Programación de Interfaces Gráficas de Usuario con GTK+ 3 Documentation

Versión 1

Noe Misael Nieto Arroyo

09 de junio de 2023

Índice general

Contenido:

CAPÍTULO 1

Prefacio

1.1 Historia

Hola.

Entre 2003 y 2004, organicé un curso titulado **«Programación de interfases gráficas de usuario con Gtk+»** en el Instituto Tecnológico de Puebla gracias a la ayuda de la Dra. Georgina Flores Becerra, docente del departamento de Informática. De la compilación de las notas y ejercicios realizados en el curso salió el primer borrador de éste manual.

Después de algún tiempo lo re-escribí en OpenOffice y generé un archivo PDF [que aún se encuentra disponible en línea.](http://www.developarts.com/programacion-de-interfaces-graficas-de-usuario-con-gtk/). Pero perdí el archivo original de OpenOffice.

Pero ahora el manual necesita de una buena actualización por que el ecosistema de desarrollo de aplicaciones con Gtk+ ha añadido tecnologías interesantes como [GObjectIntrospection](https://developer.gnome.org/gi/stable/) y [Vala](https://wiki.gnome.org/Projects/Vala)

Es por eso que aprovecho la oportunidad de actualizar éste manual de desarrollo usando nuevas herramientas como Git, GitHub, ReadtheDocs, RestructuredText y Sphinx.

1.2 Objetivos

Mi objetivo es ayudarte a que escribas más y mejores programas en Linux y el ambiente de escritorio [GNOME](http://www.gnome.org/). Escribo en español para eliminar un poco la resistencia a adoptar GTK+ 3, Linux y a GNOME como una plataforma en la que se puedan escribir programas interesantes y útiles.

No soy escritor profesional, así que es muy probable que encuentres inconsistencias, faltas de ortografía, ejemplos que no funcionan y un largo etcétera. Siéntete libre de usar GitHub para reportar un problema o mejor aún, haz un *fork* y envíame un *pull request*.

CAPÍTULO 2

Introducción

2.1 ¿Qué es Gtk+?

GTK+ es una librería para la creación de interfaces gráficas de usuario. Su licencia permite desarrollar programas libres así como aplicaciones comerciales. Todo esto sin tener que pagar licencias o regalías para su uso.

GTK es el acrónimo de *GIMP Toolkit* debido a que originalmente fue escrito para desarrollar el *Programa de manipulación de imágenes de GNU* (GNU Image Manipulation Program, GIMP)

GTK+ se usa en gran cantidad de proyectos. Algunos de los más famosos son: GIMP, Inkscape, Pitivi, Pidgin, XChat, LibreOffice, VLC y Elementary OS.

GTK+ se ha construido en base a tres librerías: GDK (*GIMP Drawing Kit*) para interactuar con el sistema gráfico, gdk-pixbuf para manipular imágenes y Glib para integración con el Sistema Operativo(SO).

Desde el punto de vista de GTK+ una aplicación gráfica se construye mediante una conjunto de objetos que se comunican mediante eventos y retro llamadas. Su diseño, en constante mejora, le permite lograr una gran velocidad de ejecución, interactuar con diferentes lenguajes de programación como C, C++, Python, C#, Perl o Ada, y ser un ambiente de desarrollo multi-plataforma (UNIX, Linux, Windows TM , MacOS X TM).

2.2 Componentes de GTK+

GTK es la parte gráfica que interactúa con el usuario. GTK nos ofrece todo lo necesario para el desarrollo de interfaces gráficas de usuario como botones, cajas de texto, menús, ventanas, etc.. Otras formas o widgets mucho más complejos pueden ser creados para satisfacer nuestras necesidades.

Los componentes de GTK+ se interrelacionan de la siguiente manera:

- GLIB es la base del proyecto GTK+ y GNOME. GLIB provee las estructuras de datos en el lenguaje C. Contiene la infraestructura necesaria para construir envoltorios e interfaces para tales funciones como un bucle de ejecución de eventos, hilos, carga dinámica de módulos y un sistema de objetos.

- GObject es la parte de Glib que implementa las características de lenguajes orientados a objetos que no están disponibles intrínsecamente en el lenguaje C. De esta manera GTK+ puede ser una caja de herramientas orientada a objetos sin tener que implementarse en C++ u otro lenguaje.
- GDK interactúa con las primitivas gráficas de cada SO y las prepara para su uso con GTK+. En los sistemas UNIX soportados como Linux y Solaris, GDK interactúa con el sistema X-Win dow[1], mientras que en sistemas Windows TM lo hará con GDI[2] y en MacOS X TM con Quartz TM [3]. El programador de aplicaciones no notará esto y, salvo raras excepciones, un programa escrito para GTK+ en Linux, por ejemplo, puede compilarse en Windows sin mayor modificación.
- GdkPixbuf contiene convenientes rutinas para el manejo de imágenes(png, gif, jpeg, etc), tales como abrir y guardar, trabajar con mapas de imágenes en memoria, escalado y animaciones entre otras características.
- Pango es un entorno de trabajo para el procesamiento y despliegue de texto internacional utilizando UTF8 como código base[4]; está integrado por completo en GTK+ y permite desplegar caracteres en distintos alfabetos y estilos de escritura (occidental, cirílico, árabe, chino, etc), texto bidireccional y con atributos como color, tamaño y estilo o rotación.
- ATK proporciona un conjunto de interfaces de accesibilidad. Si uno soporta interfaces ATK en un programa, tendremos la posibilidad de utilizar diferentes herramientas como magnificadores de pantalla, texto a diálogo o diferentes métodos de entrada para personas con capacidades diferentes.

2.3 Glib

Glib es una librería de utilidades de propósito general, la cual provee tipos de datos útiles, así como macros, conversiones de tipo, utilidades de cadenas y muchas más. Glib se encarga de interactuar con el SO anfitrión sin distraer al programador de las diferencias más significativas entre las distintas variedades de estos.

La siguiente lista reúne las principales características de Glib:

- Tipos de datos: Los tipos de datos (char, int, float, double), pueden diferir entre los diferentes SO ó arquitecturas de hardware. Glib libera al programador de prestar atención a estos detalles y en su lugar ofrece un conjunto de tipos propios (gchar, gint, gint32, guint, guint32, guchar, etc).
- Gestión de memoria: La gestión de memoria es una tarea especialmente delicada, por lo que Glib la administra automáticamente. Si el programador necesita mayor control de la memoria entonces tendrá a su disposición gran cantidad de métodos que le permitirán un control más exhaustivo de ésta.
- Estructuras de datos: Las estructuras de datos más comunes como listas enlazadas, arreglos dinámicos, tablas de tablas de claves, pilas y colas, ya están disponibles en Glib.
- Sistema de objetos: El sistema de Objetos de Glib se ha diseñado con miras a flexibilidad y capacidad de adaptarse a las necesidades del usuario. Se ha creado con la intención de integrarse fácilmente con otros lenguajes de programación diferentes de C. Todos los objetos de GTK+ se derivan de la clase fundamental de Glib: Gobject.
- Bucle de ejecución: En lugar de crear un bucle de eventos nosotros mismos, podemos dejarle la tarea a Glib para centrarnos en el diseño y desarrollo de nuestras aplicaciones asíncronas. Glib se encarga de la distribución de señales y mensajes a las diferentes partes de nuestro programa. También se encarga de administrar alarmas (temporizadores para aplicaciones síncronas), momentos de inactividad de la aplicación, eventos de entrada y salida en tuberías, sockets, o descriptores de archivos, así como hilos.

2.4 Vala

Extracto de la Traducción al castellano del tutorial de Vala

Vala es un nuevo lenguaje de programación que permite utilizar modernas técnicas de programación para escribir aplicaciones que funcionan con las bibliotecas de tiempo de ejecución de GNOME, particularmente GLib y GObject. Esta plataforma ha proporcionado durante mucho tiempo un entorno de programación muy completo, con características como un sistema de tipado dinámico y gestión asistida de memoria. Antes de crear Vala, la única manera de programar para la plataforma era con la API nativa de C, que expone muchos detalles no deseados, con un lenguaje de alto nivel que tiene una máquina virtual auxiliar, como Python o el lenguaje C# de Mono o, alternativamente, con C++ a través de una biblioteca contenedora (wrapper)

Vala es diferente a todas estas otras técnicas, ya que genera código C que se puede compilar para ejecutarse sin necesidad de bibliotecas externas aparte de la plataforma GNOME

CAPÍTULO 3

Introducción

3.1 Tipos de datos

Dentro de `Glib` hallaremos tipos de datos útiles que nos harán más fácil la tarea de construir programas en `GTK+`. `Glib` crea sus propios tipos de datos con el objetivo de hacer más sencillo transportar programas entre diferentes tipos de sistemas operativos donde los tipos estándar de C pudieran diferir en tamaño o alineación de bytes; `Glib` se encargará de esas diferencias.

Los tipos que se usan en `Glib` no son muy diferentes de los que usamos en el estándar de C. Así que realmente no resultará nada complicado habituarse a ellos. La Tabla 1 muestra una comparación entre los tipos de datos de `Glib` y su equivalencia en ANSI C[5].

Definidos en `Glib`	Equivalencia en el estándar ANSI de C	Equivalencia en VALA
gchar	typedef char gchar;	public struct char
gshort	typedef short gshort;	public struct short
glong	typedef long glong;	public struct long
gint	typedef int gint;	public struct int
gboolean	typedef int gboolean;	public struct long
gpointer	typedef void* gpointer;	??

Algunos tipos no tan conocidos como `void *`, son realmente útiles para pasar diferentes estructuras de datos a funciones. Otros tipos definidos en `Glib` se muestran en la siguiente tabla.

Tipo definido en `Glib`	Equivalencia en ANSI C
gint8	typedef signed char gint8;
gint16	typedef signed short gint16;
gint32	typedef signed int gint32;
gint64	typedef signed long long gint64;
guint8	typedef unsigned char guint8;
guint16	typedef unsigned short guint16;
guint32	typedef unsigned int guint32;
guint64	typedef unsigned int guint64;
guchar	typedef unsigned char guchar;
gushort	typedef unsigned short gushort;
gulong	typedef unsigned long gulong;
gsize	typedef unsigned int gsize;
gssize	typedef signed int gssize;
`gconstcopointer`	`typedef const void *gconstpointer;`

Tabla 2: Otros tipos de datos definidos en `Glib`.

En lugar de incrementar la complejidad a las aplicaciones, los tipos de datos de `Glib` compactan complicadas definiciones de tipos en identificadores de tipos de datos que se reconocen más fácilmente, pues son auto descriptivos. Esto elimina gran cantidad de trabajo a la hora de mantener programas muy grandes y el código se hace más legible.

3.2 Límites de tipos de datos

Por cada tipo de dato definido en la tablas 1 y 2 `Glib` ha definido valores indicando los valores mínimos y máximos que pueden contener, exceptuando los tipos de datos `gpointer`, `gconstpointer`, `gssize` y `gboolean`.

En lugar de memorizar nombres complicados podemos adivinar los límites de un tipo de datos mediante su nombre. Por ejemplo, `gint` está definido entre los límites `G_MAXINT` y `G_MININT` mientras que un `gdouble` se encuentra acotado entre `G_MAXDOUBLE y ``G_MINDOUBLE`. Por conveniencia la tabla 3 agrupa los tipos de datos y sus límites definidos.

Tabla 3: Límites de tipos de datos

Tipo de dato	Límites
`gint`	`G_MININT` a `G_MAXINT`
`gshort`	`G_MINSHORT` a `G_MAXSHORT`
`glong`	`G_MINLONG` a `G_MAXLONG`
`guint`	0 a `G_MAXUINT`
`gushort`	0 a `G_MAXUSHORT`
`gulong`	0 a `G_MAXULONG`
`gint8`	`G_MININT8` a `G_MAXINT8`
`gint16`	`G_MININT16` a `G_MAXINT16`
`gint32`	`G_MININT32` a `G_MAXINT32`
`gint64`	`G_MININT64` a `G_MAXINT64`
`guint8`	0 a `G_MAXUINT8`
`guint16`	0 a `G_MAXUINT16`
`guint32`	0 a `G_MAXUINT32`
`guint64`	0 a `G_MAXUINT64`
`gfloat`	`G_MINFLOAT` a `G_MAXFLOAT`
`gdouble`	0 a `G_MAXUSHORT`
`gsize`	0 a `G_MAXSIZE`

3.3 Macros y definiciones útiles

Glib viene empacada con una gran cantidad de Macros que nos evitan gran cantidad de trabajo. A continuación revisaremos un conjunto de Macros y definiciones útiles y de uso común.

3.3.1 Macros básicas

Las dos macros más sencillas, sin embargo útiles son TRUE y FALSE. Solamente definen un cero para un valor de tipo gboolean falso y diferente de cero para un valor positivo. Sin embargo es mucho mas intuitivo y explicativo trabajar con TRUE y FALSE. Veamos una comparación.

3.4 Macros referentes al Sistema Operativo

Este conjunto de macros son útiles al escribir programas que funcionarán en diferentes Sistemas Operativos. G_OS_WIN32, G_OS_BEOS, G_OS_UNIX solo están definidas en su respectiva plataforma y deben utilizase entre bloques de directivas del compilador «#ifdef ... #elif ...#endif», lo que causará que el compilador construya secciones de código diferentes para cada plataforma. Veamos el siguiente ejemplo.

```
#include <glib.h>
/* ... */
#ifdef G_OS_WIN32
    const gchar *dispositivo = "COM1";
#elif G_OS_BE_OS
    const gchar *dispositivo = "/dev/usb0";
#else // G_OS_UNIX
    const gchar *dispositivo = "/dev/ttyS0";
#endif
/* ... */
```

El ejemplo anterior definirá una compilación condicional en la cual, dependiendo de la plataforma donde se compile el programa, la cadena dispositivo tendrá diferente valor en cada Sistema Operativo.

Las macros G_DIR_SEPARATOR y G_DIR_SEPARATOR_S contienen el carácter separador de directorios. Su valor es '/' en sistemas tipo UNIX y '\' en sistemas Windows. La segunda macro contiene la misma información que la primera pero en formato de cadena: '/' y '\'. G_IS_DIR_SEPARATOR(c) acepta un carácter c y determina si es el carácter separador de directorios. Esta macro devuelve TRUE si el carácter es '/' en sistemas UNIX o '\' en Sistemas Windows.

G_SEARCHPATH_SEPARATOR y G_SEARCHPATH_SEPARATOR_S devuelven el carácter separador de rutas en forma de carácter o cadena respectivamente. Este carácter es ':' para sistemas UNIX y ';' para Windows.

3.4.1 Macros y constantes matemáticas

Existen ciertas operaciones matemáticas comunes que no se encuentran disponibles en la biblioteca estándar de C.

MIN(a, b) y *MAX(a, b)* calculan el valor mínimo y máximo de entre dos números *a* y *b*, mientras que *ABS(n)* calcula el valor absoluto de un número *n*.

CLAMP(x, a, b) se asegura de que el número x se encuentre dentro de los límites a y b. Si x se encuentra dentro de estos límites, CLAMP() devolverá el número x, si esto no se cumple y x es mayor que el límite superior b, CLAMP() regresará este valor, de lo contrario x es menor que el límite inferior a, CLAMP() regresará el valor de límite inferior a. Esta macro resulta confusa, pero es útil al posicionar objetos gráficos en la pantalla y simular cierta resistencia al movimiento.

La siguiente tabla muestra constantes matemáticas predefinidas en `Glib`. En la documentación de `Glib` existen uniones para acceder al signo, la mantisa y el exponente de números de tipo coma flotante que cumplan con el estándar IEEE 754.

Símbolo matemático	Definición en `Glib`	Valor
π	`G_PI`	`3.14159265358979323846264338327950288419 71`
$\frac{\pi}{2}$	`G_PI2`	`1.57079632679489661923132169163975144209 85`
$\frac{\pi}{4}$	`G_PI4`	`0.78539816339744830961566084581987572104 92`
$\sqrt{}$	`G_SQRT2`	`1.41421356237309504880168872420969807856 96`
e	`G_E`	`2.71828182845904523536028747135266249775 72`
$\ln(2)$	`G_LN2`	`0.69314718055994530941723212145817656807 55`
$\ln(10)$	`G_LN10`	`2.30258509299404568401799145468436420760 11`
$\log_{10}(2)$	`G_LOG2_BASE10`	`2.30258509299404568401799145468436420760 11`

Tabla 4: Constantes matemáticas predefinidas en `Glib`.

3.4.2 Macros para verificación de errores, excepciones y depurado

Un buen diseño de software no viene de la noche a la mañana. Parte importante del tiempo de desarrollo de un programa se consume en la depuración de errores. También es cierto que parte importante del total del código fuente escrito de un programa robusto se dedica a la validación y corrección de posibles errores, es decir, que las cosas que deban estar en orden realmente lo estén.

Los desarrolladores de `Glib` nos ofrecen diferentes herramientas: 7 macros para ayudarnos a mejorar nuestros programas.

La macro g_assert() recibe como parámetro una expresión, tal y como se usa en el condicional `if... then ... else ...` Si la condición especificada falla o es `FALSE`, el programa termina especificando un mensaje de error.

Un buen ejemplo de aplicación de estas macro se daría en un función que transforma cadenas provenientes, por ejemplo, de una comunicación serial.

```
#include <glib.h>
/* ... */
g_assert (cadena == NULL);
/* ... */
```

En el ejemplo anterior, el programa terminara con un mensaje de error si la cadena es null.

Estas macros puede desactivarse en compilaciones finales mediante la definición de G_DISABLE_ASSERT al momento de compilar la aplicación.

`g_return_if_fail()` toma una expresión y regresa de la función si tal expresión no resulta verdadera o TRUE.De lo contrario registra un mensaje de aviso y regresa de la función.

`g_return_if_fail()` sólo se puede utilizar en funciones que no regresan ningún valor. Para aquellas funciones que debe regresar un valor, esta `g_return_val_if_fail(expr, val)`, que regresa el valor `val` en función del la expresión `expr` al igual que `g_return_if_fail()`.

Parecido al par anterior, `g_return_if_reached()` y `g_return_val_if_reched()` regresan de la función si alguna vez son ejecutadas. La primera macro no toma ninguna expresión mientras que la segunda espera como parámetro el valor que ha de regresar la función.

Por último `G_BREAKPOINT` inserta una instrucción de punto de rompimiento con el objeto de depurar el programa. Esta macro solo está disponible en la arquitectura x86.

3.4.3 Macros para manejo de memoria

Como hemos discutido previamente, `Glib` maneja la memoria de de los objetos que nosotros creamos, pero también nos ofrece la posibilidad de tomar el control de la memoria en nuestras manos. Esto es conveniente si trabajamos con vectores o matrices que cambian de tamaño o estamos implementando un nuevo objeto. Gran parte de las funciones de `Glib` se basan en la implementación disponibles en la librería estándar de C de UNIX. Una región de memoria tiene un ciclo de vida simple, como el mostrado en la Figura .

Figura aca

Comencemos con la macro que define un puntero nulo: `NULL`. Está definida en prácticamente cualquier implementación de C. Esta macro es útil para inicializar punteros a memoria o estructuras vacías, por ende, un objeto que no está inicializado contiene un puntero nulo.

Kernighan y Ritchie establecieron tres funciones para manejar memoria de manera dinámica: `malloc()`, `calloc()` y `free()`. Estas pueden cubrir por completo el proceso mostrado en la figura .

