Encontramos en un par de casos extraños que esta opción daba código de máquina incorrecto al usarse el compilador Watcom C. Si agrega esto en su versión de **emake.bat** podría obtener una pequeña mejoría en velocidad, con un leve riesgo de obtener código con errores. Para DJGPP debería probar −O6 en lugar de −O2.

Para DOS usamos la opción /fpc de Watcom que genera llamadas a las rutinas en tiempo de ejecución para realizar operaciones en punto flotante. Si la máquina tiene hardware de punto flotante, será usado por la rutina, en caso contrario se usará una emulación de software. Esto hace las cosas un poco más lentas, y no se necesita en los Pentiums, pero garantiza que su programa ejecutará en máquinas 386 y 486, aún cuando carezcan de hardware de punto flotante. La librería en tiempo de ejecución para DOS, **ec.lib**, ha sido hecha de esta forma, por lo tanto, simplemente no puede quitar esta opción.

En Linux o FreeBSD debería probar la opción O3 del gcc en lugar de O2. Esto hace pequeñas rutinas "in−line" mejorando levemente la velocidad, pero creando un ejecutable más grande.

11. Problemas Comunes

Muchos programas grandes se tradujeron y compilaron exitosamente usando cada uno de los compiladores de C soportados y el Traductor es ahora completamente estable.

Nota:

> En Windows, si llama una rutina de C que usa la convención de llamadas cdecl (en lugar de stdcall), tiene que especificar un carácter '+' al comienzo del nombre de la rutina en **define_c_proc()** y **define_c_func()**. Si no lo hace, la invocación puede funcionar al ejecutar el Intérprete **exw**, pero probablemente falle al traducir y compilar con Borland o Lcc.

En algunos casos, se quiere traducir a C una enorme rutina de Euphoria, pero resulta ser demasiado grande para que el compilador de C la procese. Si tiene este problema, haga su rutina Euphoria más pequeña y más simple. También pruebe desactivando la optimización de C en **emake.bat** solamente para el archivo .c que falle. También puede ayudar, dividir una declaración única de varias variables, en declaraciones separadas de una variable por vez. Euphoria no tiene limitación en el tamaño de las rutinas, o para el tamaño de un archivo, pero la mayoría de los compiladores de C la tienen. El Traductor automáticamente producirá múltiples archivos .c a partir de un archivo de Euphoria grande para evitar saturar el compilador de C. Si prefiere evitar esto, divida una rutina grande en varias más pequeñas.

Envíe los informes de errores a rds@RapidEuphoria.com
En particular, permítanos saber si cualquier programa traducido no se ejecuta del mismo modo de cuando está interpretado o cuando está compilado.

Sistema de base de datos de Euphoria (EDS)

Introducción

Mucha gente ha mostrado interés en acceder a bases de datos usando programas Euphoria. Algunos querían acceder a sistemas de gestión de base de datos de marca conocida desde Euphoria, otros querían una base de datos orientada a Euphoria simple y fácil de usar. EDS es lo más reciente. **Provee un sistema de base de datos simple y extremadamente flexible, para usar con programas Euphoria.**

Estructura de una base de datos EDS

En EDS, una **base de datos** es un archivo único con extensión **.edb**. Una base de datos EDS contiene 0 o más **tablas**. Cada tabla tiene un **nombre**, y contiene 0 o más **registros**. Cada registro consta de una parte de **clave**, y otra de **datos**. Laclave puede ser **cualquier** objeto de Euphoria − un átomo, una secuencia, secuencias anidadas, etc. Del mismo modo, los datos pueden ser **cualquier** objeto de Euphoria. **No** hay restricciones en el tamaño o estructura de la clave o los datos. Dentro de una tabla, las claves son únicas. Esto significa, no hay dos registros que tengan la misma clave.

Los registros de una tabla se almacenan en el orden ascendente del valor de la clave. Se utiliza una eficiente búsqueda binaria para referirse a los registros por clave. También se puede acceder a un registro directamente, sin búsqueda alguna, conociendo el **número de registro** dentro de la tabla. Los números de registro son enteros desde 1 al tamaño de la tabla (cantidad de registros). Incrementando el número de registro se pueden recorrer eficientemente todos los registros. Sin embargo tenga en cuenta que el número de registro puede cambiar toda vez que se inserte un nuevo registro, o cuando un registro se borra.

Las partes de datos y claves se almacenan en forma compacta, pero **no** se pierde exactitud al guardar o recuperar números de punto flotante o **cualquier** otro dato de Euphoria.

database.e trabajará sin cambios, en **Windows**, **DOS**, **Linux** o **FreeBSD**. El código se ejecuta casi el doble de rápido en Linux/FreeBSD que en DOS/Windows. **Los archivos de la base de datos EDS se pueden copiar y compartir entre programas que corren en Linux/FreeBSD y DOS/Windows.** Asegúrese de hacer una copia exacta byte por byte usando el modo de copiado "binario", en lugar del modo "texto" o "ASCII" porque podrían cambiar los terminadores de línea.

```
Ejemplo:
   base de datos: "misdatos.edb"
         primera tabla: "accesos"
              registro #1:  clave: "García"   dato: "gordito"
              registro #2:  clave: "Pérez"    dato: "delgado"

         segunda tabla: "repuestos"
              registro #1:  clave: 134525    dato: {"martillo", 15.95, 500}
              registro #2:  clave: 134526    dato: {"sierra", 25.95, 100}
              registro #3:  clave: 134530    dato: {"destornillador", 5.50, 1500}
```

Este ejemplo es para interpretar el significado de la clave y de los datos. **En consonancia con el espíritu de Euphoria, Ud. tiene flexibilidad total.** A diferencia de otros sistemas de bases de datos, ***no*** se necesita que un registro EDS tenga un número fijo de campos, o campos con una longitud máxima predefinida.
En algunos casos no habrá ningún valor natural para la clave en los registros. En esos casos, se debería crear una clave entera sin significado alguno, pero única. Recordar que siempre se puede acceder a los datos por número de registro. Es fácil recorrer los registros buscando el valor de un campo en particular.

¿Cómo acceder a los datos?

Para reducir la cantidad de parámetros que se tienen que pasar, existe la noción de **base de datos en uso**, y **tabla en uso**. La mayoría de las rutinas usan este valor ***en uso*** (o actual) automáticamente. Normalmente se empieza por abrir (o crear) un archivo de base de datos, entonces se selecciona la tabla con la que se trabajará.

Usando **db_find_key()**, se puede mapear una clave a un número de registro. Este comando usa una eficiente búsqueda binaria. La mayoría de las otras rutinas a nivel de registro, esperan como argumento el número de registro. Dado el número de registro, se accede muy rápidamente a cualquier registro. También se puede acceder a todos los registros comenzando por el número 1 y recorriéndolos todos hasta el número de registro devuelto por **db_table_size()**.

¿Cómo se recicla el almacenamiento?

Al borrar algo, como un registro, el espacio de ese ítem se marca como disponible en una lista áreas libres para uso posterior. Las áreas libres adyacentes se combinan en grandes áreas libres. Al necesitar más espacio y no teniéndolo disponible en la lista de áreas libres, el archivo crece en tamaño. Normalmente no hay un método automático para comprimir el tamaño del archivo, pero se puede usar **db_compress()** para reescribir completamente la base de datos, removiéndole los espacios sin uso.

Seguridad / Acceso multiusuario

Esta versión provee una forma sencilla de bloqueo de la base de datos para prevenir de accesos inseguros de parte de otros procesos.

Escalabilidad

Los punteros internos son de 4 bytes. En teoría, el tamaño máximo del archivo de una base de datos es 4 Gb. En la práctica, el límite es de 2 Gb debido a limitaciones en varias funciones de archivo de C usadas por Euphoria. Bajo pedido de los usuarios, las bases de datos EDS podrían expendirse en el futuro, más allá de los 2 Gb.

El algoritmo actual asigna 4 bytes de memoria por registro en la tabla actual. Por lo tanto, se necesitan como mínimo 4Mb de RAM por cada millón de registros en disco.

La búsqueda binaria por claves debería funcionar razonablemente bien con tablas grandes.

Las inserciones y borrados tardan levemente un poco más, según la tabla se haga más grande.

En el otro extremo de la escala, es posible crear bases de datos extremadamente pequeñas sin incurrir en mucho gasto de espacio en disco.

Renuncia

No almacenar datos importantes sin realizar una copia de seguridad. RDS no se hará responsable de cualquier daño o pérdida de información.

Rutinas de base de datos

En las descripciones posteriores, se usan los siguientes prefijos para indicar el tipo de **objeto** que se puede pasar o devolver:

x – un objeto general (átomo o secuencia)

s – una secuencia

a – un átomo

i – un entero

fn – un entero usado como número de archivo

st – una secuencia de cadena, o un átomo de caracter simple

db_create – crear una nueva base de datos
db_open – abrir una base de datos existente
db_select – seleccionar una base de datos para convertirla en la base de datos en uso
db_close – cerrar una base de datos
db_create_table – crear una nueva tabla en una base de datos
db_select_table – seleccionar una tabla para convertirla en la tabla en uso
db_delete_table – borrar una tabla
db_table_list – obtener un listado de todas las tablas de la base de datos
db_table_size – obtener la cantidad de registros de la tabla en uso
db_find_key – encontrar rápidamente el registro con un cierto valor de clave
db_record_key – obtener la clave de un registro
db_record_data – obtener los datos de un registro
db_insert – insertar un nuevo registro en la tabla en uso
db_delete_record – borrar un registro de la tabla en uso
db_replace_data – reemplazar los datos de un registro

db_create

Sintaxis: include database.e
i1 = db_create(s, i2)

Descripción: Crea una nueva base de datos. Una nueva base de datos se creará en el archivo con ruta dada por 's'. 'i2' indica el tipo de bloqueo que se aplicará al archivo creado. 'i1' es un código de error que indica éxito o fracaso. Los valores de 'i2' pueden ser tanto DB_LOCK_NO (sin bloqueo) o DB_LOCK_EXCLUSIVE (bloqueo exclusivo). Si 'i1' es DB_OK, significa que la base de datos se creó exitosamente. Esta base de datos se convierte en la **base de datos en uso** a la que se aplicarán las demás operaciones de base de datos.

Comentarios: Si la ruta 's' no termina en .edb, esta extensión se agregará automáticamente.
Si la base de datos ya existe, no se sobreescribirá. ***db_create()*** devolverá DB_EXISTS_ALREADY.

El número de versión se almacena en el archivo de base de datos. De esta forma, las futuras versiones del software de la base de datos podrán reconocer el formato, leerlo y operar con él de alguna manera.

Ejemplo:

```
if db_create("misdatos", DB_LOCK_NO) != DB_OK then
    puts(2, "No se puede crear la base de datos!\n")
    abort(1)
end if
```

Ver también: ***db_open***, ***db_close***

db_open

Sintaxis: include database.e
i1 = db_open(s, i2)

Descripción: Abre una base de datos Euphoria existente. El archivo que contiene la base de datos está dado por 's'. 'i1' es un código de retorno que indica el éxito o fracaso de la operación. 'i2' indica el tipo de bloqueo que quiere aplicar a la base de datos mientras la abre. Esta base de datos se convierte en la **base de datos en uso** a la que se aplicarán las demás operaciones de base de datos.
Los códigos de retorno son:

```
global constant DB_OK = 0   -- éxito
         DB_OPEN_FAIL = -1  -- no se puede abrir el archivo
         DB_LOCK_FAIL = -3  -- no se puede bloquear el archivo de la
                            --    forma pedida
```

Comentarios: Los tipos de bloqueo que se pueden usar son: DB_LOCK_NO (sin bloqueo), DB_LOCK_SHARED (bloqueo compartido para acceso de lectura solamente) y DB_LOCK_EXCLUSIVE (para acceso de lectura/escritura). DB_LOCK_SHARED está soportado solamente por **Linux/FreeBSD**. Permite leer la base de datos, sin escribir absolutamente nada en ella. Si se solicita DB_LOCK_SHARED en **WIN32** o **DOS32**, será tratado como si el pedido se hubiera efectuado con DB_LOCK_EXCLUSIVE.
Si el bloqueo falla, su programa debería esperar unos segundos antes de reintentarlo. Otro proceso podría estar accediendo a la base de datos en ese momento.

Típicamente, los programas DOS mostrarán un mensaje de "error crítico" si intentan acceder a una base de datos que está bloqueada.

Ejemplo:

```
tries = 0
while 1 do
    err = db_open("misdatos", DB_LOCK_SHARED)
    if err = DB_OK then
        exit
    elsif err = DB_LOCK_FAIL then
        tries += 1
        if tries > 10 then
            puts(2, "demasiados intentos, lo siento\n")
            abort(1)
        else
            sleep(5)
        end if
    else
```

```
            puts(2,"No se puede abrir la base de datos!\n")
            abort(1)
        end if
    end while
```

Ver también: ***db_create***, ***db_close***

db_select

Sintaxis: include database.e
i = db_select(s)

Descripción: Selecciona una nueva base de datos ya abierta, para convertirla en la **base de datos en uso**. Las operaciones subsecuentes de base de datos se aplicarán a esta base. 's' es la ruta del archivo de base de datos que originalmente se abrió con ***db_open()*** o ***db_create()***. 'i' es un código de retorno que indica el éxito de la operación (DB_OK) o su fracaso.

Comentarios: Al crear (***db_create()***) o abrir (***db_open()***) una base de datos, ésta se convierte en la **base de datos en uso**. Use ***db_select()*** cuando quiera cambiar entre bases de datos abiertas, por ejemplo, para copiar registros de una a otra.
Después de seleccionar una nueva base de datos, se debería seleccionar una tabla de esa base, usando ***db_select_table()***.

Ejemplo:

```
if db_select("empleados") != DB_OK then
    puts(2, "No se puede seleccionar la base de datos de empleados\n")
end if
```

Ver también: ***db_open***

db_close

Sintaxis: include database.e
db_close()

Descripción: Desbloquea y cierra la **base de datos en uso**.

Comentarios: Llamar a este procedimiento cuando se haya terminado de usar la **base de datos en uso**. Se removerá cualquier bloqueo, permitiendo a otros procesos acceder al archivo de base de datos.

Ver también: ***db_open***

db_create_table

Sintaxis: include database.e
i = db_create_table(s)

Descripción: Crea una tabla nueva dentro de la **base de datos en uso**. El nombre de la tabla está dado por la secuencia de caracteres 's' y no puede ser igual al de ninguna otra tabla existente en la **base de datos en uso**.

Comentarios: La tabla que se crea tiene inicialmente 0 registros y además, se convierte en la **tabla en uso**.

Ejemplo:

```
if db_create_table("mi_nueva_tabla") != DB_OK then
    puts(2, "No se puede crear mi_nueva_tabla!\n")
end if
```

Ver también: ***db_delete_table***

db_select_table

Sintaxis: include database.e
i = db_select_table(s)

Descripción: La tabla cuyo nombre está dado por 's', se convierte en la **tabla en uso**. El código de retorno 'i' será DB_OK si la tabla existe en la **base de datos en uso**, de lo contrario obtendrá DB_OPEN_FAIL.

Comentarios: All record−level database operations apply automatically to the tabla en uso.

Ejemplo:

```
if db_select_table("salario") != DB_OK then
    puts(2, "No se puede hallar la tabla salario!\n")
    abort(1)
end if
```

Ver también: *db_create_table*, *db_delete_table*

db_delete_table

Sintaxis: include database.e
delete_table(s)

Descripción: Borrar una tabla de la **base de datos en uso**. El nombre de la tabla está dado por 's'.

Comentarios: Se borran todos los registros y el espacio ocupado por la tabla se libera. Si la tabla es la **tabla en uso**, ésta queda indefinida.
Si no existe una tabla con el nombre dado por 's', no ocurre nada.

Ver también: *db_create_table*, *db_select_table*

db_table_list

Sintaxis: s = db_table_list()

Descripción: Devuelve una secuencia con todos los nombres de tablas de la **base de datos en uso**. Cada elemento de 's' es una secuencia de caracteres que contiene el nombre de una tabla.

Ejemplo:

```
sequence names

names = db_table_list()
for i = 1 to length(names) do
    puts(1, names[i] &'\n')
end for
```

Ver también: *db_create_table*

db_table_size

Sintaxis: include database.e
i = db_table_size()

Descripción: Devuelve la cantidad de registros de la **tabla en uso**.

Ejemplo:

```
-- recorre todos los registros de la tabla en uso
for i = 1 to db_table_size() do
    if db_record_key(i) = 0 then
        puts(1, "No se encontró la clave\n")
        exit
    end if
end for
```

Ver también: *db_select_table*

db_find_key

Sintaxis: include database.e
i = db_find_key(x)

Descripción: Busca el registro de la **tabla en uso** con el valor de clave 'x'. Si se encuentra, se devuelve el número de registro. Si no se encuentra, se devuelve el número de registro que debería tener la clave, como número negativo.

Comentarios: Se usa una búsqueda binaria para encontrar la clave en la ***tabla en uso***. La cantidad de comparaciones es proporcional al logaritmo de la cantidad de registros en la tabla.
Se puede seleccionar un rango de registros, buscando los valores primero y último de la clave. Si esos valores de la clave no existen, al menos se obtiene un valor negativo que muestra donde debería estar si existiesen. Suponer que se quiere saber cuales registros tienen claves mayores que "GGG" y menores que "MMM". Si se obtiene −5 para la clave "GGG", significa que un registro de clave "GGG" debería insertarse como registro número 5. −27 para "MMM" significa que un registro de clave "MMM" debería insertarse como registro número 27. Esto indica rápidamente que todos los registros , >= 5 y < 27 cumplen la condición.

Ejemplo:

```
rec_num = db_find_key("Millennium")
if rec_num > 0 then
    ? db_record_key(rec_num)
    ? db_record_data(rec_num)
else
    puts(2, "No se encuentra, pero si lo insertara,\n")
    puts(2, "su número de registro sería #%d\n", -rec_num)
end if
```

Ver también: ***db_record_key***, ***db_record_data***, ***db_insert***

db_record_key

Sintaxis: include database.e
x = db_record_key(i)

Descripción: Devuelve la parte de clave del número de registro indicado por 'i' de la **tabla en uso**.

Comentarios: Cada registro en una base de datos Euphoria consta de una parte de clave y otra de datos, pudiendo ser cada uno de ellos cualquier átomo o secuencia de Euphoria.

Ejemplo:

```
puts(1, "La clave del registro Nº 6 es: ")
? db_record_key(6)
```

Ver también: ***db_record_data***

db_record_data

Sintaxis: include database.e
x = db_record_data(i)

Descripción: Devuelve la parte de datos del número de registro indicado por 'i' de la **tabla en uso**.

Comentarios: Cada registro en una base de datos Euphoria consta de una parte de clave y otra de datos, pudiendo ser cada uno de ellos cualquier átomo o secuencia de Euphoria.

Ejemplo:

```
puts(1, "El dato del registro Nº 6 es: ")
? db_record_data(6)
```

Ver también: ***db_record_key***

db_insert

Sintaxis: include database.e
i = db_insert(x1, x2)

Descripción: Inserta un nuevo registro en la **tabla en uso**. La clave del registro es 'x1' y el dato del registro es 'x2'. Tanto 'x1' como 'x2' pueden ser cualquier objeto de datos de Euphoria, átomos o secuencias. El código de retorno 'i1' será DB_OK si el registro se inserta.

Comentarios: Dentro de una tabla, todas las claves tienen que ser únicas. Si ya existe un registro con ese mismo valor de clave, ***db_insert()*** fallará obteniendo DB_EXISTS_ALREADY.

Ejemplo:

```
if db_insert("Pérez", {"Pedro", 100, 34.5}) != DB_OK then
    puts(2, "Falló la inserción del registro!\n")
end if
```

Ver también: *db_find_key*, *db_record_key*, *db_record_data*

db_delete_record

Sintaxis: include database.e
db_delete_record(i)

Descripción: Borra el número de registro indicado por 'i' de la **tabla en uso**.

Comentarios: El número de registro 'i' tiene que ser un entero entre 1 y la cantidad de registros de la **tabla en uso**.

Ejemplo:

```
db_delete_record(55)
```

Ver también: *db_insert*, *db_table_size*

db_replace_data

Sintaxis: include database.e
db_replace_data(i, x)

Descripción: Reemplaza los datos de número de registro 'i' en la **tabla en uso**, por 'x'. 'x' puede ser cualquier secuencia o átomo de Euphoria.

Comentarios: El número de registro 'i' tiene que estar entre 1 y la cantidad de registros de la ***tabla en uso***.

Ejemplo:

```
db_replace(67, {"Pedro", 150, 34.5})
```

Ver también: *db_delete_record*

db_compress

Sintaxis: include database.e
i = db_compress()

Descripción: Comprime la **base de datos en uso**. La **base de datos en uso** se copia a un nuevo archivo, de forma tal que los bloques de espacio sin usar se eliminan. Si no ocurre ningún error, 'i' se establece a DB_OK, y el nuevo archivo de base de datos mantendrá el mismo nombre. Por seguridad, el archivo original descomprimido se renombrará con la extensión .t0 (o .t1, .t2 ,..., .t99). Si la compresión no tiene éxito, la base de datos permancerá sin cambios y la copia de seguridad no se realizará.

Comentarios: Al borrar ítems de una base de datos, se crean bloques de espacio vacío dentro del archivo. El sistema los tiene en cuenta, intentando usarlos como nuevo espacio de almacenamiento cada vez que se realiza una inserción de datos. ***db_compress()*** copiará la **base de datos en uso** sin incluir esas áreas libres. El tamaño del archivo de base de datos, por ende, disminuirá.
Si los archivos de copia de seguridad alcanzan la extensión .t99, se deberán borrar alguno de ellos.

Ejemplo:

```
if db_compress() != DB_OK then
    puts(2, "Falló la compresión!\n")
end if
```

Ver también: *db_create*

db_dump

Sintaxis:

include database.e
db_dump(fn, i)

Descripción: Vuelca los contenidos de una base de datos Euphoria ya abierta. Los contenidos se descargan en un archivo o dispositivo 'fn'. Se muestran todos los registros de todas las tablas. Si 'i' no es cero, entonces también se muestra el volcado byte a byte de bajo nivel. El volcado de bajo nivel solamente será significativo para quienes conozcan el formato interno de una base de datos Euphoria.

Ejemplo:

```
if db_open("misdatos", DB_LOCK_SHARED) != DB_OK then
    puts(2, "No se puede abrir la base de datos!\n")
    abort(1)
end if
fn = open("db.txt", "w")
db_dump(fn, 0)
```

Ver también: ***db_open***

db_fatal_id

Sintaxis: include database.e
db_fatal_id = i

Descripción: Se pueden capturar ciertos errores fatales de base de datos, instalando un gestor de errores. Simplemente se sobreescribe la variable global ***db_fatal_id*** con un identificador de una rutina creada por el programador. El procedimiento tiene que tomar un argumento único, como una secuencia. Al ocurrir ciertos errores, se llamará a este procedimiento con un mensaje de error como argumento. Este procedimiento debería terminar con una llamada a ***abort()***.

Ejemplo:

```
procedure mi_error_fatal(sequence msg)
    puts(2, "Ocurrió un error fatal - " &msg &'\n')
    abort(1)
end procedure

db_fatal_id = routine_id("mi_error_fatal")
```

Ver también: ***db_close***

Encriptación y Enlazado

(solamente para la Edición Completa)

El Comando Shroud

Sinopsis:

```
shroud [-clear] [-list] [-quiet] [-out archivo_encriptado] [archivo]
```

El comando **shroud** convierte un programa Euphoria, que típicamente consta de un archivo principal más algunos archivos include, en un archivo único y enmascarado (es decir, encriptado) que se puede distribuir fácilmente a terceros. Por defecto, el comando **shroud** realizará los siguientes pasos:

1. − Su archivo **.ex**, **.exw** o **.exu** principal se combina con los demás archivos **.e** que directa o indirectamente incluye. Esto resulta en un archivo Euphoria sin instrucciones include.
2. − Después de realizar una pasada al programa entero, cualquier constante o rutina no utilizada, se marca para borrado. Aquí pueden aparecer constantes y rutinas adicionales que no se usan. Este proceso de marcado se repite hasta que no hayan más constantes o rutinas para borrar. En una segunda pasada, aquellas constantes o rutinas marcadas se saltean, es decir, no se copian al archivo encriptado. Es común que un programa incluya muchos archivos, pero sólo una parte de las rutinas o constantes realmente se usen.
3. − Se eliminan todos los comentarios, líneas en blanco, espacios y tabuladores superfluos.
4. − Para ahorrar espacio, se reemplazan todas las palabras clave y todos los nombres de rutinas internas por códigos de byte simple.
5. − Todos los nombres definidos por el usuario se convierten a códigos breves sin significado (una o dos letras) elegidos por el programa ***shroud***.
6. − De los pasos 1 a 5, resulta un archivo muy compacto y encriptado que es completamente ilegible y prácticamente inviolable.

Las *opciones* pueden ser:

−clear − Mantiene el código fuente legible. Las rutinas y constantes no utilizadas serán borradas y los comentarios y algunos espacios removidos sin que se altere el código. Se mantienen los nombres originales de variables y rutinas, excepto cuando exista un conflicto entre varios archivos. Use esta opción cuando quiera distribuir un solo archivo de código, pero sin importarle que se pueda leer el código. Si ocurre un error mientras el programa se está ejecutando, el archivo **ex.err** será perfectamente legible. Si el programa está encriptado, el archivo **ex.err** contendrá nombres breves sin significado, lo que lo hará muy difícil de comprender.

−list − Produce un listado en **deleted.txt** de las rutinas y constantes borradas, así como de cualquier símbolo que haya sido renombrado.

−quiet − Suprime los mensajes normales y las estadísticas. Solamente informa los errores.

−out *archivo_encriptado* − Escribe la salida a archivo_encriptado. Si no se especifica la opción ***−out***, se le pedirá un nombre para el archivo de salida.

Si escribe simplemente:

```
shroud
```

sin ninguna opción o nombre de archivo, se le pedirá toda información.

shroud solamente realiza una verificación muy superficial de la sintaxis de su programa. Deberá probar su programa extensamente antes de encriptarlo o enlazarlo.

Se puede distribuir un archivo de inclusión ***.e*** **encriptado** para que la gente pueda incluirlo en sus programas sin que puedan leer su código fuente. Los símbolos declarados como **global** en su archivo **.e** principal no se renombrarán o borrarán, por lo tanto sus usuarios podrán acceder a rutinas y variables mediante nombres convencionales.

Se puede **encriptar** o **enlazar** un programa que incluye un ***archivo de inclusión encriptado***, sin embargo no se permite usar la opción ***−clear***, debido a que esto reduciría la seguridad del archivo de inclusión encriptado.

Por seguridad, el intérprete Euphoria no realizará ningún trazado sobre un archivo encriptado, salvo que haya sido encriptado con la opción ***−clear.*** En la mayoría de los casos es mucho mejor trazar el código fuente original.

Solamente RDS tiene el conocimiento necesario para deshacer la encriptación de un programa, pero no tenemos una herramienta para hacerlo. Aún cuando alguien pudiera deshacer la encriptación, sólo podría recuperar la versión fuente obtenida trás la aplicación de los pasos 1 a 5. Los comentarios y los nombres originales de las rutinas y variables no se pueden recuperar ***nunca***. **Siempre mantenga una copia de sus archivos fuente originales!**

El Comando Bind

Synopsis:

```
bind  [-clear] [-list] [-quiet] [-out archivo_ejecutable] [archivo.ex]
bindu [-clear] [-list] [-quiet] [-out archivo_ejecutable] [archivo.exu]
bindw [-clear] [-list] [-quiet] [-out archivo_ejecutable] [-icon archivo.ico] [archivo.exw]
```

bind (**bindw** or **bindu**) hace lo mismo que **shroud**, y tiene las mismas opciones. Combina su archivo encriptado (o texto limpio) con la Edición de Dominio Público de **ex.exe**, **exw.exe** o **exu** para hacer un archivo **ejecutable único e independiente** que Ud. puede usar y distribuir convenientemente. Sus usuarios no necesitan tener instalado Euphoria. Cada vez que su ejecutable corre, se realiza una verificación rápida de integridad para detectar cualquier intento de alteración o corrupción del archivo.

Por seguridad, el intérprete Euphoria no realizará ningún trazado sobre un archivo enlazado, salvo que haya sido enlazado con la opción ***−clear.*** En la mayoría de los casos es mucho mejor trazar el código fuente original.

Las *opciones* pueden ser:

−clear	− Lo mismo que en **shroud**. El .exe contendrá código legible. Si ocurre un error, el archivo **ex.err** también será legible.
−list	− Lo mismo que en **shroud**.
−quiet	− Lo mismo que en **shroud**.
−out *archivo_ejecutable*	− Esta opción permite elegir el nombre del archivo ejecutable creado por ***bind***. Sin esta opción, ***bind*** elegirá un nombre basado en el nombre del principal archivo Euphoria fuente.
−icon *archivo[.ico]*	− **(solamente bindw)** Cuando enlaza un programa, puede sustituir el ícono original del archivo ***exw.exe*** por otro personalizado de 32x32 usando 256 colores. El original es una figura que se asemeja a **E)**. Windows mostrará en los listados de archivos, esta figura próxima a ***exw.exe***, a su programa enlazado. También se puede cargar este ícono como un recurso, usando el nombre "exw" (ver ***euphoria\demo\win32\window.exw*** como ejemplo). Al enlazar su programa, puede sustituir el ícono original por otro de 32x32, 256 colores de 2238 bytes o menos de tamaño. Se pueden usar otras dimensiones, en tanto su tamaño sea menor o igual a 2238 bytes. El archivo debe contener un único ícono de imagen (según sea necesario, Windows lo agrandará o achicará). En la Edición Completa se incluye el archivo de ícono por defecto **E)**, euphoria.ico. Puede enlazarlo, o distribuirlo separadamente, con o sin cambios.

Si escribe simplemente:

```
bind (o bindw o bindu)
```

sin ninguna opción o nombre de archivo, se le pedirá toda información.

Sólo se pueden enlazar los intérpretes de Edición de Dominio Público. Los usuarios de la Edición Completa de Euphoria para DOS32 + WIN32 tendrán **ex.exe** (Ed. Completa) y **pdex.exe** (Dominio Público), así como **exw.exe** (Ed. Completa) y **pdexw.exe** (Dominio Público) en **euphoria\bin**. El programa **bind** (**bindw**) usará el archivo **pdex.exe** (**pdexw.exe**) para enlazar. En Linux o FreeBSD, tendrá **exu** (Ed. Completa) y **pdexu** (Dominio Público), con **pdexu** usado para enlazar.

Un programa Euphoria de una sola línea creará un archivo tan grande como el intérprete con el que está binding, pero el tamaño se incrementa muy lentamente a medida que agrega líneas al programa. **Al enlazar, el editor entero de Euphoria, ed.ex, agrega sólo 18K al tamaño del intérprete.** Los tres intérpretes están comprimidos para reducir su tamaño. **exw.exe** y **exu** están comprimidos con **UPX** (ver http://upx.sourceforge.net). **ex.exe** está comprimido con la herramienta que viene con el expansor CauseWay DOS. **ex.exe** es el más grande de los tres, debido a que incluye varias

rutinas gráficas que no forman parte de **exw.exe** o **exu**. Nota: En algunos casos, un ejecutable comprimido puede disparar un mensaje de alerta desde el programa antivirus. Esto se debe a que el programa antivirus ve al archivo comprimido como anormal. Si ***demo\sanity.ex*** se ejecuta correctamente, puede ignorar con total seguridad esas alertas. En caso contrario contáctese con RDS.

Los primeros dos argumentos devueltos por la rutina de librería **command_line()** serán levemente diferentes cuando el programa está enlazado. Ver **library.doc** por más detalles.

Un archivo **ejecutable** **enlazado** ***puede*** manejar la redirección de las entrada y salida estándares, por ejemplo:

```
myprog.exe <file.in > file.out
```

Si escribió un pequeño archivo **.bat** de DOS (por ejemplo, **miprog.bat**) que contenía la línea "**ex miprog.ex**" *no* podrá redirigir la entrada y salida estándares de la siguiente forma:

```
miprog.bat <file.in > file.out     (no funciona en DOS!)
```

Sin embargo, *debería* usar la redirección en líneas individuales dentro de archivos **.bat**.

Consejos de rendimiento de Euphoria

Consejos generales

- Si su programa es suficientemente rápido, olvídese de acelerarlo. Solo hágalo simple y legible.
- Si su programa es demasiado lento, los consejos de más abajo probablemente no solucionen su problema. Debería encontrar un mejor algoritmo.
- La forma más fácil de ganar un poco de velocidad es desactivar la verificación de tipos en tiempo de ejecución. Inserte esta línea:

```
without type_check
```

  al comienzo de su archivo principal **.ex**, delante de cualquier sentencia de inclusión. Típicamente ganará entre el 0 y 20%, dependiendo de los tipos definidos y los archivos que incluya. La mayoría de los archivos de inclusión estándares hace alguna verificación de tipos definidos por el usuario. Un programa que está completamente libre de verificaciones, debería acelerar ligeramente.
  También, para ***asegurarse*** quite o comente cualquier sentencia

```
with trace
with profile
with profile_time
```

  with trace (aún sin ninguna llamada a ***trace()***), y **with profile** pueden provocar fácilmente una caida del 10% o más. **with profile_time** podría disminuir la velocidad en el 1%. Cada una de estas opciones, también, consumirá memoria adicional.
- Los cálculos usando enteros son más rápidos que los basados en números de punto flotante .
- Declare las variables como enteros, en lugar de átomos, toda vez que sea posible y secuencias, en lugar de objetos. Esto le hará ganar un poco de velocidad.
- En una expresión que realiza cáculos en punto flotante, normalmente es más rápida si escribe los números constantes en forma de punto flotante, por ejemplo, si 'x' tiene un valor en punto flotante, digamos , x = 9.9

  cambie:

```
x = x * 5
```

  a:

```
x = x * 5.0
```

  Esto le evita al intérprete tener que convertir el entero 5 al punto flotante 5.0
- Euphoria hace la evaluación de ***corto circuito*** de las condiciones de **if**, **elsif**, y **while** que involucran a **and** y **or**. Euphoria detendrá la evaluación de cualquier condición una vez que determine si la condición es verdadera o no. Por ejemplo, en la ***sentencia if***:

```
if x > 20 and y = 0 then
    ...
end if
```

  La prueba "y = 0" solamente se hará si "x > 20" es verdadera.
  Para obtener máxima velocidad, puede ordenar sus pruebas. Haga "X > 20" primero, si es más probable que sea falso que "y = 0".

  En general, con una condición "A and B", Euphoria no evaluará la expresión B cuando A es falsa (cero). Análogamente, con una condición como "A or B", B no se evaluará si A es verdadera (no cero).

  Las sentencias if simples están altamente optimizadas. Con la versión actual del intérprete, los 'if' simples anidados que comparan enteros, son comúnmente un poco más rápidos que una sentencia if única de corto circuito, por ejemplo:

```
if x > 20 then
    if y = 0 then
        ...
    end if
end if
```

- La velocidad de acceso a las variables privadas, locales y globales es la misma.
- No hay penalización de rendimiento por definir constantes, en lugar de escribir números literales en el código. La velocidad de:

```
y = x * MAX
```

  es exactamente la misma que:

```
y = x * 1000
```

donde se definió previamente:

```
constant MAX = 1000
```

- No hay penalización de rendimiento por tener muchos comentarios en su código. Los comentarios se ignoran por completo. No se ejecutan de ningún modo. Podría demorar unos pocos milisegundos más la carga inicial del programa, pero esto es un costo muy pequeño a pagar por la futura mantenibilidad del código, y cuando **enlaza** su programa, todos los comentarios se eliminan, por lo que el costo se hace absolutamente nulo.

Midiendo el rendimiento

En cualquier lenguaje de programación, y especialmente en Euphoria, **tiene que hacer realmente mediciones antes de sacar conclusiones acerca del rendimiento.**

Euphoria provee tanto **análisis por conteo de ejecución**, como **por tiempo** (solo **DOS32**). Lea **refman.doc**. Frecuentemente se sorprenderá con los resultados de estos perfiles. Concentre sus esfuerzos en los lugares del programa que usan un alto porcentaje del tiempo total (o al menos, los que se ejecutan una gran cantidad de veces). No tiene sentido reescribir una sección del código que usa el 0.01% del tiempo total. Comúnmente habrá un lugar, o unos pocos donde este código haga una diferencia significativa.

También puede medir la velocidad del código usando la función **time()**, por ejemplo:

```
atom t
t = time()
for i = 1 to 10000 do
    -- pequeña porción de código
end for
? time() - t
```

Tendría que reescribir la pequeña porción de código de formas diferentes, para determinar cual es la forma más rápida.

¿Cómo acelerar los ciclos?

El **análisis de perfiles de ejecución** le mostrará los ***puntos calientes*** de su programa. Normalmente están dentro de los ciclos. Observe cada cálculo dentro de un ciclo y pregúntese si realmente es necesario que ocurran cada vez que el ciclo se ejecuta, o si podría hacerse una vez y fuera del ciclo.

Convirtiendo multiplicaciones en sumas con un ciclo

La suma es más rápida que la multiplicación. A veces puede reemplazar una multiplicación por la variable de ciclo con una suma. Algo como:

```
for i = 0 to 199 do
    poke(screen_memory+i*320, 0)
end for
```

se convierte en:

```
x = screen_memory
for i = 0 to 199 do
    poke(x, 0)
    x = x + 320
end for
```

Guardando resultados en variables

- Es más rápido guardar el resultado de un cálculo en una variable, que recalcular todo más tarde. Aún a veces en cosas tan simples como una operación con índices, o sumar 1 a la variable son dignas del ahorro.
- Cuando tiene una secuencia con múltiples niveles de índices, es más rápido cambiar el código:

```
for i = 1 to 1000 do
    y[a][i] = y[a][i]+1
end for
```

a:

```
ya = y[a]
```

```
for i = 1 to 1000 do
    ya[i] = ya[i] + 1
end for
y[a] = ya
```

Por lo que hace 2 operaciones con índices por repetición, en lugar de 4. Las operaciones, 'ya = y[a]' y 'y[a] = ya' son muy livianas. **Solo copian un puntero.** No copian la secuencia completa.

- Hay un leve costo cuando crea una nueva secuencia usando **{a,b,c}**. Si es posible, mueva esta operación fuera de un ciclo crítico, almacenándola en una variable antes del ciclo y referenciando la variable dentro del ciclo.

Llamadas de rutinas en−línea

Si tiene una rutina pequeña y rápida, pero se la llama una enorme cantidad de veces, ahorrá tiempo haciendo la operación ***en−línea***, en lugar de llamar la rutuina. Su código puede hacerse menos legible, por lo que sería mejor usar la operación en−línea, solamente en los lugares que generan un montón de llamadas a la rutina.

Operaciones sobre secuencias

Euphoria le permite operar en una secuencia grande de datos usando una sentencia única. Esto le ahorra de escribir un ciclo donde se procesa un elemento por vez. Por ejemplo:

```
x = {1,3,5,7,9}
y = {2,4,6,8,10}

z = x + y
```

contra:

```
z = repeat(0, 5)  -- si es necesario
for i = 1 to 5 do
    z[i] = x[i] + y[i]
end for
```

En la mayoría de los lenguajes interpretados, es mucho más rápido procesar una secuencia (array) completa en una sentencia, que hacer operaciones escalares dentro de un ciclo. Esto se debe a que el intérprete tiene una gran cantidad de consumo de recursos para cada sentencia que ejecuta. Euphoria es diferente. Euphoria es muy liviano, con poco consumo de recursos interpretativos, asique las operaciones sobre secuencias no siempre ganan. La única solución es medir el tiempo en ambas formas. El costo por elemento es comúnmente más bajo cuando se procesa una secuencia dentro de una sentencia, pero hay consumos de recursos asociados con la asignación de desasignación de secuencias que puede alterar la escala de otro modo.

Algunos casos especiales de optimizaciones

Euphoria automáticamente optimiza ciertos casos especiales. Las 'x' e 'y' de más abajo, podrían ser variables o expresiones arbitrarias.

```
x + 1      -- más rápido que x + y general
1 + x      -- más rápido que y + x general
x * 2      -- más rápido que x * y general
2 * x      -- más rápido que y * x general
x / 2      -- más rápido que x / y general
floor(x/y) -- donde x e y son enteros, es más rápido que x/y
floor(x/2) -- más rápido que floor(x/y)
```

La 'x' de abajo es una variable simple, y la 'y' cualquier variable o expresión:

```
x = append(x, y)   -- más rápido que z = append(x, y) general
x = prepend(x, y)  -- más rápido que z = prepend(x, y) general

x = x &y          -- donde x es mucho más grande que y,
                   -- es más rápido que z = x &y general
```

Cuando usted escribe un ciclo que "hace crecer" una secuencia, agregando o concatenando los datos en ella, el tiempo, en general, crecerá en proporción con el ***cuadrado*** del número (N) de elementos que está agregando. Sin embargo, si usted puede utilizar una de las formas optimizadas especiales de ***append()***, ***prepend()*** o de concatenación enumeradas arriba, el tiempo crecerá solo en proporción con N (aproximadamente). Esto podía ahorrarle una cantidad de tiempo ***enorme*** al crear una secuencia extremadamente larga (podría también utilizar ***repeat()*** para establecer el tamaño máximo de la secuencia, y después completar los elementos en un ciclo, según se discute más abajo).

Asignación con operadores

Para obtener mayor velocidad, convierta:

```
lado-izquierdo = lado-derecho op expresión
```

a:

```
lado-izquierdo op= expresión
```

toda vez que el lado−izquierdo contenga al menos 2 índices, o por lo menos, un índice y un subrango. En los casos más simples, las dos formas ejecutan a la misma velocidad (o a una velocidad muy próxima).

Consejos para los modos gráficos de píxel

- El modo 19 es el modo más rápido para **gráficos animados** y **juegos**.
- La CPU no provee cache para la memoria de video (en modo 19). Generalmente lleva más tiempo leer o escribir datos a la pantalla que a un área general de memoria que usted asigne. Esto se suma a la eficacia de las ***pantallas virtuales***, donde usted hace que toda su imagen se actualice en un ***bloque de memoria*** que usted consigue con ***allocate()***, y entonces periódicamente copia con ***mem_copy()*** la imagen que resulta a la verdadera ***memoria de pantalla***. De esta manera usted nunca tiene que leer la memoria (lenta) de pantalla.
- Al dibujar píxeles, puede encontrar que los modos 257 y superiores son rápidos cerca del borde superior de la pantalla y lentos cerca del inferior.

Consejos para el modo de texto

Escribir texto en la pantalla usando **puts()** o **printf()** es bastante lento. Si es necesario, en **DOS32**, puede hacerlo mucho más rápido escribiendo en la **memoria de video**, o usando **display_text_image()**. Hay un consumo muy grande de recursos en cada **puts()** a la pantalla, y un costo relativamente pequeño por caracter. El consumo de recursos con **exw** (WIN32) es especialmente alto (en Windows 95/98/ME, al menos). Linux y FreeBSD están en medio de DOS32 y WIN32 en relación a la velocidad de salida. Por lo tanto, tiene sentido armar una cadena grande antes de llamar a **puts()**, en lugar de llamarla para cada caracter. Sin embargo, no hay ventaja en armar una cadena grande que una línea.

La lentitud de la salida de texto se debe principalmente al consumo de recursos del sistema operativo.

Rutinas de librería

Algunas rutinas comunes son estremadamente rápidas. Probablemente no podría hacer el trabajo más rápido de ninguna otra forma, aún si usa lenguaje C o ensamblador. Algunas de ellas son:

- **mem_copy()**
- **mem_set()**
- **repeat()**

Otras rutinas son razonablemente rápidas, pero en algunos casos en que la velocidad es crucial, tendría que ser capaz de hacer ese trabajo más rápido.

```
x = repeat(0,100)
for i = 1 to 100 do
    x[i] = i
end for
```

es algo más rápido que:

```
x = {}
for i = 1 to 100 do
    x = append(x, i)
end for
```

porque ***append()*** tiene que asignar y reasignar el espacio mientras que 'x' crece de tamaño. Con ***repeat()***, el espacio para 'x' se asigna una vez al comienzo. (***append()*** es suficientemente inteligente para no asignar espacio con ***cada append()*** a 'x'. Asignará algo más de lo que necesita, para reducir la cantidad de reasignaciones).
Puede reemplazar:

```
remainder(x, p)
```

con:

```
and_bits(x, p-1)
```

para mayor velocidad cuando 'p' es una potencia positiva de 2. 'x' tiene que se un entero no negativo que quepa en 32 bits.
arctan() es más rápida que **arccos()** o **arcsin()**.

Búsqueda

La función **find()** de Euphoria es la forma más rápida de buscar un valor en una secuencia de hasta 50 elementos. Más allá de esto, debería considerar el uso de una *tabla hash* (**demo\hash.ex**) o de un *árbol binario* (**demo\tree.ex**).

Ordenamiento

En la mayoría de los casos puede usar la rutina *shell sort* de **include\sort.e**.

Si tiene una enorme cantidad de datos para ordenar, debería probar uno de los ordenamientos de **demo\allsorts.e** (por ejemplo, ***great** sort*). Si los datos son demasiado grandes para entrar en memoria, no se fie de la capacidad de intercambio de memoria de Euphoria. En su lugar, ordene unos pocos de miles de registros por vez y escríbalos en una serie de archivos temporales. Luego, una todos los archivos temporales en otro ordenado completamente.

Si sus datos son enteros solamente, y están dentro de un rango razonablemente acotado, pruebe el *bucket sort* de **demo\allsorts.e**.

Sacando ventaja de la memoria cache

A medida que las CPU incrementan su velocidad, la brecha entre la velocidad de la memoria cache del chip y la velocidad de la memoria principal o DRAM (memoria dinámica de acceso aleatorio) se hace cada vez mayor. Podría tener 256 Mb de DRAM en su computadora, pero el cache del chip es probable que sea solo de 8K (datos) más 8K (instrucciones) on a Pentium, o 16K (datos) más 16K (instrucciones) en un Pentium con MMX o un Pentium II/III. La mayoría de las máquinas tienen también un cache de "nivel 2" de 256K o 512K.

Un algoritmo que recorre de una secuencia larga de varios miles de elementos o más, muchas veces, realizar desde el comienzo hasta el final una operación pequeña en cada elemento, no hará buen uso del cache de datos del chip. Puede ser que sea mejor recorrerla una vez, aplicando varias operaciones a cada elemento, antes de moverse al elemento siguiente. El mismo argumento se mantiene cuando su programa comienza a intercambiar, y los datos menos recientemente usados se mueven al disco.

Estos efectos del cache no se sienten tanto en Euphoria, ya que están en lenguajes compilados de bajo nivel, pero son mensurables.

Usando código de máquina y C

Euphoria le permite llamar rutinas escritas en código de máquina Intel de 32 bits. En **WIN32**, **Linux** y **FreeBSD** puede llamar rutinas de C en archivos **.**dll o **.**so, y esas rutinas de C pueden llamar a sus rutinas Euphoria. Podría necesitar llamar rutinas de C o en código de máquina porque algo no se puede hacer directamente en Euphoria, o debería hacerlo para mejorar la velocidad.

Para aumentar la velocidad, el código de máquina o las rutinas de C necesitan hacer una cantidad significativa de trabajo en cada llamada, de lo contrario el consumo de recursos de configurar los argumentos y hacer las llamadas, ocupará la mayor parte del tiempo, y puede ser que no representen ninguna ganacia.

Muchos programas tienen alguna operación interna que consume la mayor parte del tiempo de CPU. Si puede codificar esto en C o código de máquina mientras deja el grueso del programa en Euphoria, puede ser que alcance una velocidad comparable a C, sin sacrificar la flexibilidad y seguridad de Euphoria.

Usando el Traductor Euphoria a C

Puede descargar el Traductor Euphoria a C desde el sitio web de RDS. Este traducirá cualquier programa Euphoria en un conjunto de archivos fuente de C que luego podrá compilar usando un compilador de C.

El archivo ejecutable que obtiene al usar el Traductor debería correr igual, pero más rápido que cuando usa el intérprete. El incremento de velocidad puede ir desde un procentaje muy bajo hasta un factor de 5 o más.

Guía de resolución de problemas de Euphoria

Si se encuentra atrapado, aquí hay algunas cosas que puede hacer:

1. − Escriba: **guru**
   seguido por alguna palabra clave asociada con su problema.

   Por ejemplo,
   guru declare global include

2. − Verifique la lista de problemas comunes (debajo).

3. − Lea las partes relevantes de la documentación, es decir, **refman.doc** o **library.doc**.

4. − Intente correr su programa con las sentencias:

```
with trace
trace(1)
```

   al comienzo de su archivo **.ex** principal para ver que pasa.

5. − La **lista de correo de Euphoria** tiene una utilidad de búsqueda. Puede buscar el archivo de todos los mensajes previos. Hay una buena probabilidad que su pregunta ya haya sido discutida.

6. − Deje un mensaje en la lista de correo.

Aquí están algunos problemas comúnmente informados (P:) y sus soluciones (S:).

P: Corrí mi programa con **exw** y la ventana de consola desapareció antes que pudiera leer la salida.

S: La ventana de consola solamente aparecerá si se la necesita, y desaparecerá inmediatamente cuando su programa finalice. Quizás debería codificar algo como:

```
puts(1, "\nPresione Enter\n")
if getc(0) then
end if
```

al final de sus programa.

P: Quisiera cambiar las propiedades de la ventana de consola.

S: Haga click derecho sobre **c:\windows\system\conagent.exe** y elija "propiedades". Puede cambiar el fondo y varios ítems más.

P: Al correr **ex.exe** en una caja DOS, hace que mi pequeña ventana DOS vaya a pantalla completa.

S: Esto ocurrirá solamente la primera vez que ejecute **ex.exe** después de crear una nueva ventana DOS pequeña. Puede hacer que la ventana se haga pequeña otra vez presionando **Alt+Enter**. Después de esto, quedará pequeña. Su programa Euphoria puede mantener pequeña la ventana al ejecutar:

```
if graphics_mode(-1) then
end if
```

al inicio de la ejecución. Esto puede causar un breve parpadeo. Su programa puede forzar una ventana de texto a pantalla conpleta, ejecutando:

```
if graphics_mode(3) then
end if
```

P: Mi programa CGI Euphoria se "cuelga" o no tiene salida.

S: Busque un archivo ***ex.err*** en su directorio cgi−bin. Active **with trace** / **trace(3)** para ver que sentencias se están ejecutando (mire ***ctrace.out*** en su cgi−bin). Inserte **without warning** al comienzo de su programa. En ***Windows***, cuando hay advertencias, Euphoria emitirá una indicación antes de terminar, provocando que su programa se cuelgue. En Windows siempre debería usar **exwc.exe** para correr programas CGI, o puede tener problemas con la salida estándar (vea ***euphoria\bin\makecon.exw***). Con el servidor web Apache, puede tener una primera línea en su programa de: #!.\exwc.exe para ejecutar su programa usando exwc.exe en el directorio actual (cgi−bin). En ***Linux/FreeBSD***, cuide que su primera línea termine en LF, y no en CR−LF, o el #! no se manejará correctamente. En Linux también tiene que establecer correctamente permisos de ejecución en su programa, y ***ex.err*** y ***ctrace.out*** tienen que ser escribibles por el proceso servidor o no serán actualizados.

P: ¿Cómo hago para leer/escribir en los puertos?

S: Consiga la colección de rutinas del sistema para DOS y nivel de máquina de **Jacques Deschenes** en la página web de Euphoria. Lea el archivo **ports.e**.

P: Tengo problemas corriendo un programa gráfico **DOS32**. Presiono Ctrl+Break y el sistema parece quedar muerto.

S: Algunos programas gráficos no funcionarán a menos que los inicie desde DOS o desde una ventana de pantalla completa del DOS bajo Windows. A veces, tiene que editar el código fuente del programa para usar un modo gráfico de resolución más baja, como el modo 18. Algunos modos gráficos SVGA pueden no trabajar bajo una ventana DOS, pero funcionarán cuando reinicie su máquina en modo MS−DOS. Un mejor controlador para su tarjeta de video puede solucionar esto.
Debería detener el programa usando el método que la documentación recomienda. Si aborta su programa con Ctrl+C o Ctrl+Break puede encontrar que su pantalla queda en un modo gráfico que usa divertidos colores. Cuando intenta escribir algo, puede resultar difícil o aún imposible leer lo que escribió. *El sistema puede parecer muerto, cuando de hecho **no** lo está*.

Pruebe los siguientes comandos DOS, en este orden, hasta que se alcaren las cosas:

1. escriba: ***Alt+Enter*** para obtener una ventana normal (no pantalla completa) nuevamente.

2. escriba: **cls**
Aún cuando no pueda ver ninguna tecla presionada en pantalla, éste puede limpiar la pantalla.

3. escriba: **ex**
El intérprete Euphoria intentará restablecer la pantalla modo de texto normal.

4. escriba: **exit**
Si está corriendo bajo Windows, esto terminará la sesión DOS.

5. escriba: **Ctrl+Alt+Supr**
Esto destruirá su actual sesión DOS bajo Windows, o reiniciará su computadora si está ejecutando bajo DOS.

6. Si todo falla, reinicie o apague y vuelva a encender la computadora. Inmediatamente debería ejecutarse **scandisk** cuando el sistema reinicie.

P: Cuando ejecuto programas Euphoria SVGA, la salida en la pantalla se abarrota en la parte superior de la pantalla.

S: Pruebe: **use_vesa(1)** en **machine.e**.

Pruebe descargando el último controlador para su tarjeta de video desde Internet. El sitio de ATI es:

http://www.atitech.com

Otros que han tenido problemas de video, lo solucionaron instalando la última versión de DirectX.

P: Mi programa deja la ventana de DOS en un estado sucio. Quisiera que dejara una ventana normal de texto.

