COPIA LITERAL

Extracto del libro “GAMBAS, programación visual con Software Libre”, de la editorial EDIT LIN EDITORIAL S.L., cuyos autores son Daniel Campos Fernández y José Luis Redrejo.

LICENCIA DE ESTE DOCUMENTO

Se permite la copia y distribución de la totalidad o parte de esta obra sin ánimo de lucro. Toda copia total o parcial deberá citar expresamente el nombre del autor, nombre de la editorial e incluir esta misma licencia, añadiendo, si es copia literal, la mención “copia literal”. Se autoriza la modificación y traducción de la obra sin ánimo de lucro siempre que se haga constar en la obra resultante de la modificación el nombre de la obra originaria, el autor de la obra originaria y el nombre de la editorial. La obra resultante también será libremente reproducida, distribuida, comunicada al público y transformada en términos similares a los expuestos en esta licencia.

9. Acceso a la API

9. Acceso a la API
9. Acceso a la API
9. Acceso a la API

En realidad, el título *Acceso a la API* está tomado de los programadores de Visual Basic™, en los cuales es muy frecuente realizar llamadas a funciones externas, alojadas en librerías también externas escritas en lenguaje C. Tal vez, en este caso no sea correcto, dado que Linux es tan sólo un núcleo y el resto de librerías (incluyendo la estándar de C, *glibc*) no son propiamente la API de este sistema operativo, sino campos de utilidades que se añaden al sistema. Pero, de todas formas, sirve de introducción para el asunto que vamos a tratar.

Un sistema GNU/Linux puede contar con librerías distintas de otro, pero en la práctica siempre vamos a encontrar como mínimo la librería de C (*glibc*), posiblemente una librería para diseño de interfaces de texto (*ncurses*) y, si hay un sistema gráfico instalado, las librerías de X-Window y otras como *glib*, *gtk+*, *atk*, etc. Al margen

de éstas, todo sistema suele tener instaladas librerías como *libcurl*, para acceso a diversos protocolos de red, o *libaspell*, para corrección ortográfica.

Desde Gambas podemos sacar provecho de las funciones aportadas por estas librerías, declarando funciones externas e indicando la librería en la que se deben buscar. El programa aporta, además, ciertos elementos para la gestión de memoria y punteros. Las librerías accesibles serán aquellas escritas en lenguaje C, que sigan el estándar definido (ABI), o bien escritas en otros lenguajes si respetan dicho estándar (por ejemplo, escritas con FreePascal). No es posible, en este momento, acceder a funciones de librerías escritas en C++, al menos de forma directa.

Si bien en sistemas Windows™ es muy habitual recurrir a este modo de trabajo, hay que matizar que en Linux muchas tareas se pueden resolver simplemente llamando a procesos externos, labor más sencilla, por lo general, y menos problemática en la depuración de los programas. Por tanto, recurrir a la gestión de procesos cuando sea posible, en lugar de a este sistema, es, en general, una buena idea.

Hay que tener en cuenta que la llamada a funciones externas, tanto desde Gambas como desde otros lenguajes interpretados (Mono, por ejemplo), requiere un trabajo de conversión de tipos, entre otras tareas, lo cual consume un tiempo de procesador. Tratar, pues, de emplear llamadas a funciones con el fin de incrementar la velocidad del programa, puede no ser una buena idea en todos los casos.

9.1 Declarar una función externa

La declaración se debe realizar al principio de una clase o módulo, del mismo modo que las variables globales. Al principio se declarará la función como *PUBLIC* o *PRIVATE*, para definir su ámbito, limitado a la clase donde se aloja o a todo el programa. Por defecto son privadas.

A continuación, la palabra clave **EXTERN** indica que vamos a declarar una función externa, seguida por su nombre, que será el nativo de la función en la librería.

Hay que tener en cuenta que C sí distingue entre mayúsculas y minúsculas y, por tanto, tratar de declarar *Printf* en lugar de *printf* para la librería *glibc*, dará como resultado un error.