El primer paso del ciclo de vida de un bloque de memoria es la función estándar de C `malloc()`:

```
void *malloc(size_t n);
```

La función `malloc()` toma como único parámetro el número de bytes de memoria a reservar. Si tal petición no pudo completarse regresará entonces el puntero `NULL`.

Por otro lado se encuentra `calloc()`, cuyo prototipo es:

```
void *calloc(size_t n, size_t size);
```

La funcion `calloc()` reservará memoria para un arreglo de `n` estructuras de tamaño `size`. Como `malloc()` y `calloc()` regresan punteros de tipo `void`, se hace necesario hacer un *casting* o moldeado al tipo deseado. Ve el siguiente ejemplo.

```
int *ip;
ip = (int *) calloc(n, sizeof(int));
```

Con el objetivo de no recibir quejas del compilador de C, debemos moldear correctamente el puntero a la memoria reservada que nos entrega `calloc()`.

Cerrando el ciclo de vida de una región de memoria creada dinámicamente, se encuentra `free()`, el cual libera la memoria asignada a un puntero en especial.

`Glib` ofrece `g_malloc()` y `g_free()`; ambas funciones operan de igual manera que sus homólogas en la librería estándar de C, sólo que trabajan con el tipo `gpointer`. Además de las dos funciones anteriores, existe un abanico de posibilidades que ahorran gran cantidad de trabajo al crear una región de memoria.

Para reservar memoria para una colección de estructuras, `GLib` tienen las macros `g_new()` y *g_new0()*‘. Estas macros reservan memoria para un numero de estructuras determinado por `n_structs`. El tipo de esas estructuras esta determinado por el parametro: `struct_type`.

La diferencia entre las dos macros es que `g_new0()` inicializará a cero la región de memoria.

Ambas macros regresan un puntero a la memoria reservada, este puntero ya estará moldeado a `struct_type`. Si ocurriera un error al reservar reservar el número indicado de estructuras en memoria el programa se abortara con un mensaje de error.

La versión más segura de las macros anteriores se encuentran en `g_try_new()` y `g_try_new0()` las cuales regresarán un puntero `NULL` moldeado a `struct_type`, en lugar de abortar el programa.

El ciclo de memoria dinámica incluye cambiar el tamaño de ésta, para ello tendremos dos macros:

Ambas cambian el tamaño de una región de memoria a la que apunta `mem`. La nueva región de memoria contendrá `n_structs` de tipo `struct_type`.

La función `g_try_renew()` regresa un puntero `NULL` moldeado a `struct_type` en caso de error, mientras que `g_renew()` abortaría el programa. En ambos casos, cuando la memoria ha podido ser reservada, se regresa un puntero a la nueva región de memoria.

Existen otras macros como `g_memove()` o `g_newa()`.

3.4.4 Macros de conversión de tipos

Las aplicaciones escritas en `GTK+` usualmente necesitan pasar datos entre las diferentes partes del programa.

Conforme avancemos veremos que será muy común convertir un tipo de dato en otro; es por eso que `Glib` define seis macros básicas de conversión de tipos casi cualquier objeto o widget que usemos; son simples casting o moldeado en C, esta técnica permite que `GTK+` se comporte como una librería orientada a Objetos.

La manera de pasar datos de una parte de la aplicación a otra generalmente se hace utilizando `gpointer`, el cual es lo equivalente a un puntero `void`.

Pero existe una limitante al querer pasar números en lugar de estructuras de datos. Si, por ejemplo, deseáramos pasar un número entero en lugar de una estructura de datos deberíamos de hacer algo lo siguiente:

```
gint *ip = g_new (int, 1);
*ip = 42;
```

Los punteros tienen un tamaño de al menos 32 bits en las plataformas que `Glib` está disponible. Si vemos con detalle, el puntero‘‘ip‘‘ es puntero a una constante de tipo `gint`. Es decir, hay un puntero que apunta a una región de memoria de 32 bits, al menos. Nosotros tendremos que hacernos cargo de liberar la memoria del número entero, en base a esto podríamos tratar de asignar el valor que queremos pasar a un puntero:

```
gpointer p;
int i;
p = (void*) (long) 42;
i = (int) (long) p;
```

Pero esto es incorrecto en ciertas plataformas y en tal caso habría que hacer lo que sigue:

```
gpointer p;
int i;
p = (void*) (long) 42;
i = (int) (long) p;
```

Esto se vuelve demasiado complicado como para llevarlo a la práctica, por eso los desarrolladores de `Glib` han creado las macros `GINT_TO_POINTER()`, `GUINT_TO_POINTER()` y `GSIZE_TO_POINTER()` para empacar un `gint`, `guint` o `gsize` en un puntero de 32 bits.

Análogamente `GPOINTER_TO_GINT()`, `G_POINTER_TO_GUINT()` y `GPOINTER_TO_GSIZE()` sirven para obtener el número que se ha empacado en el puntero de 32 bits. El ejemplo anterior se cambia a:

```
#include <glib.h>
gpointer p;
17gint i;
p = GINT_TO_GPOINTER(42);
i = GPOINTER_TO_GINT(p);
```

No es buena idea tratar de empacar en un puntero otro tipo de dato que no sea `gint` o `guint`; la razón de esto es que estas macros solo preservan los 32 bits del entero, cualquier valor fuera de estos límites será truncado.

De igual manera es incorrecto guardar punteros en un entero, por las mismas razones expuestas arriba, el puntero será truncado y conducirá a gran cantidad de fallos en el programa.

3.4.5 Tratamiento de mensajes

`Glib` contiene funciones para mostrar información tales como mensajes del programa o mensajes de error. Normalmente podríamos llamar a `printf()` y desplegar toda aquella información que deseemos `Glib` tiene un sistema de tratamiento de mensajes mucho más sofisticado, pero a la vez sencillo de usar.

Para comenzar, debes saber que existen tres niveles de despliegue de mensajes:

1. Despliegue de información variada . Este tipo de mensajes se considera inocuos o de carácter meramente informativo, como por ejemplo el estado de un proceso. 2. Registro de mensajes y advertencias . Mensajes que contienen información crucial para el funcionamiento interno del programa; los eventos que generan estos mensajes no son fatales y el programa puede continuar su ejecución. 3. Registro y despliegue de errores . Los mensajes de error se consideran fatales y solo deben ser utilizados cuando el evento que se esta reportando ha sido de tal impacto que el programa no debe continuar. Como ejemplo tenemos problemas de direccionamiento y asignación de 18memoria, fallas en el hardware y problemas de seguridad. El resultado de desplegar un mensaje de error es la terminación definitiva del programa.

3.4.6 Despliegue de información variada

Comenzamos con g_print(). `g_print()` funciona de manera idéntica a `printf()` de C.

Pero a diferencia de `printf()`, que manda cualquier mensaje directamente a la salida estándar de C (stdout), `g_print()` lo hace a través de un manejador. Este manejador, que usualmente es `printf()`, puede ser cambiado a conveniencia. Este manejador puede, en lugar de sacar mensajes a `stdout`, hacerlo a un archivo o a una terminal en un puerto serial. El explicar como registrar el manejador de `g_print()` allanará el camino para el siguiente capítulo. Un manejador (handler, en el idioma anglosajón), es el puntero a una función escrita por el programador. El prototipo de la función que servirá como manejador de `g_print()` es el siguiente:

```
void mi_manejador (const gchar *string);
```

El puntero de esta función es simplemente su nombre. Este puntero se provee como parámetro de otra función que lo registra como manejador de `g_print()`: g_set_print_handler()

En el siguiente ejemplo mostraremos la facilidad de uso y versatilidad de `g_print()` usando un manejador simple.

Listado de Programa 2.3.1

```
/*****************************************************************************
*
Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo: glib-gprint.c
* Descripcion: Uso del manejador de g_print()
* Comentarios: Demuestra el funcionamiento de g_print() y g_print_handler()
*
*
*****************************************************************************/
#include <glib.h>
/*Para usar g_printf()*/
#include <glib/gprintf.h>

/* Funcion manejadora de g_print */
void mi_manejador (const gchar *string){
```

(continué en la próxima página)

(proviene de la página anterior)

```
    g_fprintf(stdout,"mi_manejador:");
    g_fprintf(stdout,string);
}
/* Programa principal */
int main (int argc, char **argv){

    GPrintFunc viejo;
    g_print("Usando g_print() sin manejador\n");
    g_print("Estableciendo el nuevo manejador de g_print() ..\n\n");
    viejo = g_set_print_handler(&mi_manejador);
    g_print ("Impresion Normal\n");
    g_print ("Impresion de numeros: %i, %f, 0x%x\n",1,1.01,0xa1);
    g_print("Restableciendo el antiguo manejador de g_print() ..\n\n");
    viejo = g_set_print_handler(viejo);
}
g_print("Fin\n");
return (0);
```

El programa listado imprime un par de mensajes usando el manejador por defecto de `g_print()`, lo cual no presenta demasiada dificultad. La parte más importante viene a continuación. Usando la variable `viejo` guardamos el puntero al manejador por defecto de `g_print()` e inmediatamente establecemos el nuevo manejador, el cual es nuestra propia función: `mi_manejador()`. Inmediatamente se pone a prueba nuestro nuevo manejador imprimiendo algunos mensajes de texto y números. Tomemos en cuenta que el manejador solo recibe una cadena y no tiene que estar lidiando con parámetros variables y quien se encarga de esto es `Glib`. Posteriormente se restablece el manejador original de `g_print()` y todo vuelve a la normalidad. La comprensión de este sencillo ejemplo es vital para todo el curso, pues no estamos trabajando con instrucciones comunes y corrientes en el lenguaje C, si no con punteros a funciones y estructuras complejas de datos. Este tipo de tópicos por lo general es evitado en los cursos universitarios del lenguaje C.

El siguiente ejemplo es un método interactivo para seleccionar el comportamiento de `g_print()`.

Listado de Programa 2.3.2

```
/*****************************************************************************
*
Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo: glib-gprint2.c
* Descripcion: Uso del manejador de g_print()
* Comentarios: Ejemplo alternativo para el uso del manejador
* de g_print()
*
*****************************************************************************/
#include <glib.h>
/*Para usar g_printf()*/
#include <glib/gprintf.h>

/* Funcion manejadora de g_print */
void mi_manejador (const gchar *string){
    g_fprintf(stdout,"mi_manejador: ");
    g_fprintf(stdout,string);
}

void muestra_ayuda( void ) {
    printf("\nError, no ha indicado ningun parametro, o es invalido.\n");
    printf("uso:\n\t--normal g_print normal\n\t--manejador g_print con manejador\n");
```

(continué en la próxima página)

(proviene de la página anterior)

```
}

/* Programa principal */
int main (int argc, char **argv) {
    GPrintFunc viejo;

    if (argc <= 1){
        muestra_ayuda();
        return 0;
    }

    if (g_str_equal(argv[1],"--normal")){
        printf("--=== Usando tratamiento normal de mensajes ===--\n");
    } else if (g_str_equal(argv[1],"--manejador")) {
        printf("--=== Usando tratamiento con manejador ===--\n");
        viejo = g_set_print_handler(&mi_manejador);
    } else {
        muestra_ayuda();
        return 0;
    }

    /*Imprime algunos mensajes*/
    g_print ("Hola mundo!\n");
    if (g_str_equal(argv[1],"--manejador")) {
        g_set_print_handler(viejo);
    }

    return 0;
}
```

El manejador de `g_print()` es el mismo que en el listado de programa 2.3.1. Este ejemplo es un programa pensado para la línea de comandos. Si se ejecuta este programa sin ningún parámetro se ejecutará la función `muestra_ayuda()`. Ocurre lo mismo si no se especifican los parámetros correctos. Solo se aceptan dos parámetros que permiten elegir entre usar o no el manejador de `g_print()`.

3.4.7 Registro de mensajes y advertencias

Es muy buena práctica el clasificar nuestros mensajes debido a su severidad. Para esta tarea `GTK+` nos ofrece tres herramientas:

- `g_message()` es una macro que registra e imprime un mensaje en la salida estándar. Este mensaje se considera informativo e inocuo.
- `g_debug()` es una macro que registra e imprime un mensaje en la salida de error estándar. Este mensaje es útil para propósito de depurado de la aplicación.
- `g_warning()` se utiliza normalmente para avisar acerca de algún evento que ha ocurrido el cual no es lo suficientemente fatal como para que el programa no pueda continuar.

Veamos el siguiente ejemplo:

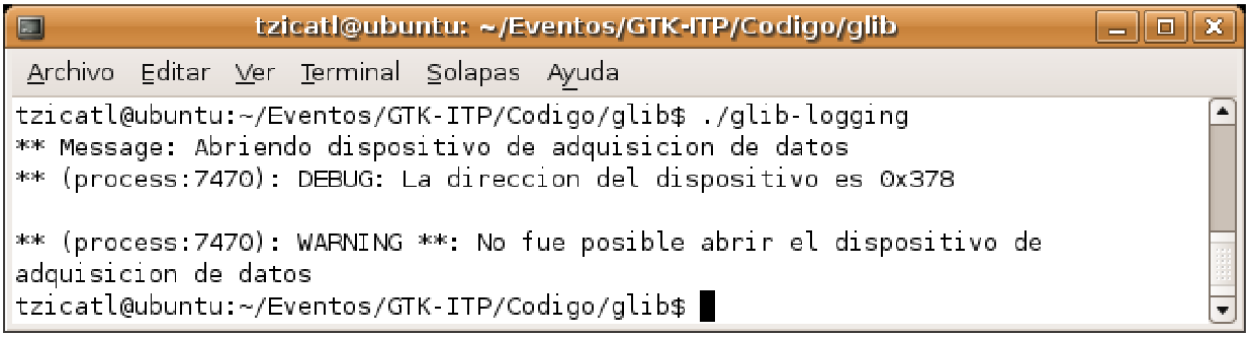
```
/*****************************************************************************
*
Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo: glib-logging.c
```

(continué en la próxima página)

(proviene de la página anterior)

```
* Descripcion: Uso de macros de registro de mensajes de ``Glib``
*
*********************************************************************/
#include <glib.h>
int main (int argc, char **argv) {
    g_message("Abriendo dispositivo de adquisicion de datos");
    g_debug ("La direccion del dispositivo es 0x378");
    g_warning ("No fue posible abrir el dispositivo de adquisicion de datos");
    return 0;
}
```

Si ejecutamos este programa obtendremos la siguiente salida:

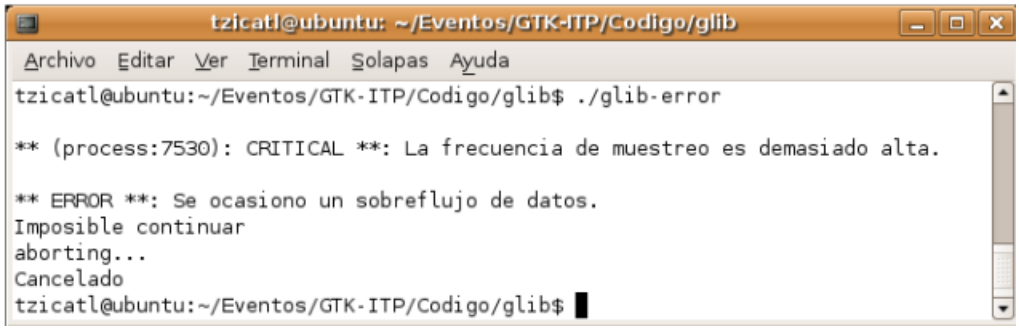


3.4.8 Registro y despliegue de errores

Estas son macros de `Glib` para el registro de errores:

- `g_critical()` avisa de algún error crítico en la aplicación. Un error crítico se define dependiendo de cada aplicación, para algunos un error critico es recuperable y para otros no. Este error se dirige a la salida de error estándar.
- `g_error()` avisa de un error grave en un programa. Sólo se debe utilizar g_error() para avisar para comunicar errores que de todas formas harían que la aplicación terminara. El uso de esta macro ocasionará que la aplicación termine.

```
/*********************************************************************
*
Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo: glib-error.c
* Descripcion: Uso de macros de registro de mensajes de ``Glib``
* Comentarios: Estos mensajes son de indole grave o fatal.
*
*
*********************************************************************/

#include <glib.h>

int main (intargc, char **argv)
{
    g_critical("La frecuencia de muestreo es demasiado alta.");
    g_error("Se ocasiono un sobreflujo de datos. \nImposible continuar ");
    return 0;
}
```



3.4.9 Tratamiento de cadenas

Según `Kernighan & Ritchie <http://es.wikipedia.org/wiki/El_lenguaje_de_programaci%C3%B3n_C>` una cadena es arreglo o vector de caracteres terminados con el carácter nulo ``'\0'`` para que los programas puedan encontrar el final de la cadena.

El uso de cadenas comienza a volverse peligroso cuando se subestima su poder. Una cadena puede ser un vector o un puntero. La diferencia sutil entre estas dos características puede determinar si el programa gotea memoria o que reviente.

Por ejemplo, una mala práctica de programación, que es usual entre programadores no experimentados, es utilizar regiones de memoria estáticas para almacenar cadenas de texto: si por alguna razón escribimos datos más allá de los límites de la cadena seguramente estaremos escribiendo en el espacio de otra variable o incluso en parte del código del programa. Esto conduce a errores muy difíciles de depurar. Además de lo anterior, las regiones de memoria estáticas representan un riesgo de seguridad, pues su debilidad inherente es ampliamente usada para instrumentar ataques informáticos llamados Buffer Overflow. En este procedimiento el atacante , previo conocimiento de la vulnerabilidad del sistema, sobreescribe a voluntad otras celdas de memorias que contienen datos o código del programa, haciendo que éste falle o se comporte de forma determinada.

Por otro lado, el tratamiento clásico de cadenas goza de gran popularidad. El tratamiento de cadenas es un tópico importante para cualquier programa. `Glib` aborda el problema desde dos perspectivas diferentes:

- Perspectiva procedimental: `Glib` ofrecer una vasta colección de rutinas de manejo de cadenas similares a las encontradas en la librería string.h de la librería estándar de C. Algunas adiciones buscan facilitar las tareas del programador.
- Perspectiva orientada a objetos: `Glib` pone a disposición de nosotros GString, un objeto cuyo funcionamiento esta basado en las cadenas del estándar de C, pero tratando de mejorar los problemas que encontremos al manejar cadenas de la manera tradicional.

3.4.10 Perspectiva procedimental

Existe una gran variedad de funciones de tratamiento de cadenas en `Glib`. Resultaría ineficaz el tratar todas en este documento. A continuación haremos reseña de un pequeño conjunto de funciones útiles en el tratamiento de cadenas demostrando el uso de g_strdup(), g_strrstr(), g_strstr_len(), g_str_has_prefix(), g_str_has_suffix(), g_str_equal(),

Ejemplo de `g_strdup`.

```
gchar* g_strdup (const gchar *str);
```

Descripción: Duplica una cadena.

Parámetros:

- *str*: un puntero a la cadena a duplicar.

- **Valor de retorno**: La cadena duplicada en otra región de memoria. Si NULL se ha especificado como parámetro de entrada, el valor de retorno también será NULL. El programador es responsable de liberar la memoria de la nueva cadena.

Ejemplo de `g_strrstr`.

```
gchar* g_strrstr (const gchar *haystack, const gchar *needle);
```

Descripción: Busca una aguja(needle) dentro de un pajar (haystack). Las cadenas de entrada debe estar terminadas con el carácter nulo.