S: Cuando su programa termina, debería llamar la función graphics_mode(−1) para regresar la ventana DOS a la normalidad. Por ejemplo:

```
if graphics_mode(-1) then
end if
```

P: Cuando ejecuto mi programa desde el editor y presiono Ctrl+C, el programa se muere con un error del sistema operativo.

S: Esto es un problema conocido. Ejecute su programa desde la línea de comandos, fuera del editor, si tiene que presionar Ctrl+C o Ctrl+Break.

P: Al ejecutar mi programa no hay errores, pero nada ocurre.

S: Probablemente se olvidó de llamar al procedimiento principal. Necesita una sentencia de alto nivel que venga después de su procedimiento principal para llamarlo e iniciar la ejecución.

P: Estoy tratando de llamar a una rutina documentada en **library.doc**, pero sigue diciendo que la rutina no está declarada.

S: ¿Recordó incluir el archivo **.e** necesario del directorio **euphoria\include**? Si la sintaxis de la rutina dice, por ejemplo, "include graphics.e", entonces su programa tiene que tener "**include graphics.e**" (sin las comillas) antes del lugar donde llama por primera vez a la rutina.

P: Tengo un archivo de inclusión con una rutina a la que quiero llamar, pero cuando intento llamarla, dice que la rutina no ha sido declarada. Pero ***sí*** ha sido declarada.

S: ¿Recordó definir la rutina con "**global**" delante de ella en el archivo de inclusión? Sin "global", la rutina no es visible ***fuera*** de su propio archivo.

P: ¿Cómo ingreso una línea de texto desde el usuario?

S: Vea **gets()** en **library.doc**. ***gets(0)*** leerá una línea desde la **entrada estándar**, que normalmente es el **teclado**. La línea siempre tiene que terminar en **\n**. Para quitar el caracter \n sobrante, haga:

```
line = line[1..length(line)-1]
```

También vea **prompt_string()** en **get.e**.

P: Después de ingresar una cadena desde el usuario con **gets()**, la siguiente línea que aparece en pantalla no comienza en el margen izquierdo.

S: Su programa debería emitir un caracter de *nueva línea*, por ejemplo, **puts(SCREEN, '\n')** después de hacer un ***gets()***. Esto no ocurre automáticamente.

P: ¿Cómo hago para convertir un número en una cadena?

S: Use **sprintf()** en **library.doc**, por ejemplo:

```
string = sprintf("%d", number)
```

Además de **%d**, también puede probar otros formatos, como **%x** (hexadecimal) o **%f** (punto flotante).

P: ¿Cómo hago para convertir una cadena en un número?

S: Use **value()** en **library.doc** o, si está leyendo desde un archivo o el teclado, use **get()**.

P: Dice que estoy tratando de redefinir ni variable del ciclo for.

S: Las variables de los ciclos for se declaran automáticamente. Aparentemente, ya tiene una declaración anterior con el mismo nombre en su rutina o programa. Quite esa declaración anterior o cábiele el nombre a la variable del ciclo.

P: Obtengo el mensaje "caracter de escape desconocido" en una línea donde estoy tratando de especificar un nombre de archivo.

S: ***No*** escriba "C:\TMP\MYFILE", sino **"C:\\TMP\\MYFILE"**. La barra invertida se usa para caracteres de escape como **\n** o **\t**. Para especificar una barra sola en una cadena, necesita escribir \\.

P: Estoy tratando de usar la entrada del ratón en un modo gráfico SVGA pero apenas trabaja .

S: **DOS** no soporta la entrada del ratón en modos más allá del modo gráfico 18 (640x480 16 colores). **DOS 7.0 (parte de Windows 95/98)** parece dejarle leer, por lo menos, las coordenadas x−y y los botones en modos de alta resolución, pero usted puede tener que dibujar su propio indicador de ratón en los modos de alta resolución. **Graeme Burke**, **Peter Blue** y otros tienen buenas soluciones a este problema. Ver sus archivos en la página web "Archivo" de Euphoria.

P: Estoy tratando de imprimir una cadena usando **printf()** pero solo sale el primer caracter.

S: Vea la descripción de **printf()** en **library.doc**. Puede necesitar poner llaves alrededor de su cadena para que sea vista como un único valor a imprimir, por ejemplo, escribió:

```
printf(1, "Hello %s", mystring)
```

donde debería haber escrito:

```
printf(1, "Hello %s", {mystring})
```

P: Cuando imprimo números usando **print()** o **?**, solamente se muestran 10 dígitos significativos.

S: Euphoria normalmente solo muestra 10 dígitos. Internamente, todos los cálculos se realizan usando al menos 15 dígitos significativos. Para ver más dígitos, tiene que usar **printf()**. Por ejemplo,

```
printf(1, "%.15f", 1/3)
```

Esto mostrará 15 dígitos.

P: Se queja de mi declaración de rutina, diciendo "aquí se esperaba un tipo".

S: Al declarar los parámetros de la subrutina, Euphoria le requiere proporcionar un tipo explícito para cada parámetro individual. Por ejemplo:

```
procedure foo(integer x, y)         -- MAL
procedure foo(integer x, integer y) -- BIEN
```

En el resto de los contextos es aceptable hacer una lista:

```
atom a, b, c, d, e
```

P: Declarando algunas variables en medio de una rutina, me da un error de sintaxis.

S: Todas las declaraciones de variables *privadas* tienen que estar al comienzo de su **subrutina**, antes de cualquier sentencia ejecutable. (En el nivel más alto de un programa, **fuera de cualquier rutina**, está bien declarar variables en cualquier parte).

P: Aparece: Error de sintaxis − esperaba ver posiblemente ' xxx ', no ' yyy '

S: En este punto en su programa se escribió una variable, palabra clave, número o símbolo de puntuación, yyy, que no se ajusta sintácticamente con lo que venía de él. El compilador le está ofreciendo un elemplo, xxx, de algo que sería aceptado en este punto, en lugar de yyy. Observe que puede haber muchas otras (y mucho mejores) posibilidades legales en este punto que xxx, pero xxx pudo por lo menos darle una pista en cuanto a lo que "está pensando" el compilador.

P: Tengo problemas para ejecutar Euphoria con ***DR−DOS***.

S: Su **config.sys** debería tener solamente HIMEM.SYS pero no EMM386.EXE.

P: Intento de correr un programa con **exw** y dice "este es un ejecutable de modo caracter de Windows NT".

S: **exw.exe** es un **programa Windows de 32 bits**. Tiene que ejecutarse bajo Windows o en una ventana DOS. No funcionará bajo DOS puro en un sistema viejo. **ex.exe *funcionará*** bajo **DOS puro.**

Notas de las versiones Euphoria

Versión 2.4 Edición Oficial, 3 de Julio de 2003:

- Se extendieron **define_c_proc()** y **define_c_func()**, por lo que además de definir rutinas de C en .dll's externas y librerías compartidas, ahora puede definir los parámetros y valor de retorno para una rutina en código de máquina que su programa ponga en su propia memoria. Puede llamar a la rutina en código de máquina usando **c_proc()** o **c_func()**. Gracias a Daniel Kluss.

- ***Mejora de rendimiento:*** **get4()** y **put4()** en **database.e** se aceleraron levemente. Son muy importantes en la velocidad final de **EDS**. Gracias a Derek Parnell.

- ***Mejora de rendimiento:*** **get_bytes()** es mucho más rápida cuando la cantidad de bytes requeridos excede bastante la cantidad de bytes que quedan en el archivo. Gracias a Pier Feddema.

- ***Traductor:*** Al traducir una rutina Euphoria enorme (varias cientas de sentencias Euphoria), el Traductor ahora llamará a una rutina desreferenciada, en lugar de usar sentencias de C en−línea. Esto reduce la posibilidad de exceder el límite de tamaño impuesto por el compilador C (especialmente Watcom C). También reduce el tamaño del **.exe**. Puesto que la rutina desreferenciada es más probable que esté en el cache que las sentencias en−línea, la diferencia de velocidad no es tan grande.

- **Código Fuente del Intérprete:** se agregó **runtime windows=4.0** en **watexw.bat**, al comando de enlace para generar **exw.exe**.

- Se agregaron algunas verificaciones para tipos de argumento inválido y tipo de retorno inválido en las llamadas a **define_c_func()** y **define_c_proc()**.

- Algunos mensajes de errores de sintaxis son más descriptivos cuando están involucrados los identificadores de espacio de nombres.

- Se modificó el programa tutorial ***filesort.ex*** para hacerlo más usable bajo ***Linux*** y ***FreeBSD***.

- Por defecto, **safe.e** es ahora menos estricto, verificación "solo bordes" de corrupción de la memoria, cuando la plataforma es **WIN32**. Frecuentemente, los programas Windows acceden a memoria que no se asignó usando **allocate()** de Euphoria.

- ***error corregido:*** Cuando una constante literal de punto flotante en un programa Euphoria era más grande que 1e308, el Traductor ponía "inf" en **init_.c**. Esto provocaba qye el compilador de C emitiera un error de símbolo no definido. Gracias a Juergen Luethje.

- ***error corregido:*** En un caso raro, el Traductor fallaba al emitir el código C para hacer una copia de una secuencia con múltiples referencias a ella, antes de sobrescribir un elemento de esa secuencia. Gracias a Juergen Luethje.

- ***error corregido:*** En ciertos casos, cuando un programa Euphoria intercambiaba datos con una **.dll** escrita en Euphoria, los datos podían no liberarse (hasta que el programa terminaba). Gracias a Wayne Overman (Euman).

- ***error corregido:*** Si utilizaba "asm" en su programa como una variable privada o nombre de parámetro, el Traductor usaría "_asm" en el código C. Esto no es aceptado por algunos compiladores de C. El Traductor ahora evitará de usar "_asm", tanto como "_try", "_Seg16", "_stdcall" y varios otros nombres de comienzan con un guión de subrayado que están reservados por varios compiladores de C. Gracias a George Papadopolous y Matt Lewis.

- ***error corregido:*** Si el parámetro HOT_KEYS en **ed.ex** fuera fijado a FALSO, entonces **Esc h Enter** no bring up el indicador de ayuda. Gracias a J. Brown.

Versión 2.4 Edición Beta, 10 de Abril de 2003:

Esta edición actualiza los productos **Intérprete Euphoria**, **Traductor Euphoria a C**, y el **Código Fuente del Intérprete**, para **Windows**, **DOS**, **Linux** y **FreeBSD**.

Nuevas Características

- **bind** y **shroud** ahora tienen la opción **−out** para especificar el archivo de salida, por lo tanto no se le pedirá que lo informe. Gracias a Jonas Temple, Rusty Davis y otros.

- **bind** y **shroud** ahora tienen la opción **−quiet** que elimina las estadísticas y mensajes normales, y por lo tanto elimina la ventana emergente que normalmente aparece. Solamente se informan los errores. Gracias a Jonas Temple.

- El mensaje de error de espacio de nombres que se emite cuando usted se refiere a un símbolo global que se define en dos o más archivos, ahora le da una lista de todos los archivos donde ese símbolo está definido. Gracias a Derek Parnell y Irv Mullins.

- ***Traductor:*** En muchos casos, el código C generado para **remainder()**, la multiplicación entera y **compare()** es más pequeño y más veloz.

- **exw, ecw −wat**: La desasignación del espacio para los números enormes de objetos pequeños (átomos o secuencias pequeñas) es mucho más rápida que en 2.4 Alpha. Gracias a Andy Serpa. (Observe que la *asignación* de números enormes de objetos pequeños en ***exw***, o ***ecw −wat***, se aceleró mucho en 2.4 Alpha, y ahora es mucho más rápida).

- Cuando se crea el archivo **ex.err**, cualquier advertencia emitida por el programa se listará al final del archivo **ex.err**. Gracias a Al Getz.

- El nombre del archivo ahora se incluye en el mensaje de alerta que obtiene para algunas de las advertencias comunes (variable no usada, variable no asignada a). Gracias a Al Getz.

- **Nuevo ícono**: En **Windows**, los archivos de inclusión de Euphoria ahora están rotulados con la versión de escala de grises del icono **E)** de Euphoria. Esto le permite distinguir fácilmente los archivos ejecutables de los de inclusión. Gracias a Wolfgang Fritz.

- Los subíndices de punto flotante fuera de límite se informaban después de redondear hacia el entero inferior. Ahora se informa el valor antes de redondear.

- **Euphoria Database System (EDS)**: **db_rename_table()** verifica si la tabla destino existe, antes de renombrarla. Gracias a Mike Nelson.

- El primer valor devuelto por **rand()** (en ausencia de **set_rand()**) es ahora más "aleatorio". Gracias a Aku.

Errores corregidos

- ***error corregido:*** Debido a cambios hechos en 2.4 Alpha, la rutina **dir()** para Borland y Lcc concatenaba los caracteres de atributos de archivos (si existían) al nombre del archivo. Gracias a Dr. Juan R. Herguijuela.

- ***error corregido:*** No se emitía un mensaje de error cuando ')' seguía inmediatamente a ',' en una lista de declaración de parámetros de una rutina. Gracias a Brage Moris.

- ***error corregido:*** Los subíndices positivos fuera de límite enormes (sobre 2 billones) se informaban como valores negativos enormes.

- ***error corregido:*** **repeat(0, size)**, donde 'size' era un número de punto flotante enorme, se informaba incorrectamente : "la cuenta de repetición no puede ser negativa". Ahora informa: "la cuenta de repetición es demasiado grande". Gracias a Martin Stachon.

- ***error corregido:*** **machine_proc(x, 5)**, donde 'x' era un número de punto flotante muy grande, informaba incorrectament: "se esperaba un entero, no una secuencia". Ahora informa: "El primer argumento para machine_proc/func tiene que ser un entero positivo pequeño". Gracias a Martin Stachon.

Versión 2.4 Edición Alfa, 21 de Febrero de 2003:

Nuevas Características

- La mayoría de las excepciones a nivel de máquina (peek/poke a direcciones incorrectas, etc.) tanto en el programa principal como en las .dll, ahora las captura **exw** y **exu**, y se informan de la forma usual con un trazado completo yel volcado de variables en **ex.err**. Esto es una gran mejora respecto de los crípticos mensajes a nivel de máquina que normalmente obtiene al usar lenguajes compilados y en la mayoría de los lenguajes interpretados como "violación de segmento", "instrucción ilegal", etc. Gracias a Martin Stachon.

- Además de **__stdcall**, ahora se soporta la convención de llamadas de C **__cdecl,** para llamadas a rutinas de C en .dll y también en call−backs a rutinas de Ephoria desde código C.

- El soporte de nombres largos de Euphoria en ***DOS*** se extendió a ***Windows XP***.

- La ventana de **trazado** le muestra secuencias grandes en una vista formateada en una pantalla separada. Puede desplazarse a través de la secuencia completa.

- **pretty_print()** se agregó a **misc.e**, mostrándole los objetos Euphoria con una vista agradable, estructurada y con muchas opciones de formato.

- La impresión formateada de secuencias se hace ahora en **ex.err**, el comando **?** y **db_dump()**.

- **Euphoria Database System (EDS)**: **db_rename_table(name, new_name)** se agregó a **database.e**. La rutina fue provista por Jordah Ferguson, e incluida solamente con cambios triviales.

- **Linux/FreeBSD**: **system()** no inicializa más curses cuando todavía no existe la ventana de la consola. Gracias a Daniel Johnson.

- Se elevó a 30 la cantidad de niveles anidados de los archivos de inclusión (desde 10). Gracias a Tone Skoda.

- Las rutas de las sentencias Include, pueden estar encerradas entre comillas dobles, de forma que las rutas que contienen espacios en blanco se pueden manejar correctamente. Esto se implementó realmente en 2.3, pero nunca fue documentado.

- **exw.exe**, y cualquier ejecutable producido por el **Traductor** con Watcom, ahora tiene el subsistema establecido a 4.0, en lugar de 3.1. Esto mejora la apareciencia de la interfaz de usuario en algunos casos. La utilidad **make31.exw** creará una versión de **exw.exe** que soporte la interfaz de usuario de Windows 3.1 como antes, en el caso inverosímil que haya problemas de compatibilidad con Euphoria 2.3. Gracias a H. W. Overman, Brian Broker y otros por recomendar este cambio.

- **makecon.exw** − creará una versión de **exw.exe** que opera como aplicación de consola − no emerge una ventana de consola, y se pueden redireccionar **stdin/stdout**.

- **trace(1, 2 y 3)** se permiten ahora con **bind −clear** (todavía no está permitido con **shrouded bind** por cuestiones de seguridad). Gracias a Jonas Temple.

- ***Traductor:*** Ahora puede hacer una .dll de Euphoria usando Lcc y usarla con programas interpretados corriendo bajo **exw**, y programas traducidos usando Borland y Watcom. Previamente, el programa principal se tuvo que compilar con Lcc.

- **Traductor:** no usa más las opciones **−m486** o **−mpentium** disponibles con GCC y DJGPP. Esas opciones causaban advertencias y el compilador de C, aparentemente establece el modelo de máquina correctamente por sí mismo. Gracias a Ken Rhodes.

- **Traductor:** realizará ahora llamadas automáticas a tipos definidos por el usuario, en el caso inusual donde la rutina de tipos tenga efectos colaterales (establecer variables globales, operaciones E/S, etc.). Gracias a Andy Serpa.

- **euphoria\demo\bench** compara el **Intérprete de Euphoria** y el **Traductor Euphoria a C** contra más de otros 20 lenguajes interpretados.

Migración

- Hemos migrado el **Intérprete** y **Traductor** a **FreeBSD.** El fuente tiene ahora varios #ifdef de C para ***FreeBSD***.

- Andy Cranston migró Euphoria a ***HP Unix***, y planea hacerlo para ***Sun Unix***.

Optimizaciones

− Intérprete −

- Los típicos subrangos grandes son más rápidos. Cerca del 30% más rápidos para subrangos de 100 a 50000 en longitud (el consumo de recursos domina en los subrangos pequeños, y la carencia de cache afecta a los subrangos más grandes). Esto asume que el subrango es principalmente de números enteros (generalmente cierto), y se hace una copia real de los datos (generalmente cierto desde v1.4b).

- Las sentencias que contienen múltiples concatenaciones **son más rápidas.**

```
por ejemplo, en lugar que:
     result = a &b &c
     sea evaluada como:
       1. copia a y b en temp
       2. copia temp y c en result
          (a y b se copian dos veces!)

     Ahora hacemos:
       1. copia a, b y c directamente en result
```

Por tanto, hay menos copiado de datos y pocas secuencias temporales para crear. Los operadores adicionales en una expresión, (cuanto mayor es la aceleración), por ejemplo, con 3 operadores algunos de los datos se copiaron 3 veces, etc. Jordah Ferguson precisó que esto era lento.

- El tiempo implicado en llamar y volver de una rutina de call−back de Euphoria ha sido reducido cerca de 10%.

- En **exw** y **ecw −wat**, la asignación del espacio para una gran cantidad de objetos es más rápida. Puede ser enormemente más rápida cuando están implicados centenares de millares o de millones de objetos.

- Una compresión mejor UPX ha bajado algunos Kb de **exw.exe** contra 2.3 (aunque se agregó nuevo código). Gracias a Wolfgang Fritz.

− Sistema de Base de Datos Euphoria −

- Las claves y registros se leen más rápido, debido a la rutina **decompress()** más rápida. Casi dos veces más rápida cuando la clave o los registros que se recuperarán constan principalmente de secuencias de caracteres o de números enteros pequeños. Este caso es absolutamente común.

- El asignación del nuevo espacio en una base de datos es mucho más rápido, hasta 4 veces más, especialmente en bases de datos grandes con una lista grande de bloques libres.

- La insersión y borrado de registros en tablas enormes son ahora mucho más rápidos. Combinado con los subrangos más veloces en Euphoria 2.4, **database.e** es ahora cerca de un 25% más rápida para una tabla con 10,000 registros y más de 3 veces más rápida para una tabla con 100,000 registros. Esto importa realmente solamente si usted está intentando insertar/borrar cientos de registros por segundo. En el caso típico de un operador humano que ingresa datos a través de la interfaz de usuario, usted nunca notaría el tiempo de insersión/borrado para un registro (algunos milisegundos). Derek Parnell precisó la lentitud.

- **db_select_table()** es significativamente más rápida.

- **get4()** es más rápido.

− Otras Optimizaciones −

- **bytes_to_int()** en **machine.e** es más de 2 veces más rápido.

- **gets()** es un 5% más rápido.

- **sort()** y **custom_sort()** son un poco más rápidas. Gracias a Ricardo Forno por optimizar el algoritmo Shell sort.

- Se agregaron al **Traductor** varias optimizaciones más. Esto produce ejecutables que son más pequeños y más rápidos que los de la versión 2.3. El Traductor 2.4 se probó exitosamente en cientos de miles de líneas de código Euphoria sin conocerse hasta hoy errores en la generación del código. Algunos resultados de las pruebas de rendimiento del Traductor están en **euphoria\demo\bench**.

Errores Reparados

− Código Fuente −

- ***error corregido:*** El archivo por lotes **gnubsd** refería a **syncolor.c** y **syncolor.o**. (**gnuexu** estaba bien).

- ***error corregido:*** Karl Bochert indicó un cambio necesario en el código de C para hacer que **poke()** trabaje con versión más reciente de Lcc. El cambio corrigió el **Traductor** (con versiones recientes de Lcc) y el **Código Fuente del Intérprete** (compilado con Lcc).

− Intérprete −

- ***error corregido:*** Una cancelación del programa puede ocurrir en situaciones cuando una rutina call−back se llama indirectamente en forma recurrente. Gracias a Matthew Lewis y George Papadopoulos.

- ***error corregido:*** En un ciclo for, en el nivel superior de un programa (fuera de cualquier rutina), que incrementaba la variable del ciclo en un entero distinto del valor +1 por defecto, el **end for** tardaba 15 veces más que lo necesario debido a la corrección de un error que se hizo en noviembre de 1999. El **end for** solamente era lento, no el código contenido enel cuerpo del ciclo. Antoine Tammer detectó esto.

- ***error corregido:*** En ***XP***, cuando abre una nueva ventana DOS que tiene más de 25 líneas, la Máquina Virtual de DOS (VDM, por sus iniciales en inglés), al principio se confunde respecto de la cantidad de líneas reales. La primera vez (solamente) en que ejecuta un programa Euphoria en esa ventana, si lo ejecuta cerca del mismo fondo de la pantalla, la salida puede desaparecer, o el VDM o Euphoria puede informar un error, etc.

Euphoria (**ex.exe**) ahora detecta los casos raros en que el VDM se confunde y limpia la pantalla, el cual elimina la confusión. Un problema similar existió en NT, y se lo solucionó unos pocos años atrás.

- ***error corregido:*** El intérprete se refería a "call−back desde Windows" en **ex.err**, aún en ***Linux*** o ***FreeBSD***. Ahora dice "call−back desde una fuente externa" en aquellos sistemas. Gracias a Pete Eberlein.

- ***error corregido:*** Cuando no se podía encontrar un archivo de inclusión, el mensaje de error se refería a "euphoria\include". Ahora usa %EUDIR%\include.

- ***error corregido:*** No se generará más un mensaje de error en ninguna plataforma para **without profile_time**. Gracias a Alan Oxley.

− Traductor −

- ***error corregido:*** Al asignar el resultado de un cálculo aritmético (típicamente una multiplicación) que involucra a dos enteros, a una variable declarada como átomo, donde la variable átomo (en el mismo bloque básico) se asigna a un valor entero, el Traductor no podía emitir ningún código para comprobar si hay desbordamiento entero (resultado fuera de +/− mil millones). Esto provocaba una cancelación del programa. Gracias a Andy Serpa.

- ***error corregido:*** el comando strip.exe de DJGPP en **emake.bat** fallaría en ***XP***, ***2000*** debido a un error en DJGPP. Ahora **emake.bat** tiene: **SET LFN=n** para trabajar alrededor del error en strip.exe.

- ***error corregido:*** El código traducido compilado con Borland C no producía INF's y NAN's, como Watcom y Lcc. En su lugar, se cancelaba al ocurrir un desbordamiento de punto flotante (más de 1e308), o se calculaba un resultado indefinido de punto flotante. El **Código Fuente del Intérprete** también se corrigió para aquellos que desean compilar **exw.exe** usando Borland C. Gracias a Andy Serpa.

- ***error corregido:*** En el primer bloque básico de una rutina Euphoria (es decir, antes de cualquier sentencia de control de flujo) **peek4u()**, **peek4s()**, y la operación "add integer 1", a veces se descuidaría comprobar si hay un posible desbordamiento entero de 31 bits al asignar a una variable privada declarada como átomo, a menos que la variable hubiera sido inicializada previamente. Gracias a Mike Duffy.

- ***error corregido:*** En algunos casos, al asignar un elemento de la secuencia a una variable declarada como entero, y conocida para tener un valor entero en este punto, no se manejaba correctamente el caso donde el elemento era un valor entero almacenado en forma de double de C.

- ***error corregido:*** En casos extraños, el Traductor puede emitirdos operadores unarios negativos en una fila, el cuál sería interpretado por un compilador de C como el operador "−−" de decremento de C.

- ***error corregido:*** Las .dll de Euphoria no siempre liberaban correctamente el almacenamiento asignado por el programa principal y viceversa. La memoria se podría agotar, con lo que podría obtener una cancelación a nivel de máquina. Gracias a H. W. Overman.
  Nota: Debido a esta corrección, cualquier .dll de Euphoria creada con el Traductor versión 2.3 o anterior, tiene que ser re−traducida con la versión 2.4, y recompilada, para interconectar con el intérprete Euphoria 2.4 (o posterior) o código traducido. Las nuevas .dll creadas con la versión 2.4 o posterior, no funcionarán con el intérprete de la versión 2.3 o anteriores, excepto en casos triviales.
- ***error corregido:*** La función **sleep(x)** solamente "dormía" por 'x' milisegundos al usar la librería en tiempo de ejecución de Lcc. Ahora "duerme" por 'x' segundos, para consistir con la documentación de Euphoria para **sleep()**. Gracias a Wolfgang Fritz.

- ***error corregido:*** En algunas versiones de ***Linux***, un programa Euphoria traducido/compilado podía cancelarse si la salida estándar se redirigía, por ejemplo para CGI.

- ***error corregido:*** En algunas versiones de ***Linux***, un programa Euphoria traducido/compilado podía cancelarse si se llamaba a máquina (M_GET_SCREEN_CHAR, {row, col}).

- ***error corregido:*** En algunos casos, el código no era correcto cuando se le asignaba a una variable entera el menos unario de una variable átomo.

- ***error corregido:*** En un caso muy raro, un valor no inicializado en memoria se podía usar para determinar si un valor literal de punto flotante se debería tratar como entero o no. Podía resultar un código incorrecto.

− Enlazador −

- ***error corregido:*** El enlazador se estrellaría después de ver un comentario sin carácter de nueva línea, solo EOF, en la última línea de un archivo. Algunas versiones de Win32Lib.ew tenían esto. Gracias a Henri Goffin.

- ***error corregido:*** El informe del uso de **bind**/**shroud** todavía decía "−scramble", en lugar de "−clear" y tenía otros errores en ***Linux***/***FreeBSD***. Gracias a Chris Bensler.

- ***error corregido:*** **bind/shroud −clear** podía dejar de renombrar una variable privada, cuando una variable local anterior se renombraba con el mismo nombre. Gracias a Pete Lomax.

- ***error corregido:*** Cuando a un archivo de inclusión le faltaba **\n** en la última línea, **bind/shroud −clear** olvidaba dejar un cierto espacio en blanco antes de la palabra siguiente en el archivo principal. Gracias a Pete Lomax.

- ***error corregido:*** Si usted definió una constante que nunca se utilizaba, y fue definida usando una expresión que contenía un operador menos binario, un error del sintaxis podría ocurrir en el archivo enlazado o encriptado que usted crea. Gracias a Chris Bensler.

− Rutinas de Librería −

- ***error corregido:*** **walk_dir("/",...)** fallaba en ***Linux***. Gracias a Ricardo Forno.

- ***error corregido:*** **db_compress()** de EDS tenía un error si el archivo de base de datos o su ruta contenía un caracter en blanco. El caracter en blanco es correcto hoy en todas las plataformas, excepto ***DOS***. Gracias a Virtual B.

- ***error corregido:*** **wildcard_file()** en **euphoria\include\wildcard.e** ahora es sensible a las mayúsculas en ***Linux***/***FreeBSD*** (pero todavía es insensible en ***DOS***/***Windows***). Gracias a Jeff Fielding.

- ***error corregido:*** **dir()** no siempre informaba correctamente el tamaño de archivos mayores a 1 Gb. Ahora maneja hasta 4 Gb. Posiblemente fallaría la aritmética utilizada en el tamaño del archivo. Gracias a Ricardo Forno.

- ***error corregido:*** **where()** no siempre informaba correctamente las posiciones del archivo mayores a 1 Gb. Posiblemente fallaría la aritmética utilizada en la posición. Ahora maneja hasta 2 Gb.

- ***error corregido:*** **ex, exw, ec −wat, ecw −wat**: La función **dir()** para ***DOS*** y ***Windows*** no manejaba correctamente los comodines cuando aparecía una barra invertida al final del archivo o de la ruta del directorio. Gracias a Juergen Luethje.

− Espacio de nombres −

- ***error corregido:*** Si intentaba declarar una nueva variable, usando un calificador de espacio de nombres, por ejemplo, **integer y:x** (que es ilegal) no lo informaría, y solo ignoraría la parte "y:", mientras y:x era una rutina (no una variable) en un archivo anterior. Esto ahora se captura como error. Gracias a Martin Stachon.

- ***error corregido:*** Al declarar el tipo de un parámetro, usando un tipo global que estaba definido en más de un archivo, obtendría un mensaje de error confuso que indica que "se espera un tipo aquí". Ahora usted obtendrá un mensaje que precisa que el tipo requiere un identificador de espacio de nombres para resolverlo. Gracias a Juergen Luethje.

- ***error corregido:*** Se ha mejorado el mensaje de error que obtiene si especifica un calificador del espacio de nombres y ': ', pero olvida seguirlo con un identificador adecuadamente constituido. Gracias a Carl White.

− Programas de Demostración −

- ***error corregido:*** En el programa de demostración **window.exw**, **allocate(64)** y **allocate(16)** causaban una pérdida de almacenamiento. Gracias a Wolfgang Fritz y Jordah Ferguson.

− Trazado/Depuración −

- ***error corregido:*** Cuando se ejecutaba **trace(0)**, seguido luego por **trace(1)**, sin E/S a pantalla en el medio, los valores de algunas variables en la pantalla de trazado no se actualizaban. Gracias a Ricardo Forno.

- ***error corregido:*** **with trace** / **with profile** / **trace(3)**, usados juntos, producían líneas fuente basura en **ctrace.out**. Gracias a Kat.

Versión 2.3 Edición Oficial, 11 de Febrero de 2002:

Esta versión actualiza el Intérprete de Euphoria, el Traductor Euphoria a C, y el Código Fuente del Intérprete, para todas las plataformas.

- La versión DJGPP del código fuente del intérprete ahora usa los rótulos dinámicos de GNU C, igual que la versión Linux. Esto permite que alcance plena velocidad, sin la necesidad de ninguna optimización a nivel de ensamblador. Gracias a Bernie Ryan.

- The Código Fuente del Intérprete incluye ahora un documento introductorio que describe cómo funciona el intérprete.

- En la Edición Completa, bind.bat y bindw.bat ahora usan ***exw*** para ejecutar el enlazador/encriptador. Esto evita problemas con los nombres largos en algunos sistemas. Mientras esté corriendo el enlazador/encriptador, aparecerá una ventana de consola. Gracias a "void", Bruce Axtens, y otros.

- **error corregido:** Debido a un error en la librería de WATCOM C 10.6, los intérpretes ***ex*** y ***exw***, y el código traducido a C y compilado con Watcom, podría obtener un resultado incorrecto de la función ***where()*** de Euphoria cuando el archivo se abre en modo append, y el puntero de archivo estaba en la porción no comprometida del archivo (no escrito al disco aún). El error se solucionó al tener ***flush()*** de Euphoria el archivo en este caso particular, antes de llamar la rutina de Watcom. Gracias a Bob Elia.

- **error corregido:** Un error introducido en el enlazador de 2.3 beta, podía provocar que una llamada de una función en la última línea del programa se ignorara. Gracias a Wolfgang Fritz.

- **error corregido:** Varios archivos Euphoria en la distribución ***WIN32***+***DOS32*** tenían terminadores de línea estilo Linux (\n solamente). Esto complicaba su edición usando el Bloc de Notas y otros editores. Gracias a Pete Lomax.

- **error corregido:** Si "with type_check" estaba activo, ***ed.ex*** obtenía una falla de type_check cuando se presionaba la tecla Esc. Gracias a Kenneth Orr.

Versión 2.3 Edición Beta, 15 de Enero de 2002:

Esta versión actualiza el Intérprete de Euphoria, el Traductor Euphoria a C, y el Código Fuente del Intérprete, para todas las plataformas.

- Ahora puede sobreescribir una rutina incorporada de Euphoria con su propia variable o el identificador del espacio de nombres del mismo nombre. Esto fue permitido previamente solamente para las rutinas definidas por el usuario. Además de dar a programadores más libertad, permitirá que RDS agregue nuevas rutinas incorporadas en el futuro sin alterar el código existente.

- Se quitó la advertencia sobre tener símbolos globales múltiples con el mismo nombre en diversos archivos de fuente. Dejó de ser necesario puesto que se le pedirá proveer un identificador del espacio de nombres si usted hace realmente una referencia ambigua a un símbolo global.

- Ahora puede tener las barras \ (o las de Linux /) en el final de todos los nombres de directorio en ***EUINC***, y usted puede tener espacios en blanco en el nombre de directorio.

- Para eliminar la confusión, el enlazador/encriptador ahora suprimirá el archivo de salida si un error fatal ocurre durante el enlazado o la encriptación.

- Se hicieron numerosas mejoras y correcciones en la documentación. Gracias a Igor Kachan.

- Las antiguas definiciones de funciones pre−ANSI del código fuente del intérprete, se actualizaron al estilo ANSI que es más compatible con C++.

- **error corregido:** Con DJGPP C, al compilar el código producido por el Traductor, o compilando el código de fuente de Intérprete, había un error de asignación de memoria que podría hacer perder un poco de tiempo, o, en casos raros, causar una caida.

- **error corregido:** En ***Windows***, usando el Intérprete, o en un programa Traducido, a veces se necesitaba presionar Enter dos veces para salir de la ventana de consola. Gracias a Tone Skoda.

- **error corregido:** La función Euphoria ***dir()***, implementada para Lcc o Borland, no manejaba correctamente los directorios cuando tenían marcado un atributo adicional, como READ_ONLY. Gracias a Euman, quien encontró el problema, y luego mostró como corregir el código fuente del Intérprete para ***dir()***.

- **error corregido:** Ahora puede declarar un identificador de espacio de nombres con el mismo nombre que una función incorporada, sin causar muchos errores. Gracias a Martin Stachon (aunque él recomendó una solución diferente).

- **error reparado − Enlazador:** se le agregó el soporte para la nueva variable de entorno, **EUINC.** Gracias a Ross Boyd.

- **error reparado − Enlazador:** Bind −clear no funcionaba bien al agregar archivos de recursos al .exe enlazado. Gracias a Wolfgang Fritz.

- **error reparado − Enlazador:** Al usar el binder interactivamente, podría obtener un error de "variable no

inicializada" al intentar sustituir un icono de Windows. Gracias a Tony Steward.

- **error reparado − Enlazador:** En algunos casos la palabra clave "constant" se excluiría de la salida encriptada, cuando la línea anterior de entrada tiene una declaración de constante terminada en ']'. Gracias a Ross Boyd.

- **error reparado − Enlazador:** Cuando una expresión general, (no solo una cadena entre comillas), se usaba como argumento de ***routine_id()***, las rutinas locales que eran potencialmente el blanco de esa expresión, deberían tener que cambiar sus nombres (a menos que se usara −clear), causando así que ***routine_id()*** devolviera −1 en tiempo de ejecución. Las rutinas globales eran correctas.

- **error reparado − Enlazador:** El enlazador/encriptador continuaba corriendo aunque faltase un archivo de inclusión. Gracias a Ross Boyd.

- **error reparado − Enlazador para Linux:** El error de la ruta de búsqueda en Linux para ejecutables enlazados, supuestamente corregido en la versión 2.3 Alpha, no lo estaba del todo. Ahora corregido. Gracias a Ken Rhodes.

- **error reparado − Enlazador para Linux:** bindu ***−clear*** y shroud ***−clear*** con un archivo conteniendo terminadores de línea ***\r\n*** estilo ***DOS/Windows,*** daba errores de "caracter ilegal" cuando se ejecutaba su programa encriptado, y "no enlazado correctamente" al ejecutar su programa enlazado.

- **error reparado − Fuente del Intérprete:** El comando link para construir el intérprete ***DOS*** con WATCOM C, listaba un archivo .obj inexistente.

Versión 2.3 Edición Alfa, 30 de Noviembre de 2001:

Esta versión actualiza el Intérprete de Euphoria: para ***WIN32***, ***DOS32*** y ***Linux***. También actualiza el Traductor Euphoria a C para todas las plataformas, y presenta un nuevo producto − el Código Fuente del Intérprete Euphoria.

- Cambiaron los precios e incentivos de registración.
    - ♦ El Intérprete tiene solo una opción:: ***WIN32***, ***DOS32*** *y* ***Linux***, antes $59, por solo $39.
    - ♦ El código fuente del Intérprete (menos unas pocas funcionalidades registradas) está disponible ahora por $49. Vea más detalles en licencia del fuente y register.doc.
    - ♦ El Traductor continua costando $29.
    - ♦ Cuando los usuarios de Dominio Público alcanzan 300 sentencias, no perderán más el diagnóstico del error en tiempo de ejecución. Solamente perderán la utilidad ***trace()***.

  register.doc tiene más detalles, incluyendo los requisitos para obtener actualizaciones gratuitas.
- Ahora, el intérprete Euphoria puede generarse con 6 compiladores distintos, sobre 3 plataformas.

- Los nuevos *calificadores del espacio de nombres* eliminan el conflicto de nombres entre símbolos globales idénticos declarados en diferentes archivos de inclusión. También ahora, los símbolos locales sobrescribirán los símbolos globales del mismo nombre, en lugar de provocar un error. Ver Reglas de ámbito.

- El Intérprete de la Edición Completa viene con un nuevo enlazador/encriptador de 2 pasos que elimina todas las rutinas y constantes no utilizadas, dando por resultado archivos ejecutables más pequeños. También tiene la opción clear para enlazar el fuente, por lo que puede conseguir mensajes de error comprensibles de sus usuarios.

- Se introdujo una nueva variable de entorno, EUINC. Si está presente, especifica una lista adicional de directorios en los que se buscarán archivos de inclusión. El directorio que contienen el archivo principal es el primero en que que se busca, luego en los directorios en EUINC, y entonces en ***euphoria\include***.

- El Intérprete ahora soporta un nuevo modo de trazado: ***trace(3)***. Este modo registra en un archivo todas las sentencias Euphoria ejecutadas, por lo que verá la última sentencia ejecutada en el momento de cualquier colapso, así como las 499 sentencias que la preceden. Esto es particularmente útil en el caso de los colapsos a nivel de máquina donde Euphoria no es capaz de escribir un archivo ex.err. Gracias a Matthew Lewis.

- El Intérprete Euphoria ahora puede pasar más datos Euphoria (átomos y secuencias), a archivos .dll codificados en Euphoria y generados por el Traductor. Use los nuevos tipos E_ en dll.e

- En ***Linux*** se agregó RTLD_GLOBAL en la llamada a ***dllopen()***. Esto le permite enlazar correctamente con más librerías compartidas.

- En ***Linux***, al usar #! en la primera línea, el archivo fuente se convierte directamente en ejecutable, por lo que no se necesita más la extensión .exu del archivo. Los programadores Linux prefieren frecuentemente que sus archivos ejecutables no tengan extensión.

- Las rutinas call−back de Windows ahora pueden tener 9 argumentos (eran 8). Gracias a Matt Lewis.

- Además del C_DOUBLE (punto flotante de 8 bytes), ahora se soporta C_FLOAT (punto flotante de 4 bytes) para argumentos y valores de retorno desde rutinas C. Gracias a David Guy.

- Las .dlls de Windows abiertas con ***open_dll()***, ahora se cierran automáticamente cuando termina el programa. Esto evita una pequeña pérdida de memoria. Gracias a Euman.

- **safe.e** tiene una nueva opción para comprobar solo los bordes de bloques registrados y no preocuparse si se utilizan otros bloques de memoria.

- ***get_bytes()*** es un 30% más rápida.

- Se aceleró ***allocate_string()***. Gracias a Aku.

- El programa de demostración ***mydata.ex*** ahora usa una base de datos EDS.

- **error reparado − Traductor:** Cuando una variable global o local que contiene una secuencia o un número de punto flotante se asignaba al resultado de una función, en algún punto durante la llamada, la variable global o local se sobreescribía, produciendo cierto daño que provocaría más adelante el colapso del programa. Gracias a Sergio Gelli.

- **error reparado − Intérprete para Linux:** Los programas de más de mil líneas tenían la posibilidad (quizá el 20%) de tener un colapso de sus sentencias al ejecutarse.

- **error reparado − Intérprete: s[i][j]...[k] = s** causaba el colapso del intérprete, es decir, una asignación de una secuencia completa al elemento de un elemento de sí misma, usando 2 o más niveles de subíndices. Gracias a Henri Goffin.

- **error reparado − Enlazador para Linux:** (Ken Rhodes) Los programas ejecutables enlazados guardados en alguna parte de la ruta de búsqueda no funcionaban correctamente salvo que estuvieran en el directorio actual o se le especificara la ruta completa al ejecutable.

- **error reparado − Enlazador:** "with profile_time" en un programa enlazado o encriptado, causaría una caida.

- **error reparado − Intérprete DOS:** En ***image.e***, se corrigió ***put_screen_char()*** para tener: **if overflow > 0 then ...** en lugar de: **if overflow then ...**

- **error reparado − Intérprete:** Una optimización de tiempo para subrangos podría causar en casos raros, la pérdida de mucho espacio. Se ajustó la optimización para manejar esos casos. Gracias a Brian Clausing.

- **error reparado − safe.e:** ***free()*** y ***free_low()*** no liberaban realmente el bloque de memoria, y en ***Linux free()*** podía causar una violación de segmento. Las mismas rutinas en **machine.e** estaban bien.

- **error reparado − Traductor para DJGPP:** Se corrigieron un par de diferencias de menor importancia de ***ex.exe*** en la exhibición del texto.

- **error reparado − define_c_var():** trabaja también en ***WIN32***. Se corrigió la documentación.

Versión 2.2 Edición Oficial para WIN32+DOS32, 14 de Enero de 2000:

- Ahora se emiten mensajes de error mejores desde el intérprete, y desde el programa de enlazado, para algunos errores típicos que los usuarios puedan cometer al enlazar un programa.

- Se mejoró la documentación en algunos lugares.

- El programa de demostración **window.exw** muestra como cargar el icono de Euphoria contenido en ***exw.exe***

- Language War usa la versión mejorada de **putsxy.e** de Jiri Babor.

Versión 2.2 Edición Beta para WIN32+DOS32, 23 de Diciembre de 1999:

La mayoría de las nuevas rutinas desarrolladas para Euphoria 2.2 en ***Linux***, se migraron a ***WIN32*** y ***DOS32***. Ellas son: ***sleep()***, ***chdir()***, ***flush()***, ***lock_file()***, ***unlock_file()***, ***crash_file()***, ***get_screen_char()*** y ***put_screen_char()***. Referirse más abajo a las notas para Linux, para obtener una descripción de esas rutinas, o LIBRARY.DOC.

Algunos arreglos de errores de plataforma cruzada y otras mejoras misceláneas fueron llevados a cabo durante la migración a Linux. Estas correcciones y mejoras se migraron nuevamente a ***WIN32+DOS32***. Ver las notas de la versión para Linux (debajo).

Además, se hicieron las siguientes mejorías específicamente para ***WIN32*** y ***DOS32***:

- **exw.exe** contiene el icono Euphoria que ***Windows*** ahora lo muestra automáticamente. El icono es una contribución de Gary Dumer. Los usuarios registrados pueden cambiar este icono cuando enlazan un programa.

- **exw.exe** es ahora un ejecutable comprimido de solo 73K. Se lo comprimió usando la herramienta de compresión UPX para archivos ejecutables.

- **ex.exe** se actualizó con la última versión del expansor DOS CauseWay. Se eliminó un problema en el que CauseWay a veces lo limitaría a 64Mb de memoria bajo algunas configuraciones de ***DOS*** y se corrigieron algunos errores menores.

- **error corregido:** El seguimiento del error podía estrellarse o imprimirse con errores a veces al ocurrir una falla type_check. Podía suceder solamente cuando 1 se sumaba expresión, y el resultado no entero de la expresión se asignaba a una variable declarada como número entero.

- **error corregido:** Si ***text_rows()*** se llamaba como primera rutina que necesita una ventana de la consola WIN32, Euphoria no podría fijar el nuevo número de líneas del texto en la consola.

Versión 2.2 Edición Oficial para Linux, 22 de Noviembre de 1999:

- **Todas las plataformas: error corregido:** Si una rutina Euphoria se llamaba a sí misma recursivamente desde el interior de un ciclo for, y en un nivel de recursión el ciclo for contaba hacia el límite superior, y en otro nivel de recursión el ciclo for contaba hacia el límite inferior, el ciclo for funcionaría incorrectamente probablemente en uno de los niveles. Gracias a Delroy Gayle.

- Se mejoró la documentación en muchos lugares, especialmente con respecto a la plataforma ***Linux***.

Versión 2.2 Edición Beta para Linux, 22 de Octubre de 1999:

*La mayoría de estas características y arreglos de errores también estarán disponibles en la Versión 2.2 para **WIN32** + **DOS32**.*

- **platform()** se movió de ***misc.e*** a ***exu*** para eliminar el consumo de recursos en la llamada a la función. ***platform()*** ahora computa sin perder tiempo. El compilador simplemente agrega el valor constante apropiado.

- **lock_file()** y **unlock_file()** se han agregado para permitir que los procesos múltiples compartan el acceso a los archivos. Esto puede ser importante en la programación CGI y otras áreas.

- **flush()** forzará el contenido del búfer de memoria hacia un archivo o un dispositivo.

- **chdir()** cambiará a un nuevo directorio actual y le permitirá saber si la operación fue correcta.

- **sleep()** suspenderá la ejecución de su programa por una cantidad de segundos, y le permitirá al sistema operativo programar otro proceso.

- **put_screen_char()** escribirá un caracter y sus atributos (colores, etc.) a la pantalla.

- **get_screen_char()** leerá un caracter y sus atributos desde la pantalla.

- **save_text_image()** ahora funciona en ***Linux*** (como en ***DOS32***). Copia una imagen rectangular de texto de la pantalla.

- **display_text_image()** ahora funciona en ***Linux*** (como en ***DOS32***). Escribe una imagen rectangular de texto en la pantalla.

- La advertencia de "corto−circuito" ahora da el nombre del archivo y el número de línea de la llamada posiblemente corto−circuitada. Se hicieron aclaraciones menores en algunos otros mensajes de error.

- Se hicieron mejoras menores a **ed** y **search**.

- Se solucionó un problema de portabilidad en **how2reg.ex**.

- Se comprimió mejor a **exu**. Ahora es realmente un poco más pequeño, aunque se le agregó funcionalidad.

Versión 2.2 Edición Alfa de pruebas para Linux, 24 de Agosto de 1999:

*Muchas de estas funcionalidades y corrección de errores estarán disponibles también en la versión 2.2 para **WIN32***

+ DOS32.

- La documentación se actualizó para incluir información específica de **Linux** para las rutinas de la biblioteca y Euphoria en general.
- Ahora existe una Edición Completa para **Linux**, incluyendo **enlazado** y **encriptación**. Ver **register\register.doc**.
- Ahora hay soporte de ratón en el **modo de texto** usando **get_mouse()**. Tiene que tener corriendo el **servidor GPM**. Este funciona en una consola de texto o en una ventana **xterm**.
- **Linux: define_c_var(nombre)** devolverá la dirección de una variable global de C en una librería compartida.
- Se confirmó que puede llamar rutinas Euphoria desde las rutinas C de **Linux** sando exactamente el mismo mecanismo que Euphoria de **WIN32**. Ver **euphoria/demo/linux**.
- Se agregó un ejemplo de como crear sus propias rutinas de la librería compartida y de llamarlas desde Euphoria. Ver **euphoria/demo/linux**.
- **Todas las plataformas: crash_file(archivo)** provocará que se escriban los mensajes de diagnóstico en **archivo**, en lugar de **ex.err**. Puede usar **crash_file("/dev/null")** para obtener diagnósticos en pantalla pero no a un archivo. **crash_file("")** equivale a "sin diagnósticos" (a pantalla o ***ex.err***).
- El modo de **Trazado** en **xterm** ahora detecta las teclas **F1/F2**.
- **time()** ahora informa la hora la del CPU.
- **search**, **guru** y **cdguru** ahora ponen su salida en el directorio $HOME en lugar del actual.
- **#!** quedó restringido a la primera línea de un archivo.
- **Todas las plataformas:** En **ed**, los comandos **Esc−N**, **Esc−D**, **Esc−F** y **Esc−R** mostrarán inmediatamente su última elección. Puede presionar **flecha hacia arriba/fecha hacia abajo** para ver otras opciones, o limpiar la elección. Si usted comienza a escribir sin corregir, limpiará la opción y tomará su nueva entrada.
- **free_console()** fijará los parámetros de la terminal de nuevo a normal. Normalmente, al funcionar un programa Euphoria, los parámetros se fijan en la manera que **curses** los desea y se fijan de nuevo a normal cuando el programa termina. Si su programa necesita terminar de una manera extraña (con excepción de llamar a ***abort()***), se debería llamar primero a **free_console()**.
- **error corregido: get()** ahora considera a '**\r**' como un espacio. Esto es importante al leer archivos de DOS.
- **Todas las plataformas: error corregido:** No se mostraba inmediatamente una falla **type_check** cuando se sumaba 1 a una variable entera que estaba establecida al máximo valor entero (1070 millones). Gracias a Jeff Fielding.
- **Todas las plataformas: error corregido:** No siempre se detectaba un exponente incorrecto en un número de punto flotante. Gracias a Lionel Wong.
- **Todas las plataformas:** El funcionamiento del asignador de almacenamiento se ha mejorado en ciertos casos. Se corrigió un error que podría hacer al intérprete estrellarse cuando casi se queda sin memoria.

Versión 2.2 Edición pre−Alfa #4 para Linux, 15 de Julio de 1999:

- Puede llamar rutinas de C en librerías compartidas de **Linux** (archivos **.so**). Ver algunos ejemplos en **euphoria/demo/linux/callc.exu**.
- Si su programa no emite nada en la ventana **xterm**, **exu** no mostrará el mensaje "Presione Enter".
- **Todas las plataformas: ed** ahora le permite recordar los comandos anteriores, usando **flecha hacia arriba** y **flecha hacia abajo**, similar a *doskey* en ***DOS*** y a *shell history* en ***Linux***. Esto funciona con cualquier cadena que escriba para **Esc−N** (nuevo archivo), **Esc−D** (comando de ***Linux***), **Esc−F** (encontrar cadena) o **Esc−R** (reemplazar cadena).Además, ahora puede usar las **teclas de flecha**, **Inicio**, **Fin**, **Supr,** etc. para editar cadenas antes de presionar **Enter**.

Versión 2.2 Edición pre−Alfa #3 para Linux, 8 de Julio de 1999:

- En una ventana **xterm**, **exu** lo invitará a presionar ***Enter*** antes de terminar. Sin esto, **xterm** restaura la pantalla tan rápidamente que usted no ve ninguna salida o mensaje de error.

- Se hizo un cambio en el código interno de la función **rand()** de Euphoria (El algoritmo no cambió). Esperanzadamente esto permitirá a ***rand()*** trabajar en todas las distribuciones de **Linux**. Permítanos saber si ***rand()*** aún falla.

- **ed:** El comando **Esc−H** mostrará los archivos de ayuda de Euphoria. Esto se cortó en pre−alpha#2.

- En una ventana **xterm**, **video_config()** de Euphoria ahora informa la cantidad correcta de líneas y columnas − esto ayuda a que **ed** funcione mucho mejor. **ed** trabajará con el tamaño inicial de la ventana en efecto cuando **ed** inicia.

- **ed:** Las teclas **F1**, **F2**, **F3**, **F4**, **Inicio**, **Fin**, y **Supr** ahora funcionan en **xterm** (bajo Red Hat 5.2 al menos). Las otras teclas F ya estaban funcionando. **Re Pag/Av Pag** y algunas otras teclas no funcionarán todavía − considérese libre de agregar sus propias teclas "alternativas".

- **exu** es aún más pequeño − solo 82K.

Versión 2.2 Edición pre−Alfa #2 para Linux, 6 de Julio de 1999:

- Se enlazó estáticamente la librería **ncurses** en **exu**.

- **exu** es ahora un ejecutable comprimido (97K).

- **error corregido: ed** ahora puede editar archivos cuyo nombre está en mayúsculas.

- Se quitó la demora de una fracción de segundo al presionar la tecla **Esc** en **ed**.

Versión 2.2 Edición pre−Alfa #1 para Linux, 1 de Julio de 1999:

- Se liberó la primera versión de Euphoria para **Linux**.

Versión 2.1 Edición Oficial para WIN32 + DOS32, 29 de Marzo de 1999:

- Se optimizaron las actualizaciones de la pantalla de **trazado**. Se eliminaron los refrescos innecesarios del código fuente y de las variables a la pantalla. Cuando un refresco es necesario, ahora es ligeramente más veloz. Esto hace una notable diferencia en **exw.exe**, y también en **ex.exe** en los modos de gráficos de píxel. Para **ex.exe** en los modos de texto, reduce el parpadeo de la pantalla levemente.

- El programa **install** ya no necesita que la ruta tenga menos de 128 caracteres. Se le advertirá simplemente si no lo es. Las nuevas versiones de DOS permiten rutas más extensas. Gracias a Steve Adams.