Como en el resto de funciones de Gambas, se indicarán los parámetros que reciben entre paréntesis, y después el tipo de dato devuelto. Opcionalmente, se puede indicar al final la librería en que se encuentra la función, así como un *alias*, o nombre alternativo, a utilizar desde este programa, para llamar a la función, que se empleará, por ejemplo, en ciertas funciones cuyo nombre coincide con palabras clave del lenguaje Gambas. Supongamos la función **strcmp** de la librería estándar de C, que se define en este lenguaje como:

```
int strcmp(const char *s1, const char *s2);
```

La declaración en Gambas será:

```
EXTERN strcmp(s1 AS STRING,s2 AS STRING) AS INTEGER IN
"libc:6"
```


Cómo denominar la librería

Tras la palabra clave IN, se presenta entre comillas el nombre de la librería. En principio, se puede indicar un nombre simple, como *libc* o *libgstreamer*. En ese caso, Gambas tratará de encontrar la versión más moderna instalada en el sistema. A veces, sin embargo, puede interesarnos una versión concreta de una librería, en cuyo caso se reflejará indicando el símbolo : tras el nombre y, a continuación, el número de versión, por ejemplo *libgl:1.0.7667* o *libc:6*.

En sistemas GNU/Linux, dentro de las carpetas */lib* o */usr/lib* encontraremos la mayoría de las librerías que hay en nuestro sistema. A partir de esos nombres de archivo, podemos deducir el nombre de la librería, por ejemplo:

```
/usr/lib/libglib-2.0.so.0.600.4    ->    "libglib-
2.0:0.600.4"
```

Como podemos observar, el número de versión es siempre la serie de números que se encuentra tras el texto **.so.** en el nombre de archivo, y que todo lo que viene antes forma parte del nombre de la librería. En este caso, el valor **2.0** tras el texto **libglib-** es parte del nombre y no de la versión. Especificar un número de versión puede ser problemático cuando se trata de instalar un programa entre diferentes equipos con distintas distribuciones GNU/Linux instaladas, que pueden tener versiones ligeramente diferentes de estas librerías o cambiar tras una actualización del sistema. Por tanto, hay que aplicar esta característica sólo en casos imprescindibles, o tal vez emplear el número mayor y dejar el menor y la revisión a elección de Gambas.

Si en un módulo o clase se han de definir varias funciones de una misma librería, no es necesario especificarla en todas las declaraciones, basta con indicarla como librería por defecto con la palabra clave **LIBRARY** y Gambas la empleará en todas las declaraciones en las cuales no se especifique la librería.


```
LIBRARY "libglib-2.0"

EXTERN g_utf8_strlen(p AS String, Mx AS Integer) AS
Integer
EXTERN g_utf8_strup(Buf AS String, Mx AS Integer) AS
Pointer
EXTERN getchar() AS Integer IN "libc:6"
```

Aquí, las dos primeras funciones pertenecen a **libglib**, y la tercera a **libc**.

El uso de los alias

Supongamos la función *free()* de la librería estándar de C, que se emplea para liberar memoria asignada, por ejemplo, con *malloc()*. El problema para declararla en Gambas es que ya existe *Free* como palabra clave reservada, por tanto, tendremos que indicar un nombre alternativo:

```
EXTERN FreePointer(Ptr AS Pointer) IN "libc:6" EXEC
"free"
```

A la hora de escribir código Gambas, llamaremos siempre a la función **FreePointer**, que internamente llamará a la original *free* de la librería de C.

Tipos de datos

Existe una correspondencia entre los tipos de datos básicos de Gambas y los de C. Hay que tener cuidado con los *falsos amigos*, un tipo *long* de C, no es un tipo *Long* de Gambas, sino un tipo *Integer*; y un tipo *float* de C no es el *Float* de Gambas, sino el *Single*. Las siguientes correspondencias son aplicables a Gambas sobre GNU/Linux 32 bits/Intel.

Correspondencia de datos entre C y Gambas

C	Gambas	C	Gambas
char	*Byte*	*long long*	*Long*
short	*Short*	*float*	*Single*
int, long	*Integer*	*double*	*Float*

El tipo *Pointer* de Gambas se emplea para definir un puntero. En Gambas-GNU/Linux-32 bits es exactamente igual a *Integer*, pero en otras arquitecturas podría cambiar, por lo cual es recomendable utilizar siempre el tipo de dato *Pointer* en lugar de *Integer* para referirse a punteros. El tipo de datos *String* sólo debe emplearse si la cadena a pasar como parámetro no va a ser modificada o reasignada por la función de C, en otro caso debe utilizarse un dato tipo *Pointer*. Por último indicar que es posible pasar datos tipo *Object*, pero no está permitido usar tipos *Variant*.