Parámetros:

- haystack: La cadena donde se busca (pajar).
- needle: El texto que se busca (aguja).

Valor de retorno: Se regresa un puntero a donde se encontró la primera ocurrencia de la aguja dentro del pajar. Si no se encontraron coincidencias entonces se regresa NULL.

Ejemplo de `g_strstr_len`.

```
gchar*
g_strstr_len
(const gchar *haystack,
gssize haystack_len,
28const gchar *needle);
```

Descripción: Esta es una versión de la función g_strstr(). Esta versión limitará su búsqueda en el pajar a un número de caracteres igual a haystack_len.

Parámetros:

- haystack: La cadena donde se busca (pajar).
- haystack_len: Número máximo de caracteres que se examinarán del pajar.
- needle: El texto que se busca (aguja).

Valor de retorno: Se regresa un puntero a donde se encontró la primera ocurrencia de la aguja dentro del pajar. Si no se encontraron coincidencias entonces se regresa NULL.

Ejemplo de `g_str_has_prefix`.

```
gboolean
g_str_has_prefix
(const gchar *str,
const gchar *prefix);
```

Descripción: Nos dice si la cadena str tiene el prefijo especificado.

Parámetros:

- str: La cadena de quien se desea determinar el prefijo.
- prefix: El prefijo.

Valor de retorno: Regresa TRUE si la cadena comienza con prefix. FALSE en caso contrario.

Ejemplo de `g_str_has_suffix`.

```
gboolean
g_str_has_suffix
(const gchar *str,
const gchar *suffix);
```

Descripción: Nos dice si la cadena str tiene el sufijo especificado.

Parámetros:

- str: La cadena de quien se desea determinar el sufijo.
- suffix: El sufijo.

Valor de retorno: Regresa TRUE si la cadena termina con suffix. FALSE en caso contrario.

Ejemplo de `g_str_equal`.

```
gboolean
g_str_equal
(gconstpointer v1,
gconstpointer v2);
```

Descripción: Esta función verifica que las dos cadenas sean iguales.

Parámetros:

- v1: Una cadena.
- v2: Otra cadena que se comparará contra v1..

Valor de retorn: Regresa TRUE si ambas cadenas son idénticas. Esta función esta preparada para ser usada en estructuras de datos que necesiten comparación, como listas enlazadas, tablas de claves o arboles binarios 5 .

3.4.11 Perspectiva Orientada a Objetos: `GString`

`GString` es un objeto que se encarga de los detalles de la administración de memoria, de tal manera que el programador no tenga que ocuparse de liberar o reservar memoria.

Recordemos que GLib nos provee de lo necesario para hacer programación orientada objetos, pero en un lenguaje procedural como C. Decimos que `GString` es un objeto, pero en realidad esta implementado como una estructure. Visto desde ese aspecto, `GString` define tres miembros públicos a los que se puede acceder directamente.

```
typedef struct {
    gchar *str;
    gsize len;
    gsize allocated_len;
} GString;
```

La propiedad `str` contendrá el texto de la instancia, mientras que `len` contendrá la longitud de la cadena, sin contar los caracteres de terminación de cadena.

El constructor de clase de `GString` es el siguiente:

```
GString* g_string_new(const gchar *init);
```

Opcionalmente toma un parámetro: `init` que será la cadena con que se inicializará el objeto. Si quieres que la cadena este vacía puedes pasar la macro NULL como parámetro. Veamos un ejemplo:

```
#include <glib.h>
/*....*/
Gstring *cadena, cadena_vacia;
cadena = gstring_new("Hola");
cadena_vacia = gstring_new(NULL);
```

Por conveniencia, GLib provee otros constructores: g_string_new_len() y g_string_sized_new() <https://developer.gnome.org/glib/2.41/glib-Strings.html#g-string-sized-new>‘_

Todos los constructores regresan el puntero a una nueva instancia de GString.

Una vez que tenemos una instancia del objeto GString podemos manipular su contenido mediante algunas de las funciones del API de GString, como por ejemplo *g_string_assign() <https://developer.gnome.org/glib/unstable/glib-Strings.html#g-string-assign>* _, g_string_append(), g_string_append_c(), g_string_prepend(), g_string_prepend_c(), g_string_ascii_up() o *g_string_ascii_down() <https://developer.gnome.org/glib/unstable/glib-String-Utility-Functions.html#g-string-ascii-down>*. Veamos un ejemplo.

```
# Define un nuevo valor para la cadena,
g_string_assign(cadena, "Nuevo valor");

# Añade caracteres al inicio y al final de la cadena almacenada en Gstring
g_string_append_c(cadena,'Z');
g_string_prepend_c(cadena,'A');

#Añade otra cadena al final de GString
g_string_append (cadena, "Añadiendo valor al final");
g_string_prepend(candena,"Añadiendo valor al Principio");

# Tambien es posible truncar la longitud de la cadena,
# por ejemplo 0 significa que la cadena se limpia...
g_string_truncate(cadena,0);

# Convertir la cadena a mayúsculas o minúsculas ...
g_string_ascii_up(cadena);
g_string_ascii_down(cadena);
```

Finalmente, cuando llegue el momento de destruir la instancia de GString deberemos usar g_string_free().

```
g_string_free(cadena, TRUE);
```

Nota: Debemos tener cuidado con el segundo parámetro de g_string_free(). Éste parámetro define si junto con el valor de la cadena también se destruye el la instancia del objeto. Pasa el parámetro FALSE si la instancia se está usando en algún otro lado del programa. Si ya no planeas utilizar más este objeto pasa TRUE como parámetro.

Finalmente, aca pongo el ejemplo completo de manipulacion de cadenas.

```
/*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo: glib-gstring1.c
* Descripcion: Ejemplo de tratamiento de cadenas con GLib
* Comentarios: Revision del ciclo de vida de GString
*
*****************************************************************************/
#include <glib.h>
```

(continué en la próxima página)

(proviene de la página anterior)

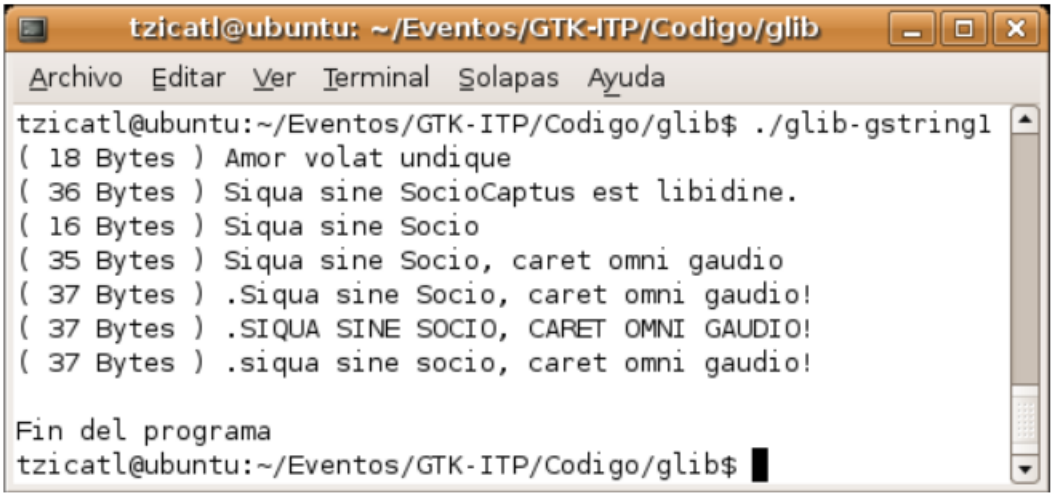
```
int main () {
    GString *cadena;

    /* Se crea una instancia de GString con un valor"*/
    cadena = g_string_new("Amor volat undique");
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);

    /*Reemplazando el contenido de la cadena*/
    g_string_assign(cadena, "Captus est libidine.");

    /* Inserta algun texto al principio de la cadena*/
    g_string_prepend(cadena,"Siqua sine Socio");
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);

    /*El valor de la cadena se trunca*/
    g_string_truncate(cadena,16);
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);

    /*Se inserta algun texto al fin de la cadena*/
    g_string_append(cadena,", caret omni gaudio");
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);

    /*Se insertan caracteres al incio y al fin de la cadena*/
    g_string_append_c(cadena,'!');
    g_string_prepend_c(cadena,'.');
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);

    /*Se convierte la cadena a Mayusculas */
    g_string_ascii_up(cadena);
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);

    /*Se convierte la cadena a Mayusculas */
    g_string_ascii_down(cadena);
    g_print("( %i Bytes ) %s\n", cadena->len, cadena->str);
    g_print("\nFin del programa\n");
    g_string_free(cadena,TRUE);

    return 0;
}
```

Compila el ejemplo anterior con el siguiente comando:

```
gcc `pkg-config --cflags glib-2.0` glib-gstring1.c -o glib-gstring1.c `pkg-config --
↪libs glib-2.0`
```

Finalmente, ejecuta el programa

```
./glib-gstring1
```



3.5 Estructuras de datos: Listas enlazadas simples

Las estructuras de datos son imprescindibles en el desarrollo de cualquier programa. Nos permiten abordar de una manera razonada y metódica un problema en particular.

3.5.1 Listas enlazadas simples

Las listas enlazadas, al igual que los arreglos y vectores se utilizan para almacenar colecciones de datos. Un buen artículo de listas enlazadas está disponible en la librería de educación de la facultad de ciencias de la computación en la universidad de Stanford[6].

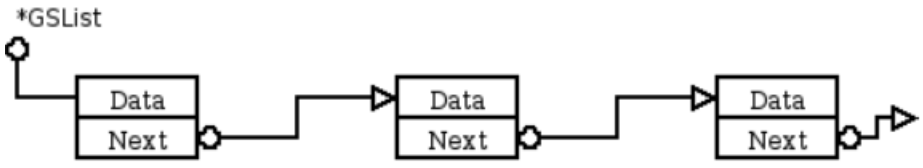
La biblioteca `Glib` incluye una implementación de listas enlazadas en.

Propiedades

La estructura GSList tiene un esquema similar al que se muestra en la Figura 2.5.1, mientras que su estructura en C es la siguiente:

```
typedef struct {
    gpointer data;
    GSList *next;
} GSList;
```

El puntero **data* almacena los datos que se desean coleccionar, mientras que next apunta hacia al siguiente elemento de la lista enlazada.



3.5.2 Constructor de clase

Una lista enlazada simple no tiene constructor de clase en si, pues un puntero con el valor NULL se interpreta como una lista vacía.

El puntero **GSList* siempre se debe inicializar con NULL. El fin de una lista enlazada se encuentra cuando el puntero next contiene el puntero NULL. De ahí que una lista vacía sólo es un puntero NULL. 2.5.3

3.5.3 Funciones asociadas o Métodos de clase

La estructura de datos de GSList indica que nuestras listas enlazadas simples pueden contener cualquier dato. Además de cualquier dato, también contienen un puntero a la siguiente estructura. Los datos contenidos en la estructura de datos pueden ser, por ejemplo, un entero usando cualquiera de las macros de conversión de tipo que se revisaron en el Capitulo 2.2.6, o un puntero a otro tipo de datos como un objeto o una cadena.

Una lista enlazada simple sólo permite recorrer la estructura de datos en una sola dirección (no hay ningún lugar donde diga como regresar a elemento anterior.

Es importante no olvidar estos detalles por que todas los funciones asociadas asumen que el puntero que se les entrega es el inicio de la lista. Así mismo, las funciones que modifican las listas enlazadas pueden cambiar la lista de tal manera que una referencia antigua ya no apunte al nuevo inicio de la lista.

Con las consideraciones anteriores podemos comenzar con nuestra reseña. El siguiente conjunto de funciones sirven para añadir y eliminar elementos.

```
GSList*
g_slist_append
(GSList *list,
gpointer data);
```

Descripción: Añade un elemento al final de la lista. Note que esta función tiene que recorrer toda la lista hasta el final para añadir el elemento. Una lista lo suficientemente larga puede crear problemas de velocidad de ejecución y cuellos de botella, principalmente cuando se añaden varios elementos a la vez. Para estos casos se puede insertar todos los elementos al inicio para posteriorment invertir el orden de la lista.

Parámetros:

- **list**: Una lista enlazada simple.
- **data**: Los datos del elemento a insertar.

Valor de retorno: El nuevo inicio de la lista enlazada simple.

```
GSList*
g_slist_prepend
(GSList *list,
gpointer data);
```

Descripción: Añade un elemento al inicio de la lista. Note que el puntero al nuevo inicio de la lista pudo haber cambiado. Asegúrese de guardar el nuevo valor.

Parámetros:

- **list**: Una lista enlazada simple.
- **data**: Los datos del elemento a insertar.

Valor de retorno: El nuevo inicio de la lista enlazada simple.

```
GSList*
g_slist_insert
(GSList *list,
gpointer data,
gint position);
```

Descripción: Inserta un elemento al en la posición especificada. Note que el puntero al nuevo inicio de la lista pudo haber cambiado. Asegúrese de guardar el nuevo valor.

Parámetros:

- **list**: Una lista enlazada simple.
- **data**: Los datos del elemento a insertar.
- **position**: La posición del elemento a insertar. El elemento se inserta al final si la posición es negativa o es mayor al número de elementos de la lista.

Valor de retorno: El nuevo inicio de la lista enlazada simple.

```
GSList*
g_slist_insert_before
(GSList *slist,
GSList *sibling,
gpointer data);
```

Descripción: Inserta un elemento antes de algún otro elemento. Note que el puntero al nuevo inicio de la lista pudo haber cambiado. Asegúrese de guardar el nuevo valor.

Parámetros:

- **list**: Una lista enlazada simple.
- **sibling**: El elemento del que deseamos que se inserte datos antes de él.
- **data**: Los datos del elemento a insertar.

Valor de retorno: El nuevo inicio de la lista enlazada simple.

```
GSList*
g_slist_insert_sorted
(GSList *list,
gpointer data,
GCompareFunc func);
```

Descripción: Inserta un elemento de manera ordenada. La ordenación se lleva a cabo mediante la función de comparación especificada.

Parámetros:

- list: Una lista enlazada simple.
- data: Los datos del elemento a insertar.
- func: La función que será usada para ordenar lo datos de la lista. Esta función deberá tomar dos parámetros y deberá regresar un valor mayor a cero si el primer parámetro debe ir después del segundo parámetro.

Valor de retorno: El nuevo inicio de la lista enlazada simple.

```
GSList* g_slist_remove (GSList *list, gconstpointer data);
```

Descripción: Remueve un elemento de la lista. Si dos elementos contienen los mismos datos, sólo se removerá el primero. Si no se encuentra el elemento a eliminar entonces la lista queda sin cambios.

Parámetros:

- **list**: Una lista enlazada simple.
- **gconstpointer**: Los datos del elemento a eliminar de la lista.

Valor de retorno: El nuevo inicio de la lista enlazada simple. El siguiente conjunto de funciones son para localizar elementos dentro de la lista enlazada simple.

```
GSList*
g_slist_last
(GSList *list);
```

Descripción: Entrega el último elemento de la lista.

Parámetros:

- **list**: Una lista enlazada simple.

Valor de retorno: El último elemento de la lista enlazada simple.

```
#define
g_slist_next(slist)
```

Descripción: Una macro que entrega el siguiente elemento de la lista. Equivale a slist- >next.

Parámetros:

- **list**: Una lista enlazada simple.

Valor de retorno: El siguiente elemento de la lista enlazada simple. NULL si la lista esta vacía o se ha llegado al último elemento.

```
GSList*
g_slist_nth
(GSList *list,
guint n);
```

Descripción: Entrega el n-ésimo elemento de la lista.

Parámetros:

list: Una lista enlazada simple.

Valor de retorno: El n-ésimo elemento de la lista enlazada simple. NULL si la lista esta vacía o se ha llegado al último elemento.

```
GSList*
g_slist_nth
(GSList *list,
guint n);
```

Descripción: Entrega el n-ésimo elemento de la lista.

Parámetros:

- **list**: Una lista enlazada simple.
- **n**: la posición del elemento, iniciando desde 0.

Valor de retorno: El n-ésimo elemento de la lista enlazada simple. NULL si la lista esta vacía o la posición buscada está fuera de los límites de la lista.

```
gpointer
g_slist_nth_data
(GSList *list,
guint n);
```

Descripción: Entrega los datos del n-ésimo elemento de la lista.

Parámetros:

- **list**: Una lista enlazada simple.
- **n**: la posición del elemento, iniciando desde 0.

Valor de retorno: Los datos del n-ésimo elemento de la lista enlazada simple. NULL si la lista esta vacía o la posición buscada está fuera de los límites de la lista.

```
GSList*
g_slist_find
(GSList *list,
gconstpointer data);
```

Descripción: Encuentra el elemento que contiene los datos especificados.

Parámetros:

- **list**: Una lista enlazada simple.
- **data**: los datos que se buscan.

Valor de retorno: El elemento que contiene los datos. NULL si no se encuentra nada.

```
GSList*
g_slist_find_custom
(GSList *list,
gconstpointer data,
GCompareFunc func);
```

Descripción: Encuentra un elemento aplicando el criterio de la función especificada. La lista se recorre y en cada paso se llama a la función especificada la cual debe regresar 0 cuando se halla encontrado el elemento deseado.

Parámetros:

- list: Una lista enlazada simple.
- data: los datos que se buscan.
- func: la función que se llama por cada elemento. Esta función debe de tomar dos punteros de tipo gconstpointer, los cuales son los datos del nodo que se esta iterando y los datos que se buscan, respectivamente

Valor de retorno: El elemento que contiene los datos. NULL si no se encuentra nada. 44Las siguientes funciones servirán para encontrar el índice de un elemento dentro de la lista

```
gint
g_slist_position
(GSList *list,
GSList *llink);
```

Descripción: Encuentra la posición de un nodo dentro de una lista enlazada simple.

Parámetros:

- **list**: Una lista enlazada simple.
- **llink**: un elemento/nodo dentro de la lista enlazada simple.

Valor de retorno: El índice del nodo dentro de la lista ó -1 si no se encuentra nada.

```
gint
g_slist_index
(GSList *list,
gconstpointer data);
```

Descripción: Encuentra la posición del elemento que contiene los datos especificados.

Parámetros:

- **list**: Una lista enlazada simple.
- **data**: los datos que se buscan.

Valor de retorno: El índice del elemento que contiene los datos ó -1 si no se encuentra nada. Si deseamos recorrer, iterar o caminar a lo largo de la lista debemos usar la siguiente función.

```
void
g_slist_foreach
(GSList *list,
GFunc func,
gpointer user_data);
```

Descripción: Recorre toda la lista enlazada simple ejecutando una función para cada nodo de la lista.

Parámetros:

- **list**: Una lista enlazada simple.
- **func**: La función que se llamará con cada elemento. Esta función debe tomar dos punteros de tipo gpointer. El primero corresponde a los datos del elemento iterado, el segundo a los datos extras proporcionados por el programador.

2.5.4 **user_data**: datos extras proporcionados por el programador.