- Se agregó una verificación de error adicional a **unregister_block()** en **safe.e**. Gracias a David Guy.

Versión 2.1 Edición Beta, 5 de Marzo de 1999:

- Se convirtieron a HTML los archivos en el directorio **euphoria\doc**. Todo archivo **.doc** en el directorio **doc** tiene su correspondiente archivo **.htm** en el directorio **euphoria\html**. Se hicieron muchas mejoras y aclaraciones en la documentación.

- Ahora se le advertirá cuando tiene código inmediatamente después de las sentencias **exit**, **return** o **abort()**. Este código puede que nunca se ejecute. Sugerido por Gabriel Boehme.

- **safe.e** ya no incluye a **graphics.e**. Esto elimina posibles conflictos de nombres cuando se sustituye a **safe.e** con **machine.e**.

- Usando código suministrado por David Guy, **safe.e** le permitirá agregar o quitar de la "lista de direcciones seguras" los bloques de memoria asignados externamente. **Ver las nuevas rutinas de librería: register_block()** y **unregister_block()**.

- **message_box()** ahora usa el **identificador de la ventana activa,** en lugar de NULL. Esto fuerza al usuario a contestar a su mensaje antes de que él pueda continuar interactuando con su programa. Lo no prevendrán de interactuar con otros programas. Gracias a Austin C.

- Se aceleraron un 5% **get()** y **value()**. Gracias a Gabriel Boehme.

- Se hizo a **exw.exe** menos propenso a colgarse misteriosamente cuando lo ataca un virus.

- **sanity.ex** verifica su instalación de Euphoria. Se le avisará si las variables PATH o EUDIR no existen, o si

sus archivos **ex.exe**, **exw.exe**, **pdex.exe**, o **pdexw.exe** están dañados o incorrectamente instalados en **euphoria\bin**.

- La seguridad de los programas **bound** y **scrambled** se ajustó un poco más. Gracias a Rusty Davis.

- Para ahorrar espacio en **euphor21.zip**, el programa de **instalación** *genera* los archivos HTML y DOC desde una fuente común, usando el **generador de documentación** de Junko Miura. En el proceso, se borra el generador, pero puede descargarlo desde el sitio de RDS.

- Al ocurrir un error **type_check** se le advertirá si el tipo devolvió erróneamente una *secuencia* para el resultado "true/false". Previamente, un resultado de secuencia era informado simplemente como un error type_check. Sugerido por Ralf Nieuwenhuijsen.

- El código de **demo\win32\winwire.exw** se aclaró considerablemente.

- El programa **install** le advertirá de cambiar su archivo **autoexec.bat** cuando instale una nueva versión de Euphoria en una unidad *diferente*.

Versión 2.1 Edición Alfa de pruebas, 15 de Enero de 1999:

- Hemos hecho algunos cambios en el embalaje, precios e incentivas de registración para el producto Euphoria:
  - ♦ Se redujo el precio del paquete Plataforma Dual (***DOS32+WIN32***) de $53 a $39 U.S.

  - ♦ Se discontinuó el paquete de Plataforma Unica (solo ***DOS32***), anteriormente $32.

  - ♦ Se discontinuó el manual impreso. En su lugar, existe una versión oficial del mismo en HTML, incluido en el archivo .zip de la Edición de Dominio Público.

  - ♦ Todos los archivos de inclusión útiles de terceras partes, como **Win32Lib.ew** y algunos otros, estarán "estampados" por RDS con un número de código que los hace ***libres***, igual que los archivos en **euphoria\include**. Ellos no sumarán a su sentencia de conteo, con tal que usted no los modifique perceptiblemente. Esto permitirá también a desarrolladores de terceras partes obtengan una información de diagnóstico mejor de sus usuarios.

  - ♦ **Binding**, **shrouding** y **profiling** ahora son parte de la Edición Completa únicamente. Estas son funcionalidades que los novatos no necesitan, pero que los programadores si pueden valorar.
- Se agregaron los operadores de asignación abreviados `+=` `-=` `*=` `/=` `=`. Por ejemplo, en lugar de decir:

```
count = count + 1
```

Ahora puede decir:

```
count += 1
```

En lugar de decir:

```
matrix[row][column] = matrix[row][column] * 5.0
```

Ahora puede decir:

```
matrix[row][column] *= 5.0
```

En lugar de decir:

```
test_scores[start..finish] = test_scores[start..finish] / 100
```

Ahora puede decir:

```
test_scores[start..finish] /= 100
```

Ver más detalles en **refman.doc**.
- Euphoria usa ahora la evaluación de "corto−circuito" de expresiones **and** y **or** en condiciones **if/elsif/while.** Por ejemplo, en una condición **and**:

```
if A and B then ...
```

el intérprete salteará la evaluación de la expresión B toda vez que la expresión A sea 0 (falso), puesto que

sabe que el resultado total tiene que ser falso. En una condición **or:**

```
while A or B do ...
```

el intérprete salteará la evaluación de la expresión B toda vez que la expresión A sea no nula (verdadero), ya que sabe que el resultado total tiene que ser verdadero.
El código de Euphoria escrito antes de la versión 2.1 puede no funcionar correctamente si la expresión B contiene una función con ***efectos secundarios*** tales como fijar una variable global, usar E/S, etc. Esta clase de código es en la práctica muy rara, pero en caso de ocurrir, ahora una advertencia se emitirá si una función con efectos secundarios puede cortocircuitarse.

Al saltear la evaluación de B, la evaluación de cortocircuito es típicamente más rápida, y permitirá escribir sentencias como:

```
if atom(x) or length(x)=1 then ...
```

que generaría un error en las versiones más viejas de Euphoria siempre que x fuera un **atom**, puesto que el ***length()*** no se define para átomos.

Ver más detalles en **refman.doc**.

- Se agregaron varias nuevas rutinas.

Incorporadas a ex.exe/exw.exe:

profile()	−	activa/desactiva el **análisis de perfiles,** por lo que puede apuntar la ejecución de **profile** y **profile_time** en eventos en particular dentro del programa.
system_exec()	−	el devuelve el código de salida de la llamada de un archivo **.exe** o **.com**, u otro programa Euphoria.
equal()	−	compara por igualdad 2 objetos Euphoria. Es equivalente a: ***compare(a,b) = 0*** pero más legible.

Agregadas a varios archivos de inclusión:

walk_dir()	−	viaja recursivamente a través de un directorio y sus subdirectorios, ejecutando la rutina que se le suministra.
reverse()	−	devuelve una secuencia en orden inverso.
sprint()	−	devuelve la representación de cadena de un objeto Euphoria.
arcsin()	−	función trigonométrica inversa.
arccos()	−	función trigonométrica inversa.
get_bytes()	−	devuelve los siguientes 'n' bytes desde un archivo.
prompt_number()	−	pide al usuario ingresar un número.
prompt_string()	−	pide al usuario ingresar una cadena.
instance()	−	**WIN32:** devuelve el handle de instancia de un programa.
PI	−	se agregó la constante PI − 3.14159... a **misc.e**.

Ver más detalles en **library.doc**.

- Ahora se puede ver la documentación principal de Euphoria con el navegador de Internet. Los archivos de texto plano **refman.doc** y **library.doc** aún están en el subdirectorio **doc**, pero ahora tenemos **refman.htm** y **library.htm** en el nuevo subdirectorio **html**. Desarrollamos una herramienta (escrita en Euphoria) que nos permite fácilmente mantener actualizadas las versiones HTML y de texto plano de **refman** y **library**.

- Se agrandó y clarificó la documentación en varias partes.

- **WIN32:** se pueden crear una cantidad **ilimitada** de rutinas call−back en Euphoria, mientras cada rutina es una función con 0 a 8 parámetros. Ver **platform.doc**. En la vesión 2.0 usted podría solamente tener una rutina call−back que tenía que tener exactamente 4 parámetros.

- Se agregó la palabra clave **xor** para complementar a: **and/or/not** y **xor_bits()**, por ejemplo:

```
if a xor b then...
```

xor también trabaja con secuencias. Es similar a **or**.

- La rutina de librería **dir(path)** ahora soporta oficialmente el uso de **comodines *** y **?** en la ruta que se le suministra. Esta característica estaba siempre disponible, pero no fue documentada hasta este momento. Por ejemplo:

```
info = dir("mydata\\*.d?t")
```

- **optimización:** se redujo en un 30% en promedio, el consumo de recursos en la subrutina call+return. El incremento de velocidad ocurre para todas las llamadas normales a funciones/procedimientos/tipos, verificaciones de tipos definidas por el usuario, llamadas a ***call_proc()/call_func()*** usando **routine id**, y call−backs de Windows. Solamente las llamadas recursivas tienen el mismo consumo que antes. Los programas con una razonablemente alta tasa de llamadas, pueden ser un 10% más rápidos debido a esto.

- **optimización:** Se implementó la ***rectificación*** de saltos. El compilador optimizará los saltos en el código interno tal como un salto de A−>B, donde la posición B contiene un salto a la posición C, se optimizará directamente como un salto de A−>C. Aún algo como A−>B−>C−>D se puede rectificar a A−>D. Esto ocurre a menudo en los ciclos while que contienen sentencias if.

- **optimización:** En muchos casos, las verificaciones de inicialización de variables se reemplazan ahora por "no−ops" después que se realiza la primera verificación. Euphoria ya tenía optimizadas muchas verificaciones en tiempo de compilación.

- **optimización:** **get()** y **value()** son mucho más rápidos gracias a Jiri Babor y algunas otras optimizaciones de RDS. La nueva v2.1 de **ex.exe** con el nuevo **get.e** v2.1 es:
  1.45 veces más rápido leer una secuencia de números de punto flotante desde un archivo y
  2.25 veces más rápido cuando se lee una secuencia de enteros desde un archivo.
- **optimización:** **power(x,2)** se convierte internamente a x*x, que es mucho más rápido en todos los casos, especialmente cuando x es un entero grande o un número de punto flotante.

- **optimización:** Gracias a Jiri Babor, **int_to_bits()** es un 15% más rápido en la mayoría de los casos.

- **optimización:** Trazar una secuencia extensa de píxeles en modos gráficos de 16 colores, es un 3% más rápido.

- **optimización:** **draw_line()** aumentó su velocidad un poco.

- **Language War** ha tenido una cirugía estética importante. Ahora corre en el **modo 18 de gráficos de píxel** (640 x 480 x 16 colores) en lugar de **modo de texto**. También tiene paralelismo de ***grano fino***, es decir, virtualmente cualquier cosa puede ocurrir en paralelo a cualquier otra. Se pueden dibujar simultáneamente varios torpedos, fasores, etc., mientras las naves se mueven, se ingresan comandos, hay cosas explotando, etc. Incluso la sincronización necesaria para los efectos sonoros del altavoz de la PC es manejada por el ***planificador de tareas***. ***No*** se ejecutan ciclos de demora durante el juego. La exploración de la galaxia ahora muestra una imagen a escala de la galaxia completa, en lugar que solamente un grupo de números.

- El formato de impresión por defecto para átomos se cambió de "%g" a "%**.**10g". ***print()***, **?**, la utilidad **trace**, y los volcados en **ex.err** usan este formato. Esto permite que los enteros grandes de −9,999,999,999 a +9,999,999,999 se impriman como enteros, en lugar de usar notación científica. También aporta 10 dígitos de exactitud, en lugar de 6, al mostrar números fraccionarios. Art Adamson y otros dejaron claro que se tenían que mostrar más dígitos.

- El estado de todos los parámetros **with/without** se guarda al comienzo de un **archivo de inclusión**, y se recupera al final de un **archivo de inclusión**. Un archivo de inclusión puede cambiar los parámetros, pero se restablecerán al terminar el archivo. Por ejemplo, las advertencias se tienen que desactivar dentro de un archivo de inclusión (y dentro de todos los archivos que incluya). Consecuentemente algunos programas ahora muestran advertencias donde antes no se mostraban.

- Las advertencias se muestran ahora ***después*** que su programa termina la ejecución, por lo que no serán borradas por ***clear_screen()***, ***graphics_mode()***, etc. Consecuentemente algunos programas ahora muestran advertencias donde antes no se mostraban.

- La seguridad del código encriptado y del código enlazado se mejoró gracias a las ideas aportadas por Rusty Davis. Cuando un programa enlazado inicia la ejecución, se realiza una verificación rápida de su integridad para detectar cualquier daño o violación. Aún se pueden agregar datos al final de un archivo ***.exe*** **enlazado**, mientras que su última línea es **abort(x)**.

- El editor **ed** le permite ahora ver y editar más allá de la columna 80.

- **ed** tiene un nuevo comando: **Esc−M** (modificaciones). Demostrará las diferencias entre el archivo original en disco y el búfer de edición actual. Esto puede ser muy útil cuando usted se ha olvidado de qué cambios hizo, y se está preguntando si es seguro guardarlos.

- La ventana de **trazado** ahora proporciona un comando**Q** mayúscula que le permite al programa ejecutarse hasta el final, ignorando cualquier comando ***trace(1)***. El comando **q** minúscula le permite ejecutarse hasta el

siguiente ***trace(1)***.

- Se mejoró **safe.e** (versión de depuración de **machine.e**). Ahora capturará automáticamente los casos adicionales donde los datos se escriben ilegalmente ***momentos antes***, o ***enseguida después***, de los límites de un bloque asignado de la memoria. Esto puede ser particularmente útil en **WIN32** donde Windows pudo sobreescribir uno de sus bloques de tamaño insuficiente. Sin una herramienta tal como **safe.e**, este tipo de error podría tomar horas o aún días encontrarlo.

- Se creó el directorio **euphoria\tutorial** para mantener varios programas tutoriales pequeños.

- El límite en la cantidad de archivos abiertos se elevó de 15 a 25. Tres de esos archivos son 0,1,2: entrada estándar, salida estándar y error estándar, por lo que ahora tiene hasta 22 archivos propios abiertos simultáneamente (por lo que sabemos, nadie excedió nunca el viejo límite, pero parecía sabio elevarlo).

- Cuando el usuario simplemente escribe **ex** o **exw** y se le pregunta el nombre del archivo Euphoria **.ex** o **.exw** a ejecutar, **command_line()** ahora se actualizará para incluir como segundo argumento de la línea de comandos, el nombre del archivo, como si el usuario hubiera escrito originalmente: **ex <nombre de archivo>**. Gracias a Mathew Hounsell por sugerir esto.

- **mset.ex** ahora guarda imágenes en formato **.bmp**. Antes usaba un formato comprimido no estándar.

- **lines.ex** (**lines.bat**) ahora informa también las líneas no−en−blanco/sin−comentarios. Esto ***no*** es lo mismo que la "sentencia de conteo" usada por Euphoria para el límite del diagnóstico, pero está generalmente dentro del +/− 10%, asumiendo que usted escribe una sentencia por línea.

- Los literales numéricos mayores que 1e308 (groseramente), se fijan ahora a **+/− inf**. Causaban un error de tiempo de compilación.

Versión 2.0 Edición Oficial, 25 de Marzo de 1998:

- Cambió el procedimiento de instalación. El archivo Euphoria .zip contiene un gran archivo **bundle.dat** que contiene más de 100 archivos. Esto hace más sencillo de localizar los archivos importantes: **readme.doc**, **install.bat**, etc. que se deberían ver antes de realizar la instalación. Como resultado, el archivo .zip es 35K más pequeño.

- **shroud** le avisará que use **bind/bindw** si intenta crear un archivo fuente encriptado con la extensión "**.**exe".

Versión 2.0 (Beta), 26 de Febrero de 1998:

- El intérprete de WIN32, **exw.exe**, es ahora un **verdadero** programa **WIN32 GUI**. En la versión 2.0 Alpha era un programa ***WIN32*** de ***consola*** que siempre estaba asociado con una ***consola*** o ventana DOS. Ahora se creará una ventana de consola de estilo DOS, solamente si su programa necesita una.**exw** creará automáticamente una nueva ventana de consola la primera vez que su programa escriba en pantalla , lea desde el teclado. o llame a cualquier rutina de librería que necesite una consola para funcionar. La consola desaparecerá automáticamente al teminar la ejecución del programa.

- Una nueva rutina de librería, **free_console()**, borrará inmediatamente la ventana de consola si existe una.

- La Edición Completa de Euphoria provee la opción **−scramble** de **bind** y **shroud** para mejorar la seguridad de los programas que distribuye.

- Ahora puede pasar **átomos** Euphoria a rutinas de C como argumentos de punto flotante de 64 bits de C, y recibir un resultado de punto flotante desde una función de C.

- **exw.exe (beta)** corre de 10% a 15% más rápido que **exw.exe (alpha)** (basado en **sieve.ex**, **shell.ex**, etc.). El compilador WATCOM C estaba haciendo mal el trabajo de optimización de una sección crítica del intérprete cuando generaba **exw.exe**, pero producía un código excelente al generar **ex.exe**. Con algunos cambios triviales en el código del intérprete de C, WATCOM produce código excelente para ambas plataformas.

- El programa promedio tiene 60K más de memoria disponible, antes de tener que usar el archivo de intercambio.

- Se eliminó el límite en el tamaño de un procedimiento simple, función o tipo.

- Se eliminó el límite en el tamaño de una sentencia simple de alto nivel.

- Se incrementó de 150 a 256 la cantidad total de archivos de inclusión que un programa puede tener.

- Se agregaron algunas **optimizaciones**. Las siguientes nuevas formas de expresiones son más rápidas:

```
2 * x
x * 2
1 + x
```

donde 'x' puede ser cualquier expresión de tipo átomo o secuencia.

- El nuevo archivo de documentación, **perform.doc**, trae muchos consejos para programadores obsesionados con el rendimiento.

- Si llama una rutina de C usando **c_func()**, pero la enlaza usando **define_c_proc()**, obtendrá un mensaje de error. Similarmente, si la llama usando **c_proc()**, pero la enlazó usando **define_c_func()** obtendrá un mensaje de error. Esta restricción estaba documentada, pero no realmente obligada a cumplirse en Alfa 2.0. Algunos programas escritos para la versión Alfa tendrán que ser corregidos.

- Ahora verá el nombre real de la rutina C o Euphoria a la que está intentando llamar, al obtener un mensaje de error desde **call_proc()**, **call_func()**, **c_proc()**, o **c_func()**.

- La nueva opción **−clear_routines** de **bind** y **shroud** dejará los nombres de todas las rutinas sin encriptar. Esto es necesario si su programa llama a **routine_id()**. Se le avisará si usa **routine_id()** y no tiene elegida esta opción (los usuarios registrados pueden usar **−scramble** junto con **−clear_routines** para recuperar un alto nivel de encriptado).

- Si un conflicto de nombres se presenta con un símbolo global, el **encriptador** le avisará, y después elige un nuevo nombre. Se usaba para abortar con un mensaje.

- Ya no es más posible **trazar** o **analizar** código encriptado.

- Se agregó un nuevo programa de demostración (**hash.ex**) a **euphoria\demo**.

- Se movió **freq.ex** de **euphoria\bin** a **euphoria\demo** y se lo renombró como **tree.ex**.

- Un nuevo archivo de documentación, **bind.doc** describe las características de **bind.bat** y **shroud.bat**. Se redujo la descripción anterior en **refman.doc.**

- El archivo **overview.doc** da un vistazo rápido a todos los archivos de documentación.

- La descripción de **get_mouse()** en **library.doc** discute el problema de los modos gráficos de ancho 320 (tiene que dividir el valor de la coordenada 'x' por 2).

Versión 2.0 Edición Alfa, 5 de Noviembre de 1997:

- Se agrega una ***nueva plataforma***. **exw.exe** correrá programas Euphoria usando el sistema operativo **WIN32** (Windows de 32 bits). **ex.exe** correrá programas usando **DOS32** (***DOS*** extendido). Ver más detalles en **platform.doc**.

- Se agregaron las siguientes rutinas de librería.

Tanto para DOS32 como WIN32:

platform()	− informa soobre cual plataforma se está ejecutando (la constante PLATFORM está disponible en Euphoria 2.2 y siguientes).
routine_id()	− obtiene un identificador entero pequeño para un procedimiento o función Euphoria.
call_proc()	− llama a un procedimiento Euphoria usando su ID.
call_func()	− llama a una función Euphoria usando su ID.
custom_sort()	− ordena una secuencia usando una función de comparación que usted esopecifica.
poke4()	− almacena un número en 4 bytes de memoria. ***poke4(address, value)*** es al menos 10 veces más rápida que: ***poke(address, int_to_bytes(value))***. ***poke4()*** también funciona con secuencias de valores.
peek4s()	− lee 4 bytes de memoria como un entero con signo (funciona también con secuencias).
peek4u()	− lee 4 bytes de memoria como un entero sin signo (funciona también con secuencias). ***peek4u(address)*** es 10 veces más rápido que: ***bytes_to_int(peek({address, 4}))***.
allocate_string()	− asigna y almacena una cadena termina en 0 en la memoria.

Para WIN32 solamente:

open_dll() – abre un archivo dll de Windows.

define_c_proc() – define una rutina C que se llamará desde Euphoria (sin valor de retorno).

define_c_func() – define una rutina C que se llamará desde Euphoria (con valor de retorno).

call_c_proc() – llama a una rutina de C desde Euphoria (sin valor de retorno).

call_c_func() – llama a una rutina de C desde Euphoria (con valor de retorno).

call_back() – obtiene una ***dirección de call−back***, por lo que Windows puede llamar a su rutina Euphoria cuando el usuario interactua con su ventana.

message_box() – muestra una ventana simple **Sí/No/Cancelar.**

- Nuevos programas de demostración:
  - **csort.ex**
  - **email.exw**
  - **window.exw**
  - **winwire.exw**
  - **dsearch.exw**

- Nuevos archivos de inclusión:

safe.e – versión de depuración de **machine.e**

misc.e – misceláneas

dll.e – acceso a dll's

msgbox.e – Caja de mensajes de Windows

- Se hicieron las siguientes mejoras en la versión **DOS32**:
  - En sistemas Pentium y superiores, los cálculos en punto flotante son 20% más rápidos (en **exw** es un 20% más rápido que **ex** para 2.0).
  - **printf()** a la pantalla y **print()** a la pantalla son significativamente más rápidos en la mayoría de los casos.
  - La ventana de **trazado** se actualiza más rápido.
  - El **análisis de prfiles por tiempo** es más exacto regarding **getc()**.
  - El demo **mset.ex** corre un 30% más rápido.

Items Destacados de Antiguas Ediciones

Versión 1.5a, 13 de Junio de 1997:

- Se optimizan muchas operaciones y rutinas de librería.
  - **get_key()** es 100 veces más rápida que cuando no hay una clave en el búfer.
  - **get_all_palette()** es casi 100 veces más rápida, lo que hace a **save_screen()** mucho más rápida.
  - Se han incorporado las siguientes rutinas directamente en **ex.exe**, para evitar el consumo de recursos llamando a ***machine_proc()*** o ***machine_func():*** **pixel()**, **get_pixel()**, **mem_set()**, **mem_copy()**.
  - **poke()** de una secuencia larga en memoria, other than video memory, es un 50% más rápida.
  - **pixel()** es 4 veces más rápida en modo 19.
  - **get_pixel()** es más rápida en todos los modos.
  - **display_image()** es alrededor 30% más rápida en la mayoría de los modos y hasta 4 veces más rápida en modo 19 debido a que ***pixel()*** es más veloz.
  - Todas las operaciones aritméticas y de bits aplicadas a secuencias de enteros son un 29% más rápidas.
  - a **b (concatenación) es un 15% más rápida en la mayoría de los casos, y drásticamente más rápida en el caso en que una secuencia muy grande crece debido a la concatenación de muchas pequeñas.**
  - **getc()** es un 12% más rápida.
  - **match()** es un 8% más rápida en casos típicos.
  - **append()/prepend()** son un 15% más rápidas en la mayoría de los casos.
  - **find()** de un entero dentro de una secuencia de enteros es un 64% más rápida.
  - La formación de una secuencia de dos elementos **{a,b}** es un 11% más rápida.
  - El copiado interno de una secuencia compartida cuando ya no se puede compartir es un 15% más rápido.

Versión 1.5, 21 de Marzo de 1997:

- Se agregan las siguientes rutinas de librería, que se describen completamente en **library.doc**.
  - **allow_break()**
  - **check_break()**
  - **mem_copy()**

- **mem_set()**
- **atom_to_float32()**
- **atom_to_float64()**
- **float32_to_atom()**
- **float64_to_atom()**
- **get_all_palette()**
- **save_bitmap()**
- **save_screen()**
- **arctan()**
- **and_bits()**
- **or_bits()**
- **xor_bits()**
- **not_bits()**
- **get_vector()**
- **set_vector()**
- **lock_memory()**
- **tick_rate()**

Se agrega **with profile_time** (análisis de perfiles por tiempo para ***DOS32***).

Versión 1.4b, Octubre de 1996:

- **mset.ex** tiene una caja de selección más visible en su grilla. También tiene más velocidad.
- **ed.ex** permite que los caracteres especiales mayores que el ASCII 127 se ingresen presionando la tecla **Alt** y escribiendo dígitos en el teclado numérico.

Versión 1.4a, Julio de 1996:

- Se agrega la rutina de librería **crash_message()**.
- Los programas **enlazados** por los usuarios registrados, producirán ahora diagnósticos de errores en tiempo de ejecución sin importar el tamaño de los programas.
- **shroud.bat** tiene una nueva opción: **−full_keywords**.

Versión 1.4, Mayo de 1996:

- Ahora se puede convertir cualquier programa Euphoria en un archivo **ejecutable_autónomo.exe.**
- Se elimina el archivo de expansión del ***DOS***, DOS4GW.EXE.
- Soporte para los nombres largos de Windows 95.
- Soporte para interrupciones de software en ***DOS***.
- Nuevos programas utilitarios: **key.ex**, **where.ex**, **ascii.ex**, **guru.ex**.
- Nuevo programa de demostración: **dosint.ex**.
- Nuevas rutinas de librería: **set_rand()**, **use_vesa()**.
- Se pueden usar ***peek()*** y ***poke()*** con secuencias enteras de bytes.
- Mejoras en el editor.
- Reducción en el consumo de recursos en las secuencias.

Versión 1.3, Junio de 1995:

- Ahora se pueden editar varios archivos, usando varias ventanas de edición.
- Se agregan 20 nuevas rutinas de librería.
- Se mejorán sustancialmente las prestaciones gráficas.

Versión 1.2, Marzo de 1994:

- Se elimina un problema que impide a Euphoria correr en ***DOS*** bajo ***Windows***.

Versión 1.1, Enero de 1994:

- Se agregan varias características al lenguaje y programas de demostración.

Versión 1.0, Julio de 1993:

- Euphoria fue lanzado, luego de tres años de investigación y desarrollo y seis meses de pruebas Beta. Muchas de las ideas detrás de Euphoria vienen de la Tesis de Maestría en Ciencias de la Computación de Robert Craig de la Universidad de Toronto. Aquella tesis estaba muy influenciada por el trabajo de John Backus sobre lenguajes de programación funcional.

Rapid Deployment Software (RDS)
Licencia del Código Fuente del Intérprete Euphoria
para la versión 2.4

A cambio del pago, RDS le proveerá el código fuente C que puede usar para generar una versión del Intérprete Euphoria de RDS, usando cualquiera de los 7 compiladores C soportados en las 4 plataformas. Se retiró el código fuente de ciertas características registradas por RDS. Estas características son: el trazador (tracer), análisis de perfiles de ejecución (profiling) y el código relacionado con el enlazado (binding).

Hemos probado el fuente con 7 diferentes compiladores de C, y estamos satisfechos porque se puede compilar y enlazar correctamente con todos ellos. Sin embargo, no podemos garantizar que sea capaz de compilar o enlazar el fuente, u obtener el mismo nivel de velocidad o confiabilidad. Si tiene algún problema, le proveeremos asistencia durante el período de de soporte técnico de 3 meses.

Se aplicarán las siguientes restricciones:

1. Copiado

Tiene permiso para hacer copias de respaldo para su propio uso. No puede distribuir copias de este código fuente, con o sin cambios, a nadie excepto RDS u otras personas de quienes esté seguro que están autorizadas para ver esta versión de nuestro fuente. Si tiene alguna duda, debe contactar a RDS para averiguar si la persona está o no autorizada ver el fuente. Si son varios programadores, sea tanto en forma individual como en una compañía, que desean estudiar o trabajar juntos en el fuente, cada uno de ellos tiene que pagar a RDS una copia.

2. Cambios aceptables al fuente

RDS entrega ciertas características de Euphoria gratuitamente, mientras que cobra por otras características (registradas). Tiene permiso y, los alentamos, a agregar nuevas características a este código fuente. Sin embargo, sin permiso escrito de RDS, no tiene permiso a distribuir ninguna característica que sea lo suficientemente similar a aquellas registradas, que ayuden a reducir significativamente el estímulo de las personas a pagar a RDS por esa función. Las características registradas actualmente incluyen:

- **enlazado:** No debe proveer una manera de anexar archivos al Intérprete Euphoria, convirtiendo así al programa Euphoria en un único archivo ejecutable. (La ***encriptación*** no está restringida, porque se puede realizar sin este código fuente, y sin ningún conocimiento contenido en el mismo).
- **utilidad de trazado:** No debe proveer la capacidad de depuración del fuente, o un mecanismo de archivo de registro para trazar la ejecución de las sentencias.
- ***análisis de perfiles de ejecución*:** No se debe proveer ninguna forma de análisis de código por tiempo o de conteo de instrucciones.
- **Traductor a C:** No puede usar este fuente para crear y distrbuir un programa que traduzca los programas Euphoria en programas en C/C++.
- **Librería del Traductor:** No puede usar este fuente para crear y distribuir una librería que se use con el Traductor Euphoria a C.

En el futuro, se podrán agregar otras características a esta lista y algunas otras ser eliminadas de ella. Ud. está limitado por cualquier característica que esté listada en el Acuerdo de Licencia en el momento que compra el fuente.

Advierta que lo estamos limitando a no tener permiso para *distribuirlo* a otros, pero no a que no tenga permiso a trabajar en la privacía de su máquina.

3. Portación

Lo alentamos a portar este software a nuevas máquinas y sistemas operativos.

4. Distribución de ejecutables

Siempre que sus cambios sean aceptables, puede compilar el fuente usando cualquier compilador, y distribuir copias del archivo ejecutable resultante (pero no del fuente o archivos objeto intermedios). Puede entregar gratuitamente sus ejecutables, o cobrar el cargo que decida por ellos.

5. Propiedad de nuevas características

Ud es dueño del código fuente para cualquier característica o cambio que haga, y no se le exige que haga público cualquier código que desarrolle. Si hace público su código fuente, a otro que no sea RDS, o aquellos de los que esté seguro que están autorizados para nuestro código fuente, no debe exponer más que unas pocas líneas del código fuente de RDS con el suyo. El hecho de que haya implementado una característica no imposibilita a RDS o a ningún otro de implementar la misma característica u otra similar.

6. Discusión pública

Puede discutir públicamente los algoritmos usados por su código fuente en foros abiertos y mencionar los nombres de rutinas C, variables y otros identificadores en el código, pero no debe dar a conocer las líneas del código fuente.

7. Reconocimiento

Se le exige reconocer el uso de nuestro código fuente, listar nuestro sitio web, http://www.RapidEuphoria.com, e indicar la naturaleza general de cualquier cambio que haga a nuestro código, en cualquier programa derivado que distribuya. El reconocimiento podría ser mostrado por el programa, o estar acompañando la documentación, pero debe ser plenamente visible por la mayoría de los usuarios.

8. Daños

RDS no se hace responsable frente a Ud., por cualquier daño que surja del uso de este código fuente, y debe informárselo a sus usuarios, en su documentación o en sus programas derivados, que RDS no se hace responsable de ellos por cualquier daño.

9. Intenciones malignas

No puede usar este fuente para crear ningún virus, gusanos, troyanos o cualquier otro software cuya intención sea la de causar daño a cualquier sistema de computación o red, o que pudiera perjudicar intencionalmente la reputación de Euphoria o de RDS.

Ordenando la Edición Completa de Euphoria

1. Productos

Euphoria actualmente corre bajo las plataformas **Windows**, **DOS**, **Linux** y **FreeBSD**. Están disponibles las Ediciones Completas de los siguientes productos, soportando cada una de ellas las 4 plataformas.

```
Intérprete Euphoria ..................... u$s 29
Traductor Euphoria a C .................. u$s 29
Código fuente del Intérprete Euphoria .. u$s 49
```

Todos los precios están en dólares estadounidenses. Los costos de envío y trámites son u$u 7.00 por producto.Si no quiere el CD o un recibo, le recomendamos que descargue el software de nuestro sitio Web y le evitamos el cargo del envío y la demora.

El Intérprete Euphoria incluye:

- versiones de **ex.exe (DOS32)**, **exw.exe (WIN32)**, **exu (Linux)** y **exu (FreeBSD)** con soporte para depuración **trace**() completo, para más de 300 sentencias.
- la capacidad de **enlazado** y **encriptación** de los programas Euphoria, para una fácil distribución.
- **analizador de perfiles de ejecución por conteo de instrucciones**, para analizar la lógica de su programa y mejorar las prestaciones.
- **analizador de perfiles de ejecución por tiempo**, para descubrir cuellos de botella en las prestaciones (**DOS32** solamente)
- un descuento en todas los futuros lanzamientos del Intérprete Euphoria.
- 3 meses de soporte técnico gratuito desde Rapid Deployment Software.
- 12 meses de actualizaciones gratuitas del Intérprete.

El Traductor Euphoria a C incluye:

- Eliminación de demoras y mensajes cuando arrancan los programas.
- La capacidad de trazar la ejecución y averiguar porque se "estrelló" el programa.
- Sentencias Euphoria insertas como comentarios en el código fuente de C.
- un descuento en los futuros lanzamientos del Traductor Euphoria a C.
- 3 meses de soporte técnico gratuito desde Rapid Deployment Software.
- 12 meses de actualizaciones gratuitas del Traductor.

El Código Fuente del Intérprete Euphoria incluye:

- Código fuente C comentado para el Intérprete Euphoria en todas las plataformas. *Se aplican algunas restricciones. Ver la Licencia del Código Fuente. El código de la utilidad de trazado, enlazado y análisis de ejecuciones no está incluido. El código del Traductor tampoco.*
- 7 archivos de comandos .bat para facilitar la compilación y el enlazado del Intérprete completo en las 4 plataformas, usando 7 compiladores de C diferentes.
- un descuento en todas los futuros lanzamientos del Código fuente del Intérprete.
- 3 meses de soporte técnico gratuito desde Rapid Deployment Software.
- 12 meses de actualizaciones gratuitas del Código fuente del Intérprete.

Además, todos los usuarios registrados de Euphoria, reciben:

- la posibilidad de votar en *Euphoria Micro−Economy* y animar a otros que desarrollen software que usted encuentra útil. (Ver el sitio web de RDS para detalles sobre Micro−Economy).

2. Costo de actualizaciones gratuitos o reducidos

Si compró (es decir, no recibió una actualización gratuita) el Intérprete Euphoria 2.3, para cualquier plataforma, tanto como una actualización o una nueva registración, está autorizado a actualizar gratuitamente a la versión 2.4 del Intérprete.

Los demás usuarios registrados del Intérprete, (desde v1.0 en adelante), tienen un descuento de u$s 10 en el precio del Intérprete.

Todos aquellos que hayan pagado por el Traductor 2.3 están autorizados a actualizar gratuitamente al Traductor 2.4. Los demás usuarios registrados del Traductor, tienen un descuento de u$s10 en el precio del Traductor.

Aquellos que hayan pagado por el código fuente del Intérprete 2.3, están autorizados a actualizar gratuitamente al código fuente del Intérprete 2.4.

Las actualizaciones gratuitas consisten en la descarga del nuevo software desde la Web. No se envía el CD.

Si ha ganado dinero en *Euphoria Micro−Economy*, por favor contacte a RDS (rds@RapidEuphoria.com) por detalles de como reducir el precio.

3. Métodos de registración

3.1 Ordenes de tarjeta de crédito

Puede registrar o actualizar Euphoria mediante **DigiBuy** usando su tarjeta MasterCard, Visa, American Express, Discover Card o Diner's Card. Vaya a nuestro sitio web:

```
http://www.RapidEuphoria.com
```

y haga clic en "Instant Registration". DigiBuy tiene un servidor **seguro** que le permite a su navegador web encriptar la información de su tarjeta de crédito antes de enviarla por Internet.
Si elige descargar la Edición Completa de un producto, le enviaremos instrucciones a la dirección de correo electrónico que le proveyó a DigiBuy. Normalmente, esto ocurre en unos pocos minutos, pero puede tomar horas con órdenes de actualización.

Con DigiBuy, *no* hay impuestos adicionales que pagar, salvo que sea residente de Connecticut, Minnesota o del estado de Washington.

3.2 Pagando con cheque o una orden de pago

En **euphoria\register** hay un formulario de orden en blanco que puede editar e imrpimir, o imprimir primero y llenar luego mediante un bolígrafo. El archivo del formulario de orden se llama **MYORDER.TXT**. Si no tiene impresora, escriba únicamente la información relevante en una hoja de papel y adjúntela con su cheque u orden de pago en el envío a RDS.

Preferimos moneda estadounidense o canadiense. Sin embargo, podemos aceptar cheques personales de cualquiera de los siguientes países:

> Australia, Austria, Barbados, Bélgica, Gran Bretaña, Canadá, Dinamarca, Finlandia, Francia, Grecia, Alemania, Hong−Kong, Irlanda, Japón, México, Países Bajos, Nueva Zelandia, Noruega, Portugal, España, Suecia, Suiza, Estados Unidos
>
> Haga por favor una conversión justa de nuestros precios a dólares estadounidenses en su moneda local.

Haga su cheque u orden de pago pagadera a:

> ***Rapid Deployment Software***
>
> **130 Holm Crescent**
> **Thornhill, Ontario**
> **L3T 5J3**
> **CANADA**

No hay impuestos adicionales que pagar.

3.3 Pagando con una transferencia de dinero de Western Union

Hay miles de oficinas de Western Union alrededor del mundo. Puede ir a una oficina y pagar en efectivo en su moneda local y el dinero será transferido casi inmediatamente a RDS en Canadá. Contáctenos vía e−mail si desea usar Western Union.

Lenguaje de Programación Euphoria

versión 2.4

Manual de Referencia

TABLA DE CONTENIDOS

Parte I − Fundamentos del Lenguaje

Parte II − Rutinas de Librería

Parte I − Fundamentos del Lenguaje

1. Introducción

Euphoria es un nuevo lenguaje de programación con las siguientes ventajas sobre otros lenguajes convencionales:

- una notablemente simple, flexible y potente definición del lenguaje que es fácil de aprender y usar.
- Asignación dinámica de memoria. Las variables crecen o decrecen en tamaño, sin que el programador tenga que preocuparse de liberar porciones de memoria. Los objetos de cualquier tamaño se pueden asignar a los elementos de una secuencia Euphoria (array).
- un avanzado intérprete de altas prestaciones que es ***30 veces*** más rápido que los intérpretes convencionales, tales como Perl y Python.
- un Traductor Euphoria a C optimizado, que puede acelerar la velocidad aún más, frecuentemente por un factor de 2 a 5 veces el ya rápido intérprete.
- Pre−compilación relámpago. El código fuente de su programa se lo verifica en sintaxis y se lo convierte en una forma interna eficiente a más de ***35,000 líneas por segundo*** en un lento Pentium−150. No hay necesidad de meterse con archivos "de código byte" adicionales.
- extensa verificación en tiempo de ejecución de: índices fuera de rango, variables no inicializadas, valores erróneos en los parámetros de las rutinas de librería, valores ilegales asignados a una variable y mucho más. No hay misteriosas excepciones de máquina −− siempre obtendrá una completa descripción en inglés de cualquier problema que ocurra con su programa en tiempo de ejecución, junto con un trazado de las llamadas a la pila y un volcado de todos los valores de sus variables. Los programas se pueden depurar rápidamente, fácilmente y más a fondo.
- Las características del hardware subyacente se ocultan totalmente. Los programas no tienen conocimiento de longitudes de palabra, representación a nivel de bit de valores subyacentes, orden de bytes, etc.
- Se incluyen un depurador de pantalla completa y un analizador de perfiles de ejecución, junto con un editor multiarchivo de pantalla completa. En un monitor color, el editor muestra los programas Euphoria en varios colores, resaltando comentarios, palabras reservadas funciones internas, cadenas y el nivel de anidación de paréntesis. Opcionalmente realiza el autocompletado de sentencias, evitándole el trabajo de escribir y reduciendo los errores de sintaxis. Este editor está escrito en Euphoria, y su código fuente se provee sin restricción alguna. Puede modificarlo libremente, agregarle nuevas funcionalidades y redistribuirlo como desee.
- Los programas Euphoria se ejecutan bajo ***Linux***, ***FreeBSD***, ***Windows de 32 bits***, y cualquier entorno ***DOS***, y no están sujetos a las limitaciones de memoria de 640K. Puede crear programas que usen la totalidad de la memoria de su máquina, se usará automáticamente un archivo de intercambio cuando el programa necesite más memoria de la que existe en su máquina.
- Puede generar un archivo .exe único e independiente de su programa.
- Las rutinas Euphoria son naturalmente genéricas. El programa de ejemplo de más abajo, muestra una sencilla rutina que ordenará cualquier tipo de dato −− enteros, números de punto flotante, cadenas, etc. Euphoria no es un lenguaje "orientado a objetos", sin embargo, alcanza muchas de las ventajas de esos lenguajes en una forma mucho más simple.

1.1 Programa de ejemplo

El siguiente es un ejemplo de un programa Euphoria completo.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
 sequence list, sorted_list

 function merge_sort(sequence x)
 -- poner x en orden ascendente, usando merge_sort recursivo
     integer n, mid
     sequence merged, a, b

     n = length(x)
     if n = 0 or n = 1 then
         return x  -- caso trivial
     end if

     mid = floor(n/2)
     a = merge_sort(x[1..mid])       -- ordena la primera mitad de x
     b = merge_sort(x[mid+1..n])     -- ordena la segunda mitad de x

     -- combina las dos mitades ordenadas en una
     merged = {}
     while length(a) > 0 and length(b) > 0 do
         if compare(a[1], b[1]) <0 then
             merged = append(merged, a[1])
```

```
            a = a[2..length(a)]
        else
            merged = append(merged, b[1])
            b = b[2..length(b)]
        end if
    end while
    return merged &a &b  -- datos combinados
end function

procedure print_sorted_list()
-- genera sorted_list desde la lista
    list = {9, 10, 3, 1, 4, 5, 8, 7, 6, 2}
    sorted_list = merge_sort(list)
    ? sorted_list
end procedure

print_sorted_list()     -- este comando inicia el programa
```

~~~~~~

El ejemplo de más arriba contiene 4 comandos separados que se procesan en orden. El primero declara dos variables: list y sorted_list como **secuencias** (arrays flexibles). El segundo define la **función** merge_sort(). El tercero define el **procedimiento** print_sorted_list(). El último comando llama al procedimiento print_sorted_list().

La salida del programa será:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.

merge_sort() ordenará fácilmente {1.5, −9, 1e6, 100} o {"naranjas", "manzanas", "bananas"} .

Este ejemplo está guardado en **euphoria\tutorial\example.ex**. Esta no es la forma más veloz de ordenar en Euphoria. Vaya al directorio **euphoria\demo** y escriba "ex allsorts" para ver los tiempos en varios algoritmos de ordenamiento según la cantidad de objetos. Para un ejemplo rápido de programación en Euphoria ver **euphoria\demo\bench\filesort.ex**.

1.2 Instalación

Para instalar Euphoria en su máquina, primero lea el archivo **install.doc**. La instalación involucra simplemente el copiado de los archivos **euphoria** a su disco rígido, dentro del directorio llamado "euphoria", y modificar el archivo **autoexec.bat** de modo que **euphoria\bin** esté en la ruta de búsqueda, y la variable de entorno **EUDIR** apunte al directorio de Euphoria.

Una vez instalado, el directorio **euphoria** se verá como esto:

\euphoria

readme.doc
readme.htm
\bin
    *ex.exe y exw.exe, o exu (**Linux/FreeBSD**), ed.bat, guru.bat, otras utilidades*
\include
    archivos de inclusión estándares, por ejemplo, graphics.e
\doc
    refman.doc, library.doc, y varios otros archivos de documentación de texto plano
\html
    archivos HTML correspondientes cada uno a los archivos .doc del directorio doc
\tutorial
    pequeños programas tutoriales para ayudar a aprender Euphoria
\demo
programas de demostración genéricos que corren en todas las plataformas
\dos32
    *programas de demostración específicos de **DOS32** (opcional)*
\win32
    *programas de demostración específicos de **WIN32** (opcional)*
\linux
    *programas de demostración específicos de **Linux/FreeBSD** (opcional)*
\langwar
    *juego language war (versión de gráficos de píxel para DOS, o versión texto para **Linux/FreeBSD**)*
\bench
    programas de ensayos
\register
    información de cómo ordenar la Edición Completa
~~~~~~

El subdirectorio Linux no está incluido en la distribución ***DOS/Windows***, y los subdirectorios DOS32 y WIN32 no están incluidos en la distribución ***Linux/FreeBSD***. En este manual, los nombres de directorios se muestran usando una barra invertida (\). Los usuarios de ***Linux/FreeBSD*** deberán sustituirla por la barra de división (/).

1.3 Ejecutando un programa

Los programas Euphoria se ejecutan al escribir **ex**, **exw** o **exu** seguido por el nombre del principal archivo Euphoria. Puede agregar palabras adicionales (conocidas como **argumentos**) en esta línea, llamada **línea de comandos**. Su programa puede llamar a la función interna ***command_line()*** para leer la línea de comandos. La versión ***DOS32*** del Intérprete Euphoria se llama **ex.exe,** la versión ***WIN32*** se llama **exw.exe** y la versión ***Linux/FreeBSD*** **exu**. Por convención, los archivos principales de Euphoria tienen la extensión **.ex**, **.exw** o **.exu**. Otros archivos Euphoria, que se pueden incluir en programas más grandes, terminan en **.e** o a veces en **.ew** o **.eu**. Para abreviar escritura, puede dejar de escribir el "***.ex***", y el comando **ex** lo agregará automáticamente. **exw.exe** agregará "***.exw***", y el **exu,** "***.exu***". Si no se puede encontrar el archivo en el directorio actual, será buscado en la ruta (PATH). Puede redireccionar la entrada y salida estándares al ejecutar un programa Euphoria, por ejemplo:

```
ex filesort.ex <raw.txt > sorted.txt
```

o simplemente,

```
ex filesort <raw.txt > sorted.txt
```

A diferencia de otros compiladores e intérpretes, no hay opciones especiales en la línea de comandos de **ex**, **exw** o **exu**. Solamente se espera el nombre de su archivo Euphoria, y sino lo suministra, le será pedido.
Para los programas usados frecuentemente bajo ***DOS/Windows*** you podría querer hacer un pequeño archivo **.bat** (batch) file, quizás llamado **miprog.bat**, conteniendo dos instrucciones como:

```
@echo off
ex miprog.ex %1 %2 %3
```

La primera instrucción desactiva el eco de los comandos en pantalla. El segundo ejecuta **ex miprog.ex** con hasta 3 argumentos en la línea de comando. En ***command_line()*** vea un ejemplo de cómo leer esos argumentos. Si su programa toma más argumentos, debería agregar %4, %5, etc. Teniendo un archivo ***.bat*** lo salvará del inconveniente menor de escribir **ex** (o **exw**) todas las veces, es decir, puede escribir solamente:

```
miprog
```

en lugar de:

```
ex miprog
```

Desafortunadamente, ***DOS*** no le permitirá la redirección de la entrada y salida estándares al usar un archivo **.bat**.

Bajo ***Linux/FreeBSD***, puede escribir el camino al intérprete Euphoria en la primera línea de su archivo principal, por ejemplo, si su programa se llama foo.exu:

```
#!/home/rob/euphoria/bin/exu

procedure foo()
    ? 2+2
end procedure

foo()
```

Entonces si hace ejecutable su archivo:

```
chmod +x foo.exu
```

Puede escribir solamente:

```
foo.exu
```

para correr el programa. Inclusive, podría acortar más aún el nombre a simplemente "foo". Euphoria ignora la primera línea cuando comienza con **#!**. Cuide que la primera línea termine con \n estilo ***Linux/FreeBSD***, y no con el \r\n estilo ***DOS/Windows***, o el shell de ***Linux/FreeBSD*** podría confundirse.
También puede correr **bind.bat** (***DOS32***), o **bindw.bat** (***WIN32***) o **bindu** (***Linux/FreeBSD***) para combinar su programa Euphoria con **ex.exe**, **exw.exe** o **exu**, para hacer un archivo ejecutable independiente (archivo **.exe** en ***DOS/Windows***). Con un archivo **.exe** independiente ***puede*** redirigir la entrada y salida estándares. El enlazado se discute luego en 1.5 Distribuyendo un programa.

Tanto **exu**, **ex.exe** como **exw.exe** están en el directorio **euphoria\bin,** el cual tiene que estar en su ruta de búsqueda. La variable de entorno EUDIR debería apuntar al directorio principal de Euphoria, por ejemplo, **c:\euphoria**.

1.3.1 Corriendo bajo Windows

Puede correr programas Euphoria directamente desde el entorno ***Windows***, o desde el shell ***DOS*** que tiene abierto desde e ***Windows***. Al "asociar" los archivos **.ex** con **ex.exe**, y los archivos **.exw** con **exw.exe** puede hacer doble clic en un archivo **.ex** o **.exw** para ejecutarlo. Bajo ***Windows*** debería definir un nuevo tipo de archivo para **.ex**, haciendo clic en Mi PC / Ver / Opciones / Tipos de archivos. Es posible que tenga varios programas Euphoria activos en diferentes ventanas. Si convierte un programa en un archivo **.exe**, simplemente puede hacer clic sobre él para ejecutarlo.

1.3.2 Uso del archivo de intercambio

Si ejecuta un programa Euphoria bajo ***Linux/FreeBSD*** o ***Windows*** (o en el shell ***DOS*** bajo Windows), y el programa se queda sin memoria física, usará "memoria virtual". El sistema operativo provee esta memoria virtual automáticamente para intercambiar los datos y código usados recientemente con el archivo de intercambio. Para cambiar el tamaño del archivo de intercambio de Windows, haga clic en Panel de Control / 386 Mejorado / "memoria virtual...". Bajo ***OS/2*** puede ajustar "DPMI_MEMORY_LIMIT" haciendo clic en el ícono de la máquina virtual DOS / "Configuración DOS" para asignar más memoria extendida para su programa.

Bajo ***DOS*** puro, fuera de Windows, no hay un archivo de intercambio del sistema, por lo que **ex.exe** (***DOS32***) incorpora un expansor para DOS, que creará el archivo de intercambio para el posible uso por su programa. Ver **platform.doc**.

1.4 Editando un programa

Puede usar cualquier editor de texto para editar un programa Euphoria Sin embargo, Euphoria viene con su propio editor especial que está enteramente escrito en Euphoria. Escriba: **ed** seguido por el nombre completo del archivo que desea editar (la extensión .ex/.exw/.exu no está sobreentendida). Puede usar este editor para editar cualquier tipo de archivo de texto. Al editar un archivo Euphoria, están disponibles algunas funcionalidades adicionales, tales como la coloración de la sintaxis y el autocompletado de ciertas sentencias.

Toda vez que ejecute un programa Euphoria y obtenga un mensaje de error, sea durante la compilación como la ejecución, simplemente escriba **ed** sin el nombre del archivo y será automáticamente posicionado en el archivo que contiene el error, en la línea y columna correctas, mientras en la parte superior de la pantalla verá el mensaje de error.

Bajo Windows puede asociar **ed.bat** con varios tipos de archivos de texto que quiera editar. La coloración de la sintaxis se aplica a los archivos de entesión **.ex**, **.exw**, **.exu**, **.e**, **.ew**, **.eu**, y **.pro** (profile).

La mayoría de la teclas que presiona se insertan en el archivo, en la posición del cursor. Presione la tecla **Esc** una vez para obtener la barra del menú de comandos especiales. También están activas las teclas de flechas y las de Insert / Supr / Inicio / Fin / Re Pág / Av Pág. Bajo ***Linux/FreeBSD*** algunas teclas pueden no estar disponibles, y habrán otras provistas como alternativa. Vea el archivo **euphoria\doc\ed.doc** (euphoria\html\ed.htm) una descripciónfor completa de los comandos de edición. **Esc h** (ayuda) que le permitirá leer **ed.doc** desde la sesión de edición.

Si necesita comprender o modificar algún detalle de la operación del editor, puede editar el archivo **ed.ex** en **euphoria\bin** (asegúrese de hacer una copia de seguridad para no perder el archivo original). Si el nombre **ed** le genera un conflicto con algún otro comando en su sistema, renombre el archivo **euphoria\bin\ed.bat** a algún otro. Debido a que el editor está escrito en Euphoria, es notablemente conciso y fácil de comprender. La misma funcionalidad implementada en otro lenguaje como el C, tomaría más líneas de código.

ed es un simple editor de modo texto que corre en ***DOS, Linux, FreeBSD*** y en la consola de ***Windows***. También vea el excelente editor de **David Cuny** para ***DOS*** y ***Linux/FreeBSD***. Puede descargar **ee.ex** desde el sitio web de Euphoria. También hay otros editores orientados a Euphoria que corren bajo Windows. Véalos en sitio web de Euphoria.

1.5 Distribuyendo un programa

Euphoria provee 4 formas distintas de distribuir un programa.

En el primer método, le despacha a sus usuarios archivos de Dominio Público **ex.exe** o **exw.exe** o **exu**, junto con su archivo principal Euphoria .ex, .exw, or .exu file y cualquier archivo de inclusión .e que se necesite (incluyendo cualquier de los estándares del directorio **euphoria\include**). Si los archivos Euphoria de código fuente se ubican juntos en un directorio y ex.exe, exw.exe o exu se ubican en el mismo u otro directorio de la ruta de búsqueda (path), entonces sus usuarios pueden ejecutar su programa, escribiendo **ex** (**exw**) o (**exu**) seguido por la ruta de su archivo principal .ex, .exw, o .exu. También podría proveer un pequeño archivo **.bat**, de forma que la gente no tenga que escribir **ex** (**exw**). Este método asume que quiere compartir su código fuente Euphoria con sus usuarios.

La Edición Completa le brinda dos métodos más de distribución. Puede **encriptar** su programa, o **enlazarlo**. La **encriptación** combina todos los archivos .e que su programa necesite, junto con su archivo principal para crear un archivo único .ex, .exw, or .exu. Puede tanto encriptar/compactar su programa para ocultarlo y hacerlo inviolable, como dejarlo legible para permitirles a los usuarios su inspección. El **enlazado** combina su programa encriptado con ex.exe, exw.exe, o exu para crear un archivo **ejecutable, único e independiente (.exe)**. Por ejemplo, si su programa se llama "miprog**.**ex", puede crear "myprog**.**exe" que correrá exactamente igual. Para más información acerca de la encriptación y enlazado, lea **bind.doc**.

Finalmente, con el **Traductor Euphoria a C**, puede **traducir** su programa Euphoria a C y entonces compilarlo con un compilador C para obtener un archivo ejecutable (.exe). El Traductor está disponible en una descarga separada en el sitio web de RDS.

1.5.1 Licenciamiento

Puede distribuir libremente cualquier programa Euphoria que desarrolle, sin tener que pagar regalías. También puede distribuir libremente los archivos de la Edición de Dominio Público **ex.exe**, **exw.exe** and **exu**, por lo que cualquiera puede correr sus programas Con la Edición Completa, puede **encriptar** o **enlazar** sus programas y distribuir los archivos resultantes, también sin pagar regalías.

Puede incorporar cualquier archivo de código fuente Euphoria de este paquete en su programa, tanto los originales como los que contienen alguna modificación que le haya hecho. (Probablemente necesite, al menos, unos pocos archivos de inclusión estándares en cualquier programa extenso).

Apreciaremos si le dice a la gente que desarrolló su programa usando Euphoria, y si les da la dirección de nuestro sitio web: **http://www.RapidEuphoria.com**, pero no exigimos tal reconocimiento.

Los únicos archivos relacionados con el Intérprete que ***no*** debe distribuir son **ex.exe**, **exw.exe**, **bind.ex**, **bind.bat**, **bindw.bat**, y **shroud.bat,** que vienen con la Edición Completa del Intérprete para ***WIN32*** + ***DOS32***, y **exu**, **bind.ex**, **bindu** y **shroud** que vienen con la Edición Completa del Intérprete para ***Linux/FreeBSD***. El archivo de ícono **euphoria.ico**, que se incluye con la Edición Completa para ***WIN32*** + ***DOS32***, se puede distribuir con o sin modificaciones.

El Intérprete Euphoria está escrito en ANSI C, y se puede compilar con varios compiladores de C. El fuente está disponible para la compra. Antes de comprarlo, por favor lea la **Licencia del código fuente del Intérprete**.

Algunas restricciones legales adicionales se pueden aplicar al usar el **Traductor Euphoria a C**.

2. Definición del lenguaje

2.1 Objetos

2.1.1 Atomos y secuencias

Todos los **objetos** de datos en Euphoria son tanto **átomos** como **secuencias**. Un **átomo** es un valor numérico simple. Una **secuencia** es una colección de valores numéricos.

Los **objetos** contenidos en una secuencia pueden ser una mezcla arbitraria de átomos o secuencias. Una secuencia se representa por una lista de objetos entre llaves, separados por comas. Los átomos pueden tener un entero o un valor de punto flotante de doble presición. Tienen un rango aproximado desde −1e300 (menos uno por 10 a la 300) hasta +1e300 con 15 decimales de exactitud. Aquí están algunos objetos Euphoria:

```
-- ejemplos de átomos:
0
1000
98.6
-1e6

-- ejemplos de secuencias:
{2, 3, 5, 7, 11, 13, 17, 19}
{1, 2, {3, 3, 3}, 4, {5, {6}}}
{{"juan", "perez"}, 52389, 97.25}
{}                          -- la secuencia de 0 elementos
```

Los números se pueden ingresar también en hexadecimal. Por ejemplo:

```
#FE             -- 254
#A000           -- 40960
#FFFF00008      -- 68718428168
-#10            -- -16
```

Solamente las letras mayúsculas A, B, C, D, E, F están permitidas en los números hexadecimales.

Las secuencias se pueden anidar a cualquier profundidad, es decir, puede tener secuencias dentro de secuencias dentro de secuencias y así hasta cualquier profundidad (hasta que se agote la memoria). Las llaves se usan para construir secuencias out of a lest of expressions. Esas expresiones pueden ser constantes o ser evaluadas en tiempo de ejecución, por ejemplo:

```
{x+6, 9, y*w+2, sin(0.5)}
```

La parte **"Objetos Jerárquicos"** del acrónimo Euphoria viene de la naturaleza jerárquica de las secuencias anidadas. Esto no debería ser confundido con las jerarquías de clases de ciertos lenguajes orientados a objetos.

¿Por qué los llamamos **átomos**? ¿Por qué no solamente "números"? Bien, un átomo ***es*** sólo un número, pero queríamos tener un término distintivo que enfatizara que son indivisibles. Por supuesto, en el mundo de la Física, los átomos se puedieron separar en partes más pequeñas hace algunos años atrás, pero en Euphoria no se pueden separar. Son los bloques básicos de todos los datos que un programa Euphoria puede manejar. Con esta analogía, las **secuencias** tendrían que ser las "moléculas", hechas de átomos y otras moléculas. Una analogía mejor podría ser que las secuencias son como los directorios y los átomos como los archivos. Como un directorio en su computadora puede contener tanto archivos como otros directorios, una secuencia puede contener tanto átomos como otras secuencias (y ***esas*** secuencias pueden contener átomos y secuencias, etc.).

Como descubrirá dentro de poco, las secuencias hacen a Euphoria muy simple ***y*** muy potente. **Comprender los átomos y las secuencias es la clave para comprender Euphoria.**

Nota de rendimiento:

> ¿Esto significa que todos los átomos se almacenan en memoria como números de punto flotante de 8 bytes? No. El intérprete Euphoria normalmente almacena átomos de valor entero como los enteros de la máquina (4 bytes) para ahorrar especio y mejorar la velocidad de ejecución. Cuando el resultado es fraccionario o se obtienen números demasiado grandes, la conversión a punto flotante ocurre automáticamente.

2.1.2 Cadenas de caracteres y caracteres individuales

Una **cadena de caracteres** es solamente una **secuencia** de caracteres. Se puede ingresarla usando comillas, por ejemplo:

```
"ABCDEFG"
```

Las cadenas de caracteres se pueden manipular y operar igual que cualquier otra secuencia. Por ejemplo, la cadena del ejemplo es completamente equivalente a la secuencia:

```
{65, 66, 67, 68, 69, 70, 71}
```

que contiene los correspondientes códigos ASCII. El compilador Euphoria convertirá inmediatamente "ABCDEFG" a la secuencia anterior de números. En un sentido, no hay "cadenas" en Euphoria, solamente secuencias de números. Una cadena entrecomillada es realmente una notación conveniente que le evita tener que escribir todos los códigos ASCII.
Esto "" es equivalente a {}. Ambos representan la secuencia de longitud 0, también conocida como **secuencia vacía**. Como una cuestión de estilo de programación, es natural usar "" para sugerir una secuencia de caracteres de longitud 0, y {} para indicar alguna otra clase de secuencia.

Un **caracter individual** es un **átomo**. Tiene que ingresarse con comillas simples. Hay diferencia entre un caracter individual (que es un átomo), y una cadena de caracteres de longitud 1 (que es una secuencia), por ejemplo:

```
'B'   -- equivalente al átomo 66 - el código ASCII de la letra B
"B"   -- equivalente a la secuencia {66}
```

Otra vez, 'B' es sólo una notación que es lo mismo que escribir 66. Realmente no hay ningún "caracter" en Euphoria, sólo números (átomos). Tenga presente que un átomo ***no*** es equivalente a una secuencia de un elemente conteniendo el mismo valor, aunque hay muy pocas rutinas internas que eligen tratarlos similarmente.

Los caracteres especiales se tienen que ingresar usando una barra invertida:

```
\n        nueva línea
\r        retorno de carro
\t        tabulador
\\        barra invertida
\"        comilla doble
\'        comilla simple
```

Por ejemplo, "Hola, Mundo!\n", o '\\'. El editor Euphoria muestra las cadenas de caracteres en verde.

2.1.3 Comentarios

Los comentarios empiezan con dos guiones y se extienden hasta el final de la línea, por ejemplo:

```
-- esto es un comentario
```

El compilador ignora todos los comentarios, los cuales no afectan la velocidad de ejecución. El editor muestra los comentarios en rojo.

En la primera línea (solamente) de su programa, puede usar un comentario especial, que comienza con #!, por ejemplo:

```
#!/home/rob/euphoria/bin/exu
```

Esto le dice al shell de Linux que su programa debería ser ejecutado por el intérprete de Euphoria, dándole la ruta completa del intérprete. Si hace ejecutable su programa, puede correrlo solamente escribiendo su nombre, sin necesidad de escribir"exu". En DOS y Windows estas líneas se tratan como un comentario.

2.2 Expresiones

Como otros lenguajes de programación, Euphoria le permite calcular resultados al formar expresiones. Sin embargo, en Euphoria puede realizar cáculos sobre secuencias de datos con una expresión, donde en la mayoría de los lenguajes debería construir un ciclo. En Euphoria puede manejar una secuencia como si se tratase de un simple número. Se la puede copiar, pasar a una subrutina o calcular como una unidad. Por ejemplo:

```
{1,2,3} + 5
```

es una expresión que suma la secuencia {1,2,3} y el átomo 5 para obtener la secuencia resultante {6,7,8}. Verá más ejemplos más tarde.

2.2.1 Operadores relacionales

Los operadores relacionales `<  >  <=  >=  =  !=` producen resultados de 1 (verdadero) o de 0 (falso).

```
8.8 <8.7   -- 8.8 menor que 8.7 (falso)
-4.4 > -4.3 -- -4.4 mayor que -4.3 (falso)
8  <= 7      -- 8 menor o igual a 7 (falso)
4 >= 4      -- 4 mayor o igual a 4 (verdadero)
1 = 10      -- 1 igual a 10 (falso)
8.7 != 8.8  -- 8.7 no es igual a 8.8 (verdadero)
```

Como veremos pronto, estos operadores se pueden aplicar a las secuencias.

2.2.2 Operadores lógicos

Los operadores lógicos **and**, **or**, **xor**, y **not** se usan para determinar el valor de "verdad" de una expresión, por ejemplo:

```
1 and 1     -- 1 (verdadero)
1 and 0     -- 0 (falso)
0 and 1     -- 0 (falso)
0 and 0     -- 0 (falso)

1 or  1     -- 1 (verdadero)
1 or  0     -- 1 (verdadero)
0 or  1     -- 1 (verdadero)
0 or  0     -- 0 (falso)

1 xor 1     -- 0 (falso)
1 xor 0     -- 1 (verdadero)
0 xor 1     -- 1 (verdadero)
0 xor 0     -- 0 (falso)

not 1       -- 0 (falso)
not 0       -- 1 (verdadero)
```

También puede aplicar estos operadores a otros números que no sean 1 o 0. La regla es: cero significa falso y no cero es verdadero. Entonces, por ejemplo:

```
5 and -4    -- 1 (verdadero)
not 6       -- 0 (falso)
```

Estos operadores también se aplican a secuencias. Ver debajo. En algunos casos la evaluación de **corto−circuito** se usará para expresiones que contienen **and** u **or**.

2.2.3 Operadores aritméticos

Los operadores aritméticos disponibles son: suma, resta, multiplicación, división, menos unario y más unario.

```
3.5 + 3  -- 6.5
3 - 5    -- -2
6 * 2    -- 12
7 / 2    -- 3.5
-8.1     -- -8.1
+8       -- +8
```

Al calcular un resultado que es demasiado grande (es decir, fuera del rango −1e300 a +1e300) aparecerá uno de los átomos especiales **+infinity** o **−infinity**. Aparecen como **inf** o **−inf** cuando los imprime. También es posible generar un **nan** o **−nan**. "nan" significa "no es un número" (del inglés, "not a number"), es decir, un valor indefinido (como inf dividido inf). Estos valores están definidos en el estándar de punto flotante del IEEE. Si ve uno de estos valores especiales en la salida de su programa, normalmente indica un error en la lógica de su programa, aunque generar inf como resultado intermedio puede ser aceptable en algunos casos. Por ejemplo, 1/inf es 0, que puede ser la respuesta "correcta" para su algoritmo.

La división por cero, tanto como argumentos erróneos en rutinas de librería, por ejemplo, raíz cuadrada de un número negativo, logaritmo de un número no positivo, etc, causará un mensajede error inmediato y la cancelación del programa. La única razón por la que querría usar el "más unario", sería para enfatizar la lectura de un número (como valor positivo). El intérprete realmente no calcula nada con ellos.

2.2.4 Operaciones sobre secuencias

Todos los operadores relacionales, lógicos y aritméticos descriptos antes, como las rutinas matemáticas descriptas en Parte II − Rutinas de Librería, se pueden aplicar tanto a las secuencias como a los números simples (átomos).

Al aplicar un operador unario (un operando) a una secuencia, realmente se aplica a cada elemento de la secuencia para producir una secuencia de resultados de la misma longitud. Si uno de esos elementos es en sí mismo una secuencia,

entonces la misma regla se aplica recursivamente, por ejemplo:

```
x = -{1, 2, 3, {4, 5}}   -- x es {-1, -2, -3, {-4, -5}}
```

Si un operador binario (dos operandos) tiene operandos que ambos son secuencias, entonces las dos secuencias deben tener la misma longitud. La operación binaria se aplica entonces a los elementos correspondientes tomados de las dos secuencias, y obteniendo una secuencia de resultados, por ejemplo:

```
x = {5, 6, 7, 8} + {10, 10, 20, 100}
-- x es {15, 16, 27, 108}
```

Si el operador binario tiene un operando que es una secuencia, mientras que el otro es un número simple (átomo), entonces el número simple se repite para formar una secuencia de igual longitud que la secuencia operando. Entonces se aplican las reglas para operaciones sobre dos secuencias. Algunos ejemplos:

```
y = {4, 5, 6}

w = 5 * y              -- w es {20, 25, 30}

x = {1, 2, 3}

z = x + y              -- z es {5, 7, 9}

z = x <y              -- z es {1, 1, 1}

w = {{1, 2}, {3, 4}, {5}}

w = w * y              -- w es {{4, 8}, {15, 20}, {30}}

w = {1, 0, 0, 1} and {1, 1, 1, 0}    -- {1, 0, 0, 0}

w = not {1, 5, -2, 0, 0}     -- w es {0, 0, 0, 1, 1}

w = {1, 2, 3} = {1, 2, 4}    -- w es {1, 1, 0}


-- observe que el primer '=' es de asignación, y el segundo '=' es un operador relacional que prueba la igua
```

Nota: Cuando desee comparar dos cadenas (u otras secuencias) , **no** debería (como en algún otro lenguaje) usar el operador '=':

```
if "APPLE" = "ORANGE" then  -- ERROR!
```

'=' es tratado como un operador, como '+', '*' etc., por lo que se aplica a los elementos correspondientes de las secuencias, teniendo que ser éstas de la misma longitud. Cuando sus longitudes son iguales, el resultado es una secuencia de 1 y 0. Cuando las longitudes no son las mismas, el resultado es un error. De cualquiera forma obtendrá un error, since la condición if tiene que ser un átomo, no una secuencia. En su lugar, debería usar la rutina interna ***equal()***:

```
if equal("APPLE", "ORANGE"} then  -- CORRECTO
```

En general, puede hacer comparaciones relacionales usando la rutina interna ***compare()***:

```
if compare("APPLE", "ORANGE"} = 0 then  -- CORRECTO
```

Puede usar ***compare()*** para otras comparaciones como:

```
if compare("APPLE", "ORANGE"} <0 then  -- CORRECTO
    -- entrar aquí si "APPLE" es menor que "ORANGE" (verdadero)
```

2.2.5 Indexación de secuencias

Se puede seleccionar un solo elemento de una secuencia al darle un número de elemento encerrado entre corchetes. Los números de elemento comienzan en 1. Los índices no enteros se redondean hacia abajo al entero más próximo.

Por ejemplo, si 'x' contiene {5, 7.2, 9, 0.5, 13}, entonces x[2] es 7.2. Suponga que asignamos otra cosa a x[2]:

```
x[2] = {11,22,33}
```

Entonces 'x' se convierte en: {5, {11,22,33}, 9, 0.5, 13}. Si ahora preguntamos por x[2], obtendremos {11,22,33} y si preguntamos por x[2][3] obtendremos el 33. Si intenta utilizar un índice que está fuera del rango 1 a cantidad de elementos, obtendrá un error de índice. Por ejemplo, x[0], x[-99] o x[6] causarán errores. También lo hará x[1][3], ya que x[1] no es una secuencia. No hay límite en la cantidad de índices que pueden seguir a una variable, pero la variable tiene que contener secuencias que estén anidadas con la suficiente profundidad. El array de dos dimensiones, común en muchos lenaguajes, se puede representar fácilmente con una secuencia de secuencias:

```
x = {
     {5, 6, 7, 8, 9},      -- x[1]
     {1, 2, 3, 4, 5},      -- x[2]
     {0, 1, 0, 1, 0}       -- x[3]
    }
```

en donde escribimos los números de forma tal que resulte más clara la estructura. Se puede usar una expresión de la forma x[i][j] para acceder a cualquier elemento.
Sin embargo, las dos dimensiones no son simétricas, ya que se puede seleccionar una "fila" entera con x[i], pero no hay una expresión sencilla para seleccionar una columna entera. También se pueden manejar fácil y flexiblemente otras estructuras lógicas, como arrays de n dimensiones, arrays de cadenas, estructuras, arrays de estructuras, etc:

```
array 3-D:
y = {
     {{1,1}, {3,3}, {5,5}},
     {{0,0}, {0,1}, {9,1}},
     {{-1,9},{1,1}, {2,2}}
    }
```

y[2][3][1] es 9

Array de cadenas:

```
s = {"Hola", "Mundo", "Euphoria", "", "El único"}
```

s[3] es "Euphoria"
s[3][1] es 'E'

Una estructura:

```
empleado = {
            {"Juan","Perez"},
            45000,
            27,
            185.5
           }
```

Para acceder a los "campos" o elementos dentro de una estructura, es un buen estilo de programación hacer un conjunto de constantes con los nombres de los campos. Esto hace que el programa sea más fácil de leer. Por ejemplo, arriba tendría que tener:

```
constant NOMBRE = 1
constant PRIMER_NOMBRE = 1, APELLIDO = 2

constant SALARIO = 2
constant EDAD = 3
constant ALTURA = 4
```

Entonces podría acceder al nombre de una persona con: empleado[NOMBRE], o si quiere el apellido podría decir: empleado[NOMBRE][APELLIDO].

Array de estructuras:

```
empleados = {
             {{"Juan","Perez"}, 45000, 27, 185.5},   -- a[1]
             {{"José","García"}, 57000, 48, 177.2},   -- a[2]

             -- .... etc.
            }
```

empleados[2][SALARIO] would be 57000.

Las estructuras de datos Euphoria son casi infinitamente flexibles. Los arrays en otros lenguajes están restrigidos a tener una cantidad fija de elementos, donde todos ellos tienen que ser del mismo tipo. Euphoria elimina ambas restricciones. Puede agregar fácilmente una nueva estructura a la secuencia empleado de más arriba, o almacenar un inusual nombre largo en el campo NOMBRE y Euphoria se hará cargo de esto por usted. Si desea, puede almacenar una variedad de distintas "estructuras" empleado, con diferentes tamaños, dentro de una sola secuencia.

Un programa Euphoria no solamente puede representar fácilmente todas las estructuras de datos convencionales, sino que puede crear estructuras muy útiles y flexibles, que serían extremadamente complicadas para declarar en otros lenguajes. Ver 2.3 Euphoria vs. lenguajes convencionales.

Advierta que las expresiones en general no puede indexarse, solamente las variables. Por ejemplo: {5+2,6-1,7*8,8+1}[3] ***no*** está soportado, ni algo como: date()[MONTH]. Tiene que asignar la secuencia devuelta por date() a una variable, para entonces poder obtener el mes, usando un índice.

2.2.6 Subrangos en secuencias

Se puede seleccionar una secuencia de elementos consecutivos, dándole los números de elemento inicial y final. Por ejemplo, si 'x' es {1, 1, 2, 2, 2, 1, 1, 1}, entonces x[3..5] es la secuencia {2, 2, 2}. x[3..3] es la secuencia {2}. También está permitido x[3..2]. Evalúa la secuencia de longitud 0 {}. Si 'y' tiene el valor {"alfredo", "pedro", "maría"}, entonces y[1..2] es {"alfredo", "pedro"}.

También podemos usar subrangos para sobreescribir porciones de variables. Después de x[3..5] = {9, 9, 9} 'x' sería {1, 1, 9, 9, 9, 1, 1, 1}. También podríamos haber dicho x[3..5] = 9 con el mismo efecto. Suponga que 'y' es {0, "Euphoria", 1, 1}. Entonces, y[2][1..4] es "Euph". Si decimos y[2][1..4]="ABCD", entonces 'y' se convertirá en {0, "ABCDoria", 1, 1}.

En general, un nombre de variable puede estar seguido por 0 o más índices, seguido a su vez por 0 o 1 subrangos. Solamente las variables pueden tener índices o subrangos, no así las expresiones.

Necesitamos ser un poco más precisos definiendo las reglas para los **subrangos vacíos**. Considere el subrango s[i..j], donde 's' es de longitud 'n'. Un subrango de i a j, donde j = i−1 y i >= 1 produce la secuencia vacía, aún si i = n+1. Así [1..0] y [n+1..n] y todos entre ambos, son **subrangos (vacíos)** permitidos. Los subrangos vacíos son absolutamente útiles en muchos algoritmos. Un subrango de i a j, donde j < i − 1 es ilegal , es decir, un subrango "inverso" tal como s[5..3], no está permitido.

2.2.7 Concatenación de secuencias y átomos − El operador

Dos objetos cualquiera se pueden concatenar usando el operador &. El resultado es una secuencia con una longitud igual que las suma de las longitudes de los respectivos objetos (donde se puede considerar que los átomos tienen longitud 1), por ejemplo:

```
{1, 2, 3} &4              -- {1, 2, 3, 4}

4 &5                      -- {4, 5}

{{1, 1}, 2, 3} &{4, 5}    -- {{1, 1}, 2, 3, 4, 5}

x = {}
y = {1, 2}
y = y &x                  -- y es aún {1, 2}
```

Puede borrar el elemento 'i' de cualquier secuencia 's', al concatenar las partes de la secuencia anteriores y posteriores a 'i':

```
s = s[1..i−1] &s[i+1..length(s)]
```

Esto funciona tanto 'i' sea 1, como length(s), ya que e s[1..0] es un legal rango vacío, y por ende s[length(s)+1..length(s)] también lo es.

2.2.8 Formación de secuencias

Finalmente, la formación de secuencias, usando llaves y comas:

```
{a, b, c, ... }
```

también es un operador. Toma 'n' operandos, donde n es 0 o más, y construye una secuencia de 'n' elementos con sus valores, por ejemplo:

```
x = {apple, orange*2, {1,2,3}, 99/4+foobar}
```

El operador de formación de secuencias, está listado al final del cuadro de precedencias.

2.2.9 Otras operaciones sobre secuencias

Algunas otras importantes operaciones que puede realizar sobre secuencias tienen nombres en inglés, en lugar de caracteres especiales. Estas operaciones están incorporadas en **ex.exe**/**exw.exe**/**exu**, por lo que siempre estarán presentes y también serán rápidas. Se describen en detalle en Parte II − Rutinas de Librería, pero son lo suficientemente importantes para la programación en Euphoria, que las mencionaremos aquí, antes de seguir. Se las llama como si fueran subrutinas, aunque están realmente implementadas de una forma mucho más eficiente que ellas.

length(s)

length() le dice la longitud de una secuencia 's', que es la cantidad de elementos en 's'. Algunos de sus elementos pueden ser secuencias que contienen sus propios elementos, pero **length()** solamente le da el recuento del "nivel más alto". Obtendrá un error si pide la longitud de un átomo, por ejemplo:

```
length({5,6,7})             -- 3
length({1, {5,5,5}, 2, 3})  -- 4 (no 6!)
length({})                  -- 0
length(5)                   -- error!
```

repeat(item, count)

repeat() construye una secuencia que consta de un 'item' repetido 'count' veces, por ejemplo:

```
repeat(0, 100)         -- {0,0,0,...,0}   es decir, 100 ceros
repeat("Hola", 3)     -- {"Hola", "Hola", "Hola"}
repeat(99,0)           -- {}
```

El ítem a repetirse puede ser cualquier átomo o secuencia.

append(s, item) / prepend(s, item)

append() crea una nueva secuencia, agregando un 'item' al final de la secuencia 's'. **prepend()** crea una nueva secuencia, agregando un 'item' al comienzo de la secuencia 's', por ejemplo:

```
append({1,2,3}, 4)         -- {1,2,3,4}
prepend({1,2,3}, 4)        -- {4,1,2,3}

append({1,2,3}, {5,5,5})   -- {1,2,3,{5,5,5}}
prepend({}, 9)             -- {9}
append({}, 9)              -- {9}
```

La longitud de la nueva secuencia es siempre mayor en una unidad (1) que la longitud de la secuencia original. El ítem a agregarse a la secuencia puede ser un átomo u otra secuencia.

Estas dos funciones internas, **append()** y **prepend()**, tienen alguna similaridad con el operador de concatenación **pero hay claras diferencias, por ejemplo:**

```
-- agregando una secuencia es diferente
append({1,2,3}, {5,5,5})   -- {1,2,3,{5,5,5}}
{1,2,3} &{5,5,5}          -- {1,2,3,5,5,5}

-- agregando un átomo es lo mismo
append({1,2,3}, 5)         -- {1,2,3,5}
{1,2,3} &5                -- {1,2,3,5}
```

2.2.10 Cuadro de precedencias

La precedencia de operadores en expresiones es la siguiente:

```
precedencia más alta:   función/tipo llamadas

                        unario-  unario+  not

                        *  /

                        +  -

                        &

                        < >   <=  >=  =  !=

                        and  or  xor

precedencia más baja:   { , , , }
```

Así, 2+6*3 significa 2+(6*3) en lugar de (2+6)*3. Los operadores en la misma línea de arriba que tienen igual precedencia se los evalúa de izquierda derecha.

El símbolo igual '=' usado en una ***sentencia de asignación*** no es un operador, sino parte de la sintaxis del lenguaje.

2.3 Euphoria vs. lenguajes convencionales

Al basar a Euphoria en una estructura de datos recursiva, general y simple, se evitó la tremenda cantidad de complejidad que se encuentra normalmente en otros lenguajes de programación. Los arrays, estructuras, uniones, arrays de registros, arrays multidimensionales, etc, de otros lenguajes se pueden representar muy fácilmente en Euphoria con secuencias. También, estructuras de mayor nivel como listas, pilas, colas, árboles ,etc.

Además, en Euphoria se pueden tener secuencias de tipos variados; puede asignar cualquier objeto a un elemento de una secuencia; y las secuencias crecen y decrecen fácilmente en longitud, sin tener que preocuparse del asunto de la asignación de memoria. No se tiene que declarar de antemano la disposición exacta de la estructura de datos, pudiendo cambiar dinámicamente cuando se necesite. Es sencillo escribir código genérico, donde por ejemplo, puede poner y quitar en una pila distintos tipos de objetos de datos. Hacer una lista flexible que pueda contener una variedad de distintos tipos de objetos de datos, una cuestión trivial en Euphoria, pero requiere de una buena cantidad de pesado código en otros lenguajes.

Las manipulaciones de estructuras de datos son muy eficientes, ya que el intérprete Euphoria apuntará a objetos de datos grandes, en lugar de copiarlos.

La programación en Euphoria está totalmente basada en la creación y manejo flexible de secuencias dinámicas de datos. Las secuencias son ***todo*** − no hay otra estructura de datos que aprender. Se opera en un mundo sencillo, seguro y elástico de ***valores***, que está lejos del tedioso y peligroso mundo de los bits, bytes, punteros y "colgadas" de las máquinas.

A diferencia de otros lenguajes como LISP y Smalltalk en Euphoria, la "recolección de desperdicios" de almacenamiento sin uso, es un proceso continuo que nunca causa demoras aleatorias en la ejecución de un programa y no preasigna enorme regiones de memoria.

Las definiciones de los lenguajes convencionales como C, C++, Ada, etc. son muy complejas. La mayoría de los programadores terminan usando sólo un subconjunto del lenguaje. Los estándares ANSI de esos lenguajes se perecen a complicados documentos legales.

Se verá forzado a escribir código distinto para cada tipo de dato diferente, simplemente para copiar el dato, pedir su longitud actual, concatenarlo, compararlo, etc. Los manuales de esos lenguajes contienen rutinas como "strcpy", "strncpy", "memcpy", "strcat", "strlen", "strcmp", "memcmp", etc., donde cada una solamente trabaja con alguno de todos los tipos de datos.

La mayoría de la complejidad gira en torno de los tipos de datos. ¿Cómo define nuevos tipos? ¿Qué tipos de datos se pueden mezclar? ¿Cómo convierte un tipo en otro, en una forma tal que mantenga feliz al compilador? Cuando necesita hacer algo que requiere flexibilidad en tiempo de ejecución, frecuentemente se encuentra tratando de engañar al compilador.

En esos lenguajes, el valor numérico 4 (por ejemplo) puede tener diferentes significados dependiendo si se trata de un entero, un caracter, un byte, un doble, etc. En Euphoria, 4 es el átomo 4, y punto. En algunas oportunidades, Euphoria los llama tipos como veremos más tarde, pero el concepto es mucho más simple.

Las cuestiones de la asignación y desasignación dinámica de memoria, consumen una gran parte del tiempo de codificación y depuración de los programadores en estos lenguajes, y hacen que los programas sean más difíciles de comprender. Los programas que tienen que correr continuamente, frecuentemente muestran "agujeros" de memoria, ya que toma un trabajo muy disciplinado la liberación segura y correcta de todos los bloques de memoria una vez que ya no se los necesita más.

Las variables de puntero se usan extensivamente. Se ha llamado al puntero el "ir a" de las estructuras de datos. Esto fuerza a los programadores a pensar en los datos como si estuvieran ligados a posiciones fijas de memoria, donde se los tiene que manejar de formas intrincadas, no portables y a bajo nivel. Euphoria no tiene punteros, ni los necesita.

2.4 Declaraciones

2.4.1 Identificadores

Los **identificadores**, que constan de nombres de variables y otros símbolos definidos por el usuario, pueden tener cualquier longitud. Las mayúsculas y las minúsculas son distintas. Los identificadores tienen que comenzar con una letra, a su vez seguida por otras letras, dígitos o caracteres de subrayado. Las siguientes **palabras reservadas** tienen un significado especial en Euphoria, por lo que no se las puede usar como identificadores:

```
and             end             include         to
by              exit            not             type
constant        for             or              while
do              function        procedure       with
```

```
else          global          return          without
elsif         if              then            xor
```

El editor Euphoria muestra estas palabras en azul. Se pueden usar los identificadores para nombrar:

- procedimientos
- funciones
- tipos
- variables
- constantes

Procedimientos

Realizan algún cálculo y pueden tener una lista de parámetros, por ejemplo:

```
procedure vacio()
end procedure

procedure dibujar(integer x, integer y)
    position(x, y)
    puts(1, '*')
end procedure
```

Hay una cantidad fija de parámetros con nombre, pero esto no es restrictivo, ya que cualquier parámetro puede ser una secuencia de longitud variable de objetos arbitrarios. En muchos lenguajes, no es posible tener las listas de parámetros de longitud variable. En C, tiene que utilizar unos extraños mecanismos que son lo suficientemente complejos como para que el programador medio tenga que consultar el manual o al gurú local.

Se pasa una copia del valor de cada argumento. Las variables de parámetros formales se pueden modificar dentro del procedimiento, pero no afecta al valor de los argumentos.

Nota de rendimiento:

El intérprete no copia secuencias o números de punto flotante, a menos que sea necesario. Por ejemplo:

```
y = {1,2,3,4,5,6,7,8.5,"ABC"}
x = y
```

La sentencia x = y no causará que se cree una nueva copia de 'y'. Tanto 'x' como 'y' simplemente "apuntarán" a la misma secuencia. Si luego hacemos x[3] = 9, entonces se creará una secuencia separada para 'x' en la memoria (aunque aún solo será una copia compartida de 8.5 y "ABC"). Lo mismo se aplica a las "copias" de argumentos pasados a las subrutinas.

Funciones

Son como los procedimientos, pero devuelven un valor, y se las puede usar en expresiones, por ejemplo:

```
function max(atom a, atom b)
    if a >= b then
        return a
    else
        return b
    end if
end function
```

Se puede devolver cualquier objeto Euphoria. Puede, de hecho, tener múltiples valores de retorno al devolver una secuencia de objetos. Por ejemplo:

```
return {x_pos, y_pos}
```

Usaremos el término general "subrutina" o simplemente "rutina" cuando una observación sea aplicable tanto a procedimientos como a funciones.

Tipos

Estos son funciones especiales que se usan para declarar los valores permitidos para las variables. Un tipo tiene que tener exactamente un parámtro y debería devolver un átomo que es tanto verdadero (no−cero) como falso (cero). También se puede llamar a los tipos como cualquier otra función. Ver 2.4.3 Especificando el tipo de una variable.

Variables

Se les puede asignar valores durante la ejecución, por ejemplo:

```
-- x solo puede recibir valores enteros
integer x
x = 25

-- a, b y c pueden recibir *cualquier* valor
object a, b, c
a = {}
b = a
c = 0
```

Al declarar una variable, le pone un nombre (que impide cometer errores ortográficos luego) y le especifica los valores que legalmente se le pueden asignar durante la ejecución del programa.

Constantes

Son variables que se asignan con un valor inicial que nunca puede cambiar, por ejemplo:

```
constant MAX = 100
constant mayor = MAX - 10, menor = 5
constant nombres = {"Francisco", "Gloria", "Luis"}
```

El resultado de cualquier expresión se puede asignar a una constante, aún uno que involucre llamadas a funciones previamente definidas, pero una vez que se hace la asignación, el valor de la variable constante queda "bloqueado".

No se pueden declarar constantes dentro de una subrutina.

2.4.2 Ambito

El *ámbito* de un símbolo es la parte del programa donde la declaración de ese símbolo tiene efecto, es decir, donde el símbolo es *visible*.

En Euphoria, cada símbolo se tiene que declarar antes que se lo use. Puede leer un programa Euphoria desde el comienzo hasta el final sin encontrar ninguna variable o rutina que no se la haya definido. Es posible llamar a una rutina esté más adelante en el código fuente, pero tiene que usar las funciones especiales, **routine_id()**, y **call_func()** o **call_proc()** para hacerlo. Ver Parte II − Rutinas de Librería − Llamadas dinámicas.

Se puede llamar *recursivamente* a procedimientos, funciones y tipos. La recursión mútua, donde una rutina A llama a la rutina B, la cual directa o indirectamente llama a la rutina A, requiere del mecanismo **routine_id()**.

Un símbolo está definido desde el lugar en donde se lo declara hasta el final de su **ámbito**. El ámbito de una variable declarada dentro de un procedimiento o función (una variable **privada**) termina la final del procedimiento o función. El ámbito de las demás variables, constantes, procedimientos, funciones y tipos termina al final del código fuente en el que están declarados y se los referencia como **local**, a menos que la palabra clave **global** preceda su declaración, por lo que su ámbito se extiende indefinidamente.

Cuando incluye (**include**) un archivo Euphoria en el archivo principal (ver 2.6 Sentencias especiales de alto nivel), solamente las variables y rutinas declaradas usando la palabra clave **global** serán visibles en el archivo principal. Las otras declaraciones, no globales, del archivo de inclusión no estarán visibles, y obtendrá un mensaje de error, "no declarada", si intenta usarlas en el archivo principal.

Los símbolos marcados como **global** se pueden usar externamente. Los deás símbolos solamente se pueden usar internamente dentro de sus archivos. Esta información es útil cuando se mantiene o mejora un archivo, o cuando se está aprendiendo a usarlo. Puede efectuar cambios en las rutinas y variables internas, sin tener que revisar otros archivos, ni tener que avisarles a los usuarios del archivo de inclusión.

A veces, al usar archivos de inclusión desarrollados por terceros, encontrará conflictos de nombres. El autor de un archivo de inclusión usó el mismo nombre para un símbolo global, que otro autor para otro símbolo. Si tiene el código fuente, simplemente puede editar uno de los archivos de inclusión para corregir el problema, pero tendría que repetir este proceso toda vez que se lance una nueva versión del archivo de inclusión. Euphoria tiene una manera sencilla de resolver este problema. Usando una extensión en la sentencia include, por ejemplo:

```
include archivo_de_juan.e as juan
include archivo_de_pepe.e as pepe
juan:x += 1
pepe:x += 2
```

En este caso, la variable 'x' está declarada en dos archivos diferentes, y necesita referirse a ambas en su archivo. Usando el *identificador de espacio de nombres* de tanto 'juan' como de 'pepe', puede agregar un prefijo a 'x', de forma de poder indicar a cual variable 'x' se está refiriendo. A veces decimos que 'juan' se refiere a un *espacio de nombres*,

mientras que 'pepe' se refiere a otro *espacio de nombres* distinto. Puede adjuntar un identificador de espacio de nombres a cualquier variable definida por el usuario, constante, procedimiento o función. Puede hacer esto para resolver un conflicto, o simplemente para hacer las cosas más claras. Un identificador de espacio de nombres tiene ámbito local. Es conocido solamente dentro del archivo que lo declara, es decir, el archivo que contiene la sentencia include. Distintos archivos tienen que definir diferentes identificadores de espacios de nombres al referirse al mismo archivo de inclusión.
Euphoria lo alienta a restringir el ámbito de los símbolos. Si todos los símbolos fueran globales automáticamente para el programa entero, podría tener muchos conflictos de nombres, especialmente en programas extensos formados por archivos escritos por varios programadores. Un conflicto de nombres puede provocar un mensaje de error del compilador, o generar un "bug" demasiado sutil, donde distintas partes del programa modifiquen accidentalmente la misma variable sin estar enteradas de ello. Intente usar el ámbito más restrictivo que pueda. Haga las variables **private** en una rutina toda vez que sea posible, y si no lo es entonces hágalas **local**, en lugar de **global.**

Cuando Euphoria busca la declaración de un símbolo, primero verifica la rutina actual, luego el archivo actual y finalmente los globales en otros archivos. Los símbolos que son más locales *sobreescribirán* a los símbolos que son más globales. Al final del ámbito de los símbolos locales, el más global estará visible nuevamente.

Las declaraciones de **constantes** tienen que estar fuera de cualquier rutina. Las constantes pueden ser globales o locales, pero no privadas (**private**).

Las declaraciones de variables dentro de una subrutina tienen que estar al comienzo, antes de las sentencias ejecutables de la subrutina.

Las declaraciones en el nivel más alto, fuera de cualquier subrutina, no tienen que estar anidadas dentro de ningún ciclo o sentencia if.

La variable de control usada en el ciclo for es especial. Se declara automáticamente al comienzo del ciclo, y su ámbito termina al final del ciclo for. Si el ciclo está dentro de una función o procedimiento, la variable del ciclo es una variable **private** y no puede tener el mismo nombre que otra variable **private**. Cuando el ciclo está en el nivel superior, fuera de cualquier función o procedimiento, la variable del ciclo es **local** y no puede tener el mismo nombre que otra variable **local** en ese archivo. Puede usar el mismo nombre en distintos ciclos for, mientras que los ciclos no estén anidados. Las variables de ciclo no se declaran como las otras variables. El rango de valores especificados en la sentencia for, define los valores legales de la variable del ciclo − especificar un tipo sería redundante y no está permitido.

2.4.3 Especificando el tipo de una variable

Anteriormente hemos visto algunos ejemplos de tipos de variables, pero ahora los vamos a definir más precisamente.

Las declaraciones de variables tienen un nombre de tipo, seguido por la lista de las variables que se están declarando. Por ejemplo:

```
object a

global integer x, y, z

procedure calcular(sequence q, sequence r)
```

Los tipos: **object**, **sequence**, **atom** e **integer** son **predefinidos**. Las variables de tipo **object** pueden tomar ***cualquier*** valor. Aquellas declaradas con el tipo **sequence** tienen que ser siempre secuencias. Aquellas declaradas con el tipo **atom** tienen que ser siempre átomos. Aquellas declaradas con el tipo **integer** tienen que ser átomos con valores enteros desde −1073741824 a +1073741823 inclusive. Puede efectuar cálculos exactos con valores enteros grandes, hasta 15 dígitos decimales, pero declárelos como **atom**, en lugar de ***integer***.

Nota:

En la lista de parámetros de una función o procedimiento como el calcular() de arriba, un nombre de tipo solamente puede estar seguido por un solo nombre de parámetro.

Nota de rendimiento:

Los cáculos usando variables declaradas como enteros serán normalmente más rápidos que los cálculos cuyas variables están definidas como átomos. Si su máquina tiene hardware de punto flotante, Euphoria lo usará para manejar los átomos que no se representan como enteros. Si su máquina no tiene If your machine hardware de punto flotante, Euphoria llamará rutinas de software de aritmética de punto flotante contenidas en **ex.exe** (o en Windows). Puede forzar a que ex.exe saltee cualquier hardware de punto flotante, al establecer una variable de entorno:

```
SET NO87=1
```

Se usarán rutinas de software más lentas, pero esto podría ser una ventaja si está preocupado por el "bug" de punto flotante de algunos de los primeros procesadores Pentium.

Para aumentar los tipos predefinidos, puede crear **tipos definidos por el usuario**. Todo lo que tiene que hacer es definir una función de un único parámtro, pero declararla con **type ... end type** en lugar de **function ... end function**. Por ejemplo:

```
type hora(integer x)
    return x >= 0 and x <= 23
end type

hour h1, h2

h1 = 10      -- ok
h2 = 25      -- error! el programa se aborta con un mensaje
```

Las variables 'h1' y 'h2' se pueden asignar solamente con valores enteros en el rango de 0 a 23 inclusive. Después de cada asignación a 'h1' o 'h2' el intérprete llamará a hora(), pasándole el nuevo valor. Primero se verificará el valor para ver si es un entero (debido a "integer x"). Si lo es, la sentencia return se ejecutará para probar el valor de 'x' (es decir, el nuevo valor de 'h1' o 'h2'). Si hora() devuelve verdadero, la ejecución continua normalmente. Si hora() devuelve falso, entonces el programa se aborta con un mensaje de disgnóstico.

Se puede usar a "hora" para declarar parámetros de subrutina:

```
procedure set_time(hora h)
```

set_time() se puede llamar solamente con un valor razonable de 'h', sino el programa se abortará con un mensaje.

El tipo de una variable se verificará luego de cada asignación a la variable (excepto donde el compilador puede determinar que una verificación no es necesaria), terminando el programa si la función de tipo devuelve falso. Los tipos de parámetro de la subrutina se verifican cada vez que se llama a la subrutina. Esta verificación garantiza que una variable nunca puede tener un valor que no pertenezca al tipo de dicha variable.

A diferencia de otros lenguajes, el tipo de una variable no afecta ningún cálculo en ella. El único valor de la variable que importa es en una expresión. El tipo solo sirve como una verificación de error que evita la "corrupción" de la variable.

Los tipos definidos por el usuario pueden capturar errores lógicos inesperados en el programa. No están diseñados para capturar o corregir los errores de las entradas del usuario.

La verificación de tipos se puede desactivar o activar entre subrutinas usando las sentencias especiales **with type_check** o **without type_check**. Por defecto, está activada.

Nota para probadores de prestaciones:

> Al comparar la velocidad de los programas Euphoria contra programas escritos en otros lenguajes, debería especificar **without type_check** al inicio del archivo. Esto le permite a Euphoria saltear las verificaciones de tipos en tiempo de ejecución, con lo que se ahorra algún tiempo de ejecución. All other checks are still performed, por ejemplo, verificación de índices, variables no inicializadas, etc. Aún cuando desactive la verificación de tipos, Euphoria se reserva el derecho de hacer algunas verificaciones en lugares estratégicos, ya que esto le permite ejecutar su programa ***más rápido*** en muchos casos. Por lo tanto, puede obtener una falla de verificación de tipos aún cuando la verificación de tipos esté desactivada. Ya sea que la verificación de tipos esté activada o desactivada, nunca obtendrá una excepción ***a nivel de máquina.*** **Siempre obtendrá un mensaje significativo de parte de Euphoria cuando las cosas no estén bien.** (*Esto puede no ser el caso cuando opera directamente con la memoria, o llama rutinas escritas en C o código de máquina*).

El método de Euphoria para definir tipos es más simple que otros que puede encontrar en otros lenguajes, Euphoria le da al programador la ***más amplia*** flexibilidad para definir los valores legales para un tipo de datos. Cualquier algoritmo puede usarse para incluir o excluir valores. Puede declarar una variable de tipo **objeto** que le permitirá tomar ***cualquier*** valor. Se pueden escribir rutinas para trabajar con tipos muy específicos, o tipos muy generales.

Para muchos programas, es una pequeña ventaja definir nuevos tipos, y podría desear to stick con los cuatro tipos predefinidos. A diferencia de otros lenguajes, el mecanismo de tipos de Euphoria es opcional. No lo necesita para crear un programa.

Sin embargo, para programas extensos, las definiciones de tipos estrictos puede ayudar en la depuración. Los errores lógicos se capturan cerca de su origen y no se permite que se propaguen en formas sutiles al resto del programa. Además, es más fácil razonar acerca del malfuncionamiento de una sección de código cuando está seguro que las variables involucradas siempre tienen un valor legal, sino el deseado.

Los tipos también proveen una significativa documentación acerca del programa, haciéndolo más fácil de comprender a futuro tanto por el programador como por terceros. Combinado con la verificación de índices, la verificación de variables no inicializadas, y otra verificación que siempre está presente, la verificación estricta de tipos en tiempo de ejecución, hacen la depuración mucho más fácil en Euphoria que en la mayoría de los lenguajes. Esto también incrementa la confiabilidad del prgrama final, ya que muchos bugs latentes que podrían haber sobrevivido a la fase de pruebas en otros lenguajes, serán capturados por Euphoria.

Anécdota 1:

Se descubrió una cantidad de bugs, al portar un extenso programa C a Euphoria. Aunque se creía que este programa C era completamente "correcto", encontramos una situación donde se leía una variable no inicializada; un lugar donde el número de elemento "−1" de un array se leía y escribía constantemente; y una situaciñon en la que algo se escrbía fuera de pantalla. Estos problemas se conviertieron en errores que no eran fácilmente visibles al observador casual, consecuentemente habían sobrevivido las pruebas del código C.

Anécdota 2:

El algoritmo Quick Sort presentado en la página 117 of *Escribiendo Programas Eficientes* de Jon Bentley tiene un error de índice! El algoritmo a veces leerá el elemento solo *antes* del comienzo del array a ordenar, a veces leerá el elemento solo *después* del final del array. El algoritmo funcionará cualquier cosa que sea leida − esto es probable porque el bug no fue capturado. ¿Pero que pasa si no hay memoria (virtual) justo antes o después del array? Más tarde Bentley modifica el algoritmo para eliminar el bug −− pero presentó esta versión como siendo la correcta. ***¡Aún los expertos necesitan la verificación de índices!***

Nota de rendimiento:

Cuando se usan extensivamente los tipos definidos por el usuario, la veridicación de tipos agrega de un 20% a un 40% de tiempo de ejecución. Déjelo así, a menos que necesite realmente la velocidad adicional. También debería considerar desactivarlo solamente para unas pocas rutinas de ejecución pesada. El análisis de perfiles de ejecución puede ayudar con esta decisión.

2.5 Sentencias

Están disponibles los siguientes tipos de sentencias ejecutables:

- Sentencia de asignación
- Llamada a procedimientos
- La sentencia if
- La sentencia while
- La sentencia for
- La sentencia return
- La sentencia exit

En Euphoria, no se usan los puntos y comas, pero puede poner varias sentencias como guste en una línea, o una sentencia simple en varias líneas. No puede separar una sentencia a la mitad de un identificador, cadena, número o palabra clave.

2.5.1 Sentencia de asignación

Una **sentencia de asignación** asigna el valor de una expresión a una variable simple, o a un subrango, o a un elemento de una variable, por ejemplo:

```
x = a + b

y[i] = y[i] + 1

y[i..j] = {1, 2, 3}
```

El valor previo de la variable, o elemento(s) de una variable con índices o subrangos, se descarta. Por ejemplo, suponga que 'x' era una secuencia de 1000 elementos que estaba inicializada con:

```
object x

x = repeat(0, 1000)  -- una secuencia de 1000 ceros
```

y luego le asignamos un átomo a 'x':

```
x = 7
```

Esto es perfectamente legal porque 'x' está declarada como **object**. El valor previo de 'x', la sencuencia de 1000 elementos, sencillamente desaparecerá. Realmente, el espacio consumido por la secuencia de 1000 elementos se recicla automáticamente, debido a la asignación dinámica dememoria de Euphoria.
Observe que el símbolo igual '=' se usa tanto para la asignación como para probar la igualdad. Nunca hay confusión porque una asignación en Euphoria es una sentencia solamente, no se puede usar como una expresión (como en C).

Asignación con operador

Euphoria provee también algunas formas adicionales para la sentencia de asignación.

Para ahorrar escritura y hacer el código un poco más claro, puede combinar la asignación con uno de estos

operadores:
`+ - / *`

Por ejemplo, en lugar de decir:

```
mylongvarname = mylongvarname + 1
```

Puede decir:

```
mylongvarname += 1
```

En lugar de decir:

```
galaxy[q_row][q_col][q_size] = galaxy[q_row][q_col][q_size] * 10
```

Puede decir:

```
galaxy[q_row][q_col][q_size] *= 10
```

y en lugar de decir:

```
accounts[start..finesh] = accounts[start..finesh] / 10
```

Puede decir:

```
accounts[start..finesh] /= 10
```

En general, toda vez que tenga una sentencia de la forma:

lado-izquierdo = lado-izquierdo op expresión

Puede decir:

lado-izquierdo op= expresión

donde ***op*** es uno de estas: `+ - * /`

Cuando el lado-izquierdo contiene índices o subrangos múltiples, la forma ***op=*** normalmente se ejecutará más rápido que la forma más larga. Cuando lo use, puede encontrar que la forma ***op=*** es ligeramente más legible que la forma larga, ya que la expresión del lado izquierdo aparece una sola vez.

2.5.2 Llamada a procedimientos

Una **llamada a procedimiento** inicia la ejecución de un procedimiento, pasándole una lista opcional de argumentos, por ejemplo:

```
plot(x, 23)
```

2.5.3 La sentencia if

Una **sentencia if** prueba una condición para ver si es 0 (falso) o no-cero (verdadero) y entonces ejecuta la serie de sentencias que corresponde. Pueden haber cláusulas **elsif** y **else** opcionales. Por ejemplo:

```
if a <b then
    x = 1
end if


if a = 9 and find(0, s) then
    x = 4
    y = 5
else
    z = 8
end if


if char = 'a' then
    x = 1
elsif char = 'b' or char = 'B' then
    x = 2
elsif char = 'c' then
    x = 3
else
    x = -1
```

```
end if
```

Observe que **elsif** es la contracción de **else if**, pero es más limpia porque no requiere un **end if** que la acompañe. Hay un solo **end if** para toda la sentencia **if**, aún cuando hayan varios **elsif's** en ella.

Las condiciones **if** y **elsif** se prueban usando la evaluación de corto−circuito.

2.5.4 La sentencia while

Una **sentencia while** prueba una condición para ver si es no−cero (verdadero), y mientras sea verdadero el ciclo se ejecuta, por ejemplo:

```
while x > 0 do
    a = a * 2
    x = x − 1
end while
```

Evaluación de corto−circuito

Cuando la condición probada por **if**, **elsif**, o **while** contiene los operadores **and** u **or**, se utilizará la evaluación de ***corto−circuito***. Por ejemplo:

```
if a <0 and b > 0 then ...
```

Si a < 0 es falso, entonces Euphoria no se molestará en probar si b es mayor que 0. Asumirá que todo el resto es falso. Similarmente,

```
if a <0 or b > 0 then ...
```

Si a < 0 es verdadero, entonces Euphoria decidirá inmediatamente que el resultado es verdadero, sin probar el valor de b.

En general, toda vez que tengamos una condición de la forma:

```
A and B
```

donde A y B pueden ser dos expresiones cualesquiera, Euphoria tomará un atajo cuando A es falso e inmediatamente dará como resultado falso, sin mirar la expresión B.
Análogamente, con:

```
A or B
```

cuando A es verdadero, Euphoria saltará la evaluación de la expresión B, y declarará al resultado como verdadero.

Si la expresión B contiene una llamada a una función y esa función tiene posibles **efectos colaterales**, es decir, que debería hacer más solamente devolver un valor, obtendrá una advertencia en tiempo de compilación. Las versiones más antiguas de Euphoria (anteriores a 2.1), no usaban la evaluación de **corto−circuito**, siendo pisible que algún código antiguo no funcione correctamente, aunque una búsqueda en los archivos de Euphoria concluyó que no hay ningún programa que dependa de efectos colaterales de esta manera.

La expresión B, podría contener algo que normalmente causaría un error en tiempo de ejecución. Si Euphoria salta la evaluación de B, no se descubrirá el error. Por ejemplo:

```
if x != 0 and 1/x > 10 then  -- se evita el error por división por cero

while 1 or {1,2,3,4,5} do    -- se evita la secuencia ilegal
```

B podria tener variables no inicializadas, índices fuera de rango, etc.

Esto podría verse como una codificación sloppy, pero de hecho frecuentemente le permite escribir algo en una forma más simple y más legible. Por ejemplo:

```
if átomo(x) or length(x)=1 then
```

Sin la evaluación de corto−circuito, tendría un problema cuando 'x' fuese un átomo, ya que length() no está definida para átomos. Con la evaluación de corto−circuito, se verificará length(x) solamente cuando 'x' sea una secuencia. De forma similar:

```
-- busca 'a' o 'A' en s
i = 1
while i <= length(s) and s[i] != 'a' and s[i] != 'A' do
```

```
        i += 1
    end while
```

En este ciclo, la variable 'i' podría ser eventualmente mayor que length(s). Sin la evaluación de corto−circuito, ocurriría un error de índice fuera de rango cuando s[i] se evalúa en la iteración final. Con la evaluación de corto−circuito, el ciclo terminará inmediatamente cuando i <= length(s) sea falso. Euphoria no evaluará s[i] != 'a' ni tampoco s[i] != 'A'. No ocurrirá un error de índice.
La evaluación de **corto−circuito** de **and** y **or** se realiza para las condiciones de **if**, **elsif** y **while** solamente. No se usa en otros contextos. Por ejemplo, la sentencia de asignación:

```
x = 1 or {1,2,3,4,5}  -- x debería establecerse a {1,1,1,1,1}
```

Si aquí se hubiera usado la evaluación de corto−circuito, estableceríamos 'x' a 1, y no se tendría en cuenta a {1,2,3,4,5}, lo cual está mal. La evaluación de corto−circuito se puede usar en condiciones if/elsif/while, debido a que solamente nos importa si el resultado es verdadero o falso, el cual es un átomo.

2.5.5 La sentencia for

Una **sentencia for** constituye un ciclo especial con una **variable de ciclo** controlando que corra desde un valor inicial hasta un valor final (en forma ascendente o descendente), por ejemplo:

```
for i = 1 to 10 do
    ? i   -- ? es una forma abreviada de print()
end for

-- tambien se permiten números fraccionarios
for i = 10.0 to 20.5 by 0.3 do
    for j = 20 to 10 by -2 do    -- cuenta regresiva
        ? {i, j}
    end for
end for
```

La **variable de ciclo** se declara automáticamente y existe hasta el final del ciclo. Fuera del ciclo, la variable no tiene valor, ni tampoco está declarada. Si necesita su valor final, cópiela en otra variable antes de abandonar el ciclo. El compilador no permitirá ninguna asignación a la variable del ciclo. Tanto el valor inicial, como el tope de ciclado y el incremento tienen que ser todos átomos. Si no se especifica el incremento, entonces se lo asume como 1. Los valores de tope e incremento se establecen cuando se entra en el ciclo, y no los afecta nada que ocurra durante la ejecución del ciclo. Ver también el ámbito de la variable de ciclo, en 2.4.2 Ambito.

2.5.6 La sentencia return

Una **sentencia return** regresa inmediatamente desde una subrutina. Si la subrutina es una función o tipo, entonces se tiene que devolver un valor, por ejemplo:

```
return

return {50, "ALFREDO", {}}
```

2.5.7 La sentencia exit

Una **sentencia exit** puede aparecer dentro de un ciclo while o un ciclo for. Provoca la terminación inmediata del ciclo, pasando el control a la primera sentencia que está después del ciclo, por ejemplo:

```
for i = 1 to 100 do
    if a[i] = x then
        location = i
        exit
    end if
end for
```

Es absolutamente común ver algo como esto:

```
constant TRUE = 1

while TRUE do
    ...
    if some_condition then
        exit
    end if
    ...
end while
```

es decir, un ciclo while "infinito" que termina mediante una sentencia exit en algún punto arbitrario del cuerpo

del ciclo.

Nota de rendimiento:

Euphoria optimiza este tipo de ciclos. En tiempo de ejecución, no se realiza ninguna prueba en lo más alto del ciclo. Es solamente un simple salto incondicional desde end while hacia la primera sentencia dentro del ciclo.

Con **ex.exe**, si se le ocurre crear un verdadero ciclo inifinito, sin que ocurra ninguna entrada /salida, no habrá forma de detenerlo. Tendría que presionar Ctrl+Alt+Supr, tanto sea para reiniciar la máquina, como (bajo Windows) terminar la sesión DOS. Si el programa tiene archivos abiertos para escritura, sería aconsejable ejecutar **scandesk** para verificar la integridad del sistema. Solamente cuando el programa una entrada por teclado, Ctrl+C abortará el programa (a menos que se use ***allow_break(0)***).

Con **exw.exe** o **exu**, Ctrl+C siempre detendrá el programa inmediatamente.

2.6 Sentencias especiales de alto nivel

Euphoria procesa el archivo **.ex** en un solo paso, comenzando en la primera línea y siguiendo hasta la última. Cuando se encuentra la definición de un procedimiento o función, se verifica la sintaxis de la rutina y se la convierte en una forma interna, pero no ocurre la ejecución. Cuando se encuentra una sentencia que está fuera de cualquier rutina, se verifica la sintaxis de la rutina y se la convierte en una forma interna y se la ejecuta inmediatamente. Una práctica común es inicializar inmediatamente una variable global, solo después de su declaración. Si su archivo **.ex** contiene solamente definiciones de rutinas, pero no sentencias de ejecución inmediata, nada ocurrirá cuando intente correrlo (salvo la verificación de sintaxis). Necesita tener una sentencia inmediata para llamar a su rutina principal (ver 1.1 Programa de ejemplo). Es absolutamente posible tener un archivo **.ex** con sentencias inmediatas únicamente, por ejemplo podría querer usar a Euphoria como una simple calculadora, escribiendo solamente la sentencia ***print()*** (o ***?***) dentro de un archivo y entonces ejecutarlo.

Como hemos visto, se puede usar cualquier sentencia Euphoria, incluyendo ciclos for, ciclos while, sentencias if, etc. (pero no return), en el nivel superior, es decir, ***fuera*** de cualquier función o procedimiento. En suma, las siguientes sentencias especiales ***solo*** pueden aparecer en el nivel superior:

- **include**
- **with / without**

2.6.1 Include

Cuando se escribe un programa extenso, frecuentemente es provechoso dividirlo en archivos separados lógicamente, usando **sentencias include.** A veces querrá reutilizar algún código que haya escrito previamente, o aquel que no haya escrito. En lugar de copiar este código en su programa principal, puede usar una **sentencia include** para referirse al archivo que contiene el código. La primera forma de la sentencia include es:

include nombre de archivo

Lee (compila) un archivo fuente de Euphoria.

Por ejemplo:

```
include graphics.e
```

Cualquier código de alto nivel que incluya el archivo de inclusión, será ejecutado.

Cualquier ***símbolo global*** que ya esté definido en el archivo principal estará visible en el archivo de inclusión.

N.B. Solamente aquellos símbolos definidos como **global** en el archivo de inclusión estarán visibles (accesibles) en el resto del programa.

Si se da un *nombre de archivo* absoluto, Euphoria lo usará. Cuando se da un *nombre de archivo* relativo, Euphoria buscará primero en el mismo directorio en el que está dado el archivo principal en la ***línea de comandos*** de **ex** (o **exw** o **exu**). Si no se lo ubica, y se definió la variable de entorno **EUINC**, se lo buscará en cada directorio listado en **EUINC** (de izquierda a derecha). Finalmente, si todavía no se lo encuentra, se lo buscará en **euphoria\include**. Este directorio contiene los archivos de inclusión estándares de Euphoria. La variable de entorno **EUDIR** le dice a **ex.exe/exw.exe/exu** donde encontrar el directorio **euphoria. EUINC** debería ser una lista de directorios, separados por puntos y comas (comas en Linux), similarmente a la variable PATH. Puede agregarse al archivo AUTOEXEC.BAT, por ejemplo:
SET EUINC=C:\EU\MYFILES;C:\EU\WIN32LIB
Esto le permite organizar sus archivos de inclusión de acuerdo a las áreas de aplicación, y evitando agregar numerosos archivos no relacionados al directorio ***euphoria\include***.

Un archivo de inclusión puede incluir otros archivos. De hecho, puede "anidar" archivos de inclusión hasta 10 niveles de profundidad.

Los nombres de los archivos de inclusión típicamente terminan en **.e**, o a veces en **.ew** o **.eu** (cuando se los escribe para usarlos con Windows o Linux). Es solamente una convención no exigible.

Si el nombre del archivo (o su ruta) contiene espacios en blanco, tiene que encerrarlo entre comillas dobles, en caso contrario las comillas son opcionales. Por ejemplo:

```
include "c:\program files\myfile.e"
```

Con excepción posiblemente de definir un nuevo identificador de espacio de nombres (ver debajo), una sentencia include será ignorada si un archivo con el mismo nombre ya se ha incluido.

Una sentencia include tiene que escribirse en una línea propia. Solamente puede aparecer un comentario después en la misma línea.

La segunda forma de la sentencia include es:

include nombre de archivo as identificador_de_espacio_de nombres

Esto es como un incluse simple, pero también define un *identificador de espacio de nombres* que se puede anexar a símbolos globales en un archivo de inclusión al que quiera referirlo en el archivo principal. Esto podría ser necesario para evitar referencias ambigüas para esos símbolos, o para hacer el código más legible. Ver reglas de ámbito.

2.6.2 With / without

Estas sentencias especiales afectan la manera en que Euphoria traduce su programa en la forma interna. No cambian la lógica de su programa, pero pueden afectar la información de diagnóstico que obtiene al correr el programa. Leer más información en 3. Debugging and Profiling.

with

Activa una de las opciones: profile, profile_time, trace, warning o type_check . Las opciones warning y type_check están inicialmente activadas, mientras que profile, profile_time y trace no.
Cualquier advertencia que se emita, aparecerá en pantalla después que el programa haya terminado de ejecutarse. Las advertencias indican problemas menores. Una advertencia nunca detendrá la ejecución del programa.

without
Desactiva una de las opciones mencionadas.

También hay una opción especial **with,** pocas veces utilizada, donde un número de código aparece después de **with**. En versiones anteriores, este código se usaba por RDS para hacer un archivo inmune al agregado de la sentencia count en la Edición de Dominio Público.

Puede seleccionar cualquier combinación de configuraciones, pudiendo cambiarlas, pero los cambios tienen que ocurrir ***entre*** subrutinas, no dentro de una subrutina. La única excepción es que solamente puede activar un tipo de análisis de perfiles de ejecución para una dada corrida de su programa.

Un **archivo de inclusión** hereda la configuración de **with/without** en el punto en donde es incluido. Un archivo de inclusión puede cambiar esas configuraciones, pero volverán a su estado original al final del archivo de inclusión. Por ejemplo, un archivo de inclusión puede desactivar advertencias por sí mismo y (inicialmente) para cualquier archivo que lo incluya, pero no desactivará las advertencias en el archivo principal.

3. Depuración y análisis de perfiles de ejecución

3.1 Depuración

La depuración en Euphoria es mucho más fácil que en el resto de los lenguajes de programación. La extensa verificación en tiempo de ejecución provista por el intérprete Euphoria captura muchos errores que en otros lenguajes podría tomar horas. Cuando el intérprete captura un error, siempre obtendrá un breve informe en pantalla y uno detallado en el archivo llamado **ex.err**. Estos informes incluyen la descripción de lo ocurrido, completamente en inglés, junto con un vuelco de la pila. El archivo **ex.err** también contendrá un volcado de todos los valores de las variables, y opcionalmente una lista de las sentencias ejecutadas recientemente. Para programas extremadamente extensos, solamente se muestra un volcado parcial. Si **ex.err** no es conveniente, puede elegir otro nombre de archivo, en cualquier parte de su sistema, llamando a ***crash_file()***.

Además, puede crear tipos definidos por el usuario que determinan precisamente el conjunto de valores legales para cada una de las variables. Se generará un informe de errores en el momento en que se asigne un valor ilegal a una variable.

A veces, un programa se comportará mal sin fallar en ninguna verificación en tiempo de ejecución. En cualquier lenguaje de programación puede ser buena idea estudiar el código fuente y replantear el algoritmo codificado. También puede ser útil insertar sentencias de impresión en puntos estratégicos de forma de monitorear la lógica interna del programa. Esta aproximación es particularmente conveniente en un lenguaje interpretado como Euphoria, ya que simplemente puede editar el código y volver a ejecutarlo sin tener que esperar una nueva compilación/enlazado.

El intérprete le provee poderosas herramientas adicionales para depuración. Usando ***trace(1)*** puede ***trazar*** la ejecución de su programa en una pantalla mientras es testigo de la salida del programa en otra. ***trace(2)*** es lo msimo que ***trace(1)***, pero la pantalla de trazado será monocromática. Finalmente, usando ***trace(3)***, puede registrar todas las sentencias ejecutadas en un archivo llamado **ctrace.out**.

La utilidad completa de trazado es parte de la Edición Completa de Euphoria, pero está habilitada en la Edición de Dominio Público para programas de hasta 300 sentencias.

Las sentencias especiales **with trace** / **without trace** seleccionan las partes del programa en las que quiere habilitar el trazado. Frecuentemente, insertará simplemente una sentencia **with trace** casi al comienzo del código fuente para hacer que el trazado sea completo. A veces es mejor ubicar el primer **with trace** después de todos sus tipos definidos por el usuario, por lo que no se realiza el trazado dentro de esas rutinas después de cada asignación a una variable. Otras veces, puede conocer exactamente que rutina o rutinas le interesa trazar, y solamente seleccionará alguna de ellas. Por supuesto, una vez que esté en la ventana de trazado, puede saltear la ejecución de cualquier rutina presionando la flecha abajo del teclado, en lugar del Enter.

Solamente debería ocurrir un error en tiempo de ejecución con las líneas trazables pueden aparecer en **ctrace.out** o en **ex.err** como "Traced lines leading up to the failure". Si quiere esta información y no la obtiene, debería insertar un **with trace** y volver a ejecutar su programa. La ejecución será más lenta con líneas compiladas con **with trace**, especialmente cuando se use ***trace(3)***.

Después de predeterminar las líneas que serán trazadas, el programa tiene que provocar dinámicamente que la utilidad de trazado se active, ejecutando la sentencia ***trace()***. Podría escribir:

```
with trace
trace(1)
```

al inicio del programa, por lo que se inicia el trazado no bien comienza la ejecución. Más comúnmente querrá disparar el trazado cuando se ejecute una cierta rutina, o surja alguna condición, por ejemplo:

```
if x <0 then
    trace(1)
end if
```

Puede desactivar el trazado ejecutando la sentencia ***trace(0)***. También lo puede desactivar interactivamente escribiendo 'q' para abandonar el trazado. Recuerde que **with trace** tiene que aparecer ***fuera*** de cualquier rutina, mientras que ***trace()*** puede aparecer tanto ***dentro*** de una rutina como ***fuera*** de ella.

Podría querer activar el trazado desde dentro de un tipo. Suponga que ejecuta su programa y éste falla, con el archivo **ex.err** mostrando que una de sus variables se estableció con un valor extraño, aunque no ilegal y quisiera saber cómo esto puede haber ocurrido. Simplemente cree un tipo para esa variable de manera que ejecute ***trace(1)*** si el valor que se le asigna a la variable no está comprendido entre los que espera, por ejemplo:

```
type positive_int(integer x)
    if x = 99 then
        trace(1) -- ¿cómo puede ser esto?
        return 1 -- keep going
```

```
        else
            return x > 0
        end if
    end type
```

Cuando positive_int() regresa, verá la sentencia exacta que causó que la variable se haya establecido a un valor extraño, y podrá verificar los valores de las otras variables. También podrá verificar la pantalla de salida para ver que ocurrió en ese preciso momento. Si define positive_int() para que devuelva 0 para el caso del valor extraño (99) en lugar de 1, puede forzar un volcado de diagnóstico a **ex.err**.

3.1.1 La pantalla de trazado

Cuando el intérprete ejecuta la sentencia ***trace(1)*** o ***trace(2)***, la pantalla principal de salida se guarda y aparece la **pantalla de trazado**. Ésta muestra una vista de su programa, con la sentencia próxima a ejecutarse resaltada, y mostrando también, algunas líneas anteriores y posteriores a ella. Al final de la pantalla se reservan algunas líneas para mostrar los nombres de las variables y sus valores. La línea superior muestra los comandos que puede ingresar en este punto:

F1 − muestra la pantalla principal de salida − eche un vistazo a la salida de su programa

F2 − redibuja la pantalla de trazado. Presione esta tecla mientras observa la pantalla principal de salida para regresar a la pantalla de trazado.

Enter − ejecuta solamente la sentencia que está resaltada

flecha−abajo − continua la ejecución y corta cuando se está por ejecutar la siguiente sentencia que viene después de la actual en el listado fuente. Esto le permite saltear las llamadasa subrutinas. También permite detenerse en la primera sentencia siguiente al final de un ciclo for o ciclo while sin tener que witness todas las repeticiones del ciclo.

? − muestra el valor de una variable. Después de presionar **?**, se le pedirá el nombre de la variable. Se muestran muchas variables automáticamente cuando tienen un valor asignado. Si una variable no se está mostrando, o se muestra parcialmente, puede preguntar por ella. Las secuencias grandes están limitadas a una línea en la pantalla de trazado, pero cuando pregunta por el valor de una variable que contiene una secuencia extensa, la pantalla se limpiará y podrá desplazarse a lo largo de toda la secuencia. Cuando regrese a la pantalla de trazado, se volverá a mostrar solamente una línea para la variable. Las variables que no están definidas en este punto del programa no se muestran y las que todavía no se inicializaron tendrán el mensaje "< NO VALUE >" ("SIN VALOR") al lado de su nombre.Solamente se mostrarán variables, no las expresiones generales. A medida que se vaya ejecutando el programa, el sistema actualizará cualquier valor mostrado en pantalla. Ocasionalmente, quitará las variables que queden fuera de ámbito, o las que no se hayan actualizado por largo tiempo, en comparación con las más nuevas o recientemente actualizadas.

q − abandona el trazado y reanuda la ejecución normal. El trazado comenzará nuevamente cuando se ejecute el siguiente ***trace(1)***.

Q − abandona el trazado y le permite al programa ejecutarse libremente hasta su terminación. Se ignorarán todas las sentencias ***trace()***.

! − aborta la ejecución del programa. En **ex.err** se descargarán el contenido de la pila y un volcado de los valores de las variables.

A medida que traza su programa, en la parte inferior de la pantalla aparecerán automáticamente los nombres de las variables y sus valores. Toda vez que una variable sea asignada, verá aparecer su nombre y el nuevo valor que recibe. Este valor se mantiene siempre actualizado. La variables privadas se borran automáticamente cuando las rutinas a las que pertenecen terminan. Cuando el área en que se muestran las variables se llena, las variables referenciadas menos recientemente se descartan para hacer lugar a las nuevas. El valor de una secuencia grande se cortará después de 80 caracteres.

Para su comodidad, los números que están en el rango de los caracteres ASCII imprimibles (32−127), se muestran junto con el caracter ASCII correspondiente. El caracter ASCII estará en un color diferente (o entre comillas si el monitor es monocromático). Esto se hace para todas las variables, ya que Euphoria no sabe en general, cuando usted está pensando en un número como un caracter ASCII o no. También verá caracteres ASCII (entre comillas) en **ex.err**. Esto puede hacer que la pantalla se vea algo "ocupada", pero la información ASCII frecuentemente es muy útil.

La pantalla de trazado adopta el mismo modo gráfico que la pantalla principal de salida. Esto hace la conmutación entre ellas más rápido y más fácil.

Cuando un programa trazado requiere de un ingreso por teclado, aparecerá la pantalla principal de salida para permitirle escribir la entrada, tal como lo haría normalmente. Esto funciona bien para una entrada ***gets()*** (que lee una línea). Cuando se llama a ***get_key()*** (que muestrea rápidamente el teclado), tiene 8 segundos para ingresar el caracter,

o de lo contrario asumirá que no hay entrada para esta llamada a ***get_key()***. Esto le permite probar los casos de entrada y no entrada para ***get_key()***.

3.1.2 El archivo de trazado

Cuando el programa llama a ***trace(3)***, se activa el trazado a un archivo. El archivo, **ctrace.out**, se creará en el directorio actual, conteniendo las últimas 500 sentencias Euphoria que el programa ejecutó. Este archivo funciona como un búfer circular que mantiene como máximo 500 sentencias. Toda vez que se alcanza el final del archivo **ctrace.out**, la próxima sentencia se escribe otra vez al comienzo. La última sentencia ejecutada está seguida siempre por "=== THE END ===". Debido a su naturaleza circular, la última sentencia ejecutada puede aparecer en cualquier parte de **ctrace.out**. La sentencia que sigue a "=== THE END ===" es la última de las 500 anteriores.

Esta forma de trazado está soportada tanto por el intérprete de la Edición Completa, como por el Traductor Euphoria a C. Es particularmente útil cuando ocurren errores a nivel de máquina que le evitan a Euphoria escribir en el archivo de diagnóstico **ex.err**. Al mirar la última sentencia ejecutada, puede conjeturar acerca de la razón por la que el programa se abortó. Quizás la última sentencia era un ***poke()*** a una posición ilegal de memoria. Quizás fue una llamada a una rutina de C. En algunos casos podría ser un bug en el intérprete o el Traductor.

El código fuente de una sentencia se escribe en **ctrace.out**, y se descarga justo *antes* que la sentencia se ejecute, por lo que la caida del programa ocurrirá *durante* la ejecución de la última sentencia que ve en **ctrace.out**.

3.2 Análisis de perfiles de ejecución (solamente en la Edición Completa)

Si especifica **with profile** (***DOS32, WIN32 o Linux***), o **with profile_time** (***solo DOS32***), el intérprete producirá un listado especial de su programa, llamado ***perfil***, al finalizar la ejecución. Este listado se guarda en el archivo **ex.pro** del directorio actual.

Hay dos tipos disponibles de análisis de perfiles: **análisis por conteo de ejecución**, y **análisis por tiempo**. El análisis por conteo de ejecución se obtiene al especificar **with profile**. El análisis por tiempo se obtiene al especificar **with profile_time**. No puede mezclar los dos tipos de análisis en una sola corrida del programa. Necesitará hacer dos corridas separadas.

Corrimos el programa de pruebas **sieve8k.ex** en **demo\bench** para ambos tipos de análisis. Los resultados están en **sieve8k.pro** (análisis por conteo de ejecución) y **sieve8k.pro2** (análisis por tiempo).

El análisis de perfiles por conteo de ejecución muestra en forma precisa la cantidad de veces que cada sentencia del programa se ejecutó. Si una sentencia no se ejecuta nunca, el campo de conteo estará en blanco.

El análisis de perfiles de ejecución por tiempo (***solo DOS32***) muestra una estimación del tiempo total consumido en ejecutar cada sentencia. Esta estimación se expresa como procentaje del tiempo consumido por analizar su programa. Si una sentencia no se muestreó nunca, el campo de procentaje estará en blanco. Si ve 0.00 significa que la sentecia se muestreó, pero no fue suficiente para obtener un puntaje de 0.01.

Solamente las sentencias compiladas con **with profile** o **with profile_time** se muestran en el listado. Normalmente especificará tanto **with profile** como **with profile_time** al inicio de su archivo **.ex** principal, por lo que puede obtener un listado completo. Abra este archivo en el editor de Euphoria editor para verlo en colores.

El análisis de perfiles de ejecución puede ayudarlo de muchas formas:

- le permite ver cuales sentencias se ejecutan pesadamente, como una pista para acelerar su programa
- le permite verificar que su programa está funcionando de la forma deseada
- puede proveerle estadísticas acerca de los datos de entrada
- le permite ver que partes del código no se probaron nunca − no deje que sus usuarios sean los primeros!

A veces querrá enfocarse en particular en una acción realizada por el programa. Por ejemplo, en el juego **Language War**, encontramos que el juego en general era suficientemente rápido, pero cuando explotaba un planeta, disparando 2500 píxeles en todas direcciones, el juego se hacía lento. Queríamos acelerar la rutina de la explosión. No nos importaba el resto del código. La solución fue llamar a ***profile(0)*** al comienzo de Language War, sólo después de **with profile_time**, para desactivar el análisis, y luego llamar a ***profile(1)*** al comienzo de la rutina de la explosión y ***profile(0)*** al final de ella. De esta forma pudimos correr el juego, creando numerosas explosiones, y registrando muchas muestras, solo para el efecto de la explosión. Si las muestras se cargaron contra otras rutinas de bajo nivel, sabríamos que esas muestras ocurrieron durante una explosión. Si hubiéramos analizado el programa entero, el cuadro no habría sido tan claro, ya que las rutinas de bajo nivel habrían sido usadas también para mover las naves, dibujar los phasors, etc. ***profile()*** puede ayudar en la misma forma cuando hace el análisis por tiempo.

3.2.1 Algunas notas adicionales sobre el análisis de perfiles por tiempo

Con cada pulso del reloj del sistema se genera una interrupción. Al especificar **with profile_time** Euphoria realizará un muestro del programa para ver cuales sentencias se ejecutan en el momento exacto que ocurre cada interrupción.

Estas interrupciones normalmente ocurren 18.2 veces por segundo, pero si llama a ***tick_rate()*** puede elegir una tasa mucho más alta y así obtener un análisis de perfiles por tiempo más exacto, ya que estará basado en más muestras. Por defecto, si no llama a ***tick_rate()***, entonces ***tick_rate(100)*** se llamará automáticamente cuando se inicie el análisis. Puede establecerlo aún más alto (hasta decir 1000) pero puede empezar a afectar el rendimiento del programa.

Cada muestra necesita 4 bytes de memoria y el espacio de búfer, normalmente, se reserva para 25000 muestras. Si necesita más de 25000 muestras, puede solicitarlo así:

```
with profile_time 100000
```

reservará espacio para 100000 muestras (poe ejemplo). Si el búfer se desborda verá una advertencia al comienzo de **ex.pro**. A 100 muestras por segundo su programa puede correr por 250 segundos antes de usar las 25000 muestra por defecto. No es factible para Euforia agrandar dinámicamente el búfer de muestra durante el manejo de una interrupción. Esa es la razón por la que tiene que especificarlo en su programa. Después de completar cada sentencia ejecutable de alto nivel, Euphoria procesará las muestras acumuladas hasta ahora, y liberará el búfer para más muestras. De este modo, el perfil puede basarse en más muestras que en el espacio reservado que tiene realmente.
Los procentajes mostrados en el margen izquierdo de **ex.pro**, se calculan dividiendo la cantidad de veces que una sentencia en particular se muestrea, por la cantidad total de muestras tomadas. Por ejemplo, si una sentencia se muestrea 50 veces de un total de 500 muestras, entonces el valor 10.0 (10%) aparecería en el margen al lado de la sentencia. Cuando se deshabilita el análisis de perfiles con ***profile(0)***, se ignoran las interrupciones, no se toman muestras y la cantidad total de muestras no se incrementa.

Tomando más muestras, puede obtener resultados más precisos. Sin embargo, una situación en la que hay que tener cuidado es el caso donde un programa se sincroniza con la inrerrupción del reloj, esperando a ***time()*** para avanzar. ***Nunca*** se tienen que muestrear las sentencias ejecutadas después del punto donde el reloj avanza, porque podrían dar una imagen muy distorsionada. Por ejemplo:

```
while time() <LIMIT do
end while
x += 1 -- esta sentencia no se muestreará nunca
```

A veces, verá un porcentaje significativo al lado de una sentencia return. Esto se debe generalmente al tiempo consumido en la desasignación del almacenamiento de variables temporales y privadas usadas dentro de la rutina. También puede aparecer un tiempo significativo en la desasignación del almacenamiento, cuando asigna un nuevo valor a una secuencia grande.

Si el intercambio a disco comienza a suceder, puede notar grandes tiempos atribuidos a las sentencias que necesitan tener acceso al archivo de intercambio, tal como sentencias que acceden a los elementos de una secuencia grande descargada al archivo de intercambio.

Parte II − Rutinas de librería

1. Introducción

Se provee una gran cantidad de rutinas de librería. Algunas están incorporadas dentro del intérprete, **ex.exe**, **exw.exe** o **exu**. Otras están escritas en Euphoria y tiene que incluir alguno de los archivos **.e** que están en **euphoria\include** para poder usarlas. Cuando este es el caso, se incluye el nombre del archivo adecuado en el apartado "Sintaxis" de la descripción. Por supuesto, un archivo de inclusión solamente se necesita incluirlo una única vez en el programa. El editor muestra en magenta a aquellas rutinas que están incorporadas dentro del intérprete, y que no requieren de ningún archivo de inclusión. Puede redefinir la definición de estas rutinas internas, haciendo su propia rutina con el mismo nombre. Obtendrá una advertencia suprimible si hace esto.

Para indicar que tipo de **objeto** se puede pasar y devolver, se usan estos prefijos:

- **x** − un objeto general (átomo o secuencia)
- **s** − una secuencia
- **a** − un átomo
- **i** − un entero
- **fn** − un entero usado como número de archivo
- **st** − una secuencia de cadena, o un átomo de un caracter

Algunas rutinas están solamente disponibles para uno, dos o tres de las plataformas. Esto está indicado con "Plataforma: **DOS32**" o "Plataforma: **WIN32**" o "Plataforma: **Linux**" en la descripción de las rutinas, y con (**DOS32**) o (**WIN32**) o (**Linux**) en algunos otros lugares.

Aparecerá un mensaje de error en tiempo de ejecución si se les pasa un argumento ilegal a cualquiera de esas rutinas.

---

2. Rutinas por área de aplicación

2.1 Tipos predefinidos

Además de declarar variables de esos tipos, también puede llamar a estas funciones como a cualquier otra función ordinaria para probar si un valor es de un cierto tipo.

- **integer** − comprueba si un objeto es un entero
- **atom** − comprueba si un objeto es un átomo
- **sequence** − comprueba si un objeto es una secuencia
- **object** − comprueba si un objeto es un objeto (siempre verdadero)

2.2 Manipulación de secuencias

- **length** − devuelve la longitud de una secuencia
- **repeat** − repite un objeto n veces para formar una secuencia de longitud n.
- **reverse** − invierte una secuencia
- **append** − agrega un nuevo elemento al final de la secuencia
- **prepend** − agrega un nuevo elemento al comienzo de la secuencia

---

2.3 Búsqueda y ordenamiento

- **compare** − compara dos objetos
- **equal** − comprueba si dos objetos son idénticos

find	−	busca un objeto dentro de una secuencia
match	−	busca una secuencia como una porción dentro de otra secuencia
sort	−	ordena los elementos de una secuencia en orden ascendente
custom_sort	−	ordena los elementos de una secuencia basándose en una función de comparación que se le suministra

2.4 Coincidencia de patrones

lower	−	convierte un átomo o secuencia a minúsculas
upper	−	convierte un átomo o secuencia a mayúsculas
wildcard_match	−	busca la coincidencia con un patrón conteniendo los símbolos ? y *
wildcard_file	−	busca la coincidencia de un nombre de archivo contra un patrón de símbolos

2.5 Matemáticas

Estas rutinas se pueden aplicar a átomos individuales o a secuencias de valores. Ver Parte I − Core Language − Operaciones sobre secuencias.

sqrt	−	calcula la raíz cuadrada de un objeto
rand	−	genera números aleatorios
sin	−	calcula el seno de un ángulo
arcsin	−	calcula el ángulo para un dado seno
cos	−	calcula el coseno de un ángulo
arccos	−	calcula el ángulo para un dado coseno
tan	−	calcula la tangente de un ángulo
arctan	−	calcula el arco tangente de un número
log	−	calcula el logaritmo natural
floor	−	redondea hacia abajo al entero más próximo
remainder	−	calcula el resto de la división de dos números
power	−	calcula un número elevado a una potencia
PI	−	valor matemático de PI (3.14159...)

2.6 Operaciones lógicas bit a bit

Estas funciones tratan a los números como colecciones de dígitos binarios, y las operaciones lógicas se efectuan sobre los bits correspondientes a la representación binaria de esos números. No hay rutinas para los desplazamientos de bits a izquierda o derecha, pero puede conseguir el mismo efecto mediante multiplicaciones y divisiones por potencias de 2.

and_bits	−	realiza la operación lógica AND bit a bit
or_bits	−	realiza la operación lógica OR bit a bit
xor_bits	−	realiza la operación lógica XOR bit a bit
not_bits	−	realiza la operación lógica NOT sobre todos los bits

2.7 Archivos y dispositivos de E/S

Para ingresar o extraer información de un archivo o dispositivo, primero tiene que abrirlo, luego usará las rutinas que están más abajo para leer o escribir en él, para finalmente cerrar ese archivo o dispositivo. **open()** le dará un número de archivo para usar como primer argumento con las otras rutinas de E/S. Ciertos archivos o dispositivos, como los archivos d texto, se abren en forma automática:

0 − entrada estándar
1 − salida estándar

2 − error estándar

A menos que los redirija en la línea de comandos, la entrada estándar viene desde el teclado, la salida y el error estándares van a la pantalla. Cuando escribe algo en pantalla, se escribe inmediatamente sin usar un búfer. Si escribe a un archivo, sus caracteres se ponen dentro de un búfer hasta que su cantidad es suficiente para que su escritura sea eficiente. Al cerrar (***close()***) o liberar (***flush()***) el archivo o dispositivo, cualquier caracter restante se escribe. La entrada desde un archivo también usa un búfer. Cuando termina el programa, cualquier archivo que aún está abierto, se cierra automáticamente.

Nota:

Si un programa (escrito en Euphoria o cualquier otro lenguaje) tiene un archivo abierto para escritura y, por ejemplo, se fuerza el reinicio de la computadora por cualquier razón, inmediatamente debería correr el **scandisk** para reparar cualquier daño que pudiese haber ocurrido en el sistema de archivos.

open	−	abre un archivo o dispositivo
close	−	cierra un archivo o dispositivo
flush	−	descarga los datos en el búfer al archivo o dispositivo
lock_file	−	bloquea un archivo o dispositivo
unlock_file	−	desbloquea un archivo o dispositivo
print	−	imprime un objeto Euphoria en una línea, con llaves y comas {,,} para mostrar la estructura
pretty_print	−	imprime un objeto Euphoria en una forma agradable, usando varias líneas y la sangría adecuada
? x	−	abreviatura para **print(1, x)**
sprint	−	devuelve un objeto Euphoria impreso como una secuencia de cadena
printf	−	impresión formateada a un archivo o dispositivo
sprintf	−	impresión formateada devuelta como una secuencia de cadena
puts	−	emite una secuencia de cadena a un archivo o dispositivo
getc	−	lee el siguiente caracter desde un archivo o dispositivo
gets	−	lee la siguiente línea desde un archivo o dispositivo
get_bytes	−	lee los siguientes n bytes desde un archivo o dispositivo
prompt_string	−	solicita al usuario que ingrese una cadena
get_key	−	verifica la tecla presionada por el usuario , sin espera
wait_key	−	espera que el usuario presione una tecla
get	−	lee la representación de cualquier objeto Euphoria desde un archivo
prompt_number	−	solicita al usuario que ingrese un número
value	−	lee la representación de cualquier objeto Euphoria desde una cadena
seek	−	se mueve a la posición de cualquier byte dentro de un archivo abierto
where	−	informa la posición del byte actual en un archivo abierto
current_dir	−	devuelve el nombre del directorio actual
chdir	−	cambia a un nuevo directorio actual
dir	−	devuelve la información completa de todos los archivos en un directorio
walk_dir	−	camina recursivamente a través de todos los archivos de un directorio
allow_break	−	habilita/deshabilita la terminación de un programa con Ctrl+C o Ctrl+Break
check_break	−	verifica si el usuario presionó Ctrl+C o Ctrl+Break

2.8 Soporte de ratón (DOS32 y Linux)

get_mouse	−	devuelve los eventos del ratón (clics, desplazamientos)
mouse_events	−	selecciona los eventos del ratón para esperarlos
mouse_pointer	−	muestra u oculta el puntero del ratón

2.9 Sistema operativo

time	–	cantidad de segundos transcurridos desde algún punto fijo en el pasado
tick_rate	–	establece la cantidad de pulsos del reloj por segundo (***DOS32***)
date	–	año, mes, día, hora, minuto, segundo, etc. actuales
command_line	–	línea de comandos usada para ejecutar este programa
getenv	–	obtiene el valor de una variable de entorno
system	–	ejecuta una línea de comandos del sistema operativo
system_exec	–	ejecuta un programa y obtiene su código de salida
abort	–	termina la ejecución
sleep	–	suspende la ejecución por un período de tiempo
platform	–	determina sobre cuál sistema operativo se está corriendo

---

2.10 Rutinas especiales dependientes de la máquina

machine_func	–	operaciones internas especializadas que devuelven un valor
machine_proc	–	operaciones internas especializadas que no devuelven ningún valor

---

2.11 Depuración

trace	–	activa o descativa dinámicamente el trazado
profile	–	activa o descativa dinámicamente el análisis de perfiles de ejecución

---

2.12 Gráficos &Sonido

Las siguientes rutinas le permiten mostrar información en pantalla. En DOS, la pantalla de la PC puede estar en uno de muchos modos gráficos. Para ver una descripción de estos modos, consulte la perte superior de **include\graphics.e.** **Hay dos tipos básicos de modo gráfico disponible:** **los modos de texto** que dividen la pantalla en líneas, donde cada línea tiene una cierta cantidad de caracteres y **los modos gráficos de píxel** que dividen la pantalla en filas de puntos o "pìxeles". Cada píxel puede ser de un color diferente. En los modos de texto, solamente se puede mostrar texto, eligiendo para cada caracter el color de frente y fondo. En los modos gráficos de pìxel, se pueden mostrar líneas, círculos, puntos y también texto. Cualquier píxel que quedase fuera de la pantalla, será recortado en forma segura.

Para ***DOS32*** hemos incluido una rutina para producir sonidos en el parlante de la PC. Para hacer sonidos más sofisticados, obtenga la librería para **Sound Blaster** desarrollada por **Jacques Deschenes**. Está disponible en el sitio web de Euphoria.

Las siguientes rutinas trabajan en todos los modos gráficos de píxel y de texto:

clear_screen	–	limpia la pantalla
position	–	establece la línea y la columna del cursor
get_position	–	devuelve la línea y la columna del cursor
graphics_mode	–	selecciona un nuevo modo de gráficos de píxel o de texto(***DOS32***)
video_config	–	devuelve los parámetros del modo actual
scroll	–	desplaza el texto hacia arriba o hacia abajo
wrap	–	controla el "wrapping" de las líneas en el borde derecho de la pantalla
text_color	–	establece el color del texto
bk_color	–	establece el color de fondo
palette	–	cambia el color para un número de color (***DOS32***)
all_palette	–	cambia el color para todos los números de color (***DOS32***)
get_all_palette	–	obtiene los valores de la paleta para todos los colores (***DOS32***)
read_bitmap	–	lee un archivo de mapa de bits (.bmp) y devuelve una paleta y una secuencia de píxeles de 2D
save_bitmap	–	crea un archivo de mapa de bits (.bmp) dada una paleta y una secuencia de píxeles de 2D

get_active_page – devuelve la página en la que se está escribiendo actualmente (***DOS32***)

set_active_page – cambia la página en la que se está escribiendo actualmente (***DOS32***)

get_display_page – devuelve la página que se está mostrando actualmente (***DOS32***)

set_display_page – cambia la página que se está mostrando actualmente (***DOS32***)

sound – produce un sonido en el parlante de la PC (***DOS32***)

Las siguientes rutinas trabajan en los modos de texto solamente:

cursor – selecciona la forma del cursor

text_rows – establece la cantidad de líneas en una pantalla de texto

get_screen_char – obtiene un caracter desde la pantalla (***DOS32, Linux***)

put_screen_char – pone uno o más caracteres en la pantalla (***DOS32, Linux***)

save_text_image – guarda una región rectangular desde una pantalla de texto (***DOS32, Linux***)

display_text_image – muestra una imagen en una pantalla de texto (***DOS32, Linux***)

Las siguientes rutinas trabajan en los modos gráficos de píxel solamente (DOS32):

pixel – establece el color de un píxel o de un conjunto de píxeles

get_pixel – lee el color de un píxel o de un conjunto de píxeles

draw_line – conecta una serie de puntos gráficos con una línea

polygon – dibuja una figura de n lados

ellipse – dibuja una elipse o un círculo

save_screen – guarda la pantalla en un archivo de mapa de bits (.bmp)

save_image – guarda una región rectangular desde una pantalla gráfica de píxeles

display_image – muestra una imagen en una pantalla gráfica de píxeles

---

2.13 Interfaz a nivel de máquina

Hemos agrupado aquí una cantidad de rutinas que puede usar para acceder a su máquina a bajo nivel. Con esta interfaz de bajo nivel, puede leer y escribir en la memoria. También puede construir sus propias de lenguaje de máquina 386+ y llamarlas.

Algunas de las rutinas listadas más abajo no son seguras, en el sentido que Euphoria no puede protegerlo si las usa de forma incorrecta. Su programa, sino su sistema, podría caerse. Bajo ***DOS32***, si referencia una mala dirección de memoria, frecuentemente será captura en forma segura por el expansor CauseWay DOS, y obtendrá un mensaje de error en pantalla, además de un volcado de información a nivel de la máquina en el archivo **cw.err**. Bajo ***WIN32***, el sistema operativo mostrará un diálogo de terminación, dándole un mensaje de disgnóstico además de información del registro. Bajo ***Linux*** típicamente obtendrá una violación de segmento.

Nota:

Para asistir a los programadores en la depuración del código involucrado en esas rutinas inseguras, hemos suministrado **safe.e**, una alternativa a **machine.e**. Si copia **euphoria\include\safe.e** en el directorio que contiene su programa y renombra **safe.e** como **machine.e** en ese directorio, su programa correrá más seguro (aunque más lento) las versiones de esas rutinas de bajo nivel. **safe.e** puede capturar muchos errores, tales como la escritura en una mala posición de memoria. Vea los comentarios en la parte superior de ***safe.e*** para leer las instrucciones de como usar y configurarlo óptimamente para su programa.

Esas rutinas de interfaz a nivel de máquina son importantes porque le permiten a los programadores Euphoria a acceder a características de bajo nivel del hardware y del sistema operativo. Para algunas aplicaciones esto es esencial.

Las rutinas de código de máquina se pueden escribir a mano u obtenerlas de la salida desensablada de un compilador de C o de algún otro lenguaje. Pete Eberlein escribió un "mini−assembler" para usarlo con Euphoria. Véalo en el Archivo. Recuerde que su código de máquina correrá en modo protegido de 32 bits. Vea un ejemplo en **demo\callmach.ex.**

peek – lee uno o más bytes desde la memoria

peek4s – lee valores de 4 bytes con signo desde la memoria

peek4u – lee valores de 4 bytes sin signo desde la memoria

poke – escribe uno o más bytes en la memoria

poke4 – escribe valores de 4 bytes en la memoria

mem_copy – copia un bloque de memoria

mem_set – establece un bloque de memoria a un valor

call – llama a una rutina en lenguaje de máquina

dos_interrupt – llama a una interrupción de software de DOS (***DOS32***)

allocate – asigna un bloque de memoria

free – desasigna un bloque de memoria

allocate_low – asigna un bloque de memoria baja (direcciones menores que 1Mb) (***DOS32***)

free_low – libera un bloque asignado con **allocate_low** (***DOS32***)

allocate_string – asigna una cadena de caracteres con el terminador 0

register_block – registra un bloque de memoria asignado externamente

unregister_block – desregistra un bloque de memoria asignado externamente

get_vector – devuelve la dirección de un vector de interrupciones (***DOS32***)

set_vector – establece la dirección de un vector de interrupciones (***DOS32***)

lock_memory – asegura que una región de memoria nunca será swapped out (***DOS32***)

int_to_bytes – convierte un entero a 4 bytes

bytes_to_int – convierte 4 bytes en un entero

int_to_bits – convierte un entero en una secuencia de bits

bits_to_int – convierte una secuencia de bits en un entero

atom_to_float64 – convierte un átomo en una secuencia de 8 bytes en formato IEEE de punto flotante de 64 bits

atom_to_float32 – convierte un átomo en una secuencia de 4 bytes en formato IEEE de punto flotante de 32 bits

float64_to_atom – convierte una secuencia de 8 bytes en formato IEEE de punto flotante de 64 bits en un átomo

float32_to_atom – convierte una secuencia de 4 bytes en formato IEEE de punto flotante de 32 bits en un átomo

set_rand – establece el generador de números aleatorios, por lo que generará una serie repetible de números aleatorios

use_vesa – fuerza el uso del estándar de gráficos VESA (***DOS32***)

crash_file – especifica el archivo para escribir el disgnóstico de error si Euphoria detecta un error en su programa

crash_message – especifica el mensaje a imprimirse si Euphoria detecta un error en su programa

---

2.14 Llamadas dinámicas

Estas rutinas le permiten llamar a procedimientos y funciones de Euphoria, usando un número entero único llamado **identificador de rutina**, en lugar de especificar el nombre de la rutina.

routine_id – obtiene un único número identificador para una rutina Euphoria

call_proc – llama a un procedimiento Euphoria usando un routine id

call_func – llama a una función Euphoria usando un routine id

---

2.15 Llamando funciones de C (WIN32 y Linux)

Vea en **platform.doc** una descripción de la programación en Euphoria para ***WIN32*** y ***Linux***.

open_dll – abre una librería de enlace dinámico de Windows (archivo .dll) o una librería compartida de Linux (archivo .so)

define_c_proc – define una función de C como VOID (sin valor de retorno), o cuyo valor ignorará el programa

define_c_func − define una función de C que devuelve un valor que usará el programa

define_c_var − obtiene la dirección de memoria de una variable de C

c_proc − llama a una función de C ignorando cualquier valor de retorno

c_func − llama a una función de C y obtiene su valor de retorno

call_back − obtiene una dirección de máquina de 32 bits para una rutina Euphoria para usarla como dirección de call−back

message_box − emite una pequeña ventana para obtener una respuesta Si/No/Cancelar de parte del usuario

free_console − borra la ventana de texto de la consola

instance − obtiene el identificador de instancia del programa actual

?

Sintaxis: ? x

Descripción: Es la forma abreviada de: **pretty_print(1, x, {})** − es decir, de imprimir el valor de una expresión en la salida estándar, con llaves y sangría para mostrar la estructura.

Ejemplo:

```
? {1, 2} + {3, 4}  -- mostrará {4, 6}
```

Ver también: ***pretty_print***, ***print***

abort

Sintaxis: abort(i)

Descripción: Aborta la ejecución de un programa. El argumento 'i' es un entero pequeño para devolver el estado al sistema operativo. Un valor 0 generalmente indica la terminación exitosa del programa. Otros valores pueden indicar distintos tipos de errores. Los programas por lotes DOS (.bat) pueden leer este valor usando la función errorlevel. Un programa Euphoria puede leer este valor usando ***system_exec()***.

Comentarios: ***abort()*** es útil cuando un programa está ejecutando llamadas subrutinas anidadas y es necesario terminar la ejecución inmediatamente, tal vez debido a la detección de un error severo.
Si no se usa ***abort()***, **ex.exe/exw.exe/exu** terminarán devolviendo normalmente el código de estado 0. Si el programa falla debido a un error en tiempo de compilación o ejecución detectado por Euphoria, entonces el código de estado de terminación será 1.

Ejemplo:

```
if x = 0 then
    puts(ERR, "No se puede dividir por 0 !!!\n")
    abort(1)
else
    z = y / x
end if
```

Ver también: ***crash_message***, ***system_exec***

all_palette

Plataforma: **DOS32**

Sintaxis: include graphics.e
all_palette(s)

Descripción: Especifica nuevas intensidades de color para un conjunto entero de colores en el modo gráfico en uso. 's' es una secuencia de la forma:
{{r,g,b}, {r,g,b}, ..., {r,g,b}}*

Cada elemento especifica una nueva intensidad de color {red, green, blue}* para el número de color correspondiente, comenzando por el número de color 0. Los valores para rojo, verde y azul tienen que estar dentro del rango 0 a 63.

* La terna {r,g,b}={red, green, blue} tal como se la conoce en inglés es {rojo, verde, azul} en español.

Comentarios: Esta rutina se ejecuta mucho más rápido que si se usa ***palette()*** para establecer las nuevas intensidades de los colores uno por uno. Este procedimiento se puede usar con ***read_bitmap()*** para mostrar rápidamente una imagen en pantalla.

Programa de ejemplo: **demo\dos32\bitmap.ex**

Ver también: *get_all_palette*, *palette*, *read_bitmap*, *video_config*, *graphics_mode*

allocate

Sintaxis: include machine.e
a = allocate(i)

Descripción: Asigna 'i' bytes contiguos de memoria. Devuelve la dirección del bloque de memoria, o devuelve 0 si no la pudo asignar. La dirección devuelta será de al menos 4 bytes alineados.

Comentarios: Al terminar de usar el bloque, debería pasar su dirección a *free()*. Esta función liberará el bloque y dejará la memoria disponible para otros usos. Euphoria no liberará o reusará su bloque hasta que se llame explícitamente a *free()*. Cuando termina su programa, el sistema operativo reclamará toda la memoria para usarla con otros programas.

Ejemplo:

```
buffer = allocate(100)
for i = 0 to 99 do
    poke(buffer+i, 0)
end for
```

Ver también: *free*, *allocate_low*, *peek*, *poke*, *call*

allocate_low

Plataforma: **DOS32**

Sintaxis: include machine.e
i2 = allocate_low(i1)

Descripción: Asigna 'i1' bytes contiguos de memoria baja, es decir memoria convencional (direcciones por debajo de 1 Mb). Devuelve la dirección del bloque de memoria, o devuelve 0 si no la pudo asignar.

Comentarios: Algunas interrupciones de software de DOS requieren que les pase una o más direcciones en registros. Esas direcciones tienen que estar dentro de las direcciones de la memoria convencional para que DOS sea capaz de leerlas o escribir en ellas.

Programa de ejemplo: **demo\dos32\dosint.ex**

Ver también: *dos_interrupt*, *free_low*, *allocate*, *peek*, *poke*

allocate_string

Sintaxis: include machine.e
a = allocate_string(s)

Descripción: Asigna espacio para una secuencia de cadena 's'. Copia 's' dentro de este espacio, junto con el caracter 0 de terminación. Este es el formato esperado por las cadenas de C. Se devuelve la dirección de memoria de la cadena. Si no hay suficiente memoria disponible, se devuelve 0.

Comentarios: Para liberar la cadena, use *free()*.

Ejemplo:

```
atom titulo

titulo = allocate_string("El Mago de Oz")
```

Programa de ejemplo: **demo\win32\window.exw**

Ver también: *allocate*, *free*

allow_break

Sintaxis: include file.e
allow_break(i)

Descripción: Cuando 'i' valga 1 (verdadero), Ctrl+C y Ctrl+Break pueden terminar el programa, cuando intenta leer la entrada desde el teclado. Si 'i' vale 0 (falso), no se podrá terminar el programa con Ctrl+C o Ctrl+Break.

Comentarios: El DOS mostrará ^C en pantalla, aún cuando no se pueda terminar el programa.
Inicialmente, el programa se puede terminar el programa en cualquier punto, donde intente leer una entrada desde el teclado. También podría terminarse mediante otra operación de E/S, dependiendo de las opciones que el usuario haya establecido en el archivo **config.sys**. (Consultar el manual de MS−DOS para el caso del comando BREAK). En algunos programas, esta terminación abrupta podría dejar cosas en un estado desordenado que podría provocar la pérdida de datos. ***allow_break(0)*** le permite evitar esta situación.

Se puede saber si el usuario presionó Ctrl+C o Ctrl+Break, llamando a ***check_break()***.

Ejemplo:

```
allow_break(0)  -- no permita que el usuario me aniquile!
```

Ver también: ***check_break***

and_bits

Sintaxis: x3 = and_bits(x1, x2)

Descripción: Ejecuta la operación lógica AND sobre los bits correspondientes en 'x1' y 'x2'. El bit en 'x3' será 1 solamente si los bits correspondientes en 'x1' y 'x2' son ambos 1.

Comentarios: Los argumentos de esta función pueden ser átomos o secuencias. Se aplican las reglas para operaciones sobre secuencias.
Los argumentos se tienen que poder representar como números de 32 bits, sean con o sin signo.

Si intenta manipular valores de 32 bits completos, debería declarar sus variables como **átomos**, en lugar de enteros. El tipo entero de Euphoria está limitado a 31 bits.

Los resultados se tratan como números con signo. Serán negativos cuando el bit de mayor orden sea 1.

Para comprender la representación binaria de un número debería mostrarlo en notación hexadecimal. Para ello, use el formato %x de ***printf()***.

Ejemplo 1:

```
a = and_bits(#0F0F0000, #12345678)
-- a es #02040000
```

Ejemplo 2:

```
a = and_bits(#FF, {#123456, #876543, #2211})
-- a es {#56, #43, #11}
```

Ejemplo 3:

```
a = and_bits(#FFFFFFFF, #FFFFFFFF)
-- a es -1
-- Observe que #FFFFFFFF es un número positivo,
-- pero el resultado en una operación lógica bit a bit se interpreta
-- como un número con signo de 32 bits, por lo tanto es negativo.
```

Ver también: ***or_bits***, ***xor_bits***, ***not_bits***, ***int_to_bits***

append

Sintaxis: s2 = append(s1, x)

Descripción: Crea una nueva secuencia idéntica a 's1', pero con 'x' agregado al final como último elemento. La longitud de 's2' será ***length(s1)*** + 1.

Comentarios: Si 'x' es un átomo, esto es equivalente a **s2 = s1 & x**, si 'x' es una secuencia, no.

El almacenamiento adicional se asigna automáticamente y en forma muy eficiente, debido a la asignación dinámica de memoria de Euphoria. El caso donde 's1' y 's2' son la misma variable (como en el Ejemplo 1) está altamente optimizado.

Ejemplo 1: Puede usar ***append()*** para hacer crecer una secuencia dinámicamente, por ejemplo:

```
sequence x

x = {}
for i = 1 to 10 do
    x = append(x, i)
end for
-- x ahora es {1,2,3,4,5,6,7,8,9,10}
```

Ejemplo 2: Se puede agregar cualquier tipo de objeto de Euphoria a una secuencia, por ejemplo:

```
sequence x, y, z

x = {"juan", "pedro"}
y = append(x, "maría")
-- y ahora es {"juan", "pedro", "maría"}

z = append(append(y, "laura"), {"bum", "bum"})
-- z ahora es {"juan", "pedro", "maría", "laura", {"bum", "bum"}}
```

Ver también: ***prepend***, concatenation operator sequence−formation operator

arccos

Sintaxis:
include misc.e
x2 = arccos(x1)

Descripción: Devuelve un ángulo cuyo coseno es igual a 'x1'.

Comentarios: El argumento 'x1' tiene que estar en el rango −1 a +1 inclusive.

Se devolverá un valor entre 0 y ***PI*** radianes.

Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

arccos() no es tan rápido como ***arctan()***.

Ejemplo:

```
s = arccos({-1,0,1})
-- s es {3.141592654, 1.570796327, 0}
```

Ver también: ***cos***, ***arcsin***, ***arctan***

arcsin

Sintaxis:
include misc.e
x2 = arcsin(x1)

Descripción: Devuelve un ángulo cuyo seno es igual a 'x1'.

Comentarios: El argumento 'x1' tiene que estar en el rango −1 a +1 inclusive.

Se devolverá un valor entre −***PI***/2 y +***PI***/2 (radianes).

Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

arcsin() no es tan rápido como ***arctan()***.

Ejemplo:

```
s = arcsin({-1,0,1})
-- s es {-1.570796327, 0, 1.570796327}
```

Ver también: ***sin***, ***arccos***, ***arctan***

arctan

Sintaxis: x2 = arctan(x1)

Descripción: Devuelve un ángulo cuyo tangente es igual a 'x1'.

Comentarios: Se devolverá un valor entre −*PI*/2 y +*PI*/2 (radianes).

Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

arctan() es más rápida que ***arcsin()*** o ***arccos()***.

Ejemplo:

```
s = arctan({1,2,3})
-- s es {0.785398, 1.10715, 1.24905}
```

Ver también: ***tan***, ***arcsin***, ***arccos***

atom

Sintaxis: i = atom(x)

Descripción: Devuelve 1 si 'x' es un átomo, 0 en caso contrario.

Comentarios: Sirve para definir el tipo átomo. También puede llamarla como una función ordinaria para determinar si un objeto es un átomo.

Ejemplo 1:

```
atom a
a = 5.99
```

Ejemplo 2:

```
object line

line = gets(0)
if atom(line) then
    puts(SCREEN, "fin del archivo\n")
end if
```

Ver también: ***sequence***, ***object***, ***integer***, atoms and sequences

atom_to_float32

Sintaxis: include machine.e
s = atom_to_float32(a1)

Descripción: Convierte un átomo de Euphoria en una secuencia de 4 valores de un byte. Esos 4 bytes contienen la representación de un número de punto flotante IEEE en formato de 32 bits.

Comentarios: Los átomos de Euphoria puden tener valores que son números de punto flotante IEEE de 64 bits, por lo tanto puede perder precisión al convertirlos a 32 bits (16 dígitos significativos contra 7). El rango del exponente es mucho más grande en el formato de 64 bits (10e308, contra 10e38), por lo que algunos átomos pueden ser demasiado grandes o demasiado pequeños para representarlos en 32 bits. En este caso, obtendrá uno de los valores especiales de 32 bits: inf o −inf (infinito o −infinito). Para evitar esto, puede usar ***atom_to_float64()***.
Los valores enteros se convertirán también al formato de punto flotante de 32 bits.

Ejemplo:

```
fn = open("numeros.dat", "wb")
puts(fn, atom_to_float32(157.82)) -- escribe 4 bytes a un archivo
```

Ver también: ***atom_to_float64***, ***float32_to_atom***

atom_to_float64

Sintaxis: include machine.e
s = atom_to_float64(a1)

Descripción: Convierte un átomo de Euphoria en una secuencia de 8 valores de un byte. Esos 8 bytes contienen la representación de un número de punto flotante IEEE en formato de 64 bits.

Comentarios: Todos los átomos Euphoria tiene valores que se pueden representar como números IEEE de punto flotante de 64 bits, por lo que puede convertir cualquier átomo al formato de 64 bits sin perder precisión.
Los valores enteros se convertirán también al formato de punto flotante de 64 bits.

Ejemplo:

```
fn = open("numeros.dat", "wb")
puts(fn, atom_to_float64(157.82)) -- escribe 8 bytes a un archivo
```

Ver también: ***atom_to_float32***, ***float64_to_atom***

bits_to_int

Sintaxis: include machine.e
a = bits_to_int(s)

Descripción: Convierte una secuencia de valores binarios (1 y 0) a un número positivo. El bit menos significativo es s[1].

Comentarios: Si imprime 's', los bits aparecerán en orden inverso, pero es conveniente para tener índices crecientes que le den acceso a los bits en orden ascendente de significación.

Ejemplo:

```
a = bits_to_int({1,1,1,0,1})
-- a es 23 (binario 10111)
```

Ver también: ***int_to_bits***, operations on sequences

bk_color

Sintaxis: include graphics.e
bk_color(i)

Descripción: Establece el color de fondo a uno de los 16 colores estándares. En los modos gráficos de píxel la pantalla entera se afectará inmediatamente. En los modos de texto cualquier nuevo caracter que imprima tendrá el nuevo color de fondo.

Comentarios: Los 16 colores estándares están definidos en **graphics.e**

En los modos gráficos de píxel, el color 0 que normalmente es NEGRO, será establecido al mismo valor {r,g,b}de paleta que el número de color 'i'.

En algunos modos gráficos de píxel, hay un color de *borde* que aparece en los extremos de la pantalla. En los modos de 256 colores, este es el 17º color en la paleta. Puede controlarlo como lo hace con cualquier otro color.

En los modos de texto, para recuperar el color de fondo original cuando termina su programa, por ejemplo 0 − NEGRO, tiene que llamar a ***bk_color(0)***. Si el cursor está en la última línea de la pantalla, tiene que imprimir algo antes de terminar su programa. Un '\n' puede ser suficiente.

Ejemplo:

```
bk_color(BLACK)
```

Ver también: ***text_color***, ***palette***

bytes_to_int

Sintaxis: include machine.e
a = bytes_to_int(s)

Descripción: Convierte una secuencia de 4 elementos de valores byte en un átomo. Los elementos de 's' están en el orden esperado por un entero de 32 bits en el 386+, es decir, el byte menos significativo primero.

Comentarios: El resultado podría ser mayor que lo que permite el tipo entero, por eso tiene que asignarlo a un **átomo**.
's' normalmente podría contener valores positivos que se pueden haber leido usando ***peek()*** desde 4 posiciones consecutivas de memoria.

Ejemplo:

```
atom int32

int32 = bytes_to_int({37,1,0,0})
-- int32 es 37 + 256*1 = 293
```

Ver también: ***int_to_bytes***, ***bits_to_int***, ***peek***, ***peek4s***, ***peek4u***, ***poke***

call

Sintaxis: call(a)

Descripción: Llama a una rutina de lenguaje de máquina que comienza en la dirección 'a'. Esta rutina tiene que ejecutar una instrucción RET #C3 para devolver el control a Euphoria. Esta rutina, también, debería guardar y restablecer cualquier registro que utilice.

Comentarios: Puede asignar un bloque de memoria para la rutina y entonces ponerlo en los bytes del código de máquina. Tendría que asignar otros bloques de memoria para datos y parámetros con los que pueda operar el código. Las direcciones de estos bloques podrían ser puestas dentro del código de máquina.

Programa de ejemplo: **demo\callmach.ex**

Ver también: ***allocate***, ***free***, ***peek***, ***poke***, ***poke4***, ***c_proc***, ***define_c_proc***

call_back

Plataforma: **WIN32, Linux, FreeBSD**

Sintaxis:
```
include dll.e
a = call_back(i)
o
a = call_back({i1, i})
```

Descripción: Obtiene una dirección de máquina para la rutina Euphoria con el **routine id** 'i'. Esta dirección puede la puede usar Windows, o una rutina C externa en una .dll de Windows **.**dll o una librería compartida de Linux/FreeBSD (.so), como una dirección de retorno ("call−back") de 32 bits para llamar a su rutina Euphoria. En Windows, puede especificar 'i1', que determina que la convención de llamada de C se puede usar para llamar a su rutina. Si 'i1' es '+', entonces su rutina trabajará con la convención de llamadas **cdecl.** Por defecto trabajará con la convención **stdcall**. En Linux y FreeBSD debería usar solamente la primera forma, ya que solo existe una convención de llamadas estándar.

Comentarios: Puede iniciar tantas funciones call−back como quiera, pero todas tienen que ser funciones Euphoria (o tipos) con 0 a 9 argumentos. Si su rutina no devuelve nada (realmente tendría que ser un procedimiento), solo devuelve 0 (digamos), y la rutina C llamada puede ignorar el resultado.
Cuando se llama a su rutina, los valores de los argumentos serán todos de 32 bits sin signo (positivos). Debería declarar a cada parámetro de de su rutina como **átomo**, a menos que quiera imponer una verificación más estrecha. Su rutina tiene que devolver un valor entero de 32 bits.

También puede usar una dirección call−back para especificar una rutina Euphoria como un manejador de excepciones en la función ***signal()*** de Linux/FreeBSD. Por ejemplo, tendría que capturar la señal SIGTERM, y hacer el cierre del sistema. Algunos servidores Web envían una señal SIGTERM a un proceso CGI que está usando demasiado tiempo de CPU.
Una rutina call−back que usa la convención cdecl y devuelve un resultado de punto flotante, podría no funcionar con exw. Esto se debe a que el compilador C de Watcom (usado para generar exw) tiene una forma no estándar de manejar los valores de retorno cdecl de punto flotante.

Programa de ejemplo: **demo\win32\window.exw, demo\linux\qsort.exu**

Ver también: ***routine_id***, **platform.doc**

c_func

Plataforma: **WIN32, Linux, FreeBSD**

Sintaxis: a = c_func(i, s)

Descripción: Llama a la función de C, o rutina de código de máquina con el **routine id** 'i'. 'i' tiene que ser un routine id válido, devuelto por ***define_c_func()***. 's' es una secuencia de valores de argumentos de longitud 'n', donde 'n' es la cantidad de argumentos necesitada por la

función. 'a' será el resultado devuelto por la función de C.

Comentarios: Si la función C no toma argumentos, entonces 's' debería ser {}.

Si pasa un valor de argumento que contiene parte fraccionaria, cundo la función de C espera un tipo entero de C, el argumento será redondeado a 0. Por ejemplo: 5.9 se pasará a 5 y −5.9 se pasará a −5.

La función C podría ser parte de una .dll creada por el Traductor Euphoria a C.

Ejemplo:

```
atom user32, hwnd, ps, hdc
integer BeginPaint

-- abre user32.dll - que contiene la función BeginPaint de C
user32 = open_dll("user32.dll")

-- la función BeginPaint de C, toma un argumento entero de C y
-- un puntero de C, y devuelve como resultado, un entero de C:
BeginPaint = define_c_func(user32, "BeginPaint",
                           {C_INT, C_POINTER}, C_INT)

-- llama a BeginPaint, pasando hwnd y ps como los argumentos,
-- en hdc se adigna el resultado:
hdc = c_func(BeginPaint, {hwnd, ps})
```

Ver también: ***c_proc***, ***define_c_func***, ***open_dll***, **platform.doc**

c_proc

Plataforma: **WIN32, Linux, FreeBSD**

Sintaxis: c_proc(i, s)

Descripción: Llama a una función C, o una rutina en código de máquina, mediante **routine id** 'i'. 'i' tiene que ser un routine id válido, devuelto por ***define_c_proc()***. 's' es una secuencia de valores de argumento de longitud 'n', donde 'n' es la cantidad de argumentos requeridos por la función.

Comentarios: Si la función C no toma argumentos, entonces 's' debería ser {}.

Si pasa un valor de argumento que contiene parte fraccionaria, cuando la función de C espera un tipo entero de C, el argumento será redondeado a 0. Por ejemplo: 5.9 se pasará a 5 y −5.9 se pasará a −5.

La función C podría ser parte de una .dll creada por el Traductor Euphoria a C.

Ejemplo:

```
atom user32, hwnd, rect
integer GetClientRect

-- open user32.dll - contiene ña función C GetClientRect
user32 = open_dll("user32.dll")

-- GetClientRect es una función C VOID que toma un entero de C
-- y un puntero de C como argumentos:
GetClientRect = define_c_proc(user32, "GetClientRect",
                              {C_INT, C_POINTER})

-- pasa hwnd y rect como los argumentos
c_proc(GetClientRect, {hwnd, rect})
```

Ver también: ***c_func***, ***call***, ***define_c_proc***, ***open_dll***, **platform.doc**

call_func

Sintaxis: x = call_func(i, s)

Descripción: Llama a una función Euphoria definida por el usuario, mediante **routine id** 'i'. 'i' tiene que ser un routine id válido, devuelto por ***routine_id()***. 's' tiene que ser una secuencia de valores de argumento de longitud 'n', donde 'n' es la cantidad de argumentos requeridos por la función 'i'. 'x' será el resultado devuelto por la función 'i'.

Comentarios: Si la función 'i' no toma argumentos, entonces 's' debería ser {}.

Programa de ejemplo: **demo\csort.ex**

Ver también: ***call_proc***, ***routine_id***

call_proc

Sintaxis: call_proc(i, s)

Descripción: Llama a un procedimiento Euphoria definida por el usuario mediante **routine id** 'i'. 'i' tiene que ser un routine id válido, devuelto por ***routine_id()***. 's' tiene que ser una secuencia de valores de argumento de longitud 'n', donde 'n' es la cantidad de argumentos requeridos por el procedimiento 'i'.

Comentarios: Si el procedimiento 'i' no toma argumentos, entonces 's' debería ser {}.

Ejemplo:

```
global integer foo_id

procedure x()
    call_proc(foo_id, {1, "Hola Mundo\n"})
end procedure

procedure foo(integer a, sequence s)
    puts(a, s)
end procedure

foo_id = routine_id("foo")

x()
```

Ver también: ***call_func***, ***routine_id***

chdir

Sintaxis: include file.e
i = chdir(s)

Descripción: Establece el directorio actual a la ruta dada por la secuencia 's'. 's' tiene que indicar un directorio que exista en el sistema. Si resulta bien, ***chdir()*** devuelve 1. Si falla, ***chdir()*** devuelve 0.

Comentarios: Al establecer el directorio actual, puede referirse a los archivos en este directorio usando solamente su nombre de archivo.
La función ***current_dir()*** devolverá el nombre del directorio actual.

En DOS32 y WIN32 el directorio actual es una propiedad global compartida por todos los procesos que corren bajo un shell. En Linux/FreeBSD, un subproceso puede cambiar el directorio actual por sí mismo, pero esto no afecta al directorio actual de su proceso padre.

Ejemplo:

```
if chdir("c:\\euphoria") then
    f = open("leeme.doc", "r")
else
    puts(1, "Error: No euphoria directory?\n")
end if
```

Ver también: ***current_dir***

check_break

Sintaxis: include file.e
i = check_break()

Descripción: Devuelve la cantidad de veces que se presionaron Ctrl+C or Ctrl+Break desde la última llamada a ***check_break()***, o desde el comienzo del programa si esta es la primera llamada.

Comentarios: Esta función es útil después de haber llamado a ***allow_break(0)*** que evita que Ctrl+C o Ctrl+Break terminen el programa. Puede usar ***check_break()*** para determinar si el usuario presionó alguna de esas teclas; entonces realizar alguna acción, como un cierre "elegante" de su programa.
Cuando lea el teclado, ni Ctrl+C, ni Ctrl+Break se devolverán como caracteres ingresados. Solamente se pueden detectar llamando a ***check_break()***.

Ejemplo:

```
k = get_key()
if check_break() then
    temp = graphics_mode(-1)
    puts(1, "Cerrando la aplicación...")
    save_all_user_data()
    abort(1)
```

```
end if
```

Ver también: ***allow_break***, ***get_key***

clear_screen

Sintaxis: clear_screen()

Descripción: Limpia la pantalla, usando el color de fondo actual (puede establecerse con ***bk_color()***).

Comentarios: Funciona en todos los modos de texto y gráficos de píxel.

Ver también: ***bk_color***, ***graphics_mode***

close

Sintaxis: close(fn)

Descripción: Cierra un archivo o dispositivo y descarga cualquier caracter que esté en el búfer.

Comentarios: Cualquier archivo abierto se cerrará automáticamente al terminar el programa.

Ver también: ***open***, ***flush***

command_line

Sintaxis: s = command_line()

Descripción: Devuelve una secuencia de cadena, donde cada cadena es una palabra de la línea de comandos que inició su programa. La primera palabra será tanto la ruta al ejecutable de Euphoria (**ex.exe**, **exw.exe** o **exu)**, como la de su archivo **ejecutable enlazado**. La siguiente palabra es el nombre de su archivo principal Euphoria, o (nuevamente) la ruta a su archivo ejecutable enlazado. Después de ellas, vendrá cualquier palabra adicional escrita por el usuario. Puede usar estas palabras en su programa.

Comentarios: El intérprete Euphoria en sí mismo no usa ninguna opción de la línea de comandos. Su programa puede usar libremente cualquier opción.
El usuario puede poner comillas alrededor de una serie de palabras para convertirlas en un argumento único.

Si **enlaza** su programa, encontrará que todos los argumentos de la línea de comandos son iguales, excepto los dos primeros, aún cuando el usuario no escriba "ex" en la línea de comandos (ver los ejemplos de más abajo).

Ejemplo 1:

```
-- El usuario escribe:  ex miprog mifich.dat 12345 "fin"

cmd = command_line()

-- cmd será:
      {"C:\EUPHORIA\BIN\EX.EXE",
       "miprog",
       "mifich.dat",
       "12345",
       "fin"}
```

Ejemplo 2:

```
-- Su programa se enlaza con el nombre "miprog.exe"
-- y se almacena en el directorio c:\misarchivos
-- El usuario escribe: miprog mifich.dat 12345 "fin"

cmd = command_line()

-- cmd será:
       {"C:\MISFICHEROS\MIPROG.EXE",
        "C:\MISFICHEROS\MIPROG.EXE", -- relleno
        "mifich.dat",
        "12345",
        "fin"
        }

-- Vea que todos los argumentos son los mismos que los del
-- del ejemplo 1 excepto por los primeros dos.
-- El segundo argumento es siempre igual al primero y se inserta
```

```
-- para mantener la numeración de los argumentos siguientes,
-- sea que su programa esté enlazado como un .exe o no.
```

Ver también: ***getenv***

compare

Sintaxis: i = compare(x1, x2)

Descripción: Devuelve 0 si los objetos 'x1' y 'x2' son idénticos, 1 si 'x1' es mayor que 'x2' o, −1 si 'x1' es menor que 'x2'. Se considera que los átomos son menores que las secuencias. Las secuencias se comparan "alfabéticamente" comenzando por el primer elemento hasta que se encuentra una diferencia.

Ejemplo 1:

```
x = compare({1,2,{3,{4}},5}, {2-1,1+1,{3,{4}},6-1})
-- idénticos, x es 0
```

Ejemplo 2:

```
if compare("ABC", "ABCD") <0 then   -- -1
    -- será verdadero: ABC es "menor" porque es más corta
end if
```

Ejemplo 3:

```
x = compare({12345, 99999, -1, 700, 2},
            {12345, 99999, -1, 699, 3, 0})
-- x será 1 porque 700 > 699
```

Ejemplo 4:

```
x = compare('a', "a")
-- x será -1 porque 'a' es un átomo
-- mientras que "a" es una secuencia
```

Ver también: ***equal***, operadores relacionales, operaciones sobre secuencias

cos

Sintaxis: x2 = cos(x1)

Descripción: Devuelve el coseno de 'x1', donde 'x1' está expresado en radianes.

Comentarios: Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

Ejemplo:

```
x = cos({.5, .6, .7})

-- x es {0.8775826, 0.8253356, 0.7648422}
```

Ver también: ***sin***, ***tan***, ***log***, ***sqrt***

crash_file

Sintaxis: include machine.e
crash_file(s)

Descripción: Especifica un nombre de archivo 's', para mantener el diagnóstico de error, en caso que Euphoria deba detener el programa debido a un error en tiempo de compilación o de ejecución.

Comentarios: Normalmente, Euphoria muestra en pantalla un mensaje de diagnóstico como "error de sintaxis" o "división por cero", tanto como realiza el volcado de la información de depuración en el archivo **ex.err** dentro del directorio actual. Llamando a ***crash_file()***, puede controlar en qué directorio y archivo se escribe la información de depuración.
's' puede estar vacío, es decir, "". En este caso, la información de depuración o los mensajes de diagnóstico no se mostrarán por pantalla, ni se escribirán en un archivo. También, 's' podría ser "NUL" o "/dev/null", en este caso se verán los mensajes de diagnóstico por pantalla, pero la información del archivo ***ex.err*** será descartada.

Puede llamar a ***crash_file()*** tantas veces como quiera en diferentes partes del programa. El archivo especificado en la última llamada será el que se use.

Ejemplo:

```
crash_file("\\tmp\\mibug")
```

Ver también: ***abort***, ***crash_message***, depuración y análisis de perfiles de ejecución

crash_message

Sintaxis: include machine.e
crash_message(s)

Descripción: Especifica una cadena 's', para imprimirse en pantalla cuando Euphoria tiene que detener su programa, debido a un error en tiempo de compilación o de ejecución.

Comentarios: Normalmente, Euphoria imprime un mensaje de diagnóstico como "error de sintaxis" o "división por cero" en pantalla, del mismo modo que vuelca la información de depuración en **ex.err**. Los mensajes de error de Euphoria no serán significativos para los usuarios, a menos que sean programadores Euphoria. Al llamar a ***crash_message()***, puede controlar el mensaje que aparecerá en la pantalla. La información de depuración se almacenará, igualmente, en **ex.err**. Usted no pierde ninguna información por usar esta función.
's' puede contener '\n' (caracteres de nueva línea) por lo que su mensaje puede ocupar varias línea en la pantalla. Euphoria limpiará y cambiará la línea superior de la pantalla a modo de texto antes que imprima su mensaje.

Puede llamar a ***crash_message()*** tantas veces como quiera desde diferentes partes de su programa. El mensaje especificado por la última llamada será el que se muestre.

Ejemplo:

```
crash_message("Ocurrió un error inesperado!\n"
              "Por favor, contacte a juan_perez@auxilio.com\n"
              "No borre el archivo \"ex.err\".\n")
```

Ver también: ***abort***, ***crash_file***, depuración y análisis de perfiles de ejecución

current_dir

Sintaxis: include file.e
s = current_dir()

Descripción: Devuelve el nombre del directorio actual.

Ejemplo:

```
sequence s
s = current_dir()
-- s podría valer "C:\EUPHORIA\DOC" si se estuviera en ese directorio.
```

Ver también: ***dir***, ***chdir***, ***getenv***

cursor

Sintaxis: include graphics.e
cursor(i)

Descripción: Selecciona un estilo de cursor. **graphics.e** contiene:

```
global constant NO_CURSOR = #2000,
         UNDERLINE_CURSOR = #0607,
    THICK_UNDERLINE_CURSOR = #0507,
         HALF_BLOCK_CURSOR = #0407,
              BLOCK_CURSOR = #0007
```

El segundo y cuarto dígitos hexadecimales (desde la izquierda) determinan la fila superior e infoerior de píxeles en el cursor. El primer dígito controla si el cursor es visible o no. Por ejemplo, #0407 activa las filas 4 a la 7.

Comentarios: En los modos gráficos de píxel el cursor no se muestra.

Ejemplo:

```
cursor(BLOCK_CURSOR)
```

Ver también: *graphics_mode*, *text_rows*

custom_sort

Sintaxis: include sort.e
s2 = custom_sort(i, s1)

Descripción: Ordena los elementos de la secuencia 's1', usando la función de comparación de **routine id** 'i'.

Comentarios: Su función de comparación tiene que ser una función de dos argumentos, similar a *compare()* de Euphoria. *compare()* compara dos objetos y devuelve −1, 0 o +1.

Programa de ejemplo: **demo\csort.ex**

Ver también: *sort*, *compare*, *routine_id*

date

Sintaxis: s = date()

Descripción: Devuelve una secuencia con la siguiente información:

```
             {  año,  -- desde 1900
                mes,  -- Enero = 1
                día,  -- día del mes, comenzando en 1
               hora,  -- 0 a 23
            minutos,  -- 0 a 59
           segundos,  -- 0 a 59
  día de la semana,  -- Domingo = 1
       día del año}  -- 1 de Enero = 1
```

Ejemplo:

```
now = date()
-- now vale: {95,3,24,23,47,38,6,83}
-- es decir, Viernes 24 de Marzo de 1995 a las 11:47:38pm, día 83 del año
```

Comentarios: El valor devuelto para el año es la cantidad de años desde el 1900 (***no*** son los últimos dos dígitos del año). Por ejemplo, el año 2000 es 100, el 2001 es 101, etc.

Ver también: *time*

define_c_func

Sintaxis: include dll.e
i1 = define_c_func(x1, x2, s1, i2)

Descripción: Define las características tanto de una función de C, como de una rutina de código de máquina que devuelve un valor. Se devolverá un entero pequeño, 'i1' conocido como **routine id.** Use este routine id como el primer argumento de *c_func()* cuando desea llamar a la función desde Euphoria.
Al definir una función de C, 'x1' es la dirección de la librería que contiene la función de C, mientras que 'x2' es el nombre de la función de C. 'x1' es un valor devuelto por *open_dll()*. Si no se puede encontrar a la función de C, routine id devolverá −1. En Windows, puede agregar un caracter '+' como prefijo del nombre de la función. Esto le indica a Euphoria que la función usa la convención de llamadas **cdecl.** Por defecto, Euphoria asume que la rutina de C acepta la convención **stdcall.**

Al definir una rutina de código de máquina, 'x1' tiene que ser la secuencia vacía, "" o {}, y 'x2' indica la dirección de la rutina de código de máquina. Puede poner los bytes del código de máquina en un bloque de memoria reservada usando *allocate()*. En Windows, se espera normalmente que la rutina de código de máquina siga la convención de llamadas **stdcall,** pero si desea usar la convención **cdecl** en lugar de aquella, puede escribir **{'+', dirección}** en lugar de **dirección** para 'x2'.

's1' es una lista de los tipos de parámetros de la función. 'i2' es el tipo devuelto por la función. En **dll.e** hay una lista de los tipos de C, y estos se pueden usar para definir también los parámetros del código de máquina:

```
global constant C_CHAR = #01000001,
                C_UCHAR = #02000001,
                C_SHORT = #01000002,
               C_USHORT = #02000002,
                  C_INT = #01000004,
                 C_UINT = #02000004,
```

```
  C_LONG = C_INT,
 C_ULONG = C_UINT,
C_POINTER = C_ULONG,
 C_FLOAT = #03000004,
C_DOUBLE = #03000008
```

La función de C que usted define, podría ser una creada por el Traductor Euphoria a C. En este caso puede pasarle datos a Euphoria y recibir de regreso datos Euphoria. En **dll.e** hay una lista de tipos Euphoria:

```
global constant
     E_INTEGER = #06000004,
     E_ATOM    = #07000004,
     E_SEQUENCE= #08000004,
     E_OBJECT  = #09000004
```

Comentarios: Puede pasar o devolver cualquier tipo entero o puntero de C. También puede pasar un átomo Euphoria como un double o float de C, y obtener un double o float de C devuelto, como un átomo Euphoria.
Los tipos de los parámetros que usan 4 bytes o menos se pasan de la misma forma, por lo que no es necesario ser exacto. Sin embargo, la distinción entre 'con signo' o 'sin signo' puede ser importante cuando especifica el tipo de retorno de una función.

Actualmente, no hay forma de pasar una estructura de C por valor u obtener una estructura de C como resultado de retorno. Solamente puede pasar un puntero a una estructura y obtener un puntero a la estructura como resultado.

Si no está interesado en usar el valor devuelto por la función C, debería definirla con ***define_c_proc()*** y llamarla con ***c_proc()***.

Si usa exw para llamar una rutina cdecl de C que devuelve un valor de punto flotante, esto podría no funcionar. Esto se debe a qye el compilador de C de Watcom (usado para generar exw) tiene una forma no estándar de manejar los valores de retorno cdecl de punto flotante.

Si usa ***c_func()*** para pasar valores de punto flotante a rutinas de código de máquina, la llamada resultará más rápida que si la hace mediante ***call()***, ya que no tiene que usar ***atom_to_float64()*** y ***poke()*** para obtener los valores de punto flotante en memoria.

ex.exe usa llamadas a rutinas de punto flotante de WATCOM (que luego usa instrucciones de punto flotante por hardware si está disponible), por lo que los valores de punto flotante, generalmente, se pasan y devuelven en pares de registros enteros, en lugar de registros de punto flotante. Tendría que desensamblar algún código Watcom para ver como trabaja.

Ejemplo:

```
atom user32
integer LoadIcon

-- abrir user32.dll - contiene la función C LoadIconA
user32 = open_dll("user32.dll")

-- Toma como argumentos un puntero de C y un entero de C.
-- Devuelve un entero de C como resultado.
LoadIcon = define_c_func(user32, "LoadIconA",
                         {C_POINTER, C_INT}, C_INT)
-- Usamos "LoadIconA" aquí porque sabemos que LoadIconA
-- necesita la convención stdcall, como lo hacen todas
-- las rutinas de .dll estándares en la WIN32 API.
-- Para especificar la convención cdecl, deberíamos haber usado "+LoadIconA".

if LoadIcon = -1 then
    puts(1, "No se pudo encontrar a LoadIconA!\n")
end if
```

Ver también: ***euphoria\demo\callmach.ex***, ***c_func***, ***define_c_proc***, ***c_proc***, ***open_dll***, **platform.doc**

define_c_proc

Sintaxis: include dll.e
i1 = define_c_proc(x1, x2, s1)

Descripción: Define las características tanto de una función de C, como de una rutina de código de máquina que desea llamar como procedimiento desde su programa Euphoria. Se devolverá un entero pequeño, 'i1' conocido como **routine id.** Use este routine id como el primer argumento de ***c_proc()*** cuando desea llamar a la rutina desde Euphoria.

Al definir una función de C, 'x1' es la dirección de la librería que contiene la función de C, mientras que 'x2' es el nombre de la función de C. 'x1' es un valor devuelto por ***open_dll()***. Si no se puede

encontrar a la función de C, routine id devolverá −1. En Windows, puede agregar un caracter '+' como prefijo del nombre de la función. Esto le indica a Euphoria que la función usa la convención de llamadas **cdecl.** Por defecto, Euphoria asume que la rutina de C acepta la convención **stdcall.**

Al definir una rutina de código de máquina, 'x1' tiene que ser la secuencia vacía, "" o {}, y 'x2' indica la dirección de la rutina de código de máquina. Puede poner los bytes del código de máquina en un bloque de memoria reservada usando ***allocate()***. En Windows, se espera normalmente que la rutina de código de máquina siga la convención de llamadas **stdcall,** pero si desea usar la convención **cdecl** en lugar de aquella, puede escribir **{'+', dirección}** en lugar de **dirección** para 'x2'.

's1' es una lista de los tipos de parámetros de la función. 'i2' es el tipo devuelto por la función. En **dll.e** hay una lista de los tipos de C, y estos se pueden usar para definir también los parámetros del código de máquina (como se muestra más arriba).

La función de C que usted define, podría ser una creada por el Traductor Euphoria a C. En este caso puede pasarle datos a Euphoria y recibir de regreso datos Euphoria. En **dll.e** hay una lista de tipos Euphoria, que se muestra más arriba.

Comentarios: Puede pasar cualquier tipo entero o puntero de C. También puede pasar un átomo Euphoria como un double o float de C.
Los tipos de los parámetros que usan 4 bytes o menos se pasan de la misma forma, por lo que no es necesario ser exacto.

Actualmente, no hay forma de pasar una estructura de C por valor. Solamente puede pasar un puntero a la estructura.

La función de C puede devolver un valor, pero éste será ignorado. Si quiere usar el valor devuelto por la función de C, tiene que definirla con ***define_c_func()*** y llamarla con ***c_func()***.

Ejemplo:

```
atom user32
integer ShowWindow

-- abrir user32.dll - contiene la función de C ShowWindow
user32 = open_dll("user32.dll")

-- Tiene 2 parámetros que son enteros de C.
ShowWindow = define_c_proc(user32, "ShowWindow", {C_INT, C_INT})
-- Si ShowWindow usase la convención cdecl,
-- tendríamos que escribir "+ShowWindow" aquí

if ShowWindow = -1 then
    puts(1, "No se encuentra ShowWindow!\n")
end if
```

Ver también: ***c_proc***, ***define_c_func***, ***c_func***, ***open_dll***, **platform.doc**

define_c_var

Plataforma: **WIN32, Linux, FreeBSD**

Sintaxis: include dll.e
a1 = define_c_var(a2, s)

Descripción: 'a2' es la dirección de una librería compartida de Linux o FreeBSD, o de una .dll de Windows como las devueltas por ***open_dll()***. 's' es el nombre de una variable global de C definida dentro de la librería. 'a1' será la dirección de memoria de la variable 's'.

Comentarios: Una vez que tiene la dirección de la variable de C, y conoce su tipo, puede usar ***peek()*** y ***poke()*** para leer o escribir el valor de la variable.

Programa de ejemplo: ***euphoria/demo/linux/mylib.exu***

Ver también: ***c_proc***, ***define_c_func***, ***c_func***, ***open_dll***, **platform.doc**

dir

Sintaxis: include file.e
x = dir(st)

Descripción: Devuelve información de directorio para el archivo o directorio indicado en 'st'. Si no existe un archivo o directorio con este nombre, entoces se devuelve −1. En Windows y DOS 'st' puede contener los comodines * y ? para seleccionar varios archivos.

Esta información es similar a lo que obtiene con el comando DIR del DOS. Se devuelve una secuencia, donde cada elemento es una secuencia que describe un archivo o subdirectorio.

Si 'st' indica un **directorio**, entonces 'x' puede tener entradas para "." y "..", igual que con el comando DIR del DOS. Si 'st' indica un **archivo**, entonces 'x' tendrá solo una entrada, es decir, ***length(x)*** será 1. Si 'st' contiene comodines, puede tener varias entradas.

Cada entrada contiene el nombre, atributos y tamaño del archivo, así como el año, mes, día, hora, minutos y segundos de la última modificación. Puede referirse a los elementos de una entrada con las siguiente constantes definidas en **file.e**:

```
global constant D_NAME = 1,
          D_ATTRIBUTES = 2,
                D_SIZE = 3,

                D_YEAR = 4,
               D_MONTH = 5,
                 D_DAY = 6,

                D_HOUR = 7,
              D_MINUTE = 8,
              D_SECOND = 9
```

El elemento atributo es una secuencia de cadena que contiene caracteres con valor:

```
'd' -- directorio
'r' -- archivo de solo lectura
'h' -- archivo oculto
's' -- archivo del sistema
'v' -- nombre del volumen
'a' -- archivo archivo
```

Un archivo normal sin atributos especiales podría tener una cadena vacía en este campo.

Comentarios: El directorio de nivel superior, por ejemplo, c:\ no tiene las entradas "." o "..".

Frecuentemente, se usa esta función para probar la existencia de un archivo o directorio.

Bajo **WIN32**, 'st' puede tener en cualquier parte de la ruta, un nombre largo de archivo o directorio.

Bajo **Linux/FreeBSD**, el único atributo disponible es 'd'.

DOS32: El nombre de archivo devuelto en D_NAME será un nombre DOS estándar 8.3 (Vea una solución mejor en el Archivo de la página web).

WIN32: El nombre de archivo devuelto en D_NAME será un nombre largo.

Ejemplo:

```
d = dir(current_dir())

-- d podría tener:
  {
    {".",    "d",     0  1994, 1, 18,  9, 30, 02},
    {"..",   "d",     0  1994, 1, 18,  9, 20, 14},
    {"fred", "ra", 2350, 1994, 1, 22, 17, 22, 40},
    {"sub",  "d" ,    0, 1993, 9, 20,  8, 50, 12}
  }

d[3][D_NAME] debería ser "fred"
```

Programas de ejemplo: **bin\search.ex**, **bin\install.ex**

Ver también: ***wildcard_file***, ***current_dir***, ***open***

display_image

Plataforma: **DOS32**

Sintaxis: include image.e
display_image(s1, s2)

Descripción: Muestra en el punto 's1' de una pantalla de gráficos de píxel la secuencia 2D de píxeles contenida en 's2'. 's1' es una secuencia de 2 elementos {x, y}. 's2' es una secuencia de secuencias, donde cada secuencia es una fila horizontal de colores de píxel a mostrar. El primer píxel de la primera secuencia se muestra en s1. Es el píxel superior izquierdo. Los demás píxeles aparecen a la derecha o debajo de este punto.

Comentarios: 's2' podría ser el resultado de una llamada previa a ***save_image()***, o ***read_bitmap()***, o algo creado por usted.

Las secuencias (filas) de la imagen no tienen que ser todas de la misma longitud.

Ejemplo:

```
display_image({20,30}, {{7,5,9,4,8},
                        {2,4,1,2},
                        {1,0,1,0,4,6,1},
                        {5,5,5,5,5,5}})
-- Esto mostrará una imagen pequeña conteniendo 4 filas de píxeles.
-- El primer píxel (7) de la fila superior estará en {20,30}.
-- La fila superior contiene 5 píxeles, la última tiene 6,
-- terminando en {25,33}.
```

Programa de ejemplo: **demo\dos32\bitmap.ex**

Ver también: ***save_image***, ***read_bitmap***, ***display_text_image***

display_text_image

Plataforma: **DOS32, Linux, FreeBSD**

Sintaxis:
include image.e
display_text_image(s1, s2)

Descripción: Muestra la secuencia 2D de caracteres y atributos contenida en 's2', en la línea s1[1]. columna s1[2]. 's2' es una secuencia de secuencias, donde cada secuencia es una cadena de caracteres y atributos a mostrar. El caracter superior izquierdo se muestra en 's1'. Los otros caracteres aparecen a la derecha o debajo de esta posición. Los atributos indican los colores de frente y fondo del caracter precedente. En DOS32, el atributo debería constar del color de frente más 16 veces el color de fondo.

Comentarios: Normalmente, 's2' podría ser el resultado de una llamada previa a ***save_text_image()***, aunque podría construirla usted mismo.
Esta rutina solamente funciona en los modos de texto.

Podría usar ***save_text_image()***/***display_text_image()*** en una interfaz gráfica de usuario en modo de texto, para permitir que aparezcan y desaparezcan cajas de diálogo emergentes y menúes desplegables sin perder lo que estaba previamente en pantalla.

Las secuencias de la imagen de texto no tienen que tener todas la misma longitud..

Ejemplo:

```
clear_screen()
display_text_image({1,1}, {{'A', WHITE, 'B', GREEN},
                           {'C', RED+16*WHITE},
                           {'D', BLUE}})
-- muestra:
     AB
     C
     D
-- en la esquina superior izquierda de la pantalla.
-- 'A' será blanca sobre fondo negro,
-- 'B' será verde sobre negro,
-- 'C' será rojo sobre blanco, y
-- 'D' será azul sobre negro.
```

Ver también: ***save_text_image***, ***display_image***, ***put_screen_char***

dos_interrupt

Plataforma: **DOS32**

Sintaxis:
include machine.e
s2 = dos_interrupt(i, s1)

Descripción: Llama a la interrupción de software de DOS número 'i'. 's1' es una secuencia de 10 elementos con los valores de los registros de 16 bits, para usarlos como entrada a la rutina de interrupción. 's2' es una secuencia similar de 10 elementos que contiene los valores de salida de los registros después que la llamada regresa. **machine.e** tiene la siguiente declaración que muestra el orden de los valores de los registros en las secuencias de entrada y salida.

```
global constant REG_DI = 1,
                REG_SI = 2,
                REG_BP = 3,
                REG_BX = 4,
                REG_DX = 5,
                REG_CX = 6,
```

```
REG_AX = 7,
REG_FLAGS = 8,
REG_ES = 9,
REG_DS = 10
```

Comentarios: Los valores de los registros devueltos en 's2' son siempre valores positivos entre 0 y #FFFF (65535).

El valor 'flags' en s1[REG_FLAGS] se ignora en la salida. En la salida, el bit menos significativo de s2[REG_FLAGS] tiene el indicador de arrastre (carry flag), que normalmente indica falla si vale 1.

Ciertas interrupciones necesitan que se les suministre direcciones de bloques de memoria. Estas direcciones tienen que ser direcciones convencionales de la memoria baja. Puede asignar o desasignar memoria baja usando ***allocate_low()*** y ***free_low()***.

Con las interrupciones por software de DOS puede realizar una amplia variedad de operaciones especializadas, desde formatear un disquete a reiniciar la computadora. La documentación de estas interrupciones la puede encontrar en manuales técnicos como *"La Biblia del programador de PC"*de Peter Norton o descargar desde la web *"Lista de interrupciones"* de Ralf Brown:

http://www.cs.cmu.edu/afs/cs.cmu.edu/user/ralf/pub/WWW/files.html

Ejemplo:

```
sequence registers

registers = repeat(0, 10)  -- no se necesita dar valor a los registros

-- llama a la interupcion 5 de DOS: Imprimir pantalla
registers = dos_interrupt(#5, registers)
```

Programa de ejemplo: **demo\dos32\dosint.ex**

Ver también: ***allocate_low***, ***free_low***

draw_line

Plataforma: **DOS32**

Sintaxis: include graphics.e
draw_line(i, s)

Descripción: Dibuja una línea en una pantalla de gráficos de píxel contectando dos o más puntos en 's', usando el color 'i'.

Ejemplo:

```
draw_line(WHITE, {{100, 100}, {200, 200}, {900, 700}})

-- Conecta los tres puntos de la secuencia usando
-- una línea blanca, es decir, dibuja una línea desde {100, 100} a
-- {200, 200} y otra desde {200, 200} a {900, 700}.
```

Ver también: ***polygon***, ***ellipse***, ***pixel***

ellipse

Plataforma: DOS32

Sintaxis:
include graphics.e
ellipse(i1, i2, s1, s2)

Descripción: Dibuja una elipse con color 'i1' en una pantalla de gráficos de píxel. La elipse cabrá cuidadosamente dentro del rectángulo definido por los puntos diagonales 's1' {x1, y1} y 's2' {x2, y2}. Si el rectángulo es una cuadrado, entonces la elipse será un círculo. Rellena la elipse cuando 'i2' vale 1. No la rellena cuando 'i2' vale 0.

Ejemplo:

```
ellipse(MAGENTA, 0, {10, 10}, {20, 20})

-- Debería hacer un círculo magenta bien ajustado
-- dentro del cuadrado:
--        {10, 10}, {10, 20}, {20, 20}, {20, 10}.
```

Programa de ejemplo: **demo\dos32\sb.ex**

Ver también: ***polygon***, ***draw_line***

equal

Sintaxis: i = equal(x1, x2)

Descripción: Compara dos objetos Euphoria para determinar si son iguales. Devuelve 1 (verdadero) si son iguales, 0 si son distintos.

Comentarios: Es equivalente a la expresión: ***compare(x1, x2) = 0*****.**

Ejemplo 1:

```
if equal(PI, 3.14) then
    puts(1, "déme un mejor valor de PI!\n")
end if
```

Ejemplo 2:

```
if equal(name, "Jorge") or equal(name, "JORGE") then
    puts(1, "el nombre es Jorge\n")
end if
```

Ver también: ***compare***, operador equals (=)

find

Sintaxis: i = find(x, s)

Descripción: Busca 'x' como elemento de 's'. Si lo encuentra, devuelve el índice del primer elemento de 's' que coincida, sino devuelve 0.

Ejemplo 1:

```
ubicacion = find(11, {5, 8, 11, 2, 3})
-- ubicacion se establece en 3
```

Ejemplo 2:

```
nombres = {"Alfredo", "Pedro", "Juan", "María", ""}
ubicacion = find("María", nombres)
-- ubicacion se establece en 4
```

Ver también: ***match***, ***compare***

float32_to_atom

Sintaxis:
include machine.e
a1 = float32_to_atom(s)

Descripción: Convierte una secuencia de 4 bytes en un átomo. Esos 4 bytes tienen que contener el formato IEEE de punto flotante de 32 bits.

Comentarios: Cualquier número IEEE de punto flotante de 32 bits se puede convertir en un átomo.

Ejemplo:

```
f = repeat(0, 4)
fn = open("numeros.dat", "rb") -- lee en binario
f[1] = getc(fn)
f[2] = getc(fn)
f[3] = getc(fn)
f[4] = getc(fn)
a = float32_to_atom(f)
```

Ver también: ***float64_to_atom***, ***atom_to_float32***

float64_to_atom

Sintaxis: include machine.e
a1 = float64_to_atom(s)

Descripción: Convierte una secuencia de 8 bytes en un átomo. Esos 8 bytes tienen que contener el formato IEEE de punto flotante de 64 bits.

Comentarios: Cualquier número IEEE de punto flotante de 64 bits se puede convertir en un átomo.

Ejemplo:

```
f = repeat(0, 8)
fn = open("numeros.dat", "rb")  -- lee en binario
for i = 1 to 8 do
    f[i] = getc(fn)
end for
a = float64_to_atom(f)
```

Ver también: ***float32_to_atom***, ***atom_to_float64***

floor

Sintaxis: x2 = floor(x1)

Descripción: Devuelve el mayor entero menor o igual que 'x1' (Redenodea hacia abajo un entero).

Comentarios: Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

Ejemplo:

```
y = floor({0.5, -1.6, 9.99, 100})
-- y es {0, -2, 9, 100}
```

Ver también: ***remainder***

flush

Sintaxis: include file.e
flush(fn)

Descripción: Al escribir datos a un fichero, normalmente Euphoria almacena los datos en un búfer de memoria hasta se acumule una porción lo suficientemente grande. Esta gran porción se puede escribir en el disco de forma muy eficiente. A veces, puede querer forzar (o descargar) todos los datos inmediatamente, aún cuando el búfer de memoria no esté lleno. Para hacer esto, tiene que llamar a ***flush(fn)***, donde 'fn' es el número de fichero de un fichero abierto para escritura o agregado.

Comentarios: Cuando el fichero está cerrado, (ver ***close()***), todos los datos en el búfer se descargan. Cuando termina un programa, todos los ficheros abiertos se descargan y se cierran automáticamente.
Use ***flush()*** cuando otro proceso necesite ver todos los datos escritos hasta ahora, pero sin que esté listo para cerrar el fichero aún.

Ejemplo:

```
f = open("logfile", "w")
puts(f, "Record#1\n")
puts(1, "Presione ENTER cuando esté listo\n")

flush(f)  -- Esto fuerza a "Record #1" into "logfile" on disk.
          -- Sin esto, "logfile" aparecerá teniendo
          -- 0 caracteres cuando detengamos la entrada de teclado.
```

```
s = gets(0) -- espera una entrada por teclado
```

Ver también: *close*, *lock_file*

free

Sintaxis:
include machine.e
free(a)

Descripción: Libera un bloque de memoria previamente asignado al especificar la dirección del comienzo del bloque, es decir, la dirección que devolvió *allocate()*.

Comentarios: Use ***free()*** para reciclar bloques de memoria durante la ejecución. Esto reducirá la posibilidad de quedarse sin memoria o de provocar un excesivo intercambio de memoria virtual a disco. No referencie un bloque de memoria que ya fue liberado. Cuando su programa termina, toda la memoria asignada se devolverá al sistema.
No use ***free()*** para desasignar memoria que fue asignada usando *allocate_low()*. Use *free_low()* para este fin.

Programa de ejemplo: **demo\callmach.ex**

Ver también: *allocate*, *free_low*

free_console

Plataforma: **WIN32, Linux, FreeBSD**

Sintaxis:
include dll.e
free_console()

Descripción: Libera (borra) cualquier ventana de consola asociada a su programa.

Comentarios: Euphoria creará una ventana de consola de texto para su programa la primera vez que el programa escriba algo en pantalla, lea algo desde el teclado o cualquier situación que necesite una consola (similar a la ventana del Indicador de DOS). En ***WIN32*** esta ventana desaparecerá automáticamente cuando termine el programa, pero puede llamar a ***free_console()*** para hacerla desaparecer más pronto. En ***Linux*** o ***FreeBSD***, la consola de modo de texto está siempre, pero una ventana xterm desaparecerá después que Euphoria muestre el indicador "Presionar Enter" al final de la ejecución.
En ***Linux*** o ***FreeBSD***, ***free_console()*** establecerá los parámetros nuevamente como normales, deshaciendo los efectos que curses tiene en pantalla.

En una ventana xterm de ***Linux*** o ***FreeBSD***, una llamada a ***free_console()***, sin ninguna impresión en pantalla o lectura del teclado, eliminará el indicador "Presionar Enter" que normalmente emite Euphoria al final de la ejecución.

Después de liberar la ventana de consola, puede crear una nueva ventana de consola imprimiendo alguna cosa en pantalla, o simplemente llamando a *clear_screen()*, *position()* o cualquier otra rutina que necesite una consola.

Cuando usa la utilidad de **trazado**, o cuando el programa tiene un error, Euphoria creará automáticamente una venta de consola para mostrar la información de trazado, mensajes de error, etc.

Hay una rutina de la API de ***WIN32***, ***FreeConsole()*** que hace algo similar a ***free_console()***. Debería usar ***free_console()***, porque le permite al intérprete saber si no existe más una consola.

Ver también: *clear_screen*, **platform.doc**

free_low

Plataforma: **DOS32**

Sintaxis:
include machine.e
free_low(i)

Descripción: Libera un bloque de memoria convencional previamente asignado al especificar la dirección del comienzo del bloque, es decir, la dirección que devolvió *allocate_low()*.

Comentarios: Use ***free_low()*** para reciclar bloques de memoria convencional durante la ejecución. Esto reducirá la posibilidad de quedarse sin memoria convencional. No referencie un bloque de memoria que ya

fue liberado. Cuando su programa termina, toda la memoria asignada se devolverá al sistema.
No use ***free_low()*** para desasignar memoria que fue asignada usando ***allocate()***. Use ***free()*** para este fin.

Programa de ejemplo: **demo\dos32\dosint.ex**

Ver también: ***allocate_low***, ***dos_interrupt***, ***free***

get

Sintaxis:
include get.e
s = get(fn)

Descripción:
Ingresa, desde el fichero 'fn', una cadena de caracteres humanamente legible representando un objeto Euphoria. Convierte la cadena en el valor numérico de ese objeto. 's' será una secuencia de 2 elementos: **{estado de error, valor}**. Los códigos de error son:

```
GET_SUCCESS -- el objeto se leyó existosamente
GET_EOF     -- fin de fichero antes que se lea el objeto
GET_FAIL    -- el objeto no es correcto sintacticamente
```

get() puede leer objetos Euphoria arbitrariamente complicados. Podría tener un secuencia larga de valores entre llaves, separados por comas, por ejemplo: {23, {49, 57}, 0.5, −1, 99, 'A', "Juan"}. **Una sola llamada a get() leerá secuencia entera y devolverá su valor como resultado.**
Cada llamada a ***get()*** inicia donde quedó la llamada previa. Por ejemplo, se debería necesitar una serie de 5 llamadas a ***get()*** para leer:

99 5.2 {1,2,3} "Hola" −1

En la sexta llamada a ***get()*** y cualquier otra siguiente, vería el código de estado GET_EOF. Si tiene algo como:

{1, 2, xxx}

en el flujo de entrada vería el código de error GET_FAIL, porque xxx no es un objeto Euphoria.

Si tiene varios objetos "de alto nivel" en el flujo de entrada, tiene que separarlos unos de otros mediante uno o más caracteres "separadores" (blanco, tabulador, \r o \n). Los separadores no son necesarios ***dentro*** de un objeto de alto nivel. Una llamada a ***get()*** leerá un objeto completo de alto nivel, más un caracter adicional (separador).

Comentarios:
La combinación de ***print()*** y ***get()*** se puede usar para guardar un objeto Euphoria en el disco y luego volver a leerlo. Esta técnica se puede usar para implementar una base de datos de una o más secuencias Euphoria grandes almacenadas en ficheros del disco. Las secuencias se pueden leer a memoria, actualizar y escribir nuevamente al disco después que cada serie de transacciones se complete. Recuerde escribir un caracter "separador" (usando ***puts()***) después de cada llamada a ***print()***.
El valor devuelto carece de significado, salvo que obtenga el estado GET_SUCCESS.

Ejemplo:
Suponga que el programa le pide al usuario ingresar un número desde el teclado.