9. 2 Funciones auxiliares

Gambas aporta algunas utilidades para trabajar con llamadas a funciones.

```
Puntero = Alloc ( Tamaño AS Integer [ , Cantidad AS
Integer ] )
```

La función **Alloc** es similar a la homónima de C: reserva un bloque de *Tamaño*Cantidad bytes*. En este caso, el intérprete de Gambas lleva, además, un contador de las asignaciones y liberaciones de memoria realizadas, indicando, con un mensaje de advertencia al final del programa, si quedó memoria sin liberar.

```
Nuevo Puntero = Realloc ( Antiguo puntero AS Pointer ,
Tamaño AS Integer , Cantidad AS Integer )
```

Realloc toma un puntero, previamente asignado con *Alloc* o *Realloc*, y cambia su tamaño, manteniendo los datos almacenados; devuelve un puntero a la nueva área de memoria reasignada; y, al igual que *Alloc*, mantiene la cuenta de asignaciones/liberaciones de memoria.

```
Free ( Puntero )
```

Free libera un puntero asignado con *Alloc* o *Realloc*, de igual modo que la función homónima de C, pero manteniendo la cuenta de asignaciones y liberaciones de memoria.

Muchas librerías aportan sus funciones propias para asignar o liberar memoria. Puede ser necesario que empleemos estas funciones en lugar de *Alloc* o *Free*, si así lo requiere el programa, para mantener la coherencia del código y funcionalidad de la librería.

```
Cadena = StrPtr ( puntero AS Pointer )
```

Cuando una cadena ha de tratarse como un puntero, esta función nos permite obtener una copia como tipo de dato *String*, siempre y cuando **puntero** apunte a una cadena terminada en carácter *\0* (nulo), que es lo habitual cuando se trabaja con C.

Gambas permite escribir y leer en memoria como si fuese otro flujo cualquiera. Así, utilizando un puntero como parámetro, se pueden utilizar las instrucciones

habituales en archivos: *READ*, *WRITE*, etc. Entre otras posibilidades ofrecidas por esta solución, éste es el modo actual en el que el programador puede acceder a los datos de una estructura de C.

Está previsto que la versión 2 de Gambas añada el componente *gb.api*.

Este componente se está desarrollando para dar soporte directo al trabajo con estructuras de datos y a retrollamadas, que aumentarán y simplificarán las capacidades de Gambas para manejar librerías externas.

9. 3 Un ejemplo con libaspell

Aspell proporciona una API para corrección de texto, permite analizar palabras, indicar si son válidas o no según el diccionario empleado, y ofrecer palabras alternativas a las consideradas incorrectas, entre otras capacidades.

Disponemos de más información sobre *libaspell* en su página oficial:
http://aspell.sourceforge.net/

Para poder utilizar este código, tendremos que instalar *Aspell* en el sistema, así como el diccionario de nuestro idioma.

Por ejemplo, en un sistema gnuLinex 2004 en castellano (basado en Debian), los paquetes instalados en el sistema son:

aspell
aspell-bin
libaspell-dev
aspell-es
libaspell15

Crearemos un nuevo proyecto gráfico llamado **Aspell**, con un único formulario **FMain** en el cual se contendrá un botón llamado **BtnCheck**, un *TextBox* llamado **TxtIn**, un *ListBox* llamado **Lista** y un *ProgressBar* llamado **Pbar**.

Figura 1. Formulario del proyecto *Aspell*.