3.5.4 Destructor de clase

Cuando se termine el uso de la lista enlazada simple se debe de limpiar la memoria que este usando. El destructor de GSList libera la memoria de la estructura de la lista, mas no libera la memoria que esta a la que hace referencia cada elemento de la lista. Visto de otra forma. Una lista enlazada simple es una estructura que contiene espacio para dos punteros: uno apunta al siguiente elemento, el otro apunta a cualquier tipo o estructura de datos. Cuando se libera la memoria de la lista enlazada se libera el espacio que ocupan los dos punteros de cada elemento de la lista, pero los datos y estructuras a los que hacían referencia cada elemento de la lista quedan intactos. Ahora que se ha discutido los detalles del destructor, vemos al reseña. void

```
g_slist_free
(GSList *list);
```

Descripción: Libera toda la memoria ocupada por la estructura de una lista enlazada.

Parámetros:

list: Una lista enlazada simple.

462.5.5

3.5.5 Ciclo de vida de una lista enlazada simple

Comencemos la descripción del ciclo de vida de una lista enlazada simple.

- El primer paso es declarar la estructura e inicializarla con valor NULL.

```
#include <glib.h>
GSList *lista=NULL;
/* ... */
```

- Ahora podemos manipular la lista a nuestro antojo. Podemos, por ejemplo, añadir una sola cadena al final...

```
lista = g_slist_append (lista,"Elemento 1");
```

...al principio ...

```
list = g_slist_prepend(lista,"Elemento 0");
```

... o insertar elementos en posiciones arbitrarias ...

```
list = g_slist_insert (lista, "Elemento insertado",1);
```

... y no solamente funciona con cadenas, si no también con otros tipos de objetos...

```
lista = g_slist_append (lista,G_INT_TO_POINTER(113));
lista = g_slist_append (lista,objeto);
```

- Cuando llega el momento de recavar la información guardada en la lista tendremos que recordar la estructura en C vista arriba. El mismo puntero GSList que representa la lista enlazada, es a su vez el puntero al primer nodo de la lista. El elemento data del nodo es un puntero a los datos guardados y el elemento next apunta al siguiente nodo de la lista o es NULL si ya no hay más elementos. La manera correcta de acceder a los datos que contiene un nodo es mediante la notación de punteros:

```
datos= nodo->data;
siguiente = nodo->next;
```

Una manera útil de recorrer una lista enlazada simple es mediante un ciclo utilizando for...

```
for (nodo=lista; nodo; nodo=nodo->next)
g_print("%s\n",(char *)nodo->data);
```

Otra manera de caminar a lo largo de la lista es utilizar g_slist_for_each() el cual se apoya de una función definida por el usuario que debe de corresponder con el siguiente prototipo:

```
void
GFunc
(gpointer data, gpointer extra_data);
```

En el próximo ejemplo veremos cono se debe utilizar esta función. Una vez que se ha terminado de operar con la lista enlazada es necesario liberar la memoria

- usada, para ello se encuentra g_slist_free().

3.5.6 Ejemplo

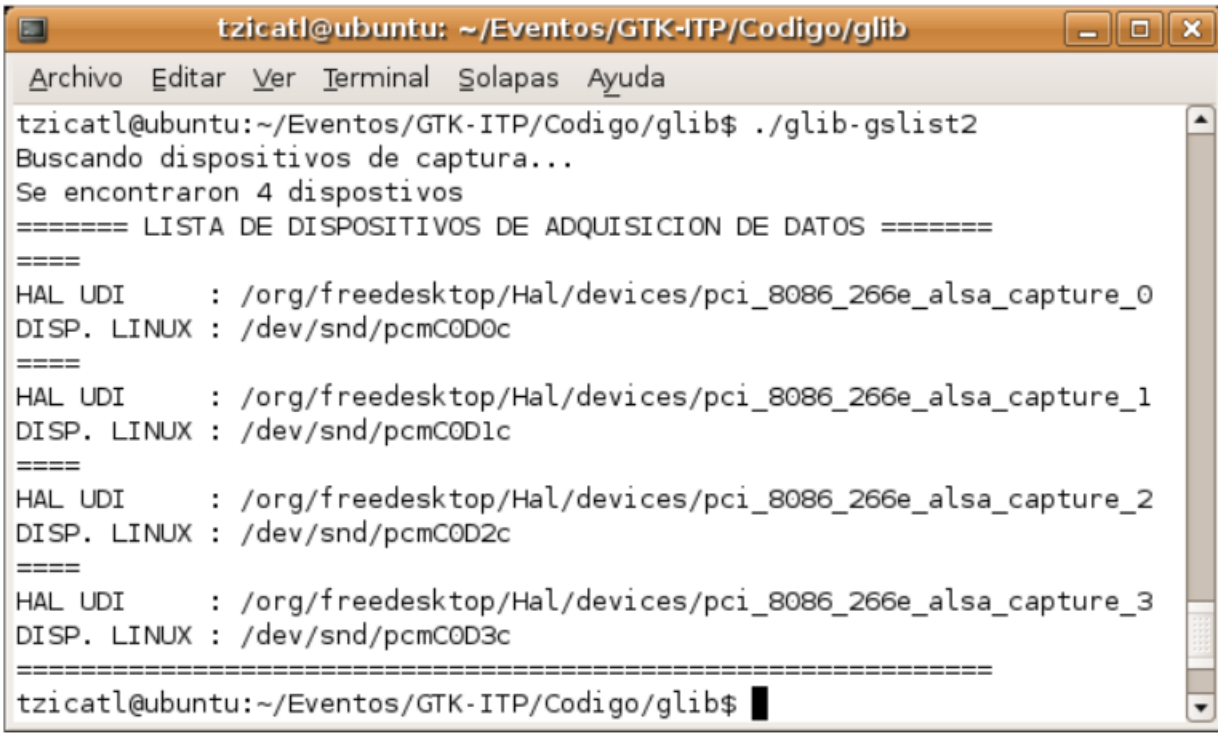
Mostraremos dos ejemplos. El primero de ellos mostrará de manera breve el ciclo de vida de GSList. Listado de Programa 2.5.1

```
/*****************************************************************************
*
Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo:
glib-gslist1.c
*
Descripcion:
Muestra de ciclo de vida de GSlist
* Comentarios:
Además muestra como caminar a traves de la
*
lista.
*
*
* TESIS PROFESIONAL
INSTITUTO TECNOLOGICO DE PUEBLA
*
INGENIERIA ELECTRONICA
* Autor: Noe Misael Nieto Arroyo
tzicatl@gmail.com
*
*****************************************************************************/
#include <glib.h>
void imprimir_lista(gpointer data, gpointer user_data){
gchar *mensaje;
mensaje = (gchar *) data;
g_print("%s\n", mensaje);
}
int main(){
GSList *lista = NULL;
GSList *nodo = NULL;
48gchar *nombre = "Nombre";
/*Inserción de diferentes tipos de elementos */
lista = g_slist_append(lista, nombre);
lista = g_slist_prepend(lista, "Elemento adicionado al principio");
lista = g_slist_insert(lista, "Elemento insertado en posicion 1", 1);
/* Primer metodo de acceso a elementos */
g_print("==-Primer metodo de acceso a los elementos de una lista-==\n");
for (nodo = lista; nodo; nodo = nodo->next)
g_print("%s\n", (char *) nodo->data);
/* segundo metodo */
g_print("==-Segundo metodo de acceso a los elementos de una lista-==\n");
g_slist_foreach(lista, (GFunc) imprimir_lista, NULL);
/*Destructor*/
g_slist_free(lista);
}
return 0;
```

En el ejemplo anterior se ha mostrado que dos métodos para recorrer toda la lista, elemento por elemento. El primero es un bucle de ejecución que itera sobre cada elemento hasta que se halle el elemento final de la lista.

El segundo método deja que `Glib` haga la caminata por la lista y llame una función designada por nosotros por cada elemento que encuentre.

Como se puede ver en la figura siguiente, los efectos de ambos métodos son iguales.



El segundo ejemplo es una aplicación práctica de las listas enlazadas simples. El objetivo de este ejemplo es realizar una lista de los dispositivos de captura de datos que existe en la computadora e imprimir una relación de estos. Listado de Programa 2.5.2

```
/*****************************************************************************
*
Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo:
glib-gslist2.c
*
Descripcion:
Aplicación práctica de GSlist
* Comentarios:
El siguiente ejemplo buscará todos los
*
dispositivos de sonido del sistema y los guardará
*
en una lista enlazada para su posterior
*
procesamiento
*
*
* TESIS PROFESIONAL
INSTITUTO TECNOLOGICO DE PUEBLA
*
INGENIERIA ELECTRONICA
* Autor: Noe Misael Nieto Arroyo
tzicatl@gmail.com
*
*****************************************************************************/
#include <glib.h>
```

(continué en la próxima página)

(proviene de la página anterior)

```
#include <glib/gprintf.h>
//void llenar_lista(GSList lista){
GSList *llenar_lista(GSList *lista){
gchar *comando = "/usr/bin/hal-find-by-property --key alsa.type --string
capture";
gchar *mi_stdout;
gchar **disps;
50gint i=0;
/*Ejecuta otro programa sin terminar este */
g_spawn_command_line_sync(comando, &mi_stdout,
NULL,NULL, NULL);
/*La salida del programa se guardó en mi_stdout.
Ahora procederemos a separar cada uno de los
resultados que vienen separados por caracteres
de nueva linea*/
disps = g_strsplit(mi_stdout,"\n",-1);
/*Despues de separados, cada uno se inserta en la lista*/
for (i=0;i< (g_strv_length(disps) -1); i++)
lista = g_slist_insert_sorted(lista,g_strdup(disps[i]),g_str_equal);
/*Liberar la memoria usada por los resultados separados*/
g_free(mi_stdout);
g_strfreev(disps);
}
return lista;
/*Esta función averiguará el dispositivo linux correspondiente a
cada dispositivo de adquisicion de datos*/
void imprimir_lista(gpointer data, gpointer user_data){
GString *comando;
gchar *mi_stdout;
/*Preparar el comando a ejecutar */
comando = g_string_new("");
g_string_printf( comando,
"/usr/bin/hal-get-property --udi %s --key linux.device_file",
(gchar *) data);
/*Ejecuta el comando programa sin terminar este */
g_spawn_command_line_sync(comando->str, &mi_stdout,
NULL,NULL, NULL);
/*Presentar los resultados*/
g_print("====\n");
g_print("HAL UDI
: %s\n", (gchar *) data);
g_print("DISP. LINUX : %s", mi_stdout);
}
/*Limpiar memoria */
g_string_free(comando,TRUE);
g_free(mi_stdout);
void limpiar_lista(gpointer data, gpointer user_data){
g_free(data);
}
int main(){
GSList *lista = NULL;
51g_print ("Buscando dispositivos de captura...\n");
lista = llenar_lista(lista);
g_print ("Se encontraron %i dispostivos\n",g_slist_length(lista));
g_print ("======= LISTA DE DISPOSITIVOS DE ADQUISICION DE DATOS =======\n");
g_slist_foreach(lista,imprimir_lista,NULL);
/*Es hora de liberar toda la memoria*/
```

(continué en la próxima página)

(proviene de la página anterior)

```
g_slist_foreach(lista,limpiar_lista,NULL);
g_slist_free(lista);
g_print ("============================================================\n");
return 0;
```

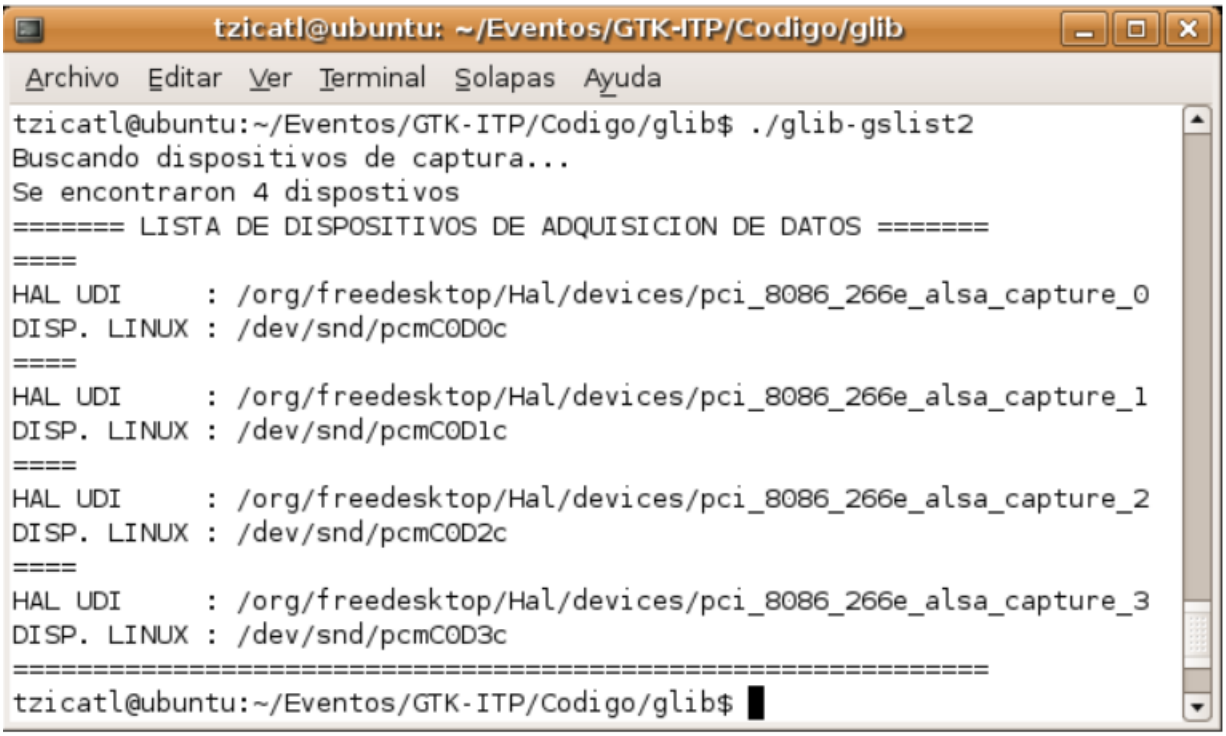
La tarea anteriormente expuesta parece difícil, pero los últimos mejoras del sistema operativo Linux hacen que nuestra tarea no sea titánica. FreeDesktop es un grupo de expertos en computación que se han reunido para establecer estándares de operación entre las diferentes versiones (distribuciones) de Linux.

Una de esas especificaciones es HAL (Hardware Abstraction Layer). Una serie de utilerías en línea de comandos permiten acceder a detalles del hardware de manera sencilla. La lógica detrás de este ejemplo es la siguiente: La función llenar_lista() usa HAL para listar a todos los dispositivos de sonido que sean de captura. Lo anterior implica la ejecución del programa hal-find-by-property, lo cual queda a cargo de la función g_spawn_command_line_sync() que ejecuta la linea de comandos, descrita en una cadena, y entrega la salida del comando en otra cadena (mi_stdout).

La salida del comandoes una lista de los dispositivos de captura de audio disponibles en el sistema y están separados por caracteres de nueva línea. Es necesario entonces dividirlos en cadenas independientes. La función g_strsplit() parte la cadena mi_stdout en un arreglo de cadenas, las cuales contienen ya, el identificador de cada dispositivo separado de todos los demás. La función g_strsplit() regresa una cadena extra vacía que podemos ignorar. Después de haber separado nuestros identificadores en cadenas de texto individuales se procede a llenar la lista enlazada simple con estos valores. Una vez preparada la lista enlazada, se libera la memoria que ya no sirve y se regresa el puntero de la nueva lista, ya llena. Llega la hora de presentar resultados. El numero de dispositivos encontrados es ahora reportado mediante g_slist_lenght(). Ya hemos visto anteriormente como caminar a través de todos los elementos de la lista; hacemos lo mismo mediante imprimir_lista() que además de imprimir los identificadores de los dispositivos, utiliza g_spawn_command_line_sync() para investigar el dispositivo Linux correspondiente a cada dispositivo.

Antes de poder liberar la memoria de la estructura de la lista enlazada simple, se debe recorrer y liberar la memoria de cada uno de los elementos de la lista en forma individual. Esto se hace fácilmente con la función limpiar_lista().

El producto de nuestro programa se muestra a continuación.



CAPÍTULO 4

Introducción

A continuación comenzaremos a estudiar la parte gráfica, de interacción con el usuario. Discutiremos el concepto genérico de widget y como utilizarlos para construir gran cantidad de objetos.

4.1 El concepto de widget

La palabra widget proviene de una contracción de la lengua inglesa: “Window Gadget6”, que se utiliza para referir a los diferentes elementos de una interfaz gráfica de usuario. Un widget de GTK+ es un componente de interfaz gráfica de usuario, es un objeto en toda la extensión de la palabra. Ejemplos de widgets son las ventanas, casillas de verificación, botones y campos editables. Los widgets (no importando de que tipo sean), siempre se definen como punteros a una estructura GtkWidget. Esta estructura es un tipo de dato genérico utilizado por todos los widgets y ventanas en GTK+.

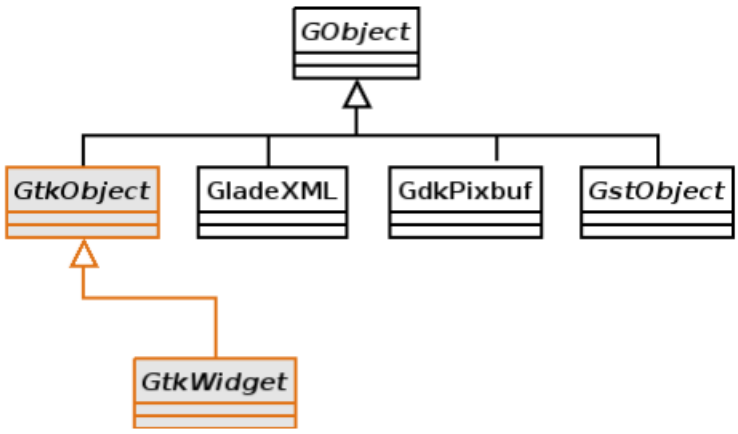
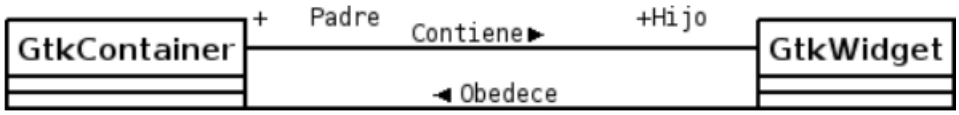


Figura 3.1.1: Diagrama UML de herencia de GObject

La librería GTK+ sigue un modelo de programación orientada a objetos(POO). La jerarquía de objetos comienza en GObject de la librería Glib del que hereda GtkObject. Todos los widgets heredan de la clase de objetos GtkWidget, que

a su vez hereda directamente de GtkObject. Un diagrama UML que muestra tales relaciones se muestra en la figura 3.1.1.

La clase GtkWidget contiene las propiedades comunes a todos los widgets; cada widget particular le añade sus propias propiedades. En GTK+ los widgets presentan una relación padre/hijo entre sí, las aplicaciones suelen tener un widget «ventana» de nivel superior que no tiene padre, pero aparte de él, todos los widgets que se usen en una aplicación deberán tener un widget padre. Al widget padre se le denomina contenedor



4.2 Nomenclatura GTK

Cualquier proyecto de software, por pequeño que sea, requiere una gran inversión en tiempo y trabajo. Considerando que el software Libre como GTK+ o GNOME son proyectos enormes que no pueden ser manejados por un grupo reducido de personas, ofrecer una nomenclatura consistente es de vital importancia para poder coordinar los esfuerzos de miles de programadores alrededor del mundo.