```
-- Si el usuario escribe 77.5, get(0) devolverá:

{GET_SUCCESS, 77.5}

-- por cuanto gets(0) devolverá:

"77.5\n"
```

Programa de ejemplo: **demo\mydata.ex**

Ver también: ***print***, ***value***, ***gets***, ***getc***, ***prompt_number***, ***prompt_string***

get_active_page

Plataforma: **DOS32**

Sintaxis:
include image.e
i = get_active_page()

Descripción:
Algunos modos gráficos en la mayoría de las tarjets de video, tienen varias páginas de memoria. Esto le permite escribir la salida de pantalla en una página, mientras muestra otra diferente.

get_active_page() devuelve el número de página actual que se está enviando a la pantalla.

Comentarios: Las páginas activa y mostrada son 0 por defecto. ***video_config()*** le dirá cuantas páginas están disponibles en el actual modo gráfico.

Ver también: ***set_active_page***, ***get_display_page***, ***video_config***

get_all_palette

Plataforma: DOS32

Sintaxis: include image.e
s = get_all_palette()

Descripción: Devuelve las intensidades de color para el conjunto completo de colores del actual modo gráfico. 's' es una secuencia de la forma:
{{r,g,b}, {r,g,b}, ..., {r,g,b}}

Cada elemento especifica una intensidad de color {red, green, blue} para el número de color correspondiente, comenzando con el número de color 0. Los valores para rojo, verde y azul estarán en el rango de 0 a 63.

Comentarios: Esta función se debería usar para obtener los valores de la paleta que necesita ***save_bitmap()***. Recuerde multiplicar esos valores por 4 antes de llamar a ***save_bitmap()***, ya que ***save_bitmap()*** espera valores en el rango de 0 a 255.

Ver también: ***palette***, ***all_palette***, ***read_bitmap***, ***save_bitmap***, ***save_screen***

get_bytes

Sintaxis: include get.e
s = get_bytes(fn, i)

Descripción: Lee los próximos 'i' bytes desde el fichero número 'fn'. Devuelve los bytes como una secuencia. La secuencia será de longitud 'i', salvo cuando haya menos de los 'i' bytes restantes a leer dentro del fichero.

Comentarios: Cuando i > 0 y ***length(s)*** < i, usted sabe que se alcanzó el fin del fichero. Eventualmente, se devolverá una secuencia vacía en 's'.
Esta función se usa normalmente con ficheros abiertos en modo binario "rb". Esto evita la situación confusa en el modo de texto donde DOS convierte el par CR LF a LF.

Ejemplo:

```
include get.e

integer fn
fn = open("temp", "rb")  -- un fichero existente

sequence whole_file
whole_file = {}

sequence chunk

while 1 do
    chunk = get_bytes(fn, 100) -- lee 100 bytes por vez
    whole_file =chunk        -- chunk debería estar vacío, ok
    if length(chunk) <100 then
        exit
    end if
end while

close(fn)
? length(whole_file)  -- debería coincidir el tamaño DIR de "temp"
```

Ver también: ***getc***, ***gets***

get_display_page

Plataforma: DOS32

Sintaxis: include image.e
i = get_display_page()

Descripción: Algunos modos gráficos en la mayoría de las tarjets de video, tienen varias páginas de memoria. Esto le permite escribir la salida de pantalla en una página, mientras muestra otra diferente. ***get_display_page()*** devuelve el número de página actual que se está mostrando en el monitor.

Comentarios: Las páginas activa y mostrada son 0 por defecto. ***video_config()*** le dirá cuantas páginas están disponibles en el actual modo gráfico.

Ver también: ***set_display_page***, ***get_active_page***, ***video_config***

get_key

Sintaxis: i = get_key()

Descripción: Devuelve la tecla que presionó el usuario, sin esperar. Devuelve −1, si no se presionó ninguna tecla. Devuelve los códigos especiales de las teclas de función, flechas del cursor, etc.

Comentarios: El sistema operativo puede mantener en el búfer del teclado, una pequeña cantidad de tecleos. ***get_key()*** devolverá el siguiente del búfer, o −1 si el búfer está vacío.
Ejecute el programa **key.bat** para ver que código de tecla se genera para cada tecla de su teclado.

Ver también: ***wait_key***, ***getc***

get_mouse

Plataforma: **DOS32, Linux**

Sintaxis:
```
include mouse.e
x1 = get_mouse()
```

Descripción: Devuelve el último evento del ratón, en la forma: **{evento, x, y}** o devuelve −1 si no ha habido ningún evento del ratón, desde la última vez que se invocó a ***get_mouse()***.
Las constntes definidas en **mouse.e** para los posibles eventos del ratón son:

```
global constant MOVE = 1,
           LEFT_DOWN = 2,
             LEFT_UP = 4,
          RIGHT_DOWN = 8,
            RIGHT_UP = 16,
         MIDDLE_DOWN = 32,
           MIDDLE_UP = 64
```

x e y son las coordenadas del puntero del ratón en el momento en que ocurrió el evento. ***get_mouse()*** inmediatamente devuelve tanto −1 como un evento del ratón. No espera que ocurra un evento. Usted tiene que consultarlo bastante frecuentemente para evitar perder algún evento. Al ocurrir el evento siguiente, el evento actual se pierde, si no lo leyó. En la práctica, no es difícil capturar la mayoría de los eventos. Perder un evento MOVE, generalmente no están serio, ya que el siguiente MOVE le dirá donde está el puntero del ratón.
A veces se informan varios eventos. Por ejemplo, si el ratón se está moviendo cuando se presiona el botón izquierdo, ***get_mouse()*** informará el valor de los eventos LEFT_DOWN+MOVE, es decir 2+1 o 3. Por esta razón, debería consultar un evento en particular usando ***and_bits()***. Vea los ejemplos de más abajo.

Comentarios: En los modos de gráficos de píxel que son de 320 píxeles de ancho, necesita dividir el valor 'x' por 2 para conseguir la posición correcta en pantalla (una característica extraña del DOS).
En los modos de texto de DOS32 necesita escalar las coordenadas x e y, para obtener las posiciones de línea y columna. En Linux, este escalamiento no se necesita − x e y corresponden a la línea y columna de la pantalla, donde (1,1) es la esquina superior izquierda.

En DOS32, necesita un controlador de ratón DOS para usar esta rutina. En Linux, el servidor GPM tiene que estar corriendo.

En Linux, los eventos de movimiento del ratón no se informan en una ventana xterm, solo en la consola de texto. LEFT_UP, RIGHT_UP y MIDDLE_UP no se distinguen unos de otros.

Puede usar ***get_mouse()*** en la mayoría de los modos de texto y gráficos de píxel.

La primera llamada que haga a ***get_mouse()*** activará el puntero del ratón o un caracter resaltado.

El DOS generalmente no soporta el uso del ratón en los modos gráficos SVGA (más allá de 640x480 píxeles). Esta restricción se quitó en Windows 95 (DOS 7.0). **Graeme Burke**, **Peter Blue** y otros, han contribuido con **rutinas de ratón** que salvan los problemas de usar el ratón en SVGA. Vea la página web "Archivo" de Euphoria.

Las coordenadas x, y devueltas pueden ser aquellas of the very tip of the mouse pointer o podrían referirse al píxel apuntado por el puntero del ratón. Probar si se está intentando leer un píxel usando ***get_pixel()***. En su lugar, puede tener que leer x−1, y−1.

Ejemplo 1: un valor de retorno de:

```
{2, 100, 50}
```

indicaría que se presionó el botón izquierdo cuando el puntero del ratón estaba en la posición x=100, y=50 de la pantalla.

Ejemplo 2: Para probar LEFT_DOWN, escriba algo así:

```
object event

while 1 do
    event = get_mouse()
    if sequence(event) then
        if and_bits(event[1], LEFT_DOWN) then
            -- se presionó el botón izquierdo
            exit
        end if
    end if
end while
```

Ver también: ***mouse_events***, ***mouse_pointer***, ***and_bits***

get_pixel

Plataforma: **DOS32**

Sintaxis: x = get_pixel(s)

Descripción: Cuando 's' es una coordenada de pantalla de 2 elementos {x, y}, ***get_pixel()*** devuelve el color (un entero pequeño) del píxel en la pantalla de gráficos de píxel en ese punto.
Cuando 's' es una secuencia de 3 elementos de la forma: {x, y, n}, ***get_pixel()*** devuelve una secuencia de 'n' colores para los puntos que comienzan en {x, y} y siguen a la derecha {x+1, y}, {x+2, y}, etc.

Los puntos fuera de la pantalla tienen valores de color impredecibles.

Comentarios: Cuando 'n' está especificado, se usa un algoritmo muy veloz para leer los colores del píxel en pantalla. Es mucho más rápido llamar a ***get_pixel()*** una vez, especificando un valor grande para 'n', que llamarlo muchas veces leyendo un color de píxel por vez.

Ejemplo:

```
object x

x = get_pixel({30,40})
-- x se establece al valor de color del punto x=30, y=40

x = get_pixel({30,40,100})
-- x se establece a una secuencia de 100 valores enteros, representando
-- los colores que empiezan en {30,40} y siguen hacia la derecha
```

Ver también: ***pixel***, ***graphics_mode***, ***get_position***

get_position

Sintaxis: include graphics.e
s = get_position()

Descripción: Devuelve la línea y columna actuales de la posición del cursor, como una secuencia de 2 elementos **{línea, columna}**.

Comentarios: ***get_position()*** trabaja tanto en los modos de texto como los de gráficos de píxel. En los modos gráficos de píxel no se muestra el cursor, pero ***get_position()*** devolverá la línea y la columna donde se mostrará el próximo caracter.
El sistema de coordenadas para mostrar texto es diferente que el que se usa para mostrar píxeles. Los píxeles se muestran de modo tal que el superior izquierdo (x=0,y=0) y la primera coordenada controla la ubicación horizontal izquierda−derecha. En los modos de gráficos de píxel, se pueden mostrar tanto texto como píxeles.

get_position() devuelve la línea y columna actuales para el texto que se está mostrando, no los píxeles que puede dibujar. No existe la rutina que obtiene la posición actual del píxel.

Ver también: ***position***, ***get_pixel***

get_screen_char

Plataforma: **DOS32, Linux, FreeBSD**

Sintaxis: include image.e
s = get_screen_char(i1, i2)

Descripción: Devuelve una secuencia 's' de 2 elementos, de la forma **{código ASCII, atributos}** para el caracter en pantalla de la línea 'i1', columna 'i2'. 's' está compuesto por dos átomos. El primero es el código ASCII del caracter y el segundo contiene los colores de frente y fondo del caracter y, posiblemente otra información describiendo la apariencia del caracter en pantalla.

Comentarios: Con ***get_screen_char()*** y ***put_screen_char()*** puede guardar y recuperar un caracter de la pantalla, junto con sus atributos.

Ejemplo:

```
-- lee el caracter y atributos de la esquina superior izquierda
s = get_screen_char(1,1)
-- almacena el caracter y atributos de la línea 25, columna 10
put_screen_char(25, 10, {s})
```

Ver también: ***put_screen_char***, ***save_text_image***

get_vector

Plataforma: **DOS32**

Sintaxis: include machine.e
s = get_vector(i)

Descripción: Devuelve la dirección lejana (far) de modo protegido actual del manejador de interrupción número 'i'. 's' será una secuencia de 2 elementos: **{segmento de 16 bits, offset de 32 bits}**.

Ejemplo:

```
s = get_vector(#1C)
-- s tendrá la dirección lejana (far) del manejar de la interrupción del
-- timer, por ejemplo: {59, 808}
```

Programa de ejemplo: **demo\dos32\hardint.ex**

Ver también: ***set_vector***, ***lock_memory***

getc

Sintaxis: i = getc(fn)

Descripción: Obtiene el siguiente caracter (byte) desde un fichero o dispositivo 'fn'. El caracter tendrá un valor entre 0 y 255. Al final del fichero se devuelve −1.

Comentarios: Con ***getc()***, se usa un búfer a la salida del fichero, es decir, ***getc()*** no va al disco por cada caracter. En su lugar, un gran bloque de caracteres se leerán de una vez y se traerán uno por uno desde el búfer de memoria.
Cuando ***getc()*** lee desde el teclado, no verá ningún caracter hasta que se presione Enter. Tenga en cuenta que el usuario puede escribir Ctrl+Z, que el sistema operativo interpreta como "fin de fichero". Se devolverá −1.

Ver también: ***gets***, ***get_key***, ***wait_key***, ***open***

getenv

Sintaxis: x = getenv(s)

Descripción: Devuelve el valor de una variable del entorno. Si la variable está indefinida, devuelve −1.

Comentarios: Debido a que se podrían devolver tanto una secuencia como un átomo (−1), probablemente debería asignar el resultado a una variable declarada como objeto.

Ejemplo:

```
e = getenv("EUDIR")
-- e será "C:\EUPHORIA" -- o tal vez D:, E: etc.
```

Ver también: ***command_line***

gets

Sintaxis: x = gets(fn)

Descripción: Obtiene la próxima secuencia (una línea, incluyendo '\n') de caracteres desde un fichero o dispositivo 'fn'. Los caracteres tendrán valores entre 0 y 255. Al final del fichero se devuelve el átomo −1.

Comentarios: Debido a que se puede devolver tanto una secuencia como un átomo (−1), probablemente debería asignar el resultado a una variable declarada como objeto.
Después de leer una línea de texto desde el teclado, comúnmente debería emitir un caracter '\n', por ejemplo ***puts(1, '\n')***, antes de imprimir alguna cosa. Solamente en la última línea de la pantalla el sistema operativo hace un desplazamiento de pantalla y avanza a la siguiente línea.

La última línea de un fichero no debería terminar con el caracter '\n' (nueva línea).

Al leer desde el teclado, el usuario puede escribir Ctrl+Z, que el sistema operativo interpreta como "fin de fichero". Se devolverá −1.

En los modos SVGA, DOS puede establecer erróneamente la posición del cursor, después de llamar a ***gets(0)*** para leer el teclado. Debería establecerlo usando ***position()***.

Ejemplo 1:

```
sequence buffer
object line
integer fn

-- lee un fichero de texto a una secuencia
fn = open("mifichero.txt", "r")
if fn = -1 then
    puts(1, "No se puede abrir mifichero.txt\n")
    abort(1)
end if

buffer = {}
while 1 do
    line = gets(fn)
    if atom(line) then
        exit   -- al final del fichero se devuelve -1
    end if
    buffer = append(buffer, line)
end while
```

Ejemplo 2:

```
object line

puts(1, "¿Cual es tu nombre?\n")
line = gets(0)  -- lee la entrada estándar (teclado)
line = line[1..length(line)-1] -- elimina el caracter \n del final
puts(1, '\n')   -- necesario
puts(1, line &" es un bonito nombre.\n")
```

Ver también: ***getc***, ***puts***, ***open***

graphics_mode

Plataforma: **DOS32**

Sintaxis: include graphics.e
i1 = graphics_mode(i2)

Descripción: Selecciona el modo gráfico 'i2'. Ver la lista de modos gráficos válidos en **graphics.e**. Si la operación resulta exitosa, 'i1' valdrá 0, en caso contrario 'i1' valdrá 1.

Comentarios: A algunos modos se los llama modos de texto porque le permiten solamente mostrar texto. A los otros modos se los llama modos de gráficos de píxel porque pueden mostrar píxeles, líneas, elipses, así

como texto.
Como facilidad para sus usuarios, es buena idea regresar del modo de gráficos de píxel al modo de texto antes de terminar el programa. Puede hacer esto con ***graphics_mode(−1)***. Si un programa de gráficos de píxel deja su pantalla hecha un desorden, puede limpiarla con el comando CLS de DOS, o ejecutando **ex** o **ed**.

Bajo ciertas condiciones, algunas tarjetas gráficas serán incapaces de entrar en algunos modos SVGA. No siempre puede decir desde el valor de 'i1', que la configuración del modo gráfico fue existosa.

En las plataformas **WIN32** y **Linux/FreeBSD**, ***graphics_mode()*** asignará una consola de modo texto plano si no existe ninguna aún. Devolverá 0, sin importar el valor pasado como 'i2'.

Ejemplo:

```
if graphics_mode(18) then
    puts(SCREEN, "se necesitan gráficos VGA!\n")
    abort(1)
end if
draw_line(BLUE, {{0,0}, {50,50}})
```

Ver también: ***text_rows***, ***video_config***

instance

Plataforma: **WIN32**

Sintaxis: include misc.e
i = instance()

Descripción: Devuelve un número identificador para el programa actual.

Comentarios: Este valor número identificador se puede pasar a varias rutinas de Windows para obtener información acerca del programa que está corriendo. Cada vez que un usuario inicia su programa, se creará una nueva instancia.
En C, este es el primer parámetro de WinMain().

En **DOS32** y **Linux/FreeBSD**, ***instance()*** siempre devuelve 0.

Ver también: **platform.doc**

int_to_bits

Sintaxis: include machine.e
s = int_to_bits(a, i)

Descripción: Devuelve los bits 'i' de orden bajo de 'a' como una secuencia de 1 y 0. Los bits menos significativos vienen primero. Para los números negativos, se devuelve el patrón en complemento a 2.

Comentarios: Puede usar subíndices, subrangos, and/or/xor/not de secuencias completas, etc. para manipular las secuencias de bits. Las rotaciones y desplazamientos de bits son fáciles de realizar.

Ejemplo:

```
s = int_to_bits(177, 8)
-- s is {1,0,0,0,1,1,0,1} -- orden "inverso"
```

Ver también: ***bits_to_int***, ***and_bits***, ***or_bits***, ***xor_bits***, ***not_bits***, operaciones sobre secuencias

int_to_bytes

Sintaxis: include machine.e
s = int_to_bytes(a)

Descripción: Convierte un entero en una secuencia de 4 bytes. Esos bytes están en el orden esperado por el 386+, es decir, el bit menos significativo primero.

Comentarios: Usted puede ser que utilice esta rutina antes de poner los 4 bytes en la memoria para el uso por un programa de lenguaje de máquina.
El número entero puede ser negativo. Se devolverán los valores byte negativos, pero después de ponerlos en memoria recién tendrá la representación correcta (complemento a dos) para el 386+.

Esta función convertirá correctamente valores enteros de hasta 32 bits. Para valores más grandes, solamente se convierten los 32 bits de orden inferior. El tipo entero de Euphoria permite solamente valores de hasta 31 bits, así que declare sus variables como **átomos** si necesita una gama más grande.

Ejemplo 1:

```
s = int_to_bytes(999)
-- s es {231, 3, 0, 0}
```

Ejemplo 2:

```
s = int_to_bytes(-999)
-- s es {-231, -4, -1, -1}
```

Ver también: ***bytes_to_int***, ***int_to_bits***, ***bits_to_int***, ***peek***, ***poke***, ***poke4***

integer

Sintaxis: i = integer(x)

Descripción: Devuelve 1 si 'x' es un entero dentro del rango −1073741824 to +1073741823. En cualquier otro caso, devuelve 0.

Comentarios: Esta función sirve para definir el tipo entero. También puede llamarla como una función ordinaria para determinar si un objeto es un entero.

Ejemplo 1:

```
integer z
z = -1
```

Ejemplo 2:

```
if integer(y/x) then
    puts(SCREEN, "y es un múltiplo exacto de x")
end if
```

Ver también: ***atom***, ***sequence***, ***floor***

length

Sintaxis: i = length(s)

Descripción: Devuelve la longitud de 's'. 's' tiene que ser una secuencia. Ocurrirá un error si 's' es un átomo.

Comentarios: La longitud de cada secuencia, la almacena internamente el intérprete para acceso rápido (en otros lenguajes esta operación requiere de una búsqueda en memoria para hallar el caracter de terminación).

Ejemplo 1:

```
length({{1,2}, {3,4}, {5,6}})   -- 3
```

Ejemplo 2:

```
length("")    -- 0
```

Ejemplo 3:

```
length({})    -- 0
```

Ver también: ***sequence***

lock_file

Sintaxis:
include file.e
i1 = lock_file(fn, i2, s)

Descripción: Cuando varios procesos acceden a un archivo simultáneamente, se puede nucesitar algún tipo de mecanismo de bloqueo para evitar mutilar el contenido del archivo, o causar datos erróneos en la lectura.
lock_file() intenta poner un bloqueo en un archivo abierto 'fn' para detener a otros procesos de usar el archivo mientras su programa lo está escribiendo o leyendo. Bajo ***Linux/FreeBSD***, hay dos tipos de bloqueos que se pueden solicitar usando el parámetro 'i2' (bajo ***DOS32*** y ***WIN32*** el parámetro 'i2' se ignora, pero debería ser un entero). Solicite un bloqueo **compartido** cuando quiera leer un archivo y desee bloquearlo temporalmente para otros procesos que lo lean. Solicite un bloqueo **exclusivo** cuando quiera escribir en el archivo y desee bloquearlo temporalmente para otros procesos que lo lean o escriban en él. Muchos procesos pueden tener bloqueos compartidos aplicados simultáneamente sobre el mismo archivo, pero solamente un proceso puede tener bloqueo exclusivo, el cual solamente puede aplicarse cuando ningún otro procesos haya aplicado ningún bloqueo de cualquier clase sobre el archivo. ***file.e*** contiene la siguiente declaración:

```
global constant LOCK_SHARED = 1,
                LOCK_EXCLUSIVE = 2
```

En ***DOS32*** y ***WIN32*** puede bloquear una porción específica de un archivo usando el parámetro 's'. 's' es una secuencia de la forma: {primer byte, último byte}. En ella se indican el primer y último byte de la porción del archivo sobre la que se aplica el bloqueo. Especifique la secuencia vacía {}, si quiere bloquear el archivo completo. En la versión actual para ***Linux/FreeBSD***, los bloqueos siempre se aplican sobre el archivo completo y debería especificar {} para este parámetro.
lock_file() devolverá 1, si se pudo obtener el bloqueo deseado; en caso contrario, se devolverá 0. ***lock_file()*** no espera otros procesos para abandonar sus bloqueos. Puede tener que llamarlo en varias ocasiones, antes que se conceda la petición del bloqueo.

Comentarios: En ***Linux/FreeBSD***, estos bloqueos se llaman bloqueos consultivos, lo que significa que no son forzados por el sistema operativo. Esto es, los procesos que utilizan un archivo particular cooperan uno con otro. Un proceso puede acceder a un archivo sin primero bloquearlo. En ***WIN32*** y ***DOS32***, los bloqueos son forzados por el sistema operativo.

En ***DOS32***, ***lock_file()*** es más útil cuando está habilitada la compartición de archivos. Típicamente devuelve 0 (falla) bajo ***MS−DOS*** plano, fuera de ***Windows***.

Ejemplo:

```
include misc.e
include file.e
integer v
atom t
v = open("visitor_log", "a")  -- abre para agregar
t = time()
while not lock_file(v, LOCK_EXCLUSIVE, {}) do
    if time() > t + 60 then
        puts(1, "Un minuto ya... No puedo esperar por siempre!\n")
        abort(1)
    end if
    sleep(5) -- permite correr a los demás procesos
end while
puts(v, "Otro visitante\n")
unlock_file(v, {})
close(v)
```

Ver también: ***unlock_file***, ***flush***, ***sleep***

lock_memory

Plataforma: **DOS32**

Sintaxis: include machine.e
lock_memory(a, i)

Descripción: Evita el bloque de la memoria virtual que comienza en la dirección 'a', de longitud 'i', siempre sea intercambiada al disco.

Comentarios: ***lock_memory()*** se debería usar solo en situaciones altamente especializadas, donde tenga que iniciar sus propios manejadores de interrupciones por hardware de DOS, usando código de máquina. Cuando ocurre una interrupción de hardware, no le es posible al sistema operativo devolver ningún código o dato que se haya intercambiado, por lo tanto, usted necesita proteger cualquier bloque de código de máquina o datos que necesite en el servicio de la interrupción.

Programa ejemplo: **demo\dos32\hardint.ex**

Ver también: ***get_vector***, ***set_vector***

log

Sintaxis: x2 = log(x1)

Descripción: Devuelve el logaritmo natural de 'x1'.

Comentarios: Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia. Tenga presente que log está definida solamente para números positivos. Si intenta tomar el logaritmo de un número negativo o de cero, su programa se abortará con un mensaje de error.

Ejemplo:

```
a = log(100)
-- a es 4.60517
```

Ver también: ***sin***, ***cos***, ***tan***, ***sqrt***

lower

Sintaxis: include wildcard.e
x2 = lower(x1)

Descripción: Convierte un átomo o secuencia a minúsculas.

Ejemplo:

```
s = lower("Euphoria")
-- s es "euphoria"

a = lower('B')
-- a es 'b'

s = lower({"Euphoria", "Programming"})
-- s es {"euphoria", "programming"}
```

Ver también: ***upper***

machine_func

Sintaxis: x1 = machine_func(a, x)

Descripción: Ver ***machine_proc()*** debajo.

machine_proc

Sintaxis: machine_proc(a, x)

Descripción: Ejecuta una operación específica de máquina, tal como un gráfico o efectos sonoros. Esta rutina normalmente se debería llamar indirectamentea través de las rutinas de librería de los archivos de inclusión de Euphoria. Una llamada directa, hecha incorrectamente, provocaría una excepción de la máquina.

Ver también: ***machine_func***

match

Sintaxis: i = match(s1, s2)

Descripción: Busca la coincidencia de 's1' contra un cierto subrango de 's2'. Si resulta exitoso, devuelve el número de elemento de 's2' donde comienza (la primera) coincidencia del subrango, sino devuelve 0.

Ejemplo:

```
posicion = match("pho", "Euphoria")
-- posicion se establece a 3
```

Ver también: ***find***, ***compare***, ***wildcard_match***

mem_copy

Sintaxis: mem_copy(a1, a2, i)

Descripción: Copia un bloque de 'i' bytes de memoria desde la dirección 'a2' a la dirección 'a1'.

Comentarios: Los byes de memoria se copiarán correctamente, aún cuando el bloque en 'a2' se solape sobre el que comienza en 'a1'.
mem_copy(a1, a2, i) es equivalente a: **poke(a1, peek({a2, i}))**, pero mucho más rápida.

Ejemplo:

```
dest = allocate(50)
src = allocate(100)
poke(src, {1,2,3,4,5,6,7,8,9})
mem_copy(dest, src, 9)
```

Ver también: ***mem_set***, ***peek***, ***poke***, ***allocate***, ***allocate_low***

mem_set

Sintaxis: mem_set(a1, i1, i2)

Descripción: Establece 'i2' byte de memoria, comenzando en la dirección 'a1' al valor de 'i1'.

Comentarios: Los 8 bits de orden inferior de 'i1' se almacenan realmente en cada byte.

mem_set(a1, i1, i2) es equivalente a: **poke(a1, repeat(i1, i2))**, pero mucho más rápida.

Ejemplo:

```
destination = allocate(1000)
mem_set(destination, ' ', 1000)
-- 1000 bytes consecutivos en memoria se establecerán a 32
-- (el código ASCII de ' ')
```

Ver también: *mem_copy*, *peek*, *poke*, *allocate*, *allocate_low*

message_box

Plataforma: WIN32

Sintaxis:
include msgbox.e
i = message_box(s1, s2, x)

Descripción: Muestra una ventana con título 's2', conteniendo la cadena de mensaje 's1'. 'x' determina la combinación de botones que estarán disponibles para que el usuario presione, y algunas otras características. 'x' puede ser un átomo p una secuencia. El valor de retorno 0, indica la falla en la inicialización de la ventana.

Comentarios: Ver la lista completa de valores posibles para 'x' e 'i' en **msgbox.e** .

Ejemplo:

```
response = message_box("¿Desea continuar?",
                       "Mi Programa",
                       MB_YESNOCANCEL)
if response = IDCANCEL or response = IDNO then
    abort(1)
end if
```

Programa ejemplo: **demo\win32\email.exw**

mouse_events

Plataforma: DOS32, Linux

Sintaxis:
include mouse.e
mouse_events(i)

Descripción: Use este procedimiento para seleccionar los eventos del ratón que desea que ***get_mouse()*** informe. Por defecto, ***get_mouse()*** informará todos los eventos. ***mouse_events()*** se puede llamar en varios lugares de la ejecución del programa, allí donde se necesite detectar los cambios en los eventos. Bajo **Linux**, ***mouse_events()*** actualmente no tiene efecto.

Comentarios: Es una buena práctica ignorar los eventos en los que no está interesado, particularmente el muy frecuente evento MOVE, para reducir la ocasión quese perderá de un evento significativo.
La primera llamada que hace a ***mouse_events()*** activará el puntero del ratón, o un caracter resaltado.

Ejemplo:

```
mouse_events(LEFT_DOWN + LEFT_UP + RIGHT_DOWN)
-- restringirá get_mouse() a informar cuando el botón izquierdo
-- sea presionado y liberado, y el botón derecho
-- sea presionado. El resto de los eventos se ignorarán.
```

Ver también: *get_mouse*, *mouse_pointer*

mouse_pointer

Plataforma: DOS32, Linux

Sintaxis:
include mouse.e
mouse_pointer(i)

Descripción: Si 'i' vale 0, se oculta el puntero del ratón, en caso contrario se activará. Varias llamadas para ocultar el puntero, necesitarán de varias llamadas para activarlo nuevamente. La primera llamada tanto a ***get_mouse()*** o a ***mouse_events()***, también activará el puntero (una vez). Bajo **Linux**, ***mouse_pointer()*** actualmente no tiene efecto.

Comentarios: Puede ser necesario ocultar el puntero del ratón en forma temporal, mientras actualiza la pantalla.
Después de llamar a *text_rows()* puede tener que llamar a ***mouse_pointer(1)*** para ver nuevamente el puntero del ratón.

Ver también: *get_mouse*, *mouse_events*

not_bits

Sintaxis: x2 = not_bits(x1)

Descripción: Ejecuta la operación lógica NOT a cada bit de 'x1'. Un bit en 'x2' será 1 si el correspondiente en 'x1' es 0 y viceversa.

Comentarios: Los argumentos para la función pueden ser átomos o secuencias. Se aplican las reglas para operaciones sobre secuencias.
Los argumentos tienen que ser representables como números de 32 bits, sean con o sin signo.

Si se prepone manipular valores 32−bit completos, debería declarar sus variables como **átomos**, en lugar de enteros. El tipo entero de Euphoria está limitado a 31 bits.

Los resultados se tratan como números con signo. Serán negativos cuando el bit de mayor orden sea 1.

Ejemplo:

```
a = not_bits(#000000F7)
-- a es -248 (es decir, FFFFFF08 interpretado como número negativo)
```

Ver también: ***and_bits***, ***or_bits***, ***xor_bits***, ***int_to_bits***

object

Sintaxis: i = object(x)

Descripción: Prueba si 'x' es de tipo objeto. Siempre será verdadero, por lo que ***object()*** siempre devolverá 1.

Comentarios: Todos los tipos predefinidos como los definidos por el usuario también se pueden usar como funciones para probar si un valor pertenece a ese tipo. ***object()*** se incluye solo para completar. Siempre devuelve 1.

Ejemplo:

```
? object({1,2,3})  -- siempre imprime 1
```

Ver también: ***integer***, ***atom***, ***sequence***

open

Sintaxis: fn = open(st1, st2)

Descripción: Abre un archivo o dispositivo, para obtener el número de archivo. Devuelve −1 si la operación falla. 'st1' es la ruta del archivo o dispositivo. 'st2' es el modo en que se abre el archivo. Los modos posibles son:
"r" − abre un archivo de texto para lectura
"rb" − abre un archivo binario para lectura
"w" − crea un archivo de texto para escritura
"wb" − crea un archivo binario para escritura
"u" − abre un archivo de texto para actualizarlo (lectura y escritura)
"ub" − abre un archivo binario para actualizarlo
"a" − abre un archivo de texto para agregar nuevos datos
"ab" − abre un archivo binario para agregar nuevos datos

Los archivos que se abren para leer o actualizar, tienen que existir. Los archivos que se abren para escribir o agregar, serán creados si es necesario. Un archivo abierto para escritura tendrá 0 bytes. La salida a un archivo abierto para agregar, comenzará en el final del archivo.

La salida a **archivos de texto** tendrá caracteres de retrono de carro, agregados automáticamente antes de los caracteres de avance de línea. En la entrada, se quitarán estos caracteres de retorno de carro. El caracter Ctrl+Z (ASCII 26) indicará el fin del archivo.

La E/S a **archivos binarios** no se modifica en forma alguna. Se puede leer o escribir cualquier valor de byte de 0 a 255.

Algunos dispositivos típicos que puede abrir son:

"CON" − consola (pantalla)
"AUX" − puerto serie auxiliar
"COM1" − puerto serie 1
"COM2" − puerto serie 2
"PRN" − impresora en el puerto paralelo

"NUL" − un dispositivo inexistente que acepta y descarta salidas

Comentarios: **DOS32:** Al correr bajo Windows 95 o posterior, puede abrir cualquier archivo existente que tenga un nombre largo de archivo o directorio en su ruta (es decir, más grande que el estándar 8.3 del DOS) usando cualquier modo de apertura − leer, escribir, etc. Sin embargo, si intenta crear un archivo ***nuevo*** (open con "w" o "a" y el archivo no existe aún) entonces se truncará el nombre al estilo 8.3. En una futura versión, esperamos soportar la creación de archivos con nombres largos.
WIN32, Linux and FreeBSD: Se soportan completamente los nombres largos en lectura, escritura y creación de archivos.

Ejemplo:

```
integer file_num, file_num95
sequence first_line
constant ERROR = 2

file_num = open("miarchivo", "r")
if file_num = -1 then
    puts(ERROR, "No se puede abrir miarchivo\n")
else
    first_line = gets(file_num)
end if

file_num = open("PRN", "w") -- abre la impresora para salida

-- en Windows 95:
file_num95 = open("nombreDeDirectorio\\nombreDeArchivo.abcdefg", "r")
if file_num95 != -1 then
    puts(1, "funcionó!\n")
end if
```

Ver también: ***close***

open_dll

Plataforma: **WIN32, Linux, FreeBSD**

Sintaxis:
include dll.e
a = open_dll(st)

Descripción: Abre un archivo de librería de enlace dinámico de ***Windows*** (**.**dll), o un archivo de librería compartida de ***Linux*** o ***FreeBSD***. Se devolverá una dirección de 32 bits, o 0 si no se puede encontrar el archivo. 'st' puede ser una ruta relativa o absoluta. ***Windows*** usará la ruta normal para ubicar los archivos **.**dll.

Comentarios: El valor devuelto por ***open_dll()*** se puede pasar a ***define_c_proc()***, ***define_c_func()***, o ***define_c_var()***.
Se puede abrir varias veces el mismo archivo **.**dll o **.**so. No se usa memoria extra y cada vez se devolverá el mismo número.

Euphoria cerrará la .dll automáticamente al final de la ejecución.

Ejemplo:

```
atom user32
user32 = open_dll("user32.dll")
if user32 = 0 then
    puts(1, "No se puede abrir user32.dll!\n")
end if
```

Ver también: ***define_c_func***, ***define_c_proc***, ***define_c_var***, ***c_func***, ***c_proc***, **platform.doc**

or_bits

Sintaxis: x3 = or_bits(x1, x2)

Descripción: Ejecuta la operación lógica OR sobre los bits correspondientes de 'x1' y 'x2'. Un bit en 'x3' valdrá 1 cuando el bit correspondiente tanto de 'x1' o 'x2' valga 1.

Comentarios: Los argumentos para la función pueden ser átomos o secuencias. Se aplican las reglas para operaciones sobre secuencias.
Los argumentos tienen que ser representables como números de 32 bits, sean con o sin signo.

Si se prepone manipular valores 32−bit completos, debería declarar sus variables como **átomos**, en lugar de enteros. El tipo entero de Euphoria está limitado a 31 bits.

Los resultados se tratan como números con signo. Serán negativos cuando el bit de mayor orden sea 1.

Ejemplo 1:

```
a = or_bits(#0F0F0000, #12345678)
-- a es #1F3F5678
```

Ejemplo 2:

```
a = or_bits(#FF, {#123456, #876543, #2211})
-- a es {#1234FF, #8765FF, #22FF}
```

Ver también: ***and_bits***, ***xor_bits***, ***not_bits***, ***int_to_bits***

palette

Plataforma: DOS32

Sintaxis: include graphics.e
x = palette(i, s)

Descripción: Cambia el color para el número de color 'i' a 's', donde 's' es una secuencia de intensidades de color RGB: {rojo, verde, azul}. Cada valor en 's' puede ir de 0 a 63. Si la operación resulta existosa, se devolverá una secuencia de 3 elementos conteniendo el color previo de 'i' y todos los píxeles en pantalla de color 'i' cambiarán al nuevo color. En cambio, si la operación falla, se devolverá el átomo −1.

Ejemplo:

```
x = palette(0, {15, 40, 10})
-- número de color 0 (normalmente negro) cambia a un matiz verdoso.
```

Ver también: *all_palette*

peek

Sintaxis: i = peek(a)
o ...
s = peek({a, i})

Descripción: Devuelve un valor de un byte en el rango de 0 a 255 desde la dirección 'a' de la máquina, o una secuencia conteniendo 'i' bytes consecutivos, comenzando por la dirección 'a' en la memoria.

Comentarios: Como las direcciones son números de 32 bits, pueden ser mayores que el entero más grande (31 bits). Las variables que mantengan direcciones tendrán que ser declaradas como **átomos**.
Es más rápido leer varios bytes de una vez usando el segundo método de ***peek()***, que leer de a un byte por vez dentro de un ciclo.

Recuerde que ***peek()*** sólo toma un argumento, el cual en el segundo método se trata realmente de una secuencia de 2 elementos.

Ejemplo: Los siguientes son equivalentes:

```
-- método 1
s = {peek(100), peek(101), peek(102), peek(103)}

-- método 2
s = peek({100, 4})
```

Ver también: *poke*, *peek4s*, *peek4u*, *allocate*, *free*, *allocate_low*, *free_low*, *call*

peek4s

Sintaxis: a2 = peek4s(a1)
o ...
s = peek4s({a1, i})

Descripción: Devuelve valor de 4 bytes (32 bits) con signo en el rango de −2147483648 a +2147483647 desde la dirección de la máquina 'a1', o una secuencia conteniendo 'i' consecutivos valores de 4 bytes (32 bits) con signo, comenzando en la dirección 'a1' de memoria.

Comentarios: Como las direcciones son números de 32 bits, pueden ser demasiado grandes para el tipo entero de Euphoria, las variables que mantengan direcciones tendrán que ser declaradas como **átomos**. El mismo caso se plantea para el valor devuelto por ***peek4s(),*** el cual también tendría que ser una variable declarada como **átomo**.

Es más rápido leer varios valores de 4 bytes de una vez usando el segundo método de ***peek4s()***, que leer de a 4 bytes por vez dentro de un ciclo.

Recuerde que ***peek4s()*** sólo toma un argumento, el cual en el segundo método se trata realmente de una secuancia de 2 elementos.

Ejemplo: Los siguientes son equivalentes:

```
-- método 1
s = {peek4s(100), peek4s(104), peek4s(108), peek4s(112)}
```

```
-- método 2
s = peek4s({100, 4})
```

Ver también: ***peek4u***, ***peek***, ***poke4***, ***allocate***, ***free***, ***allocate_low***, ***free_low***, ***call***

peek4u

Sintaxis: a2 = peek4u(a1)
o ...
s = peek4u({a1, i})

Descripción: Devuelve un valor de 4 bytes (32 bits) sin signo en el rango de 0 a 4294967295 desde la dirección de la máquina 'a1', o una secuencia conteniendo 'i' valores de 4 bytes sin signo consecutivos, comenzando por la dirección 'a1' en memoria.

Comentarios: Como las direcciones son números de 32 bits, pueden ser demasiado grandes para el tipo entero de Euphoria, las variables que mantengan direcciones tendrán que ser declaradas como **átomos**. El mismo caso se plantea para el valor devuelto por ***peek4u(),*** el cual también tendría que ser una variable declarada como **átomo**.

Es más rápido leer varios valores de 4 bytes de una vez usando el segundo método de ***peek4u()***, que leer de a 4 bytes por vez dentro de un ciclo.

Recuerde que ***peek4u()*** sólo toma un argumento, el cual en el segundo método se trata realmente de una secuancia de 2 elementos.

Ejemplo: Los siguientes son equivalentes:

```
-- método 1
s = {peek4u(100), peek4u(104), peek4u(108), peek4u(112)}

-- método 2
s = peek4u({100, 4})
```

Ver también: ***peek4s***, ***peek***, ***poke4***, ***allocate***, ***free***, ***allocate_low***, ***free_low***, ***call***

PI

Sintaxis: include misc.e
PI

Descripción: ***PI*** (3.14159...) ha sido definido como una constante global.

Comentarios: Se usaron los dígitos suficientes para lograr la máxima exactitud posible para un átomo de Euphoria.

Ejemplo:

```
x = PI  -- x es 3.14159...
```

Ver también: ***sin***, ***cos***, ***tan***

pixel

Plataforma: **DOS32**

Sintaxis: pixel(x1, s)

Descripción: Establece uno o más píxeles en una pantalla gráfica de píxeles, comenzando en el punto 's', donde 's' es una coordenada de pantalla de 2 elementos {x, y}. Si 'x1' es un átomo, un píxel se establecerá al color indicado por 'x1'. Si 'x1' es una secuencia, entonces se establecerá una cantidad de píxeles, comenzando en 's' y moviéndose hacia la derecha (incrementando el valor de x, lo mismo para el calor de y).

Comentarios: Cuando 'x1' es una secuencia, se utiliza un algoritmo muy veloz para ubicar los píxeles en pantalla. Es mucho más veloz que llamar una vez a ***pixel()***, con una secuencia de colores de píxeles, que llamarla varias veces, pintando el color del píxel de a uno por vez.
En gráficos de modo 19, ***pixel()*** está altamente optimizado.

Cualquier píxel que quede fuera de pantalla será seguramente recortado.

Ejemplo 1:

```
pixel(BLUE, {50, 60})
-- el punto {50,60} se establece al color BLUE (azul)
```

Ejemplo 2:

```
pixel({BLUE, GREEN, WHITE, RED}, {50,60})
-- {50,60} establecido a BLUE (azul)
-- {51,60} establecido a GREEN (verde)
-- {52,60} establecido a WHITE (blanco)
-- {53,60} establecido a RED (rojo)
```

Ver también: ***get_pixel***, ***graphics_mode***

platform

Sintaxis: i = platform()

Descripción: ***platform()*** es una función interna del intérprete. Indica la plataforma en la que se está ejecutando el programa: **DOS32**, **WIN32**, **Linux** o **FreeBSD**.

Comentarios: Cuando se está ejecutando **ex.exe**, la plataforma es ***DOS32***. Cuando se está ejecutando**exw.exe** la plataforma es ***WIN32***. Cuando se está ejecutando **exu** la plataforma es ***Linux*** (o ***FreeBSD***).
El archivo de inclusión **misc.e** contiene las siguiente constantes:

```
global constant DOS32 = 1,
                WIN32 = 2,
                LINUX = 3,
                FREEBSD = 3
```

Use ***platform()*** cuando quiera ejecutar código diferente según la plataforma en la que el programa se ejecute.
Se agregarán plataformas adicionales en la medida que Euphoria sea portado a nuevas máquinas y sistemas operativos.

La llamada a ***platform()*** no cuesta nada. Está optimizada en tiempo de compilación para los valores enteros adecuados: 1, 2 o 3.

Ejemplo 1:

```
if platform() = WIN32 then
    -- llama a la rutina Beep del sistema
    err = c_func(Beep, {0,0})
elsif platform() = DOS32 then
    -- hace el beep
    sound(500)
    t = time()
    while time() <t + 0.5 do
    end while
    sound(0)
else
    -- no hace nada (Linux/FreeBSD)
end if
```

Ver también: **platform.doc**

poke

Sintaxis: poke(a, x)

Descripción: Si 'x' es un átomo, escribe un valor de un byte en la dirección de memoria 'a'. Si 'x' es una secuencia, escribe una secuencia de valores de un byte en posiciones consecutivas de memoria, comenzando a partir de la dirección 'a'.

Comentarios: En memoria, se almacenan realmente los 8 bits inferiores de cada byte, es decir, **remainder(x,256).**

Es más rápido escribir varios bytes por vez usando ***poke()*** con una secuencia de valores, que escribiendo un byte por vez dentro de un ciclo.

Escribir la memoria de pantalla con ***poke()*** puede ser mucho más rápido que usar ***puts()*** o ***printf()***, pero la programación es más complicada. En la mayoría de los casos, no se necesita velocidad. Por ejemplo, el editor de Euphoria (**ed**), no usa ***poke()***.

Ejemplo:

```
a = allocate(100)   -- ubica 100 bytes en memoria

-- poke() de un byte por vez:
poke(a, 97)
poke(a+1, 98)
poke(a+2, 99)

-- poke() de 3 bytes por vez:
```

```
poke(a, {97, 98, 99})
```

Programa de ejemplo: **demo\callmach.ex**

Ver también: ***peek***, ***poke4***, ***allocate***, ***free***, ***allocate_low***, ***free_low***, ***call***, **safe.e**

poke4

Sintaxis: poke4(a, x)

Descripción: Si 'x' es un átomo, escribe un valor de 4 bytes (32 bits) en la dirección de memoria 'a'. Si 'x' es una secuencia, escribe una secuencia de valores de 4 bytes en posiciones consecutivas de memoria, comenzando a partir de la dirección 'a'.

Comentarios: El o los valores a almacenarse no deben exceder los 32 bits de tamaño.

Es más rápido escribir varios valores de 4 bytes por vez usando ***poke4()*** con una secuencia de valores, que escribiendo un valor de 4 bytes por vez dentro de un ciclo.

Los valores de 4 bytes a almacenarse pueden ser positivos o negativos. Puede leerlos, tanto con ***peek4s()*** como con ***peek4u()***.

Ejemplo:

```
a = allocate(100)   -- ubica 100 bytes en memory

-- poke4() de un valor de 4 bytes por vez:
poke4(a, 9712345)
poke4(a+4, #FF00FF00)
poke4(a+8, -12345)

-- poke4() de 3 valores de 4 bytes por vez:
poke4(a, {9712345, #FF00FF00, -12345})
```

Ver también: ***peek4u***, ***peek4s***, ***poke***, ***allocate***, ***allocate_low***, ***call***

polygon

Plataforma: **DOS32**

Sintaxis: include graphics.e
polygon(i1, i2, s)

Descripción: Dibuja un polígono de 3 o más vértices dados en 's', en una pantalla de gráficos de píxel, usando un cierto color 'i1'. Si 'i2' vale 1, se rellena el área, en cambio, con 0 no.

Ejemplo:

```
polygon(GREEN, 1, {{100, 100}, {200, 200}, {900, 700}})
-- hace un triángulo sólido de color verde.
```

Ver también: ***draw_line***, ***ellipse***

position

Sintaxis: position(i1, i2)

Descripción: Posiciona el cursor en la línea 'i1', columna 'i2', donde la esquina superior izquierda es la línea 1, columna 1. El siguiente caracter que se muestra en pantalla se imprimirá en esa posición. ***position()*** informará un error si la ubicación está fuera de la pantalla.

Comentarios: ***position()*** trabaja en los modos de texto y gráfico de píxel.

El sistema de coordenadas para mostrar texto es diferente del usado para mostrar píxeles. Los píxeles se muestran de forma que eñ superior izquierdo es (x=0,y=0) y la primera coordenada controla la ubicación horizontal superior derecha. En los modos de gráfico de píxel puede mostrar tanto texto como píxeles. ***position()*** solamente establece la línea y columna para el texto que se muestra, no los píxeles que dibuja. No hay una rutina correspondiente para establecer la posición del píxel siguiente.

Ejemplo:

```
position(2,1)
-- el cursor se mueve al comienzo de la segunda línea
-- desde arriba
```

Ver también: *get_position*, *puts*, *print*, *printf*

power

Sintaxis: x3 = power(x1, x2)

Descripción: Eleva 'x1' a la potencia 'x2'.

Comentarios: Los argumentos de esta función pueden ser átomos o secuencias. Se aplican las reglas de operaciones sobre secuencias. Las potencias de 2 se calculan muy eficientemente.

Ejemplo 1:

```
? power(5, 2)
-- se imrime 25
```

Ejemplo 2:

```
? power({5, 4, 3.5}, {2, 1, -0.5})
-- se imprime {25, 4, 0.534522}
```

Ejemplo 3:

```
? power(2, {1, 2, 3, 4})
-- {2, 4, 8, 16}
```

Ejemplo 4:

```
? power({1, 2, 3, 4}, 2)
-- {1, 4, 9, 16}
```

Ver también: *log*, *sqrt*

prepend

Sintaxis: s2 = prepend(s1, x)

Descripción: Agrega 'x' al comienzo de la secuencia 's1'. La longitud de 's2' será *length(s1)* + 1.

Comentarios: Si 'x' es un átomo, es lo mismo que **s2 = x &s1**. Si 'x' es una secuencia, definitivamente no es lo mismo.
El caso en el que 's1' y 's2' son la misma variable, se maneja muy eficientemente.

Ejemplo 1:

```
prepend({1,2,3}, {0,0})   -- {{0,0}, 1, 2, 3}

-- Compare con la concatenación:

{0,0} &{1,2,3}            -- {0, 0, 1, 2, 3}
```

Ejemplo 2:

```
s = {}
for i = 1 to 10 do
    s = prepend(s, i)
end for
-- s es {10,9,8,7,6,5,4,3,2,1}
```

Ver también: *append*, operador de concatenación operador de formación de secuencias

pretty_print

Sintaxis: include misc.e
pretty_print(fn, x, s)

Descripción: Imprime a un archivo o dispositivo 'fn', un objeto 'x' usando llaves { , , , }, indentación, y varias líneas para mostrar la estructura.
Se pueden dar varias opciones en 's' para controlar la presentación. Pase {} para elegir los valores por defecto o establezca las opciones como se muestra:
[1] muestra caracteres ASCII:
* 0: nunca
* 1: junto con cualquier entero en el rango ASCII imprimible (por defecto)
* 2: muestra como "cadena" cuando todos los enteros de una secuencia están en el rango ASCII
* 3: muestra cadenas y caracteres encomillados (solo) para cualquier entero en el rango ASCII
[2] cantidad de sangría para cada nivel de anidamiento de secuencias − valor por defecto: 2
[3] columna en la que comenzamos − valor por defecto: 1
[4] columna aproximada para aplicar "wrapping" − valor por defecto: 78
[5] formato para usar con enteros − valor por defecto:"%d"

[6] formato para usar con números de puntos flotante − valor por defecto:"%.10g"
[7] mínimo valor ASCII imprimible − valor por defecto: 32
[8] máximo valor ASCII imprimible − valor por defecto: 127

Si la longitud de 's' es menor que 8, las opciones sin especificar al final de la secuencia, mantendrán sus valores por defecto. Por ejemplo, {0, 5} seleccionará "nunca mostrar caracteres ASCII", más 5 caracteres de indentación, con valores por defecto para el resto.

Comentarios: La impresión comenzará en la posición actual del cursor. Normalmente querrá llamar a ***pretty_print()*** cuando el cursor está en la columna 1 (después de imprimir un caracter \n). Si desea comenzar en una columna diferente, debería llamar a ***position()*** y especificar un valor para la opción [3]. Esto asegurará de que la primera y última llaves en una secuencia se alineen verticalmente.

Ejemplo 1:

```
pretty_print(1, "ABC", {})

{65'A',66'B',67'C'}
```

Ejemplo 2:

```
pretty_print(1, {{1,2,3}, {4,5,6}}, {})

{
  {1,2,3},
  {4,5,6}
}
```

Ejemplo 3:

```
pretty_print(1, {"Lenguaje", "de Programación", "Euphoria"}, {2})

{
  "Lenguaje",
  "de Programación",
  "Euphoria"
}
```

Ejemplo 4:

```
puts(1, "word_list = ") -- mueve el cursor a la columna 13
pretty_print(1,
             {{"Euphoria", 8, 5.3},
              {"Programming", 11, -2.9},
              {"Language", 8, 9.8}},
             {2, 4, 13, 78, "%03d", "%.3f"}) -- primeras 6 de 8 opciones

word_list = {
                {
                    "Euphoria",
                    008,
                    5.300
                },
                {
                    "Programming",
                    011,
                    -2.900
                },
                {
                    "Language",
                    008,
                    9.800
                }
            }
```

Ver también: ***?***, ***print***, ***puts***, ***printf***

print

Sintaxis: print(fn, x)

Descripción: Imprime a un archivo o dispositivo 'fn', un objeto 'x' entre llaves { , , , } para mostrar la estructura.

Ejemplo 1:

```
print(1, "ABC")  -- la salida es:  {65, 66, 67}
puts(1, "ABC")   -- la salida es:  ABC
```

Ejemplo 2:

```
print(1, repeat({10,20}, 3))
-- la salida es: {{10,20},{10,20},{10,20}}
```

Ver también: ***?***, ***pretty_print***, ***puts***, ***printf***, ***get***

printf

Sintaxis: printf(fn, st, x)

Descripción: Imprime 'x' a un archivo o dispositivo 'fn', usando la cadena de formato 'st'. Si 'x' es un átomo, entonces se imprimirá un valor simple. Si 'x' es una secuencia, el formato en 'st' se aplicará a los ***elementos sucesivos*** de 'x'. Así ***printf()*** siempre toma 3 argumentos exactamente. Solamente variará la longitud del último argumento que contiene los valores a imprimir. Los formatos básicos son:
%d − imprime un átomo como un entero decimal
%x − imprime un átomo como un entero hexadecimal
%o − imprime un átomo como un entero octal
%s − imprime una secuencia como una cadena de caracteres o un átomo como un solo caracter
%e − imprime un átomo como un número de punto flotante con notación científica (exponencial)
%f − imprime un átomo como un número de punto flotante con punto decimal, sin exponente
%g − imprime un átomo como un número de punto flotante usando el formato %f o %e según sea el más adecuado
%% − imprime el caracter '%' en sí mismo

Se pueden agregar los anchos de los campos al formato básico, por ejemplo, %5d, %8.2f, %10.4s. El número antes del punto decimal es el ancho mínimo del campo a usarse. El número después del punto decimal es la presición a utilizarse.

Si el ancho del campo es negativo, por ejemplo, %−5d entonces el valor estará alineado a la izquierda dentro del campo. Normalmente estará alineado a la derecha. Si el ancho del campo comienza con un 0, por ejemplo, %08d se suministrarán ceros para completar el campo. Si el ancho del campo comienza con un '+', por ejemplo, %+7d entonces el signo + se imprimirá para los valores positivos.

Comentarios: Observe el siguiente error común:

```
nombre="Juan Pérez"
printf(1, "%s", nombre)     -- error!
```

Esto imprimirá solo el primer caracter del nombre (J), ya que cada elemento del nombre se toma para ser un valor separado que se ajustará a formato. Debería decir en su lugar:

```
nombre="Juan Pérez"
printf(1, "%s", {nombre})   -- correcto
```

Ahora, el tercer argumento de ***printf()*** es una secuencia de un elemento que contiene el ítem a formatearse.

Ejemplo 1:

```
tasa = 7.875
printf(miarchivo, "La tasa de interés es: %8.2f\n", tasa)

      La tasa de interés es:     7.88
```

Ejemplo 2:

```
nombre="Juan Pérez"
puntos=97
printf(1, "%15s, %5d\n", {nombre, puntos})

      Juan Pérez,    97
```

Ejemplo 3:

```
printf(1, "%-10.4s $ %s", {"ABCDEFGHIJKLMNOP", "XXX"})

      ABCD       $ XXX
```

Ver también: ***sprintf***, ***puts***, ***open***

profile

Sintaxis: profile(i)

Descripción: Habilita o deshabilita el análisis de perfiles en tiempo de ejecución. Trabaja tanto por **conteo de ejecución** como por **análisis por tiempo**. Si 'i' vale 1, entonces se habilitará el análisis y se grabarán las muestras/conteos. Si 'i' vale 0, el análisis estará deshabilitado y no habrá registración de ninguna naturaleza.

Comentarios: Después de una sentencia "**with profile**" o "**with profile_time**", el análisis se habilita en forma automática. Use ***profile(0)*** para desactivalo. Use ***profile(1)*** para activarlo nuevamente cuando la ejecución alcanza el código sobre el que desea enfocar el análisis.

Ejemplo 1:

```
with profile_time
profile(0)
  ...
procedure slow_routine()
profile(1)
```

```
  ...
profile(0)
end procedure
```

Ver también: ***trace***, análisis de perfiles de ejecución, sentencias especiales de alto nivel

prompt_number

Sintaxis: include get.e
a = prompt_number(st, s)

Descripción: Le pide al usuario que ingrese un número. 'st' es una cadena de texto que se mostrará en pantalla. 's' es una secuencia de dos valores {inferior, superior} que determina el rango de valores que el usuario puede ingresar. Si el usuario ingresa un número que está fuera de este rango, se le pedirá nuevamente el ingreso. 's' puede estar vacía, {}, si no hay restricciones.

Comentarios: Si esta rutina es demasiado simple para sus necesidades, sientase libre de copiarla y hacer una versión propia más especializada.

Ejemplo 1:

```
edad = prompt_number("¿Cuál es su edad? ", {0, 150})
```

Ejemplo 2:

```
t = prompt_number("Ingrese una temperatura en ºC:\n", {})
```

Ver también: ***get***, ***prompt_string***

prompt_string

Sintaxis: include get.e
s = prompt_string(st)

Descripción: Le pide al usuario que ingrese una cadena de texto. 'st' es una cadena que se mostrará en pantalla. La cadena que el usuario escriba se devolverá como una secuencia, sin ningún caracter de nueva línea.

Comentarios: Si el usuario escribe Ctrl+Z (indicación de fin de archivo), se devolverá "".

Ejemplo:

```
nombre = prompt_string("¿Cuál es su nombre? ")
```

Ver también: ***gets***, ***prompt_number***

put_screen_char

Plataforma: **DOS32, Linux, FreeBSD**

Sintaxis: include image.e
put_screen_char(i1, i2, s)

Descripción: Escribe cero o más caracteres en pantalla junto con sus atributos. 'i1' especifica la línea, e 'i2' la columna donde se debería escribir el primer caracter. La secuencia 's' se ve como: {código ASCII1, atributo1, código ASCII2, atributo2, ...}. Cada par de elementos en 's' describe un caracter. El átomo "código ASCII" contiene el código ASCII del caracter. El átomo de atributos, contiene el color del texto, color del fondo y posiblemente otra información dependiente de la plaraforma que controle la manera en que se muestra el caracter en pantalla.

Comentarios: La longitud de 's' tiene que ser múltiplo de 2. Si 's'tiene longitud 0, no se escribirá nada en la pantalla.
Es más rápido escribir varios caracteres en la pantalla con una llamada simple a ***put_screen_char()*** que escribir un caracter por vez.

Ejemplo:

```
-- escribe AZ en la esquina superior izquierda de la pantalla
-- (los atributos son dependientes de la plataforma)
put_screen_char(1, 1, {'A', 152, 'Z', 131})
```

Ver también: ***get_screen_char***, ***display_text_image***

puts

Sintaxis: puts(fn, x)

Descripción: Envía a un archivo o dispositivo 'fn', un byte único (átomo) o una secuencia de bytes. Los 8 bits de orden inferior de cada valor son los realmente enviados. Si 'fn' está en pantalla, verá los caracteres mostrados.

Comentarios: Cuando envía una secuencia de bytes, ésta no tiene que tener ninguna (sub)secuencia dentro de ella. Tiene que ser únicamente una **secuencia de átomos** (Típicamente una cadena de códigos ASCII).
Evite enviar 0 a la pantalla o salida estándar. Su salida se puede truncar.

Ejemplo 1:

```
puts(SCREEN, "Ingrese su nombre: ")
```

Ejemplo 2:

```
puts(output, 'A')  -- el byte 65 se enviará a output
```

Ver también: ***printf***, ***gets***, ***open***

rand

Sintaxis: x2 = rand(x1)

Descripción: Devuelve un entero al azar desde 1 a 'x1', donde 'x1' puede ir desde 1 al valor positivo más grande del tipo entero (1073741823).

Comentarios: Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

Ejemplo:

```
s = rand({10, 20, 30})
-- s tiene que ser: {5, 17, 23} o {9, 3, 12} etc.
```

Ver también: ***set_rand***

read_bitmap

Sintaxis: include image.e
x = read_bitmap(st)

Descripción: 'st' es el nombre de un archivo de mapa de bits (.bmp). El archivo debería estar en formato de mapa de bits. Están soportadas la mayoría de las variaciones del formato. Si se lee exitosamente el archivo, el resultado será una secuencia de 2 elementos. El primer elemento es la paleta, conteniendo valores de intensidad en el rango de 0 a 255. El segundo elementos es una secuencia 2D de secuencias que contiene una imagen de gráficos de píxel. Puede pasar la paleta a ***all_palette()*** (después de dividirla por 4 para escalarla). La imagen se puede pasar a ***display_image()***.
Están soportados los mapas de bits de 2, 4, 16 o 256 colores. Si el archivo no tiene el formato correcto, se devolverá en su lugar un código de error (átomo):

```
global constant BMP_OPEN_FAILED = 1,
                BMP_UNEXPECTED_EOF = 2,
           BMP_UNSUPPORTED_FORMAT = 3
```

Comentarios: Puede crear su propio archivo de imagen de mapa de bits usando el Paintbrush de Windows y algunos otros programas gráficos. También puede incorporar estas imágenes dentro de sus programas Euphoria.

Ejemplo:

```
x = read_bitmap("c:\\windows\\arcade.bmp")
-- nota: se necesita la doble barra para obtener
-- una barra simple en la cadena
```

Programa de ejemplo: **demo\dos32\bitmap.ex**

Ver también: ***palette***, ***all_palette***, ***display_image***, ***save_bitmap***

register_block

Sintaxis: include machine.e (or safe.e)
register_block(a, i)

Descripción: Agrega un bloque de memoria a la lista de bloques seguros mantenida por **safe.e** (la versión de depuración de **machine.e**). El bloque comienza en la dirección 'a'. La longitud del bloque es de 'i' bytes.

Comentarios:

Esta rutina solamente tiene sentido para usarla con **propósitos de depuración.** **safe.e** rastrea los bloques de memoria en los que a su programa se le permite ejecutar ***peek()***, ***poke()***, ***mem_copy()***, etc. Normalmente, estos son solo los bloques que se asignaron usando las rutinas Euphoria ***allocate()*** o ***allocate_low()***, y que no se liberaron usando las rutinas Euphoria ***free()*** o ***free_low()***. En algunos casos, puede adquirir bloques de memoria adicionales y externos, quizás como resultado de llamar una rutina de C. Si está depurando su programa usando **safe.e**, usted debe registrar estos bloques externos de memoria o safe.e evitará que usted los pueda acceder. Use ***unregister_block()*** para desregistrar un bloque externo cuando haya terminado de usarlo.
Al incluir **machine.e**, obtendrá distintas versiones de ***register_block()*** y ***unregister_block()*** que no hacen nada. Esto permite ir y venir rápidamente entre la ejecución y la depuración de su programa.

Ejemplo 1:

```
atom addr

addr = c_func(x, {})
register_block(addr, 5)
poke(addr, "ABCDE")
unregister_block(addr)
```

Ver también: ***unregister_block***, **safe.e**

remainder

Sintaxis: x3 = remainder(x1, x2)

Descripción: Calcula el resto de la división de 'x1' por 'x2'. El resultado tendrá siempre el mismo signo que 'x1' y la magnitud será menor que 'x2'.

Comentarios: Los argumentos para esta función pueden ser átomos o secuencias. Se aplican las reglas para operaciones sobre secuencias.

Ejemplo 1:

```
a = remainder(9, 4)
-- a es 1
```

Ejemplo 2:

```
s = remainder({81, -3.5, -9, 5.5}, {8, -1.7, 2, -4})
-- s es {1, -0.1, -1, 1.5}
```

Ejemplo 3:

```
s = remainder({17, 12, 34}, 16)
-- s es {1, 12, 2}
```

Ejemplo 4:

```
s = remainder(16, {2, 3, 5})
-- s es {0, 1, 1}
```

Ver también: ***floor***

repeat

Sintaxis: s = repeat(x, a)

Descripción: Crea una secuencia de longitud 'a', donde cada elemento es 'x'.

Comentarios: Cuando se repite una secuencia o un número de punto flotante, el intérprete realmente no hace múltiples copias en memoria, sino que una sola copia "se apunta a" una cantidad de veces.

Ejemplo 1:

```
repeat(0, 10)      -- {0,0,0,0,0,0,0,0,0,0}
```

Ejemplo 2:

```
repeat("JUAN", 4)  -- {"JUAN", "JUAN", "JUAN", "JUAN"}
-- El intérprete creará solamente una copia de "JUAN" en memoria
```

Ver también: ***append***, ***prepend***, operador de formación de secuencias

reverse

Sintaxis: include misc.e
s2 = reverse(s1)

Descripción: Invierte el orden de los elementos de una secuencia.

Comentarios:

Se crea una nueva secuencia, donde los elementos aparecen en orden inverso, en relación a la secuencia original.

Ejemplo 1:

```
reverse({1,3,5,7})          -- {7,5,3,1}
```

Ejemplo 2:

```
reverse({{1,2,3}, {4,5,6}}) -- {{4,5,6}, {1,2,3}}
```

Ejemplo 3:

```
reverse({99})               -- {99}
```

Ejemplo 4:

```
reverse({})                 -- {}
```

Ver también: ***append***, ***prepend***, ***repeat***

routine_id

Sintaxis: i = routine_id(st)

Descripción: Devuelve un número de identificación entero, conocido como **routine id**, para una función o procedimiento Euphoria definido por el usuario. El nombre de la función o procedimiento está dado por la secuencia de cadena 'st'. Si no puede encontrarse la rutina, se devuelve −1.

Comentarios: El número de identificación puede pasarse a ***call_proc()*** o ***call_func()***, para llamar indirectamente a la rutina mencionada por 'st'.
La rutina mencionada por 'st' debe estar visible, es decir llamable, en el lugar donde se usa ***routine_id()*** para obtener su número de identificación. Las llamadas indirectas a la rutina pueden aparecer en el programa antes que la definición de la rutina, pero el número de identificación se tiene que obtener *después* de difinir la rutina. Ver el ejemplo 2 más abajo.

Una vez obtenido un **routine id** válido, se lo puede usar en ***cualquier*** lugar del programa para llamar aindirectamente a una rutina mediante ***call_proc()***/***call_func()***.

Algunos usos típicos de ***routine_id()*** son:

1. Llamar a una rutina que está definida más adelante en el programa.
2. Crear una subrutina que toma como parámetro a otra rutina. (Ver el ejemplo 2 más abajo)
3. Usar una secuencia de **routine id's** para armar una sentencia case (switch).
4. Iniciar un sistema Orientado a Objetos.
5. Obteniendo el **routine id,** puede pasárselo a ***call_back()***. (Ver **platform.doc**)

Observe que las rutinas C, llamables por Euphoria, también tienen un **routine id**. Ver ***define_c_proc()*** y ***define_c_func()***.

Ejemplo 1:

```
procedure foo()
    puts(1, "Hola Mundo\n")
end procedure

integer foo_num
foo_num = routine_id("foo")

call_proc(foo_num, {})  -- es lo mismo que llamar a foo()
```

Ejemplo 2:

```
function apply_to_all(sequence s, integer f)
    -- aplica una función a todos los elementos de una secuencia
    sequence result
    result = {}
    for i = 1 to length(s) do
        -- podemos llamar a add1() aquí, aunque venga después en el programa
        result = append(result, call_func(f, {s[i]}))
    end for
    return result
end function

function add1(atom x)
    return x + 1
end function

-- add1() está visible aquí, por lo que podemos pedir su routine id
? apply_to_all({1, 2, 3}, routine_id("add1"))
-- muestra {2,3,4}
```

Ver también: ***call_proc***, ***call_func***, ***call_back***, ***define_c_func***, ***define_c_proc***, **platform.doc**

save_bitmap

Sintaxis: include image.e
i = save_bitmap(s, st)

Descripción: Crea un fichero de mapa de bits (.bmp) desde una secuencia 's' de 2 elementos. 'st' es el nombre del fichero de mapa de bits (bitmap) . s[1] es la paleta:
{{r,g,b}, {r,g,b}, ..., {r,g,b}}

Cada valor rojo, verde o azul está dentro del rango 0 a 255. s[2] es una secuencia 2D de secuencias, conteniendo una imagen gráfica de píxeles. Las secuencias contenidas en s[2] tienen que tener la misma longitud. 's' tiene el mismo formato que el valor devuelto por ***read_bitmap()***.

El resultado será uno de los siguientes códigos:

```
global constant BMP_SUCCESS = 0,
                BMP_OPEN_FAILED = 1,
               BMP_INVALID_MODE = 4 -- modo gráfico o argumento inválido
```

Comentarios: Si usa ***get_all_palette()*** para obtener la paleta antes de llamar a esta función, tiene que multiplicar las intensidades devueltas por 4 antes de llamar a ***save_bitmap()***.
Podría usar ***save_image()*** para obtener la imagen 2D para s[2].

save_bitmap() produce mapas de bits de 2, 4, 16, o 256 colores y se pueden leer con ***read_bitmap()***. Windows Paintbrush y algunas otras herramientas no soportan mapas de bits de 4 colores.

Ejemplo:

```
paletteData = get_all_palette() * 4
code = save_bitmap({paletteData, imageData}, "c:\\example\\a1.bmp")
```

Ver también: ***save_image***, ***read_bitmap***, ***save_screen***, ***get_all_palette***

save_image

Plataforma: **DOS32**

Sintaxis: include image.e
s3 = save_image(s1, s2)

Descripción: Guarda una imagen rectangular desde una pantalla de gráficos de píxel. El resultado es una secuencia 2D de secuencias, conteniendo todos los píxeles de la imagen. Puede volver a mostrar la imagen con ***display_image()***. 's1' es una secuencia de 2 elementos {x1, y1} que especifica el píxel superior izquierdo de la imagen. 's2' es una secuencia {x2,y2} que especifica el pìxel inferior derecho.

Ejemplo:

```
s = save_image({0,0}, {50,50})
display_image({100,200}, s)
display_image({300,400}, s)
-- guarda una imagen cuadrada de 51x51, entonces vuelve a mostrarla en
-- {100,200} y en {300,400}
```

Ver también: ***display_image***, ***save_text_image***

save_screen

Plataforma: **DOS32**

Sintaxis: include image.e
i = save_screen(x1, st)

Descripción: Guarda la pantalla entera o una región rectangular de ella como un fichero de mapa de bits de (.bmp). Para guardar la pantalla entera, pase el entero 0 a 'x1'. PAra guardar una región rectangular de pantalla, 'x1' deberá ser una secuencia de 2 secuencias: {{topLeftXPixel, topLeftYPixel}, {bottomRightXPixel, bottomRightYPixel}}
'st' es el nombre del fichero de mapa de bits .bmp.

El resultado será uno de los siguientes códigos:

```
global constant BMP_SUCCESS = 0,
                BMP_OPEN_FAILED = 1,
               BMP_INVALID_MODE = 4 -- modo gráfico o argumento inválido
```

Comentarios: ***save_screen()*** produce mapas de bits de 2, 4, 16, o 256 colores y se pueden leer con ***read_bitmap()***. Windows Paintbrush y algunas otras herramientas no soportan mapas de bits de 4 colores.
save_screen() solo trabaja en los modos de gráficos de píxel, no en los modos de texto.

Ejemplo 1:

```
-- guarda la pantalla entera:
code = save_screen(0, "c:\\example\\a1.bmp")
```

Ejemplo 2:

```
-- guarda una parte de la pantalla:
err = save_screen({{0,0},{200, 15}}, "b1.bmp")
```

Ver también: ***save_image***, ***read_bitmap***, ***save_bitmap***

save_text_image

Plataforma: **DOS32, Linux, FreeBSD**

Sintaxis: include image.e
s3 = save_text_image(s1, s2)

Descripción: Guarda una región rectangular de texto de una pantalla de modo de texto. El resultado es una secuencia de secuencias conteniendo caracteres ASCII y atributos de la pantalla. Puede volver a mostrar este texto usando ***display_text_image()***. 's1' es una secuencia de 2 elementos {línea1, columna1} que especifica el caracter superior izquierdo. 's2' es una secuencia {línea2, columna2} que especifica el caracter inferior derecho.

Comentarios: Debido a que los atributos del caracter también se guardan, obtendrá el correcto color del texto, color de fondo y otras propiedades para cada caracter cuando vuelva a mostrar el texto.
En ***DOS32***, un byte de atributo está formado por 2 campos de 4 bits que codifican los colores de frente y fondo del caracter. Los 4 bits de orden superior determinan el color de fondo, mientras los 4 bits de orden inferior determinan el color del texto.

Esta rutina solamente funciona en los modos de texto.

Puede ser que utilice esta función en una interfaz gráfica del usuario de modo de texto para guardar una porción de la pantalla antes de exhibir un menú desplegable, una caja de diálogo, una caja de alerta, etc.

En ***DOS32***, si usted pasando las páginas de video, observe que esta función lee en la página activa actual.

Ejemplo: Si las 2 líneas superiores de la pantalla tienen:

```
Hola
Mundo
```

Y ejecuta:

```
s = save_text_image({1,1}, {2,5})
```

Entonces 's' es algo como:

```
    {"H-o-l-a-", "M-u-n-d-o-"}
donde '-' indica los bytes de atributos
```

Ver también: ***display_text_image***, ***save_image***, ***set_active_page***, ***get_screen_char***

scroll

Sintaxis: include graphics.e
scroll(i1, i2, i3)

Descripción: Desplaza una región de texto en la pantalla, tanto hacia arriba ('i1' positivo) o hacia abajo ('i1' negativo) por 'i1' líneas. La región es la serie de líneas de pantalla desde 'i2' (línea superior) a 'i3' (línea inferior) inclusive. Aparecerán nuevas líneas en blanco por encima o por debajo.

Comentarios: Podría ejecutar la operación de desplazamiento usando una serie de llamadas a ***puts()***, pero ***scroll()*** es mucho más rápido. La posición del cursor después del desplazamiento no está definida.

Programa de ejemplo: **bin\ed.ex**

Ver también: ***clear_screen***, ***text_rows***

seek

Sintaxis: include file.e
i1 = seek(fn, i2)

Descripción: Busca (mueve) cualquier posición de byte en el fichero 'fn', o al final del fichero si 'i2' es −1. Para cada fichero abierto hay una posición actual de byte que se actualiza como resultado de las operaciones de E/S en el fichero. La posición inicial en el fichero es 0 para ficheros abiertos para lectura, escritura o actualización. La posición inicial es el fin del fichero para los ficheros abiertos para agregar (append). El valor devuelto por ***seek()*** es 0 si la búsqueda fue exitosa, y no cero si falló. Es posible que ***seek()*** se pase de la marca de fin de fichero. Si busca más allá del fin de un fichero, y escribe algún dato, los bytes indefinidos se insertarán en la brecha entre el fin de fichero original y los nuevos datos.

Comentarios: Después de buscar y leer (o escribir) una serie de bytes, puede necesitar llamar a ***seek()*** explícitamente antes de cambiar a escribir (o leer) bytes, aunque la posición del fichero debe ya ser la que usted desea. Esta función se usa normalmente con ficheros abiertos en modo binario. En el modo de texto, DOS convierte los CR LF a LF al leer, y LF a CR LF al guardar, lo cuál puede causar la gran confusión cuando usted está intentando contar los bytes.

Ejemplo:

```
include file.e

integer fn
fn = open("midata", "rb")
-- leer y muestra la primera línea del archivo 3 veces:
for i = 1 to 3 do
    puts(1, gets(fn))
    if seek(fn, 0) then
        puts(1, "el rebobinado falló!\n")
    end if
end for
```

Ver también: ***where***, ***open***

sequence

Sintaxis: i = sequence(x)

Descripción: Devuelve 1 si 'x' es una secuencia, en caso contrario devuelve 0.

Comentarios: Sirve para definir el tipo secuencia (sequence). También puede llamarla como una función ordinalria para determinar si un objeto es una secuencia.

Ejemplo 1:

```
sequence s
s = {1,2,3}
```

Ejemplo 2:

```
if sequence(x) then
    sum = 0
    for i = 1 to length(x) do
        sum = sum + x[i]
    end for
else
    -- x tiene que ser un átomo
    sum = x
end if
```

Ver también: ***atom***, ***object***, ***integer***, atoms and sequences

set_active_page

Plataforma: **DOS32**

Sintaxis: include image.e
set_active_page(i)

Descripción: Selecciona la página de video 'i' para enviarla a pantalla.

Comentarios: Con páginas múltiples puede cambiar instantáneamente la pantalla entera sin causar ningún "parpadeo visible". También puede guardar la pantalla y traerla detrás rápidamente.
video_config() le dirá cuántas páginas están disponibles en el modo de gráficos actual.

Por defecto, la página activa y la página mostrada son ambas 0.

Esto trabaja bajo DOS, o en una ventana de pantalla completa del DOS. En una ventana de pantalla parcial usted no puede cambiar la página activa.

Ejemplo:

```
include image.e

-- las páginas activa &mostrada son ambas inicialmente 0
puts(1, "\nEsto es la página 0\n")
set_active_page(1)     -- la salida de pantalla ahora irá a paginar 1
clear_screen()
puts(1, "\nAhora hemos cambiado a la página 1\n")
if getc(0) then        -- wait for key-press
end if
set_display_page(1)    -- "Ahora hemos ..." se hace visible
if getc(0) then        -- espera la presión de una tecla
end if
set_display_page(0)    -- "Esto es ..." se hace visible otra vez
set_active_page(0)
```

Ver también: ***get_active_page***, ***set_display_page***, ***video_config***

set_display_page

Plataforma: **DOS32**

Sintaxis: include image.e
set_display_page(i)

Descripción: Selecciona la página de video 'i' para mapearla en la pantalla visible.

Comentarios: Con páginas múltiples puede cambiar instantáneamente la pantalla entera sin causar ningún "parpadeo visible". También puede guardar la pantalla y traerla detrás rápidamente.
video_config() le dirá cuántas páginas están disponibles en el modo de gráficos actual.

Por defecto, la página activa y la página mostrada son ambas 0.

Esto trabaja bajo DOS, o en una ventana de pantalla completa del DOS. En una ventana de pantalla parcial usted no puede cambiar la página activa.

Ejemplo: Ver el ejemplo de ***set_active_page()***.

Ver también: ***get_display_page***, ***set_active_page***, ***video_config***

set_rand

Sintaxis: include machine.e
set_rand(i1)

Descripción: Establece el generador de números aleatorios en el estado 'i1', de modo que usted consiga una serie conocida de números al azar en llamadas subsecuentes a ***rand()***.

Comentarios: Normalmente los números devueltos por la función ***rand()*** son toalmente impredecibles, y serán diferentes cada vez que se ejecute el programa. A veces, sin embargo, puede desear repetir la misma serie de números, quizás porque está tratando de depurar el programa, o tal vez la capacidad de generar la misma salida (por ejemplo, una imagen al azar) por pedido del usuario.

Ejemplo:

```
sequence s, t
s = repeat(0, 3)
t = s

set_rand(12345)
s[1] = rand(10)
s[2] = rand(100)
s[3] = rand(1000)

set_rand(12345)  -- el mismo valor para set_rand()
t[1] = rand(10)  -- los mismos argumentos para rand() como antes
t[2] = rand(100)
t[3] = rand(1000)
-- en este punto, 's' y 't' serán idénticos
```

Ver también: ***rand***

set_vector

Plataforma: **DOS32**

Sintaxis: include machine.e
set_vector(i, s)

Descripción: Establece a 's' como la nueva dirección para el manejo de la interrupción número 'i'. 's' tiene que ser una dirección lejana (far) de modo protegido de la forma: {segmento de 16 bits, offset de 32 bits}.

Comentarios: Antes de llamar a ***set_vector()***, tiene que guardar la rutina de manejo de interrupciones de código de máquina en la posición 's' de memoria.
El segmento de 16 bits puede ser el segmento de código usado por Euphoria. Para obtener el valor de este segmento, vea **demo\dos32\hardint.ex**. El offset puede ser el valor de 32 bits devuelto por ***allocate()***. Euphoria corre en **modo protegido** con el segmento de código y el segmento de datos apuntando a la misma memoria física, pero con diferentes modos de acceso.

Las interrupciones que ocurren tanto en **modo real** como en **modo protegido** se pasarán a su manejador. Su manejador de interrupciones debería cargar inmediatamente el segmento de datos correcto antes de intentar referenciar la memoria.

Su manejador tednría que regresar de la interrupción usando la instrucción iretd, o saltar al manejador de interrupciones original. Este debería guardar y restablecer cualquier registro que se modifique.

Debería bloquear la memoria usada por su manejador, para asegurar que no ocurra un intercambio. Ver ***lock_memory()***.

Es altamente recomendable que estudie **demo\dos32\hardint.ex** antes de intentar inicializar su propio manejador.

Antes de intentar escribir su propio manejador, debería tener un buen conocimiento de programación a nivel de máquina.

Puede llamar a ***set_vector()*** con la dirección lejana (far) devuelta por ***get_vector()***, cuando quiera restablecer el manejador original.

Ejemplo:

```
set_vector(#1C, {code_segment, my_handler_address})
```

Programa de ejemplo: **demo\dos32\hardint.ex**

Ver también: ***get_vector***, ***lock_memory***, ***allocate***

sin

Sintaxis: x2 = sin(x1)

Descripción: Devuelve el seno de 'x1', donde 'x1' está expresado en radianes.

Comentarios: Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia

Ejemplo:

```
sin_x = sin({.5, .9, .11})
-- sin_x es {.479, .783, .110}
```

Ver también: ***cos***, ***tan***

sleep

Sintaxis: include misc.e
sleep(i)

Descripción: Suspende la ejecución por 'i' segundos.

Comentarios: En **WIN32** y **Linux/FreeBSD**, el sistema operativo suspenderá su proceso y programa otros procesos. En ***DOS32***, su programa entrará en un ciclo "ocupado" por 'i' segundos, tiempo durante el cual otros procesos pueden ejecutarse, pero competirán con el suyo por el uso de la CPU.

Ejemplo:

```
puts(1, "Esperando 15 segundos...\n")
sleep(15)
puts(1, "Listo.\n")
```

Ver también: ***lock_file***, ***abort***, ***time***

sort

Sintaxis: include sort.e
s2 = sort(s1)

Descripción: Ordena 's1' en orden ascendente, usando un algoritmo rápido de ordenamiento. Los elementos de 's1' pueden ser una mezcla de átomos y secuencias. Los átomos vendrán antes que las secuencias, y éstas se ordenarán "alfabéticamente", donde los primeros elementos son más significativos que los últimos.

Ejemplo 1:

```
x = 0 &sort({7,5,3,8}) &0
-- x se establece a {0, 3, 5, 7, 8, 0}
```

Ejemplo 2:

```
y = sort({"Tenorio", "Juan", "Don", 5.5, 4, 6})
-- y es {4, 5.5, 6, "Don", "Juan", "Tenorio"}
```

Ejemplo 3:

```
database = sort({{"Smith",   95.0, 29},
                 {"Jones",   77.2, 31},
                 {"Clinton", 88.7, 44}})

-- Los 3 "registros" se ordenarán por el primer "campo"
-- es decir, por nombre. Si el primer campo (elemento) es igual
-- se ordenará por el segundo, etc.

-- después de ordenar, database es:
               {{"Clinton", 88.7, 44},
                {"Jones",   77.2, 31},
                {"Smith",   95.0, 29}}
```

Ver también: *custom_sort*, *compare*, *match*, *find*

sound

Plataforma: **DOS32**

Sintaxis: include graphics.e
sound(i)

Descripción: Ejecuta un sonido de frecuencia 'i' en el parlante de la PC. Si 'i' vale 0, el parlante se desconecta.

Comentarios: En **WIN32** y **Linux/FreeBSD** no generará sonido alguno.

Ejemplo:

```
sound(1000) -- ejecuta un sonido agudo
```

sprint

Sintaxis: include misc.e
s = sprint(x)

Descripción: Devuelve la representación de 'x' como cadena de caracteres. Es exactamente lo mismo que *print(fn, x)*, excepto que la salida se devuelve como una secuencia de caracteres, en lugar de enviarla a un fichero o dispositivo. 'x' puede ser cualquier objeto Euphoria.

Comentarios: Los átomos contenidos dentro de 'x' se mostrarán con un máximo de 10 dígitos significativos, igual que con *print()*.

Ejemplo 1:

```
s = sprint(12345)
-- s es "12345"
```

Ejemplo 2:

```
s = sprint({10,20,30}+5)
-- s es "{15,25,35}"
```

Ver también: *print*, *sprintf*, *value*, *get*

sprintf

Sintaxis: s = sprintf(st, x)

Descripción: Es exactamente lo mismo que ***printf()***, excepto que la salida se devuelve como una secuencia de caracteres, en lugar de enviarse a un fichero o dispositivo. 'st' es una cadena de formato, 'x' es el valor o secuencia de valores a formatearse. ***printf(fn, st, x)*** es equivalente a ***puts(fn, sprintf(st, x))***.

Comentarios: Algunos usos comunes de ***sprintf()*** son:

1. Convertir números en cadenas.
2. Crear cadenas para pasárselas a ***system()***.
3. Crear mensajes de error que se pueden pasar a un manejador general de errores.

Ejemplo:

```
s = sprintf("%08d", 12345)
-- s es "00012345"
```

Ver también: ***printf***, ***value***, ***sprint***, ***get***, ***system***

sqrt

Sintaxis: x2 = sqrt(x1)

Descripción: Calcula la raíz cuadrada de 'x1'.

Comentarios: Está función se puede aplicar a un átomo o a todos los elementos de una secuencia. Si toma la raíz cuadrada de un número negativo, el programa se abortará con un mensaje de error en tiempo de ejecución.

Ejemplo:

```
r = sqrt(16)
-- r es 4
```

Ver también: ***log***, ***power***

system

Sintaxis: system(st, i)

Descripción: Pasa una cadena de comando 'st' al intérprete de comandos del sistema operativo. El argumento 'i' indica la forma en que se regresará de la llamada a ***system()***:
i = 0, se restablece el modo gráfico previo y se limpia la pantalla.

i = 1, sonará un bip y el programa esperará que el usuario presione una tecla antes que se restablezca el modo gráfico previo.

i = 2, ni se restablece el modo gráfico, ni se limpia la pantalla.

Comentarios: i = 2 solamente debería usarse cuando se conoce que el comando ejecutado por ***system()*** no cambiará el modo gráfico.
Puede usar a Euphoria como un lenguaje por lotes sofisticado (.bat), haciendo llamadas a ***system()*** y ***system_exec()***.

system() iniciará un nuevo shell en ***DOS*** o ***Linux/FreeBSD***.

system() le permite usar redirección de la línea de comandos de las entrada y salida estándares en la cadena de comando 'st'.

Bajo **DOS32**, un programa Euphoria comenzará con memoria extendida. Si se agota la memoria extendida, el programa consumirá memoria convencional. Si se agota la memoria convencional, el programa usarámemoria virtual, es decir, espacio de intercambio en disco. El comando DOS ejecutado por ***system()*** fallará si no hay suficiente memoria convencional disponible. Para evitar esta situación, puede reservar algo de memoria convencional (baja), escribiendo:

```
SET CAUSEWAY=LOWMEM:xxx
```

donde xxx es la cantidad de Kb de memoria convencional a reservar. Escriba esto antes de correr su programa. También puede ponerlo en el fichero **autoexec.bat**, o en un fichero **.bat** que ejecute su programa. Por ejemplo:

```
SET CAUSEWAY=LOWMEM:80
ex miprog.ex
```

Esto reservará 80K de memoria convencional, que debería ser suficiente para ejecutar comandos simples de DOS como COPY, MOVE, MKDIR, etc.

Ejemplo 1:

```
system("copy temp.txt a:\\temp.bak", 2)
-- se usa la doble barra en una cadena literal para obtener
-- una barra simple
```

Ejemplo 2:

```
system("ex \\test\\miprog.ex <indata > outdata", 2)
-- ejecuta miprog redireccionando la entrada y salida estándares
```

Programa de ejemplo: **bin\install.ex**

Ver también: ***system_exec***, ***dir***, ***current_dir***, ***getenv***, ***command_line***

system_exec

Sintaxis: i1 = system_exec(st, i2)

Descripción: Intenta ejecutar el comando dado por 'st'. 'st' tiene que ser un comando para correr un programa ejecutable, posiblemente con algunos argumentos en su línea de comandos. Si el programa se puede ejecutar, 'i1' tendrá el código de salida del programa. Si no es posible ejecutarlo, ***system_exec()*** devolverá −1. 'i2' es un código que indica que hacer con el modo gráfico cuando termina ***system_exec()***. Estos códigos son los mismos que para ***system()***:
i2 = 0, se restablece el modo gráfico previo y se limpia la pantalla.

i2 = 1, sonará un bip y el programa esperará que el usuario presione una tecla antes que se restablezca el modo gráfico previo.

i2 = 2, ni se restablece el modo gráfico, ni se limpia la pantalla.

Comentarios: En ***DOS32*** o ***WIN32***, ***system_exec()*** solamente ejecutará programas **.exe** y **.com**. PAra ejecutar ficheros **.bat**, o comandos internos de DOS, necesitará a ***system()***. Algunos comandos, tales como DEL, no son programas, sino realmente comandos internos del intérprete de comandos.
En ***DOS32*** y ***WIN32***, ***system_exec()*** no permite usar la redirección de la línea de comandos en la cadena de comando 'st'.

Los códigos de salida de programas DOS o Windows normalmente están en el rango de 0 a 255, donde 0 indica "éxito".

Puede ejecutar un programa Euphoria usando ***system_exec()***. Un programa Euphoria puede devolver un código de salida mediante ***abort()***.

system_exec() no inicia un nuevo shell de DOS.

Ejemplo 1:

```
integer exit_code
exit_code = system_exec("xcopy temp1.dat temp2.dat", 2)

if exit_code = -1 then
    puts(2, "\n no se puede correr xcopy.exe\n")
elsif exit_code = 0 then
    puts(2, "\n ejecución exitosa de xcopy\n")
else
    printf(2, "\n xcopy falló con código %d\n", exit_code)
end if
```

Ejemplo 2:

```
-- ejecuta miprog con dos nombres de ficheros como argumentos
if system_exec("ex \\test\\miprog.ex indata outdata", 2) then
    puts(2, "falla!\n")
end if
```

Ver también: ***system***, ***abort***

tan

Sintaxis: x2 = tan(x1)

Descripción: Devuelve la tangente de 'x1', donde 'x1' está en radianes.

Comentarios: Esta función se puede aplicar a un átomo o a todos los elementos de una secuencia.

Ejemplo:

```
t = tan(1.0)
-- t es 1.55741
```

Ver también: *sin, cos, arctan*

text_color

Sintaxis: include graphics.e
text_color(i)

Descripción: Establece el color del texto. Agregue 16 para obtener texto parpadeante en algunos modos. Ver la lista de los colores posibles en **graphics.e**.

Comentarios: El texto que imprime ***después*** de llamar a ***text_color()***, tendrá el color deseado.

Cuando termina su programa, el último color elegido e impreso, será el que quede activo en pantalla. Así, puede tener que imprimir algo, tal vez solamente '\n', en WHITE para restaurar el texto blanco, especialmente si está en la última línea de la pantalla, listo a desplzarla hacia arriba.

Ejemplo:

```
text_color(BRIGHT_BLUE)
```

Ver también: *bk_color*

text_rows

Plataforma: **DOS32, WIN32**

Sintaxis: include graphics.e
i2 = text_rows(i1)

Descripción: Establece la cantidad de líneas en una pantalla de modo de texto al valor de 'i1' si es posible. 'i2' se establecerá a la nueva cantidad real de líneas.

Comentarios: En la mayoría de las tarjetas de video se soportan valores de 25, 28, 43 y 50 líneas.

Ver también: *graphics_mode*

tick_rate

Plataforma: **DOS32**

Sintaxis: include machine.e
tick_rate(a)

Descripción: Especifica la cantidad de interrupciones por pulsos del reloj por segundo. Esto determina la precisión de la rutina de librería *time()*. También afecta la tasa de muestreo del análisis de perfiles por tiempo.

Comentarios: ***tick_rate()*** se ignora en **WIN32** y **Linux/FreeBSD**. La resolución de la hora en **WIN32** es siempre 100 pulsos/segundo.
En la PC, las interrupciones por pulsos del reloj ocurren normalmente a 18.2 interrupciones por segundo. ***tick_rate()*** le permite incrementar esa tasa, pero no decrementarla.

tick_rate(0) restaurará la tasa al valor normal de 18.2. Euphoria también restaurará la tasa automáticamente cuando exista, aún cuando encuentre un error en su programa.

Si un programa corre en una ventana DOS window with a tick rate other than 18.2, the *time()* function will not advance unless the window is the active window.

Mientras **ex.exe** se está ejecutando, el sistema mantendrá la hora del día en forma correcta. Sin embargo, si **ex.exe** se aborta (por ejemplo, ve un error "CauseWay...") mientras la tasa de pulsos es alta, usted (o el usuario) puede necesitar reiniciar la máquina para restaurar la tasa adecuada. Si no lo hace, el sistema puede avanzar demasiado rápido. Este problema no ocurrirá en **Windows 95/98/NT**, solamente en **DOS** o **Windows 3.1**. Siempre deberá corregir la hora del día del reloj operado con baterías, cuando inicie su sistema nuevamente.

Ejemplo:

```
tick_rate(100)
-- time() avanzará en pasos de .01 segundos
-- en lugar de los usuales .055 segundos
```

Ver también: *time, time profiling*

time

Sintaxis: a = time()

Descripción: Devuelve la cantidad de segundos transcurridos desde algún punto fijo en el pasado.

Comentarios: Tome la diferencia entre dos lecturas de ***time()*** para medir, por ejemplo, cuanto tarda una sección de código en ejecutarse.
La resolución con **DOS32** es normalmente de 0.05 segundos. En **WIN32** y **Linux/FreeBSD** la resolución es de 0.01 segundos.

Bajo **DOS32** puede mejorar la resolución llamando a ***tick_rate()***.

Bajo **DOS32** el período de tiempo que puede medir está limitado a 24 horas. Después de ese tiempo, el valor devuelto por ***time()*** se restablecerá e iniciará nuevamente.

Ejemplo:

```
constant ITERATIONS = 1000000
integer p
atom t0, loop_overhead

t0 = time()
for i = 1 to ITERATIONS do
    -- mide un ciclo vacío
end for
loop_overhead = time() - t0

t0 = time()
for i = 1 to ITERATIONS do
    p = power(2, 20)
end for
? (time() - t0 - loop_overhead)/ITERATIONS
-- calcula el tiempo (en segundos) para una llamada a power()
```

Ver también: ***date***, ***tick_rate***

trace

Sintaxis: with trace
trace(i)

Descripción: Si 'i' es 1 o 2, se activa la sentencia de pantalla interactiva trazado/depuración. Si 'i' es 3, se activa el trazado de sentencias hacia el fichero **ctrace.out**. Si 'i' es 0, se desactiva el trazado. Cuando 'i' vale 1, aparece una pantalla en color. Cuando 'i' vale 2, la pantalla de trazado es monocromática. El trazado sólo puede ocurrir en rutinas compiladas con "**with trace**", y ***trace()*** no tendrá efecto, salvo que esté dentro de una ***sección*** "**with trace**" del programa.
Para leer una discusión completa acerca de trazado / depuración, vea Parte I − 3.1 Depuración.

Comentarios: Use ***trace(2)*** si la pantalla en colores es difícil de ver en su sistema.

trace(3) está soportado solamente en la Edición Completa del Traductor Euphoria a C. El Traductor no soporta el trazado interactivo.

El intérprete de Dominio Público soporta todas las variantes de ***trace()***, pero solamente para programas de hasta 300 sentencias en total. Para programas más grandes se necesitará la Edición Completa del intérprete.

Ejemplo:

```
if x <0 then
    -- ok, aquí es donde quiero depurar...
    trace(1)
    -- etc.
    ...
end if
```

Ver también: ***profile***, depuración y análisis de perfiles de ejecución

unlock_file

Sintaxis: include file.e
unlock_file(fn, s)

Descripción: Desbloquea el archivo 'fn' abierto, o una parte de él. Tiene que haber sido bloqueado previamente usando ***lock_file()***. En DOS32 y WIN32 se puede desbloquear un rango de bytes dentro de un archivo al especificar el parámetro 's' como {first_byte, last_byte}. El mismo rango de bytes tiene que haber sido bloqueado previamente por una llamada a ***lock_file()***. En Linux/FreeBSD solamente se puede bloquear y desbloquear un archivo completo. Para desbloquear un archivo completo, el parámtro 's' debería ser {}. En Linux/FreeBSD, 's' siempre tiene que ser {}.

Comentarios: Debería desbloquear el archivo tan pronto como sea posible, para que otros procesos puedan usarlo.

Cualquier archivo bloqueado, se desbloqueará automáticamente cuando el programa termine.

Ver ***lock_file()*** por comentarios adicionales y un ejemplo.

Ver también: ***lock_file***

unregister_block

Sintaxis: include machine.e (or safe.e)
unregister_block(a)

Descripción: Quita un bloque de memoria de la lista de bloques seguros mantenida por **safe.e** (la versión de depuración de **machine.e**). El bloque comienza en la dirección 'a'.

Comentarios: Esta rutina solamente tiene sentido con **propósitos de depuración**. Usela para desregistrar bloques de memoria que previamente hayan sido registrados con ***register_block()***. Al desregistrar un bloque, se lo quita de la lista de bloques seguros mantenida por **safe.e**. Esto evita que el programa realice lecturas o escrituras adicionales de memoria dentro de un bloque.

Ver ***register_block()*** por comentarios adicionales y un ejemplo.

Ver también: ***register_block***, **safe.e**

upper

Sintaxis: include wildcard.e
x2 = upper(x1)

Descripción: Convierte un átomo o secuencia a mayúsculas.

Example:

```
s = upper("Euphoria")
-- s es"EUPHORIA"

a = upper('g')
-- a es 'G'

s = upper({"Euphoria", "Programación"})
-- s es {"EUPHORIA", "PROGRAMACIÓN"}
```

Ver también: ***lower***

use_vesa

Platform: **DOS32**

Sintaxis: include machine.e

use_vesa(i)

Descripción: ***use_vesa(1)*** forzará a Euphoria a usar los gráficos estándar VESA. Esto puede provocar que los programas Euphoria trabajen mejor en los modos gráficos SVGA con ciertas tarjetas de video. ***use_vesa(0)*** restablecerá el modo original en que Euphoria usaba la tarjeta de video.

Comentarios: La mayoría de la gente puede ignorar esto.Sin embargo, si experimenta alguna dificultad en los modos gráficos de SVGA debería probar llamando a ***use_vesa(1)*** al comienzo del programa, antes de cualquier llamada a ***graphics_mode()***.
No se deberían usar otros argumentos para ***use_vesa()*** que no sean 0 y 1.

Example:

```
use_vesa(1)
fail = graphics_mode(261)
```

Ver también: ***graphics_mode***

value

Sintaxis: include get.e
s = value(st)

Descripción: Lee la cadena de representación de un objeto Euphoria, y computa el valor de ese objeto. Devuelve una secuencia de 2 elementos, **{error_status, value}**, donde error_status puede ser:

```
GET_SUCCESS -- se encontró una representación válida de un opbjeto
GET_EOF     -- fin de la cadena alcanzado demasiado pronto
GET_FAIL    -- la sintaxis es errónea
```

Comentarios: Esta función trabaja igual que ***get()***, pero lee desde una cadena que puede suministrarle, en lugar de un archivo o dispositivo.
Después de leer una representación válida de un objeto Euphoria, ***value()*** dejará de leer e ignorará cualquier otro caracter de la cadena. Por ejemplo: "36" y "36P" le darán ambos {GET_SUCCESS, 36}.

Ejemplo 1:

```
s = value("12345"}
-- s es {GET_SUCCESS, 12345}
```

Ejemplo 2:

```
s = value("{0, 1, -99.9}")
-- s es {GET_SUCCESS, {0, 1, -99.9}}
```

Ejemplo 3:

```
s = value("+++")
-- s es {GET_FAIL, 0}
```

Ver también: ***get***, ***sprintf***, ***print***

video_config

Platform: **DOS32, Linux, FreeBSD**

Sintaxis: include graphics.e
s = video_config()

Descripción: Devuelve una secuencia de valores describiendo la configuración actual de video: {monitor color?, modo gráfico, filas de texto, columnas de texto, píxeles x, píxeles y, cantidad de colores, cantidad de páginas}
Las siguientes constantes están definidas en **graphics.e**:

```
global constant VC_COLOR   = 1,
                VC_MODE    = 2,
                VC_LINES   = 3,
                VC_COLUMNS = 4,
                VC_XPIXELS = 5,
                VC_YPIXELS = 6,
                VC_NCOLORS = 7,
                VC_PAGES   = 8
```

Comentarios: Esta rutina le hace fácil parametrizar un programa, porque trabaja con diferentes modos gráficos.

En la PC hay dos tipos de modos gráficos. El primer tipo, modo texto, que le permite imprimir solamente texto. El segundo tipo, modo gráficos de píxel, que le permite dibujar píxeles, o puntos, en varios colores, tanto como texto. Puede decir que está en modo texto, cuando los campos VC_XPIXELS y VC_YPIXELS valgan 0. Las rutinas de librería tales como ***polygon()***, ***draw_line()***, y ***ellipse()*** solamente trabajan en modo gráfico de píxel.

Example:

```
vc = video_config()  -- en modo 3 con 25 líneas de texto:
-- vc es {1, 3, 25, 80, 0, 0, 32, 8}
```

Ver también: ***graphics_mode***

wait_key

Sintaxis:
include get.e
i = wait_key()

Descripción: Devuelve la próxima tecla presionada por el usuario. No devuelve nada hasta que se presiona una tecla.

Comentarios: Podría obtener el mismo resultado usando ***get_key()*** de este modo:

```
while 1 do
    k = get_key()
    if k != -1 then
        exit
    end if
end while
```

Sin embargo, en sistemas multitarea como **Windows** o **Linux/FreeBSD**, este "ocupado" tendería a hacer más lento el sistema. ***wait_key()*** le permite al sistema operativo hacer otras tareas, mientras el programa espera que el usuario presione una tecla.
También podría usar ***getc(0)***, asumiendo que el número de archivo 0 es la entrada desde el teclado, excepto que no podría levantar los códigos especiales de teclas de función, teclas de flecha, etc.

Ver también: ***get_key***, ***getc***

walk_dir

Sintaxis:
include file.e
i1 = walk_dir(st, i2, i3)

Descripción: Esta rutina "caminará" a través de un directorio cuya ruta esté dada por 'st'. 'i2' es el **routine id** de una rutina que se suministra. ***walk_dir()*** llamará a su rutina una vez por cada archivo y subdirectorio de 'st'. Si 'i3' no es cero (VERDADERO), entonces los subdirectorios de 'st' serán "caminados" recursivamente.
La rutina que se suministra debería aceptar la ruta y ***dir_entry()*** por cada archivo y subdirectorio. Esta rutina debería devolver 0 para continuar la ejecución de ***walk_dir()***, o no cero para detenerlo.

Comentarios: Este mecanismo le permite escribir una función simple que gestiona un archivo por vez, mientras ***walk_dir()*** se encarga del proceso de caminar a través de los archivos y los subdirectorios.
Por defecto, los archivos y subdirectorios serán visitados en orden alfabético. Para usar un orden diferente, establecer el entero global **my_dir** al **routine id** de su función ***dir()*** *modificada* que ordena las entradas del directorio de forma distinta. Ver la función ***dir()*** por defecto en **file.e**.

Example:

```
function look_at(sequence path_name, sequence entry)
-- esta función acepta dos secuencias como argumentos
    printf(1, "%s\\%s: %d\n",
           {path_name, entry[D_NAME], entry[D_SIZE]})
    return 0 -- continuar el recorrido
end function
```

```
exit_code = walk_dir("C:\\MYFILES", routine_id("look_at"), TRUE)
```

Programa de ejemplo: **euphoria\bin\search.ex**

Ver también: ***dir*, *current_dir***

where

Sintaxis: include file.e
i = where(fn)

Descripción: Esta función devuelve la posición del byte actual en el archivo 'fn'. Esta posición se actualiza leyendo, escribiendo o buscando en el archivo. Su valor indica la posición del próximo byte en el archivo, en donde se leerá o escribirá.

Ver también: ***seek*, *open***

wildcard_file

Sintaxis: include wildcard.e
i = wildcard_file(st1, st2)

Descripción: Devuelve 1 (verdadero) si el nombre del archivo 'st2' coincide con el patrón 'st1'. Devuelve 0 (falso) en caso contrario. Esto es similar a la coincidencia de patrones de DOS, pero mejor en algunos casos. * significa 0 o más caracteres, ? significa un solo caracter. En Linux y FreeBSD las comparaciones de caracteres son sensibles a la mayúscula; en DOS y Windows no lo son.

Comentarios: Tiene que usar esta función para verificar la salida de la rutina ***dir()*** para los nombres de archivos que coincidan con el patrón suministrado por el usuario del programa.
En DOS "*ABC.*" coincidirá con ***todos*** los archivos. ***wildcard_file("*ABC.*", s)*** solamente coincidirá cuando la parte final del nombre del archivo contenga "ABC" (como se esperaría).

Ejemplo 1:

```
i = wildcard_file("AB*CD.?", "aB123cD.e")
-- i se establece a 1 en DOS o Windows, 0 en Linux o FreeBSD
```

Ejemplo 2:

```
i = wildcard_file("AB*CD.?", "abcd.ex")
-- i se establece a 0 en todos los sistemas,
-- debido a que la extensión del archivo tiene 2 letras en lugar de 1
```

Programa de ejemplo: **bin\search.ex**

Ver también: ***wildcard_match*, *dir***

wildcard_match

Sintaxis: include wildcard.e
i = wildcard_match(st1, st2)

Descripción: Esta función realiza la búsqueda general de una cadena contra un patrón que contiene los símbolos * y ? Devuelve 1 (verdadero) si la cadena 'st2' coincide con el patrón 'st1'. Devuelve 0 (falso) en caso contrario. * significa 0 o más caracteres. ? siginifica un solo caracter. La comparación de caracteres es sensible a la mayúscula.

Comentarios: Si desea hacer comparaciones que no tengan en cuenta mayúsculas y minúsculas, debe convertir 'st1' y 'st2' a mayúsculas con ***upper()***, o a minúsculas con ***lower()*** antes de llamar a ***wildcard_match()***.
Si quiere detectar un patrón en cualquier lugar dentro de una cadena, agregue * a cada extremo del patrón:

```
i = wildcard_match('*' &pattern &'*', string)
```

No hay forma de tratar literalmente a * o ? en un patrón.

Ejemplo 1:

```
i = wildcard_match("A?B*", "AQBXXYY")
-- i es 1 (VERDADERO)
```

Ejemplo 2:

```
i = wildcard_match("*xyz*", "AAAbbbxyz")
-- i es 1 (VERDADERO)
```

Ejemplo 3:

```
i = wildcard_match("A*B*C", "a111b222c")
-- i es 0 (FALSO) porque no coinciden las mayúsculas con las minúsculas
```

Programa de ejemplo: **bin\search.ex**

Ver también: ***wildcard_file***, ***match***, ***upper***, ***lower***, ***compare***

wrap

Sintaxis: include graphics.e
wrap(i)

Descripción: Le permite al texto enrollarse alrededor del margen derecho (i = 1) o truncarse (i = 0).

Comentarios: Por defecto, es texto estará enrollado.

Use ***wrap()*** en modos de texto o modos gráficos de píxel cuando se muestren largas líneas de texto.

Ejemplo:

```
puts(1, repeat('x', 100) &"\n\n")
-- ahora tiene una línea de 80 'x' siguiendo a una línea de 20 'x' más
wrap(0)
puts(1, repeat('x', 100) &"\n\n")
-- crea solo una línea de 80 'x'
```

Ver también: ***puts***, ***position***

xor_bits

Sintaxis: x3 = xor_bits(x1, x2)

Descripción: Realiza la operación lógica XOR (O exclusivo) sobre los correspondientes bits en 'x1' y 'x2'. El bit 'x3' será 1 cuando uno de los dos bits en 'x1' o 'x2' sea 1 y el otro sea 0.

Comentarios: Los argumentos de esta función pueden ser átomos o secuencias. Se aplican las reglas para operaciones sobre secuencias.
Los argumentos tiene que ser representables como números de 32 bits, con o sin signo.

Si intenta manejar valores de 32 bits (completos), debería declarar las variables como **átomos**, en lugar de declararlas como enteros. El tipo entero de Euphoria está limitado a 31 bits.

Los resultados se tratan como números con signo. Serán números negativos cuando el bit de mayor orden sea 1.

Ejemplo 1:

```
a = xor_bits(#0110, #1010)
-- a es #1100
```

Ver también: ***and_bits***, ***or_bits***, ***not_bits***, ***int_to_bits***, ***int_to_bytes***