Al inicio del código del formulario definimos las funciones de **libaspell**.

```
LIBRARY "libaspell"

EXTERN new_aspell_config() AS Pointer
EXTERN aspell_config_replace(Cfg AS Pointer, Var AS
String, Value AS String)
EXTERN new_aspell_speller(Cfg AS Pointer) AS Pointer
EXTERN aspell_error_number(Err AS Pointer) AS Integer
EXTERN aspell_error_message(Err AS Pointer) AS
Pointer
EXTERN to_aspell_speller(Err AS Pointer) AS Pointer
EXTERN aspell_speller_check(V1 AS Pointer, V2 AS
String, V3 AS Integer) AS Integer
EXTERN aspell_speller_suggest(V1 AS Pointer, V2 AS
String, V3 AS Integer) AS Pointer
EXTERN aspell_word_list_elements(V1 AS Pointer) AS
Pointer
EXTERN aspell_string_enumeration_next(Ptr AS Pointer)
AS Pointer
EXTERN delete_aspell_string_enumeration(Ptr AS
Pointer)
EXTERN delete_aspell_config(Cfg AS Pointer)
```

Al pulsar el botón **BtnCheck**, realizaremos el proceso de análisis del texto introducido en el cuadro **TxtIn**, para ir mostrando el resultado en la lista.

```
PUBLIC SUB BtnCheck_Click()

    DIM spell_config AS Pointer
    DIM possible_err AS Pointer
    DIM spell_checker AS Pointer
    DIM suggestions AS Pointer
    DIM elements AS Pointer

    DIM MyStr AS String[]
    DIM Bucle AS Integer
    DIM BufErr AS String
    DIM Sum AS Float
    DIM hPtr AS Pointer
    DIM sPtr AS String
```

En primer lugar, dividimos el texto introducido por el usuario en palabras, separadas por espacios.


```
    PBar.Value = 0
    Lista.Clear()
    MyStr = Split(TxtIn.Text, " ")
    Sum = 1 / MyStr.Count
```

Realizamos las primeras llamadas a la librería *libaspell*. Para ello tomamos una configuración por defecto y aplicamos dos atributos: el texto estará codificado como UTF-8 y se usará el diccionario que corresponda con el lenguaje del sistema.

Aplicamos la configuración y, de existir un error (por ejemplo, que el diccionario no se encuentre disponible), indicamos el mensaje correspondiente:

```
    spell_config = new_aspell_config()
    aspell_config_replace(spell_config, "lang",
    Application.Env["LANG"])
```

```
aspell_config_replace(spell_config, "encoding",
"utf-8")


possible_err = new_aspell_speller(spell_config)

IF aspell_error_number(possible_err) <> 0 THEN
  Message.Error(StrPtr(aspell_error_message
  (possible_err)))
  RETURN
ELSE
  spell_checker = to_aspell_speller(possible_err)
END IF
```

Después entramos en un bucle en el cual se analiza, palabra por palabra, el texto introducido. En caso de error, se dará una lista de sugerencias, pero si está todo correcto, se indicará con un *OK*.


```
FOR Bucle = 0 TO MyStr.Count - 1

    PBar.Value = PBar.Value + Sum
    MyStr[Bucle] = Trim(MyStr[Bucle])
    IF Len(MyStr[Bucle]) > 0 THEN
      IF aspell_speller_check(spell_checker,
      MyStr[Bucle], - 1) THEN
        lista.Add("OK    -> " & MyStr[Bucle])
      ELSE

        BufErr = " ( "
        suggestions = aspell_speller_suggest
        (spell_checker, MyStr[Bucle], - 1)
        elements = aspell_word_list_elements
        (suggestions)
```

```
            DO WHILE TRUE
              hPtr = aspell_string_enumeration_next(elements)
              IF hPtr = 0 THEN
                BREAK
              ELSE
                sPtr = ""
                INPUT #hPtr, sPtr
                BufErr = BufErr & sPtr & " "
              END IF

            LOOP
            delete_aspell_string_enumeration(elements)

            BufErr = BufErr & ")"
            lista.Add("ERROR -> " & MyStr[Bucle] & BufErr)

          END IF
        END IF
        WAIT 0.001
      NEXT
```

Finalmente liberamos los elementos necesarios empleados en la corrección de texto.

```
      delete_aspell_config(spell_config)
      PBar.Value = 1

    END
```

9. 4 Obtener información acerca de la librería

Normalmente, las librerías aportan multitud de constantes y macros definidas en sus ficheros de cabecera. A la hora de trabajar con ellas debemos conocer sus valores, ya

que no se puede recurrir directamente a los **#include** de C. Veamos este ejemplo, que utiliza la constante GTK_WINDOW_TOPLEVEL:

```
#include <gtk/gtk.h>
int main(void)
{
        GtkWindow *win;

        gtk_init(0,0);

        win=gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_widget_show_all(win);

        gtk_main();
}
```