Antes de continuar precisa discutir adecuadamente la nomenclatura de objetos, métodos y, en general, el estilo de programación de GTK+, Glib, Gdk, GdkPixBuf, Gobject, etc.. Cubriendo estos lineamientos tendremos el beneficio de mejorar nuestros hábitos de programación, escribiendo código que puede ser entendido por cualquier otra persona.

Seremos capaces de acostumbrarnos rápidamente a gran cantidad de librerías que utilizan este estilo de programación como el entorno de programación de GNOME. Como extracto del documento "GNOME Programming Guidelines"[7] resumimos la nomenclatura de GTK+.

- Siempre que sea posible, en GTK+ siempre se debe evitar el uso de variables globales. Esto es importante aún más para librerías, ya que las variables globales dentro de sendos módulos se exportan junto a las funciones (métodos si consideramos a una librería como un objeto). Todos estos **símbolos** se integran al espacio global de nombres de la aplicación que llama a la librería(Global Namespace). Un variable global de una librería cuyo nombre haya sido asignado de manera descuidada puede causar conflictos con otra variable con el mismo nombre que se esté usando en una parte del código que utiliza la librería.
- Los nombres de las funciones deberán tener la forma modulo_submodulo_operacion(). Por ejemplo: gtk_window_new(), g_string_truncate() ó g_tree_destroy(). Esta simple nomenclatura evita el choque de **símbolos** o nombres entre diferentes módulos de las librerías.
- Los **símbolos** (funciones y variables), deberán tener nombres descriptivos: en lugar de usar cntusr() deberemos usar contar_usuarios_activos(). Esto hace que el código fuente sea fácil de usar y casi auto-documentado.
- Los nombres de funciones deben estar en minúsculas y deberán usar guión bajo para separar palabras, por ejemplo: g_string_destroy(), gtk_window_new(), abrir_puerto().
- Las macros y enumeraciones deben escribirse en letras mayúsculas, utilizando guiones bajos para separar palabras, por ejemplo: TRUE o G_DIR_SEPARATOR .
- Tipos de datos, objetos y nombres de estructuras son una mezcla entre mayúsculas y minúsculas, por ejemplo: Gslist, GtkWidget.
- El uso de guión bajo para separar palabras hace que el código fuente luzca menos apretado y fácil de editar ya que se pueden utilizar de mejor manera los comandos de navegación por palabras de los editores de texto de forma que naveguemos más rápidamente.
- A la hora de escribir una librería se hace común compartir **símbolos** (nombres de variables o funciones), entre los diversos componentes de la librería pero no se desea que estos símbolos estén disponibles para loss usuarios

de la librería. En tal caso, se puede anteponer un guión bajo al nombre de la función o variable mientras éste sigue la nomenclatura modulo/submódulo descrita arriba. Por ejemplo: _modulo_objeto_algunmetodo().

4.3 Proceso de creación de un widget

El proceso de creación de un **widget** consta de cuatro pasos:

(1) Creación de la instancia de clase del **widget** que deseamos utilizar.

(2) Configuración de esta instancia (tamaño, clase, relación con **widgets** padres, etc..)

(3) Conexión de señales y eventos.

(4) Visualización de la instancia.

De acuerdo a la nomenclatura de la sección anterior, si la clase de un *widget* es GtkClase, su constructor de clase y todos los métodos asociados a esta tendrán la siguiente nomenclatura: gtk_clase_metodo:

- «clase» debe sustituirse por el nombre del **widget** que se desea crear.
- «metodo» describe la acción que ejecutará la instancia de la clase. Por ejemplo, el constructor de clase GtkWindow tiene la siguiente nomenclatura: gtk_window_new().

La función de creación de un **widget** gtk_clase_new() siempre debe devolver un puntero (en lenguaje C) a una instancia de tipo GtkWidget y no un puntero a una instancia del tipo creado. Por ejemplo, la función gtk_window_new() devuelve un puntero a un objeto de GtkWidget y no una instancia de tipo GtkWindow.

Es importante remarcar esta característica primordial de GTK+, ya que si recordamos que el lenguaje C no es un lenguaje orientado a objetos, nosotros deberemos hacernos cargo del correcto moldeo de tipos de clase.

La preferencia de los constructores de clase de regresar la referencia a un tipo de dato de la clase base (GtkWidget) en lugar de regresar como un puntero a la clase heredada (GtkWindow) se justifica gracias a que muchos métodos de la clase base aún aplican a la clase heredada. El mejor ejemplo lo encontramos a la hora de hacer visible la instancia del objeto de tipo GtkWindow, para ello se utiliza el método gtk_widget_show(). Si en algún momento se necesitase un puntero del tipo de la clase heredada podemos hacer uso de las macros que define cada objeto de GTK+ y que nos ayudan a moldear nuestro puntero a la clase de conveniencia.

Con nuestro ejemplo, si necesitásemos un puntero del tipo GtkWindow utilizando como base al puntero de tipo GtkWidget, recurriríamos a la macro GTK_WINDOW. Ahora un mismo objeto se puede comportar de dos formas distintas. Esto es conocido en cualquier lenguaje que soporte programación orientada a objetos como **polimorfismo**.

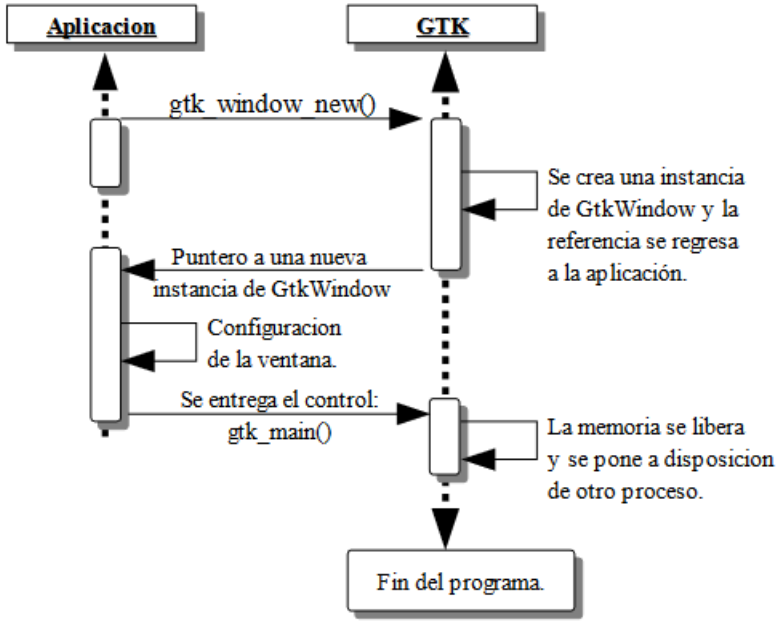
Un ejemplo no compilable, pero ilustrativo, se redacta a continuación. (Listado de programa 3.3.1)

```
/* Crear una ventana con GTK+ */
/*Primero debemos incluir la librería gtk*/
#include <gtk.h>
main(){
GtkWidget *ventana;
...
/*Crear la instancia de clase GtkWindow*/
ventana = gtk_window_new(...);
60
/*Cambiar el tamaño de la ventana, por ejemplo, para ocupar
    toda la   pantalla del monitor
    Note que se utiliza la macro GTK_WINDOW que hace que el
    objeto   ventana se comporte como GtkWindow.
*/
gtk_window_set_full_screen(GTK_WINDOW(ventana));
/*A continuación hacer visible el objeto gráfico
```

(continué en la próxima página)

(proviene de la página anterior)

```
    utilizando herencia y polimorfismo.
    Note que el objeto ventana ahora se comporta como GtkWidget.
*/
gtk_widget_show(ventana);
/*Otorgar control completo a la librería GTK+*/
gtk_main();
....
}
```



Es importante hacer notar que, en este ejemplo en específico, el objeto ventana es de tipo GtkWindow, pero a la vez es del tipo GtkWidget. Como el tipo base del puntero ventana es GtkWidget, es necesario moldearlo al tipo GtkWindow para que pueda comportarse como éste tipo de objeto. Si no se hace esto, el compilador se quejará y la aplicación terminará con una violación de segmento.

El interfaz gráfico de una aplicación se construye combinando diferentes widgets (ventanas, cuadros combinados, cuadros de texto, botones, . . .) y se establecen diversas retrollamadas (callbacks),

(Figura 3.3.1: Ciclo de vida)

Es importante hacer notar que, en este ejemplo en específico, el objeto ventana es de tipo GtkWindow, pero a la vez es del tipo GtkWidget. Como el tipo base del puntero ventana es GtkWidget, es necesario moldearlo al tipo GtkWindow para que pueda comportarse como éste tipo de objeto. Si no se hace esto, el compilador se quejará y la aplicación terminará con una violación de segmento.

El interfaz gráfico de una aplicación se construye combinando diferentes *widgets* (ventanas, cuadros combinados, cuadros de texto, botones, . . .) y se establecen diversas retrollamadas (*callbacks*) eventos asíncronos; de esta forma se obtiene la lógica requerida por el programa a medida que se producen ciertas señales que a su vez provocan las *retrollamadas*.

Las señales se producen por diversos sucesos como oprimir el botón de un ratón que se encuentra sobre un *widget* botón, pasar el cursor por encima de un *widget* u oprimir una tecla

4.4 Teoría de señales y retrollamadas

GTK+ es una librería dirigida por eventos. Desde el punto de vista del programador esto significa que se quedará en el bucle principal de ejecución (gtk_main()), hasta que algún evento o señal ocurra y el control se pase a la función apropiada.

Las señales son el medio por el cual GTK+ informa a las aplicaciones de los acontecimientos producidos en el interfaz gráfico o dentro de los objetos que componen el programa.

Las señales son importantes dentro de las aplicaciones con interfaz gráfica de usuario ya que el programa debe responder a las acciones que el usuario ejecute que por naturaleza son asíncronas y no se pueden predecir o prever.

Si el usuario mueve el ratón, presiona un botón, escribe un texto o cierra una ventana, una función retrollamada se ejecuta y se realiza el cómputo requerido por el usuario, por ejemplo: guardar un archivo.

Un procesador de textos puede tener un botón que haga que el bloque seleccionado de texto adquiera los atributos de letra negrita. La **retrollamada** asignada a ese botón contiene el código que se encargará de llevar a cabo esa tarea.

De alguna forma, antes de cerrar una aplicación se hace necesario llamar a rutinas de limpieza, guardar el trabajo del usuario o simplemente desplegar un diálogo que pregunte si realmente desea cerrar la ventana.

En una aplicación, como veremos más tarde, continuamente se están generando señales y eventos, sin embargo no todos son atendidos y sólo conectamos **retrollamadas** para aquellos eventos o señales que son de nuestro interés.

Cuando deseamos atender a la escucha de una señal o **retrollamada**, se asocia un *widget* y una función en C. Así, también se puede asociar **retrollamadas** a más de un *widget* ahorrando código que deba escribirse.

4.5 Rutinas de tratamiento de señales y eventos

En GTK+ señales y eventos se administran casi de la misma manera, la distinción entre estos dos grupos se debe a que las señales son provocadas por el sistema de objetos de Glib / GTK+ y los eventos son una corriente de mensajes que llegan desde el subsistema gráfico. Desde una perspectiva del programador resulta sencillo pensar en los eventos como cualquier señal causada por la interacción del usuario con el programa.

Dos de las señales básicas en GTK+ son delete_event y destroy. El evento delete_event generalmente se envía a una ventana cuando el usuario trata de cerrarla. Por su parte, la señal destroy se manda a un objeto cuando su método de destrucción debe ser invocado. Una ventana de nivel superior siempre debe conectar una función **retrollamada** al evento delete_event. Si el usuario quiere cerrar la ventana, entonces la aplicación deberá terminar correctamente.

Una retrollamada es una función en C como cualquier otra. Sin embargo, dependiendo de la señal o evento a escuchar es como se declarará el tipo dato de regreso y los parámetros. Una vez escrita adecuadamente, se registra esta rutina ante GTK+ usando la macro g_signal_connect().

```
#define g_signal_connect(instance, detailed_signal, c_handler, data)
```

Descripción: Conecta una función retrollamada que atenderá una señal de un objeto en particular.

Parámetros:

- **instance** : Es la referencia al *widget* u objeto del que queremos escuchar señales y eventos. Este puntero debe estar moldeado al tipo GObject ya que GtkWidget está es un derivado de éste. Para esto deberemos usar la macro G_OBJECT().
- **detailed_signal** : Es una cadena que especifica la señal o evento a escuchar.
- **c_handler** : El puntero de la función *retrollamada*. Este puntero debe estar moldeado mediante la macro G_CALLBACK() al tipo de puntero GCallback. El prototipo de cada función *retrollamada* se determina por

el contexto en el que será usada; visto de otra manera: el prototipo de cada función se determina por el tipo de señal a la que será conectada.

- **data** : Este último argumento permite adjuntar algún dato extra a la *retrollamada*, de tal manera que se evite el uso de variables globales y en su lugar se pasen estructuras o valores directamente a la función *retrollamada* cuando ésta sea invocada.

La función retrollamada cambia dependiendo de la señal que se desea escuchar, pero hay una función retrollamada prototipo que se usa como base para todas las demás:

```
void (*Gcallback) (void);
```

Lo anterior no significa que todas las funciones retrollamadas no deban tomar parámetros y regresar void.

Una función **retrollamada** muy común en GTK+ y puede tener el siguiente prototipo:

```
void funcion_retrollamada ( Gtkwidget *widget,
gpointer datos);
```

El primer argumento es un puntero al widget que recibe el evento o genera la señal. El segundo argumento es un puntero a los datos extras que se mandaron cuando se conectó la señal a la retro llamada. De nuevo hay que hacer notar que el perfil de retro llamada descrito arriba es sólo una forma general. Hay algunas retrollamadas generadas por widgets especiales que requieren diferentes parámetros.

4.6 Eventos

En complemento al mecanismo de señales descrito arriba, hay un conjunto de eventos que reflejan el mecanismo de eventos del subsistema gráfico del sistema operativo (En UNIX será X-window). Las funciones retrollamada también se pueden conectar a estos. Son

- event
- button_press_event
- button_release_event
- scroll_event
- motion_notify_event
- delete_event
- destroy_event
- expose_event
- key_press_event
- key_release_event
- enter_notify_event
- leave_notify_event
- configure_event
- focus_in_event
- focus_out_event
- map_event
- unmap_event

- property_notify_event
- selection_clear_event
- selection_request_event
- selection_notify_event
- proximity_in_event
- proximity_ouLevent
- visibility_notify_event
- client_event
- no_expose_event
- window_state_event

Para poder conectar una función retro llamada a alguno de estos eventos, se usará la función g_signal_connect() , tal y como se ha descrito arriba usando alguno de los nombres que se dan como el parámetro señal. La función retro llamada para eventos es un poco diferente a la que se usa para las señales:

```
gint funcion_retrollamada( GtkWidget *widget,
GdkEvent *event,
gpointer datos_extra );
```

En C, GdkEvent es una unión, de la cual su tipo dependerá de cual de los eventos mostrados arriba se han producido y esta construido mediante diferentes máscaras de eventos.. Para poder decirnos que tipo de evento ha ocurrido, cada una de las posibles alternativas tiene un miembro type que muestra que evento ocurrió. Los otros elementos de la estructura dependerán de que tipo de evento se generó. Las máscaras de los tipos posibles de eventos son:

```
GDK_NOTHING
GDK_DELETE
GDK_DESTROY
GDK_EXPOSE
GDK_MOTION_NOTIFY
GDK_BUTTON_PRESS
GDK_2BUTTON_PRESS
GDK_3BUTTON_PRESS
GDK_BUTTON_RELEASE
GDK_KEY_PRESS
GDK_KEY_RELEASE
GDK_ENTER_NOTIFY
GDK_LEAVE_NOTIFY
GDK_FOCUS_CHANGE
GDK_CONFIGURE
GDK_MAP
GDK_UNMAP
GDK_PROPERTY_NOTIFY
GDK_SELECION_REQUEST
GDK_SELECTION_NOTIFY
GDK_PROXIMITY_IN
GDK_PROXIMITY_OUT
GDK_DRAG_ENTER
GDK_DRAG_LEAVE
GDK_DRAG_MOTION
GDK_DRAG_STATUS
GDK_DROP_START
GDK_DROP_FINISHED
```

(continué en la próxima página)

(proviene de la página anterior)

```
GDK_CLIENTE_EVENT
GDK_VISIBILITY_NOTIFY
GDK_NO_EXPOSE
GDK_SCROLL
GDK_WINDOW_STATE
GDK_SETTING
```

En resumen: para conectar una retro llamada a uno de esos eventos, usaremos algo como lo que se presenta:

```
g_signal_connect ( G_OBJECT (button),
"button_press_event",
G_CALLBACK (button_press_callback),
NULL);
```

Si asumimos que button es un *widget*. Cuando el ratón esté sobre el botón y el botón sea presionado, se llamará a la función button_press_callback(), la cual puede ser declarada como sigue:

```
static gint button_press_callback( GtkWidget *widget,
GdkEventButton *event,
gpointer data );
```

Es preciso hacer notar que el segundo argumento lo podemos declarar como tipo GdkEventButton por que ya sabemos cuál es el evento que ocurrirá para que esta función sea invocada. El valor regresado por esta función indica si el evento se deberá propagar más allá por el mecanismo de manejo de señales de GTK+. Regresar FALSE indica que el evento ya ha sido tratado correctamente y ya no se debe propagar.

4.7 Bucle de ejecución y eventos

El bucle de eventos de GTK+ es el responsable de que el sistema de señales funcione correctamente, ya que el primero no es más que un bucle interno de GTK+, en el que se van, una y otra vez, comprobando os estados de cada uno de los elementos de la aplicación, e informando de dichos cambios a los elementos que se hayan registrado para ser informados. Este bucle de eventos GTK+ se traduce básicamente en dos funciones, que son gtk_main() y gtk_main_quit().

gtk_main() entrega el control de cualquier programa al bucle de eventos de GTK+. Esto significa que, una vez que se haya realizado la llamada a gtk_main(), se cede todo el control de la aplicación a GTK+. Aunque gtk_main() toma el control de la aplicación, es posible ejecutar otras porciones de código aprovechando el sistema se señales usando algún manejador (instalado ANTES de llamar a gtk_main()) Dentro de algún manejador o *retrollamada* se puede llamar a gtk_main_quit() que termina el bucle de eventos de GTK+. El pseudo-código de una típica aplicación GTK+ seria:

```
int main (int argc, char *argv[])
{
gtk_init (&argc, &argv);
/* creación del interfaz principal */
/* conexión a las distintas señales */
gtk_main ();
return 0;
}
```

Como puede comprobarse, el programa inicializa GTK+, crea el interfaz básico, conecta funciones a las distintas señales en las que esté interesado (llamadas a g_signal_connect()), para seguidamente entregar el control del programa a GTK+ mediante gtk_main(). Cuando en algún manejador de señal realicemos una llamada a gtk_main_quit(), gtk_main() retornará, tras lo cual la aplicación termina.

4.8 Ejemplo

A continuación se mostrará un sencillo ejemplo mostrando el proceso de creación del widget más sencillo (GtkWindow) y el uso de señales.