Con el fin de conocer el valor de esta constante, o cualquier otra aportada por una librería, podemos recurrir al menos a cuatro métodos:

1. Consultar la documentación de la propia librería, que en este caso (GTK+), es bastante completa. La podemos encontrar en la siguiente dirección: **http://developer.gnome.org/doc/API/2.0/gtk/gtk-Standard-Enumerations.html#GtkWindowType.**

2. Indicar que **GTK_WINDOW_TOPLEVEL** tiene el valor 0 como parte de la enumeración *GtkWindowType*.

3. Buscar en los ficheros de cabecera de la librería, que se suelen depositar a partir de la ruta */usr/include*. En el caso que tratamos en el ejemplo, este valor se encuentra en el fichero */usr/include/gtk-2.0/gtk/gtkenums.h*.

4. Compilar un pequeño programa en C que muestre el valor problemático que no alcanzamos a averiguar por otros sistemas.

```
#include <gtk/gtk.h>
#include <stdio.h>

void main(void)
{
  printf("%d\n",GTK_WINDOW_TOPLEVEL);
}
```

Podemos ya pasar el pequeño programa a Gambas. Para ello, creamos un programa de consola con un solo módulo *modMain* que contenga este código:

```
' Gambas module file
LIBRARY "libgtk-x11-2.0"

CONST GTK_WINDOW_TOPLEVEL AS Integer = 0

EXTERN gtk_init(Argv AS Pointer, Argc AS Pointer)
EXTERN gtk_main()
EXTERN gtk_window_new(wType AS Pointer) AS Pointer
EXTERN gtk_widget_show(wid AS Pointer)

PUBLIC SUB Main()

  DIM Win AS Pointer

  gtk_init(0, 0)

  Win = gtk_window_new(GTK_WINDOW_TOPLEVEL)
  gtk_widget_show(Win)

  gtk_main()

END
```

Si lo ejecutamos podemos observar que el resultado es equivalente al del programa en C.

9. 5 Resumen

En este momento, Gambas autoriza el acceso a funciones de librerías externas. La única condición es que sigan el ABI estándar. Las propias instrucciones del programa permiten reservar y liberar memoria. En el futuro, a través de *gb.api* (no detallado aquí, pues se encuentra en pleno desarrollo y su interfaz puede cambiar), la gestión de estructuras y retrollamadas será posible.

Pero no se recomienda el uso de esta funcionalidad, salvo que esté plenamente justificado. Gambas aporta componentes para tareas variadas que cubren buena parte de las necesidades que se encuentran en los programas habituales, y su capacidad de gestión de procesos externos lo hace apropiado para casi cualquier tarea. Trabajar con llamadas a funciones externas puede hacer el programa más difícil de depurar, sensible a los cambios en el sistema (actualizaciones de librerías, cambio de versión de la distribución...) y limita la portabilidad del código (por ejemplo, de un sistema GNU/Linux, donde se desarrolla, a una plataforma FreeBSD o, en el futuro, en la versión Windows de Gambas).

Marcas registradas

Los siguientes términos son marcas registradas en Estados Unidos u otros países:

- Linux es una marca registrada de Linus Torvalds.
- Debian es una marca registrada de Software in the Public Interest, Inc.
- SUSE es una marca registrada de SuSE AG.
- Mandriva es una marca registrada de Mandrakesoft S.A. y Mandrakesoft Corporation.
- GNOME es una marca registrada de la Fundación GNOME.
- KDE, K Desktop Environment, es una marca registrada de KDE e.V.
- Microsoft, Windows y Visual Basic son marcas registradas de Microsoft Corporation.
- gnuLinEx es una marca registrada de la Junta de Extremadura.
- Java y todos los nombres y logos basados en Java son marcas registradas de Sun Microsystems, Inc.
- UNIX es una marca registrada de The Open Group.
- Mozilla y Firefox son marcas registradas de The Mozilla Organization.
- Apple y Macintosh son marcas registradas de Apple Computer Corporation.

Otras empresas, productos y nombres de servicios pueden ser marcas registradas o servicios de otras compañías.