Comencemos recordando el capítulo 3.3. El primer *widget* que aprenderemos a usar es GtkWindow que es ventana común y corriente

(Listado de programa 3.4.1)

```
/*************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      bucle1.c
* Descripcion:            Crea una ventana.
* Widgets usados:         GtkWindow
*  Comentarios:
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*************************************************************************/
#include <gtk/gtk.h>
int main( int   argc, char *argv[] )
{
GtkWidget *window;


/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
/*Crea una nueva instancia de GtkWindow*/
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
/*Configura la instancia de GtkWindow*/
gtk_window_set_title (GTK_WINDOW (window), "bucle1.c");
gtk_widget_set_size_request(window,200,100);
/*Conectar señales.
Cuando la señal "destroy" se emita, se llamará a la
función gtk_main_quit() que termina el programa
*/
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),
    NULL);
/*Muestra la ventana en la pantalla*/

gtk_widget_show (window);
/*Cede el control de la apliación a GTK+*/
gtk_main ();

return 0;
}
```

El primer paso es inicializar la librería GTK+ con esta instrucción:

```
gtk_init (&argc, &argv);
```

De no incluirla, nuestros programas fallarían de manera inmediata. El siguiente paso es crear una instancia de una ventana y alojar la referencia al objeto en la variable window:

```
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

El constructor de clase de GtkWindow toma un parámetro, es el tipo de ventana que se desea crear. Las ventanas normales, como la ventana del navegador (Firefox ó Mozilla) o el administrador de archivos (Nautilus) son ventanas de nivel superior (GTK_WINDOW_TOPLEVEL). El siguiente paso en nuestra aplicación es establecer el título . . .

```
gtk_window_set_title (GTK_WINDOW (window), "bucle1.c");
```

. . . y el tamaño:

```
gtk_widget_set_size_request(window,200,100);
```

Observe que el método utilizado para cambiar el tamaño de la ventana es un método de GtkWidget y no de GtkWindow.

Observe también que al establecer el título de la ventana se utilizó una especie de macro con el puntero window como parámetro. ¿Por qué ocurre esto? El constructor de GtkWindow regresa la instancia de GtkWindow como un puntero de GtkWidget y no de GtkWindow. Esto es necesario para que se pueda utilizar el polimorfismo en el lenguaje C. Usando punteros al objeto más general como GtkWidget nos permite moldearlo a cualquier otro objeto derivado.

El método gtk_window_set_title() requiere que el primer parámetro sea un puntero de tipo GtkWindow; la macro GTK_WINDOW() moldea el puntero de tipo GtkWidget a puntero GtkWindow.

El método gtk_widget_set_size_request() requiere que el primer parámetro sea un puntero de tipo GtkWindow; en el caso citado anteriormente no es necesario moldear el puntero window pues ya es del tipo deseado.

¿Que ocurriría si decido no usar las macros de moldaje de tipos? El compilador se quejaría de punteros de tipos incompatibles.

A continuación viene la instrucción más importante del programa:

```
g_signal_connect (G_OBJECT (window), "destroy",
G_CALLBACK (gtk_main_quit),
NULL);
```

El prototipo de la macro g_signal_connect() es ya conocida desde el capítulo 3.4.1. El objeto window conectará la señal «destroy» a la función gtk_main_quit(). La señal «destroy» se emite cuando la ventana es cerrada.

Cuando el usuario cierre la ventana también ocasionará que el bucle de control de Gtk+ termine y con ello la aplicación.

¿Qué ocurriría si no conectáramos esta señal? Al cerrar la ventana, esta desaparecería pero el programa seguiría ejecutándose en memoria.

Por último hacemos visible la ventana y entregamos el control de la aplicación al bucle de GTK+.

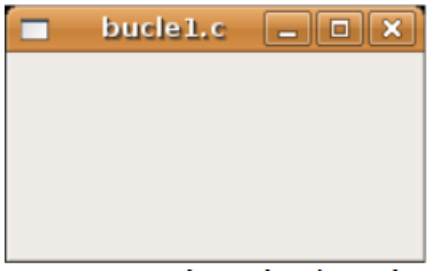
Los frutos de nuestro programa se muestran en la Figura 3.4.1.

4.9 Widgets contenedores

Uno de los conceptos fundamentales de Gtk+ son los contenedores. Un widget contenedor es aquel que es capaz de contener a otros **widgets**. Existen una gran cantidad de contenedores y GTK+ los utiliza para acomodar los **widgets** dentro de la interfaz gráfica de usuario..

Cuando se escribe una aplicación, normalmente se necesita colocar mas de un widget dentro de una ventana. En el ejemplo anterior(listado de programa 3.4.1) no necesitamos de ningún otro widget más que la ventana.

El ejemplo anterior no ofrece utilidad más allá de la didáctica, pero como no conocemos aún ningún otro widget lo tomaremos como base para extender nuestra aplicación. El diagrama de herencia de clase de GtkWindow es el siguiente.



Como podemos ver en la Figura 3.5.1 GtkWindow también puede contener otros **widgets**, pues desciende de la clase GtkContainer. Pero debido a su descendencia directo con la clase GtkBin sólo puede contener un sólo *widget*, eso significa que, a pesar de tener la capacidad de almacenar otros **widgets** por ser descendiente de GtkContainer, la clase GtkWindow sólo puede contener un sólo widget debido a su parentesco inmediato con GtkBin. Al igual que GtkWidget, GtkContainer y GtkBin son clases abstractas. Eso quiere decir que no son instanciables y sólo sirven de plantillas para otros **widgets**.

La clase GtkBin es muy simple y sólo contiene un método que se utiliza de manera errática. Usaremos, entonces, las siguientes líneas a comentar los métodos más importantes de la clase GtkContainer.

4.10 Métodos de la clase GtkContainer

```
void gtk_container_add (GtkContainer *container,
GtkWidget *widget);
```

Descripción: Inserta un *widget* dentro de un contenedor. No es posible añadir el mismo widget a múltiples contenedores.

Parámetros:

- **container** : Una instancia de un contenedor. Use la macro GTK_CONTAINER() para moldear un puntero de diferente tipo.
- widget: El widget que se quiere insertar en el contenedor.

```
void gtk_container_remove (GtkContainer *container,
GtkWidget *widget);
```

Descripción: Remueve un *widget* que ya esta adentro de un contenedor.

Parámetros:

- **container** : Una instancia de un contenedor. Use la macro GTK_CONTAINER() para moldear un puntero de diferente tipo.

- widget: El widget que se quiere remover del contenedor.

```
void gtk_container_set_border_width (GtkContainer *container,
guint border_width);
```

Descripción: Establece el ancho de borde de un contenedor.

Parámetros:

- **container** : Una instancia de un contenedor. Use la macro GTK_CONTAINER() para moldear un puntero de diferente tipo.
- **border_width** : El espacio libre que se desea dejar alrededor del contenedor. Los valores válidos van de 0 a 65535.

```
guint gtk_container_get_border_width (GtkContainer *container);
```

Descripción: Obtiene el valor actual del ancho de borde del contenedor

Parámetros:

- **container** : Una instancia de un contenedor. Use la macro GTK_CONTAINER() para moldear un puntero de diferente tipo.

Valor de retorno: El ancho de borde del contenedor.

Hasta ahora hemos visto (al menos en teoría), que es posible insertar un **widget** dentro de otro, para ello usamos el método gtk_container_add(). Pero, ¿Qué pasa si se quiere usar mas de un *widget* dentro de una ventana?, ¿Cómo se puede controlar la posición de los ***widgets***?

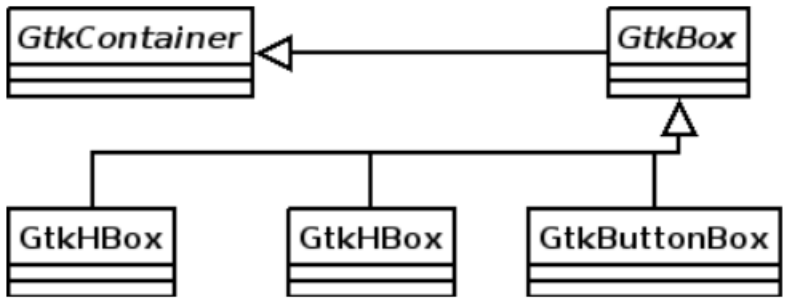
CAPÍTULO 5

Cajas

5.1 Descripción

Regresemos un poco a la realidad cotidiana: Si deseamos acomodar algún objeto como un anillo, conseguimos un recipiente adecuado que sólo aloje nuestra alhaja.

Por otra parte, si nosotros fabricáramos teléfonos y tuviéramos que enviar varios de ellos a un cliente en otro país, la acción más común sería acomodar y empacar todos ellos en una caja y enviarlos a nuestro comprador. En el mundo de GTK+ se hace la misma analogía. Una caja es un *widget* que organiza un grupo de objetos en un área rectangular: Si deseamos colocar varios de ellos en una sola ventana usaremos una caja y esta se puede insertar, a su vez, en la ventana. La ventaja principal de usar cajas es el despreocuparnos del lugar donde deben dibujarse cada uno de nuestros objetos gráficos, GTK+ toma esa responsabilidad por nosotros. Existen dos tipos de cajas: GtkHBox y GtkVBox. Ambos descienden de la clase abstracta GtkBox (Figura 2.5.2) y son invisibles.



Cuando se empaquetan **widgets** en una caja horizontal (GtkHBox) se acomodan horizontalmente de izquierda a derecha o viceversa y todos tienen la misma altura.

En una caja vertical (GtkVBox) se acomodan de arriba a abajo o viceversa y todos tienen el mismo ancho. También se puede usar una combinación de cajas dentro o al lado de otras cajas para crear el efecto deseado. GtkBox es una clase abstracta, y las clases derivadas (GtkHBox y GtkVBox) no contienen métodos de clase. Los constructores de clase son solamente para las cajas verticales u horizontales mientras que los métodos de clase son de GtkBox.

5.2 Constructor de clase

```
tkWidget* gtk_hbox_new (gboolean homogeneous,
gint spacing);
```

Descripción: Crea una nueva instancia de una caja horizontal.

Parámetros:

- **homogeneous** : Especifique TRUE si desea que todos los **widgets** (hijos) que se inserten en la caja les sea asignado un espacio por igual.
- **spacing** : El número de *pixeles* que se insertarán entre los **widgets** hijos.

Valor de retorno: una nueva instancia de GtkHBox.

```
GtkWidget* gtk_vbox_new (gboolean homogeneous,
gint spacing);
```

Descripción: Crea una nueva instancia de una caja vertical.

Parámetros:

- **homogeneous** : Especifique TRUE si desea que todos los **widgets** (hijos) que se inserten en la caja les sea asignado un espacio por igual.
- **spacing** : El número de **pixeles** que se insertarán entre los **widgets** hijos.

Valor de retorno: una nueva instancia de GtkVBox.

5.3 Métodos de clase básicos

El siguiente par de métodos permiten acomodar widgets en cualquier tipo de caja.

```
void gtk_box_pack_start_defaults (GtkBox *box,
GtkWidget *widget);
```

Descripción : Acomoda un **widget** en una caja. Los **widget** hijos se irán acomodando de arriba a abajo en una caja vertical, mientras que serán acomodados de izquierda a derecha en una caja horizontal.

Parámetros:

- **box** : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado.
- **widget** : El **widget** que será empacado.

```
void gtk_box_pack_end_defaults (GtkBox *box,
GtkWidget *widget);
```

Descripción: Acomoda un widget en una caja. Los widgets hijos se irán acomodando de abajo a arriba en una caja vertical, mientras que serán acomodados de derecha a izquierda en una caja horizontal

Parámetros:

- **box** : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado.
- **widget** : El **widget** que será empacado.

5.4 Métodos de clase avanzados

La siguiente colección de métodos exhibe toda la flexibilidad del sistema de empaquetado de GTK+.

Las dos principales funciones gtk_box_pack_start() y gtk_box_pack_end() son complejas, es por eso que se les ha aislado de las demás para una discusión más detallada.

Cinco son los parámetros que gobiernan el comportamiento de cada *widget* hijo que se acomoda en una caja:

- homogeneus y spacing que se determinan en el constructor de clase.
- expand, fill y padding que se determinan cada vez que se empaca un widget en un contenedor.

El parámetro homogeneous controla la cantidad espacio individual asignado a cada uno de los **widgets** que se empacan en una caja. Si es TRUE entonces el espacio asignado será igual para todos los **widgets** hijos. Si es FALSE entonces cada widget hijo podrá tener un espacio asignado diferente. El parámetro spacing especifica el número de pixeles que se usarán para separar a los widgets hijos.

El parámetro expand le permite al **widget** hijo usar espacio extra. El espacio extra de toda una tabla se divide equitativamente entre todos sus hijos.

El parámetro fill permite al **widget** hijo ocupar todo el espacio que le corresponde, permitiendo llenar por completo el espacio asignado. El **widget** no tiene permitido ocupar todo el espacio si el parámetro expand es FALSE. Los **widgets** hijos siempre están usando todo el espacio vertical cuando están acomodados en una caja horizontal. Asimismo usarán todo el espacio horizontal si están situados en una caja vertical.

El parámetro padding permite establecer un espacio vacío entre el **widget** hijo y sus vecinos. Este espacio se añade al establecido por spacing.

```
void gtk_box_pack_start (GtkBox *box,
GtkWidget *child,
gboolean expand,
gboolean fill,
guint padding);
```

Descripción: Acomoda un *widget* en una caja. Los *widgets* hijos se irán acomodando de arriba a abajo en una caja vertical, mientras que serán acomodados de izquierda a derecha en una caja horizontal.

Parámetros: * **box** : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado. * **child** : El widget que será empacado. * **expand** : Si es TRUE al widget hijo podrá asignársele espacio extra. * **fill** : Si es TRUE el widget podrá ocupar el espacio extra que se le asigne. * **padding** : El perímetro de espacio vació del hijo, especificado en pixeles.

```
void gtk_box_pack_end (GtkBox *box,
GtkWidget *child,
gboolean expand,
gboolean fill,
guint padding);
```

Descripción: Acomoda un *widget* en una caja. Los *widgets* hijos se irán acomodando de abajo a arriba en una caja vertical, mientras que serán acomodados de derecha a izquierda en una caja horizontal.

Parámetros:

- **box** : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado.
- **child** : El widget que será empacado.
- **expand** : Si es TRUE al widget hijo podrá asignársele espacio extra.
- **fill** : Si es TRUE el widget podrá ocupar el espacio extra que se le asigne.

- **padding** : El perímetro de espacio vació del hijo, especificado en pixeles.

```
void gtk_box_set_homogeneous (GtkBox *box,
gboolean homogeneous);
```

Descripción: Establece la propiedad «homogeneous» que define cuando los *widgets* hijos deben de tener el mismo tamaño.

Parámetros: * box : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado. * homogeneous : Especifique TRUE si desea que todos los *widgets* (hijos) que se inserten en la caja les sea asignado un espacio por igual.

```
gboolean gtk_box_get_homogeneous (GtkBox *box);
```

Descripción: Devuelve el valor al que esta puesto la propiedad «homogeneous».

Parámetros: * box : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado.

Valor de retorno: El valor de la propiedad «homogeneous».

```
void gtk_box_set_spacing (GtkBox *box,
gint spacing);
```

Descripción: Establece la propiedad «homogeneous» que define cuando los widgets hijos deben de tener el mismo tamaño.

Parámetros: * **box** : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado. * **homogeneous** : Especifique TRUE si desea que todos los *widgets* (hijos) que se inserten en la caja les sea asignado un espacio por igual.

```
gint gtk_box_get_spacing (GtkBox *box);
```

Descripción: Devuelve el valor al que esta puesto la propiedad «spacing».

Parámetros: **box** : Una instancia de GtkBox. Use la macro GTK_BOX() para moldear las referencias de cajas verticales y horizontales al tipo adecuado.

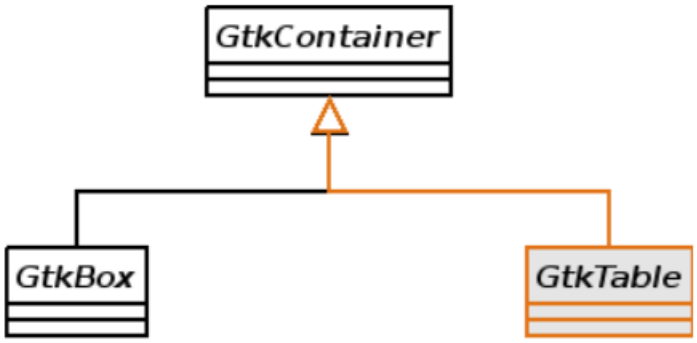
Valor de retorno: El número de *pixeles* que hay entre los *widgets* hijos de la instancia de GtkBox.

CAPÍTULO 6

Tablas

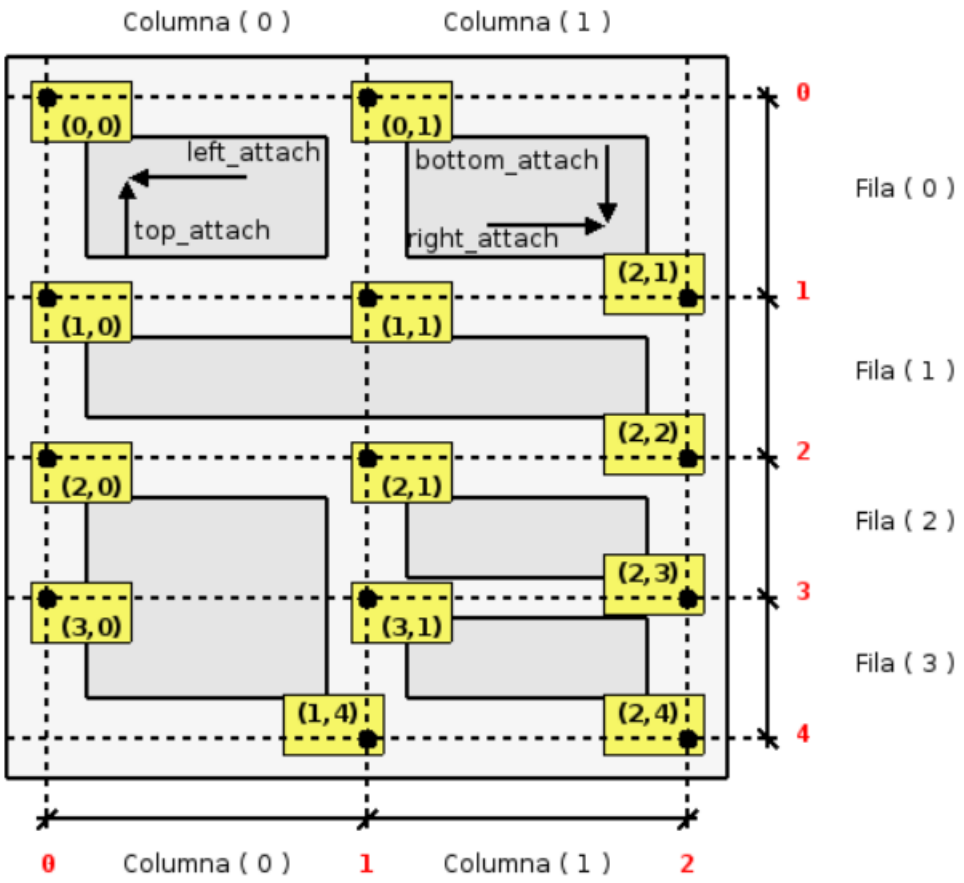
6.1 Descripción

Una tabla es una rejilla en donde se colocan widgets. Los **widgets** pueden ocupar los espacios que se especifiquen (1 o más celdas).



Como es común en GTK+, u n contenedor no tiene una representación gráfica pero afecta la posición y tamaño de los elementos que contiene Cada **widget** se inserta en un rectángulo invisible dentro de la cuadrícula de la tabla.

Según podemos ver en la Figura 3.7.2, un **widget** hijo puede ocupar el espacio de uno o más celdas de la siguiente línea o columna, o ambas. Las coordenadas de ese rectángulo definen de qué celda a qué celda ocupará un **widget**.



El sistema de espaciados contiene diferentes variables que controlar y por tanto puede ocasionar confusión a más de uno. Para una mejor explicación debemos hacer distinción entre las propiedades de la tabla y las propiedades de los **widgets** hijos.

Parámetros de comportamiento de GtkTable.

- Espaciado entre columnas. Define el espacio (en **pixeles**) que habrá entre dos columnas consecutivas. Este valor se controla mediante la propiedad «column-spacing».
- Espaciado entre filas. Define el espacio (en **pixeles**) que habrá entre dos filas consecutivas. Este valor se controla mediante la propiedad «row-spacing».
- Numero de columnas. Define el número de columnas que contendrá la tabla. Un widget puede ocupar más de dos columnas consecutivas.
- Numero de filas. Define el número de filas que contendrá la tabla. Un widget puede ocupar más de dos columnas consecutivas.
- Homogeneidad. Define si las todas las celdas de la tabla tienen el mismo ancho y alto. **Parámetros** de comportamiento de los widgets hijos de GtkTable.
- Columna. La columna donde se encuentra un widget se numera de izquierda a derecha a partir del numero cero.
- Fila. La fila donde se encuentra un widget se numera de arriba a abajo comenzando desde cero.
- Comportamiento vertical y horizontal. Definen el comportamiento de una celda dentro de una tabla. Estos comportamientos pueden ser: * Expandirse para ocupar todo el espacio extra que la tabla le pueda otorgar. * Encogerse para ocupar el espacio mínimo necesario. * Expandirse para ocupar el espacio exacto que la tabla le ha otorgado.

- Relleno vertical y horizontal. Define el espacio en pixeles que habrá entre celdas adyacentes.
- Coordenadas de la celda. Resulta común describir el inicio y el fin de una celda utilizando solamente la coordenada superior izquierda de la celda y la coordenada superior izquierda de la celda transpuesta. Coordenada superior izquierda. Estas coordenadas se forman tomando el numero de la columna que comienza a la izquierda y el numero de la fila que comienza por arriba. Coordenada inferior derecha. Estas coordenadas se forman tomando el numero de la columna que comienza a la derecha y el numero de la fila que comienza por abajo.

6.2 Constructor de clase

Sólo existe un constructor de clase para GtkTable.

```
tkWidget* gtk_table_new (guint rows,
guint columns,
gboolean homogeneous);
```

Descripción: Crea una nueva instancia de una tabla que acomodará widgets a manera de rejilla.

Parámetros:

- **rows**: El número de filas de la tabla.
- **columns**: El número de columnas de la tabla.
- **homogeneous**: Si este valor es TRUE, entonces las celdas de la tabla se ajustan al tamaño del **widget** más largo de la tabla. Si es FALSE, las celdas de la tabla se ajustan al tamaño del **widget** más alto de la fila y el más ancho de la columna.

Valor de retorno: una nueva instancia de GtkTable.

6.3 Métodos de clase

```
void gtk_table_resize (GtkTable *table,
guint rows,
guint columns);
```

Descripción: Cambia el tamaño de la tabla una vez que esta ha sido creada.

Parámetros:

- **table** : Una instancia de GtkTable.
- **rows** : El número de filas que tendrá la nueva tabla.
- **columns** : El número de columnas que tendrá la nueva tabla.

```
void gtk_table_attach_defaults (GtkTable *table,
GtkWidget *widget,
guint left_attach,
guint right_attach,
guint top_attach,
guint bottom_attach);
```

Descripción: Acomoda un **widget** en la celda de una caja. El widget se insertará en la celda definida por las coordenadas definidas por la esquina superior derecha y la esquina inferior izquierda. Para ocupar una o más celdas contiguas especifique la coordenada superior izquierda de la primera celda y la coordenada inferior de la última celda. Usando este método de clase el relleno de la celda será 0 *pixeles* y esta llenará todo el espacio disponible para la celda.

Parámetros:

- **table** : Una instancia de GtkTable.
- **widget** : El **widget** que será acomodado en una celda o celdas adyacentes.
- **left_attach** : ordenada de la esquina superior izquierda.
- **right_attach** : ordenada de la esquina inferior derecha.
- **top_attach** : abscisa de la esquina superior izquierda.
- **bottom_attach** : abscisa de la esquina inferior derecha.

```
void gtk_table_set_row_spacings (GtkTable *table,
guint spacing);
```

Descripción: Establece el espaciado de entre todas las filas de la tabla.

Parámetros:

- **table** : Una instancia de GtkTable.
- **spacing** : El nuevo espaciado en pixeles.

```
void gtk_table_set_col_spacings (GtkTable *table,
guint spacing);
```

Descripción: Establece el espaciado de entre todas las columnas de la tabla.

Parámetros:

- **table** : Una instancia de GtkTable.
- **spacing** : El nuevo espaciado en **pixeles**.

Descripción: Establece el espaciado de una sola fila de la tabla con respecto a las filas adyacentes.

Parámetros:

- **table** : Una instancia de GtkTable.
- **row** : El numero de la fila, comenzando desde cero.
- **spacing** : El nuevo espaciado en **pixeles**.

```
void gtk_table_set_col_spacing (GtkTable *table,
guint col,
guint spacing);
```

Descripción: Establece el espaciado de una sola columna de la tabla con respecto a las columnas adyacentes.

Parámetros:

- **table** : Una instancia de GtkTable.
- **col** : El numero de la columna, comenzando desde cero.
- **spacing** : El nuevo espaciado en **pixeles**.

```
void gtk_table_set_homogeneous (GtkTable *table,
gboolean homogeneous);
```

Descripción: Establece el valor de la propiedad «homogeneous».

Parámetros:

- **table**: Una instancia de GtkTable.
- **homogeneous**: TRUE si se desea que todas las celdas de la tabla tengan el mismo tamaño. Establecer a FALSE si se desea que cada celda se comporte de manera independiente.

```
guint gtk_table_get_default_row_spacing
(GtkTable *table);
```

Descripción: Devuelve el espacio que se asigna por defecto a cada fila que se añade.

Parámetros:

- **table**: Una instancia de GtkTable.

Valor de retorno: El espaciado de la fila.

```
guint gtk_table_get_default_col_spacing
(GtkTable *table);
```

Descripción: Devuelve el espacio que se asigna por defecto a cada columna que se añade.

Parámetros:

- **table** : Una instancia de GtkTable.

Valor de retorno: El espaciado de la columna.

```
guint gtk_table_get_row_spacing (GtkTable *table,
guint row);
```

Descripción: Devuelve el espacio que existe entre la fila y la fila subyacente.

Parámetros:

- **table** : Una instancia de GtkTable.
- **row** : el número de la fila comenzando desde cero.

Valor de retorno: El espaciado de la fila.

```
guint gtk_table_get_col_spacing (GtkTable *table,
guint column);
```

Descripción: Devuelve el espacio que existe entre la columna y la columna adyacente.

Parámetros: * **table** : Una instancia de GtkTable. * **column** : el número de la columna comenzando desde cero.

Valor de retorno: El espaciado de la columna.

```
gboolean gtk_table_get_homogeneous (GtkTable *table);
```

Descripción: Devuelve el estado de la propiedad «homogeneous».

Parámetros:

- **table**: Una instancia de GtkTable.

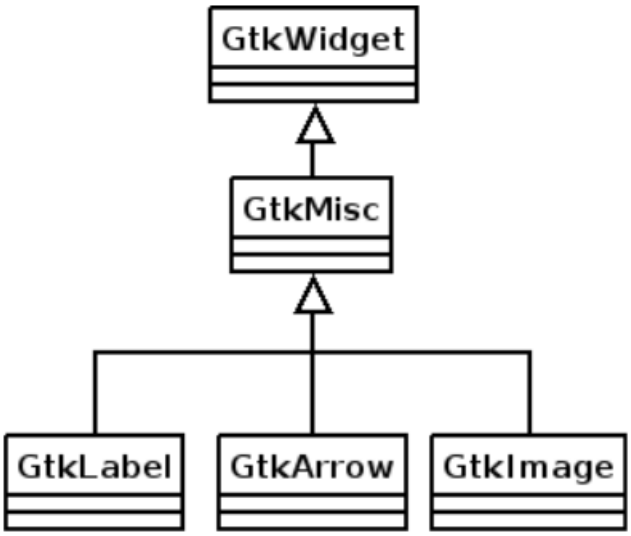
Valor de retorno: El estado de la propiedad «homogeneous».

CAPÍTULO 7

Etiquetas

7.1 Descripción

GtkLabel es útil para desplegar cantidades moderadas de información en forma de texto el cual se puede alinear a la izquierda, derecha y de forma centrada. La opción de lenguaje de marcado (similar a HTML) mejora la calidad y cantidad de información desplegada usando tipos de letra (itálica, negritas, subrayado) y colores.



7.2 Constructor de clase

Solo existe un constructor de clase para GtkLabel.

```
GtkWidget* gtk_label_new (const gchar *str);
```

Descripción: Crea una nueva instancia de una etiqueta GtkLabel que despliega el texto str.

Parámetros:

- **str**: El texto que contendrá la etiqueta. Si no se desea ningún texto adentro de la etiqueta se puede pasar NULL como parámetro para una etiqueta vacía.

Valor de retorno: una nueva instancia de GtkLabel.

7.3 Métodos de clase básicos

Los métodos de clase básicos son los que se usaran con mas frecuencia y se reducen a escribir el texto de la etiqueta y obtenerlo. Si se desea borrar el texto de una etiqueta solo es necesario escribir en ella un texto vacío.

```
void gtk_label_set_text (GtkLabel *label, const gchar *str);
```

Descripción: Establece el texto que mostrara la instancia de una etiqueta.

Parámetros:

- **label** : Una instancia de GtkLabel
- **str** : Un puntero a una cadena que contiene el texto que desplegara la etiqueta. Si especifica NULL entonces se desplegara una etiqueta vacía.

```
const gchar* gtk_label_get_text (GtkLabel *label);
```

Descripción: Obtiene el texto que esta almacenado actualmente en la instancia de la etiqueta.

Parámetros:

- **label** : Una instancia de GtkLabel.

Valor de retorno: un puntero a la cadena que esta almacenada en la etiqueta. La instancia de GtkLabel es dueña de la cadena y por tanto la esta no debe ser modificada.

7.4 Métodos de clase avanzados

La siguiente colección de métodos indican como realizar un control mas avanzado sobre la etiqueta y así mejorar la presentación y sencillez de uso de nuestros programas.

```
void gtk_label_set_justify (GtkLabel *label,
GtkJustification jtype);
```

Descripción: Establece el valor de la propiedad «justify» de GtkLabel. Esta propiedad define la alineación entre las diferentes lineas del texto con respecto unas de otras. Por defecto todas las etiquetas están alineadas a la izquierda.

Parámetros:

- **label**: Una instancia de GtkLabel.

- **jtype**: El tipo de alineación del las lineas de texto en relación con las demás. Lo anterior implica que no hay efecto visible para las etiquetas que contienen solo una linea. Las diferentes alineaciones son:
- GTK_JUSTIFY_LEFT,
- GTK_JUSTIFY_RIGHT,
- GTK_JUSTIFY_CENTER,
- GTK_JUSTIFY_FILL

Es importante hacer notar que esta función establece la alineación del las líneas texto en relación de unas con otras. Este método NO establece la alineación de todo el texto, esa tarea le corresponde a gtk_misc_set_aligment().

```
PangoEllipsizeMode gtk_label_get_ellipsize (GtkLabel *label);
```

Descripción: Describe la manera en que se esta dibujando una elipsis en la etiqueta label.

Parámetros:

- **label** : Una instancia de GtkLabel

Valor de retorno: el modo en que se esta dibujando la elipsis. Este puede ser cualquiera de PANGO_ELLIPSIZE_NONE, PANGO_ELLIPSIZE_START, PANGO_ELLIPSIZE_MIDDLE y PANGO_ELLIPSIZE_END.

```
void gtk_label_set_ellipsize (GtkLabel *label,
PangoEllipsizeMode mode);
```

Descripción: Establece el valor de la propiedad «ellipsize» de GtkLabel. Esta propiedad define el comportamiento de GtkLabel cuando no existe suficiente espacio para dibujar el texto de la etiqueta.

Parámetros:

- **label**: Una instancia de GtkLabel.
- **mode**: Se debe establecer a cualquiera de los cuatro modos definidos en la enumeración PangoEllipsizeMode, a saber: PANGO_ELLIPSIZE_NONE, PANGO_ELLIPSIZE_START, PANGO_ELLIPSIZE_MIDDLE y PANGO_ELLIPSIZE_END. Estos cuatro modos definen si se dibujara una elipsis (»...») cuando no haya suficiente espacio para dibujar todo el texto que contiene la etiqueta. Se omitirán los caracteres suficientes para insertar la elipsis. Si se especifica PANGO_ELLIPSIZE_NONE no se dibujara la elipsis. Si se especifica PANGO_ELLIPSIZE_START entonces se omitirán caracteres del principio de la cadena en favor de la elipsis. Si se especifica PANGO_ELLIPSIZE_MIDDLE los caracteres se omitirán desde la mitad de la cadena hacia los extremos. Si se especifica PANGO_ELLIPSIZE_END los últimos caracteres se eliminaran en favor de la elipsis.

```
PangoEllipsizeMode gtk_label_get_ellipsize (GtkLabel *label);
```

Descripción: Describe la manera en que se esta dibujando una elipsis en la etiqueta label.

Parámetros:

- **label** : Una instancia de GtkLabel

Valor de retorno: el modo en que se esta dibujando la elipsis. Este puede ser cualquiera de PANGO_ELLIPSIZE_NONE, PANGO_ELLIPSIZE_START, PANGO_ELLIPSIZE_MIDDLE y PANGO_ELLIPSIZE_END.

```
void gtk_label_set_markup (GtkLabel *label,
const gchar *str);
```

Descripción: Examina el texto pasado en la cadena str. El texto introducido se formatea de acuerdo al lenguaje de marcado de la librería Pango (similar a HTML). Con este método tenemos la capacidad de desplegar texto con colores o en negritas.

Parámetros:

- **label**: Una instancia de GtkLabel.
- **str**: Un puntero a una cadena que contiene el texto que desplegara la etiqueta y en el lenguaje de marcado de Pango. Si especifica NULL entonces se desplegara una etiqueta vacía. Si el texto no coincide con el lenguaje de marcado de Pango entonces recibirá un mensaje de error en tiempo de ejecución (y no en tiempo de compilación) y la etiqueta o parte de ella no se mostrar.

Vea la Tabla 5 para una breve descripción de las etiquetas válidas.

Tabla 1: Etiquetas válidas para el lenguaje de mercado Pango

Etiqueta	Descripción
Texto	Texto en negritas.
<big> Texto </big>	Texto en un tamaño mas grande en relación con otro texto.
Texto	Texto en itálica.
<s> Texto </s>	Texto rayado.
_{Texto}	Texto a subíndice.
<sup> Texto </sub>	Texto a superíndice.
Texto	Texto en un tamaño mas pequeño en relación con otro texto.
<tt> Texto </tt>	Texto monoespaciado.
<u> Texto </u>	Texto subrayado.
Texto Texto	Texto en color azul.
Texto Texto	Texto con fondo negro.

7.5 Ejemplos

El primer ejemplo sirve para demostrar el uso básico de GtkLabel. Este se muestra en el siguiente listado.

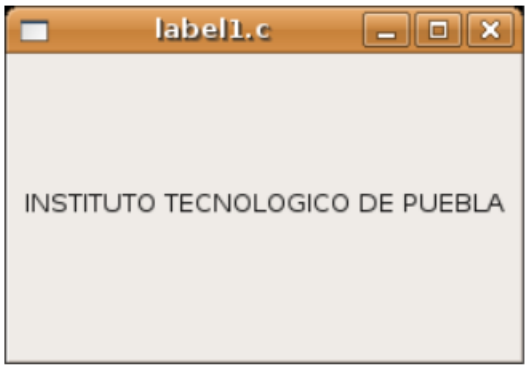
(Listado de programa 3.8.1)

```
/*************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo: label1.c
* Descripcion: Ejemplo sencillo de uso de etiquetas
* Widgets usados: GtkLabel, GtkWindow
*  Comentarios: Basado en el ejemplo disponible en el
* tutorial original de GTK.
*
*  TESIS PROFESIONAL  INSTITUTO TECNOLOGICO DE PUEBLA
* INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*************************************************************************/
#include <gtk/gtk.h>
```

(continué en la próxima página)

(proviene de la página anterior)

```
int main( int   argc, char *argv[] )
{
GtkWidget *window;
GtkWidget *label;
/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
/* Crear una instancia del objeto GtkLabel */
label = gtk_label_new("INSTITUTO TECNOLOGICO DE PUEBLA");
/*Crear una instancia del objeto GtkWidget y configurar esa instancia*/
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
/*Ya sea asociando una retrollamada a un evento ...*/
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),
    NULL);
/*... estableciendo el titulo ... */
gtk_window_set_title (GTK_WINDOW (window), "label1.c");
/*... cambiando el tamanho de la ventana ...*/
gtk_widget_set_size_request(window,250,150);
/* insertando la etiqueta en la ventana ...*/
gtk_container_add (GTK_CONTAINER (window), label);

/*Por ultimo mostramos todos los widgets que tenga la ventana*/
gtk_widget_show_all (window);
/* y otorgamos el control del programa a GTK+*/
gtk_main ();

return 0;
```



La aplicación anterior creará una ventana con una etiqueta adentro. Vea la Figura 3.8.3. Inmediatamente después de inicializar GTK+ (con gtk_init()), se crea una instancia de una etiqueta. Después de eso se crea una ventana, se conecta el evento “delete-event” con gtk_main_quit() de manera que cuando se presione el botón de cerrar la aplicación termine correctamente.

A continuación se ajustan las opciones cosméticas: (a)Establecer el titulo a label1.c y (b) definir el tamaño de la ventana a 200 pixeles de ancho por 150 de alto usando gtk_widget_set_size_request().

Una parte importante que no hay que olvidar es que una aplicación GTK+ se construye acomodando widgets adentro de otros widgets. De esa forma es como se logra relacionar el comportamiento entre diferentes partes de una interfaz gráfica. Una ventana es un contenedor que solo puede alojar un solo widget y en este ejemplo el huésped será la etiqueta que ya hemos creado. La inserción queda a cargo de gtk_container_add().

Sólo queda mostrar todos los widgets usando gtk_widget_show_all() y entregarle el control de la aplicación a GTK+. El ejemplo anterior muestra de la manera mas sencilla cómo instanciar una etiqueta e insertarla en un contenedor. El siguiente ejemplo es una muestra de las principales características avanzadas de GtkLabel.

(Listado de programas 3.8.2)

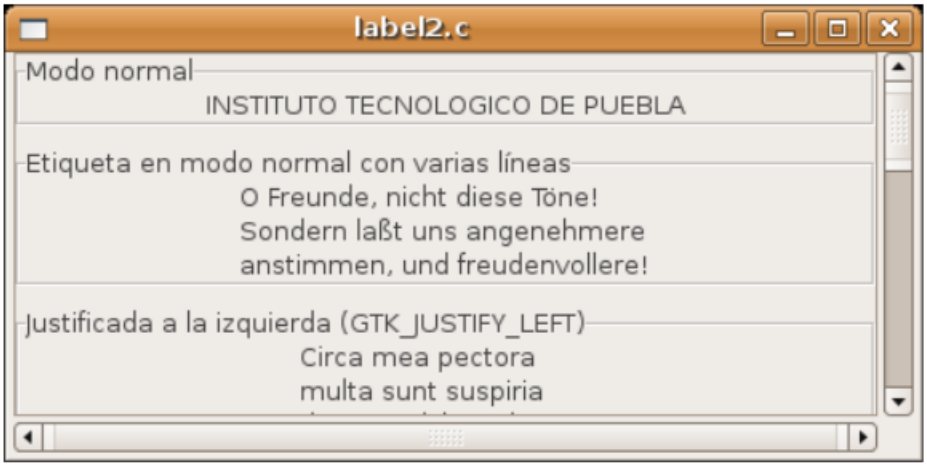
```
/*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      label2.c
* Descripcion:           Alineación del texto de etiquetas.
* Widgets usados:        GtkLabel, GtkBox(GtkVBox), GtkWindow y
*                          GtkScrolledWindow
*  Comentarios:            Basado en el ejemplo disponible en el tutorial
*                          original de GTK. (http://www.gtk.org/tutorial/)
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*****************************************************************************/
#include <gtk/gtk.h>
int main( int   argc,
        char *argv[] )
{
GtkWidget *window;
GtkWidget *vbox;
GtkWidget *frame;
GtkWidget *label;
GtkWidget *scrollw;

/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "label2.c");
scrollw = gtk_scrolled_window_new(NULL,NULL);
vbox = gtk_vbox_new(FALSE,10);
gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrollw),vbox);
gtk_container_add(GTK_CONTAINER(window),scrollw);
gtk_widget_set_size_request(window,450,200);
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),
    NULL);
frame = gtk_frame_new ("Modo normal");
label = gtk_label_new ("INSTITUTO TECNOLOGICO DE PUEBLA");
gtk_container_add (GTK_CONTAINER (frame), label);
gtk_box_pack_start_defaults (GTK_BOX (vbox), frame);

frame = gtk_frame_new ("Etiqueta en modo normal con varias líneas");
label = gtk_label_new ("O Freunde, nicht diese Töne!\n"\
                       "Sondern laßt uns angenehmere\n" \
        "anstimmen, und freudenvollere!");
gtk_container_add (GTK_CONTAINER (frame), label);
gtk_box_pack_start_defaults (GTK_BOX (vbox), frame);

frame = gtk_frame_new ("Justificada a la izquierda (GTK_JUSTIFY_LEFT)");
label = gtk_label_new ("Circa mea pectora\nmulta sunt suspiria\n"\
    "de tua pulchritudine,\nque me ledunt misere.");
gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_LEFT);
gtk_container_add (GTK_CONTAINER (frame), label);
gtk_box_pack_start_defaults (GTK_BOX (vbox), frame);
```

(continué en la próxima página)

(proviene de la página anterior)

```
frame = gtk_frame_new ("Justificada a la derecha (GTK_JUSTIFY_RIGHT)");
label = gtk_label_new ("Como quien viaja a lomos de una llegua sombría,\n"\
    "por la ciudad camino, no pregunteis a dónde\n"\
    "busco, acaso, un encuentro que me ilumne el dia.\n"\
"Y no encuentro más que puertas que niegan lo que esconden,\n");
gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_RIGHT);
gtk_container_add (GTK_CONTAINER (frame), label);
gtk_box_pack_start_defaults (GTK_BOX (vbox), frame);

frame = gtk_frame_new ("Texto distribuido en la etiqueta (GTK_JUSTIFY_FILL)");
label = gtk_label_new ("FAUSTO.- ¿Quién soy yo, pues, si no me es dado llegar "\
            "a esa corona de la humanidad a la que aspiran todos mis sentidos?\n"\
"MEFISTÓFELES. - Tú eres, en último resultado, lo que debes ser: "\
"colóca sobre tu cabeza una peluca de miles de bucles, calza tus"\
"pies con conturnos de una vara de alto, que no por ello dejarás"\
"de ser lo que eres.");
gtk_label_set_justify (GTK_LABEL (label), GTK_JUSTIFY_FILL);
gtk_label_set_line_wrap (GTK_LABEL (label), TRUE);
gtk_container_add (GTK_CONTAINER (frame), label);
gtk_box_pack_start_defaults (GTK_BOX (vbox), frame);


gtk_widget_show_all (window);
gtk_main ();

return 0;
```

La aplicación tendrá la siguienda apariencia



Este ejemplo se vuelve un poco más complicado pues ahora hacemos uso de 5 tipos de widgets: GtkWindow, GtkLabel, GtkVBox, GtkFrame y GtkScrolledWindow. Esto se ha hecho debido a que ahora debemos transmitir una mayor cantidad de información en una sola ventana(De paso aprenderemos a trabajar con nuevos objetos de los que conocemos muy poco).

Se han creado cinco diferentes etiquetas y cada una contiene un texto diferente. A cada una de estas etiquetas se le ha aplicado un modo de alineación diferente. Para evitar la confusión y mejorar la apariencia del programa se ha decorado cada una de las diferentes etiquetas con un cuadro que describe el tipo de modo que se quiere mostrar. La clase GtkFrame se comporta como un contenedor más (esta clase se describirá mas a fondo en el apartado dedicado a widgets para decoración).

Debido a que desplegaremos toda la información al mismo tiempo es necesario usar una caja vertical (GtkVBox) para

acomodar todos los marcos y las etiquetas.

Por último se utilizó la clase GtkScrolledWindow para añadir barras de desplazamiento y así evitar que la ventana tenga un tamaño grande y desgradable.

En resúmen: cinco etiquetas (GtkLabel) con diferente alineación se insertan con sendos marcos(GtkFrame), los cuales se alojan en una caja vertical(GtkVBox). Esta caja se "mete" dentro de una ventana que contiene barras de desplazamiento(GtkScrolledWindow) que a su vez se inserta en la ventana de nivel principal (GtkWindow).

Hay otros dos ejemplos que hay que mostrar. El primero(Listado de Programa 3.8.3) muestra la forma de usar el lenguaje de marcado de Pango para definir diferentes estilos de texto (colores, fuentes,etc.).

(Listado de Programa 3.8.3)

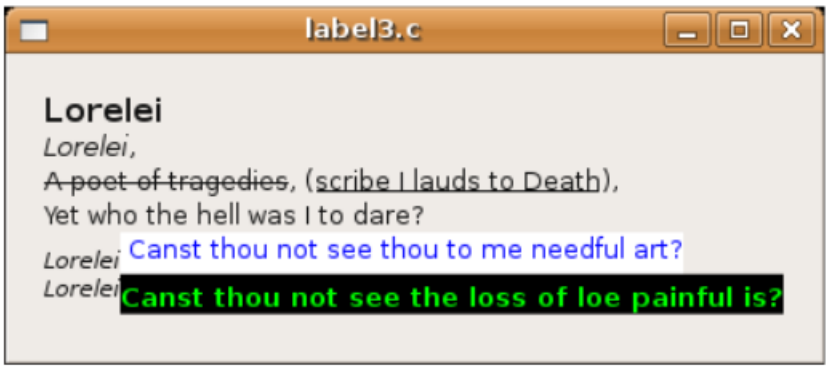
```
/*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      label3.c
*  Descripcion:            Marcado de atributos de texto
*  Widgets usados:         GtkLabel, y GtkWindow
*  Comentarios:            Este ejemplo muestra como utilizar un lenguaje de
*                          marcado de texto similar a HTML para definir el
*                          estilo de texto desplegado en cualquier etiqueta.
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*****************************************************************************/
#include <gtk/gtk.h>
int main( int   argc,
        char *argv[] )
{
GtkWidget *window;
GtkWidget *label;

/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "label3.c");
gtk_widget_set_size_request(window,400,150);
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),
    NULL);
label = gtk_label_new (NULL);
gtk_label_set_markup(GTK_LABEL(label),"<big><b>Lorelei</b></big>\n\
<i>Lorelei</i>,\n\
<s>A poet of tragedies</s>, (<u>scribe I lauds to Death</u>),\n\
Yet who the hell was I to dare?\n\
<sub><i>Lorelei</i></sub>\
<span foreground=\"blue\" background=\"white\"> \
99
Canst thou not see thou to me needful art?</span>\n\
<sup><i>Lorelei</i></sup>\
<span foreground='#00FF00' background='#000000' weight='ultrabold'>\
Canst thou not see the loss of loe painful is?</span>");
gtk_container_add(GTK_CONTAINER(window),label);

gtk_widget_show_all (window);
```

(continué en la próxima página)

(proviene de la página anterior)

```
gtk_main ();

return 0;
}
```

El Listado de Programa 3.8.3 luce como en la Figura 3.8.5



El segundo ejemplo Listado de Programa 3.8.4 muestra como funciona las elipsis.

(Listado de Programa 3.8.4)

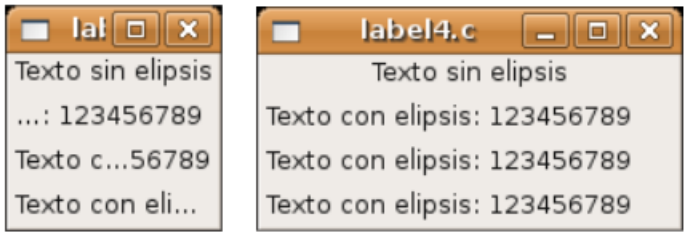
```
    /*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      label4.c
*  Descripcion:            Uso de elipsis en las etiquetas.
*  Widgets usados:         GtkLabel, GtkBox(GtkVBox)y GtkWindow
*  Comentarios:            Las elipsis son utiles para mostrar texto en
*                          una etiqueta con espacio restringido.
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*****************************************************************************/
#include <gtk/gtk.h>

Figura 3.8.5: Uso del lenguaje de marcado en etiquetas
int main( int   argc,
        char *argv[] )
{
GtkWidget *window;
GtkWidget *vbox;
GtkWidget *label;
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "label4.c");
vbox = gtk_vbox_new(FALSE,5);
gtk_container_add(GTK_CONTAINER(window),vbox);
g_signal_connect (G_OBJECT (window), "destroy", G_CALLBACK (gtk_main_quit),
    NULL);
label = gtk_label_new ("Texto sin elipsis");
gtk_box_pack_start_defaults (GTK_BOX (vbox), label);
```

(continué en la próxima página)

(proviene de la página anterior)

```
label = gtk_label_new ("Texto con elipsis: 123456789");
gtk_label_set_ellipsize(GTK_LABEL(label),PANGO_ELLIPSIZE_START);
gtk_box_pack_start_defaults (GTK_BOX (vbox), label);
label = gtk_label_new ("Texto con elipsis: 123456789");
gtk_label_set_ellipsize(GTK_LABEL(label),PANGO_ELLIPSIZE_MIDDLE);
gtk_box_pack_start_defaults (GTK_BOX (vbox), label);

label = gtk_label_new ("Texto con elipsis: 123456789");
gtk_label_set_ellipsize(GTK_LABEL(label),PANGO_ELLIPSIZE_END);
gtk_box_pack_start_defaults (GTK_BOX (vbox), label);


gtk_widget_show_all (window);
gtk_main ();
return 0;
}
```

Con esto hemos cubierto gran parte de la funcionalidad de las etiquetas. Más información se puede hallar en el manual de referencia de GTK+.



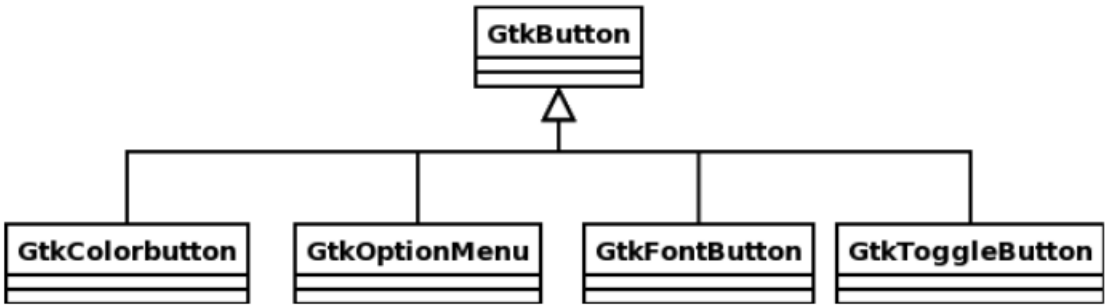
Con esto hemos cubierto gran parte de la funcionalidad de las etiquetas. Más información se puede hallar en el manual de referencia de GTK+.

CAPÍTULO 8

Botones

8.1 Descripción

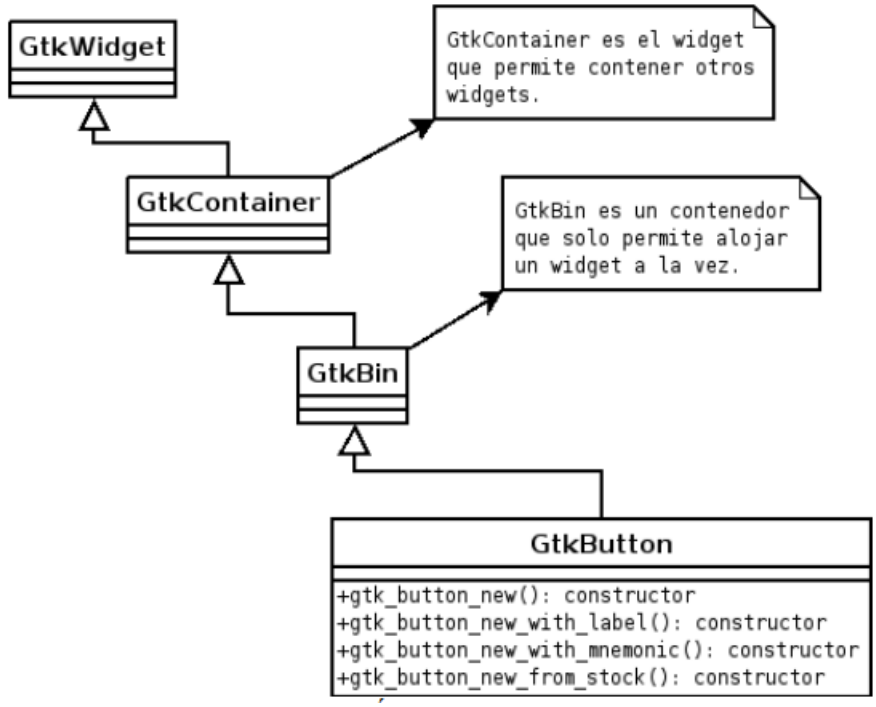
GtkButton es un widget que emite una señal cuando es presionado. Un botón es a su vez un contenedor. Por lo general contiene una etiqueta, una imagen o ambas. GtkButton es punto de partida para la creación de otros tipos de botones (Vea la Figura 3.9.1).



Más adelante analizaremos el funcionamiento de GtkToggleButton y GtkOptionMenu. Los dos restantes no serán cubiertos en este manual. GtkColorButton es un botón que al ser presionado muestra una ventana de selección de color y GtkFontButton mostrará una ventana de selección de fuente al ser presionado.

8.2 Constructores de clase

Existen cuatro constructores de clase para GtkButton. Se puede usar el constructor gtk_button_new_with_label() ó gtk_button_new_with_mnemonic() para crear un botón con etiqueta(normal y con acelerador, respectivamente); gtk_button_new_with_stock() crear un botón cuya imagen y texto estará determinado por un identificador stock_item, y por último gtk_button_new() instancia un botón vacío.




```
GtkWidget* gtk_button_new (void);
```

Descripción: Crea una nueva instancia de un botón GtkButton. Esta nueva instancia de botón no contiene nada. Si desea colocar algún widget dentro de la nueva instancia use gtk_container_add().

Valor de retorno: una nueva instancia de GtkButton.

```
GtkWidget* gtk_button_new_with_label (const gchar *label);
```

Descripción: Crea una nueva instancia de un botón GtkButton. El nuevo botón contendrá una etiqueta con el texto especificado.

Parámetros:

- **label** : El texto que contendrá la etiqueta dentro del botón.

Valor de retorno: una nueva instancia de GtkButton.

```
GtkWidget* gtk_button_new_with_mnemonic (const gchar *label);
```

Descripción: Crea una nueva instancia de un botón GtkButton. El nuevo botón contendrá una etiqueta con el texto especificado. Cualquier letra que venga precedida de un guión bajo ("_"), aparecerá como texto subrayado. La primera letra que sea precedida con un guión bajo se convierte en el acelerador del botón, es decir, presionando la tecla Alt y la letra activan el botón(Causan que se emita la señal «clicked»).

Parámetros:

- **label**: El texto que contendrá la etiqueta dentro del botón. Anteponga un guión bajo a un carácter para convertirlo en acelerador.

Valor de retorno: una nueva instancia de GtkButton.

```
GtkWidget* gtk_button_new_from_stock (const gchar *label);
```

Descripción: Crea una nueva instancia de un botón GtkButton. El nuevo botón contendrá una imagen y una etiqueta predeterminados(stock item) . Es una forma sencilla de hacer botones vistosos con mensajes usuales como si, no, cancelar y abrir. Al usar elementos predeterminados (stock items) nos aseguramos que los botones sigan el tema y el idioma elegidos en el entorno GNOME.

Parámetros:

- **label** : El nombre del elemento predeterminado (stock item). Una lista de los elementos predeterminados se muestra en el ANEXO 4.6.1.3 : STOCK ITEMS.

Valor de retorno: una nueva instancia de GtkButton.

8.3 Métodos de clase

```
void gtk_button_set_label (GtkWidget button, const gchar *label);
```

Descripción: Establece el mensaje que mostrará la etiqueta de un botón. El nuevo botón contendrá una etiqueta con el texto especificado. Si hay otro widget dentro del botón, entonces GTK+ lo eliminará y en su lugar insertará una etiqueta.

Parámetros:

- **button** : Una instancia de GtkButton.
- **label** : El texto que contendrá la etiqueta dentro del botón.

```
const gchar* gtk_button_get_label (GtkButton *button);
```

Descripción: Regresa el texto contenido en la etiqueta de un botón si el botón ha sido creado con gtk_button_new_with_label() o se ha establecido el texto de la etiqueta con el método gtk_button_set_label(). Si lo anterior no se cumple el valor regresado es NULL.

Parámetros:

- **button** : Una instancia de GtkButton.

Valor de retorno: el texto de la etiqueta del botón. La cadena regresada por este método es propiedad de Gtk+, no la libere ni la manipule u obtendrá un fallo de segmentación.

```
void gtk_button_set_use_stock (GtkButton *button,
gboolean use_stock);
```

Descripción: Si esta propiedad se establece a verdadero entonces el texto de la etiqueta del botón se usará para seleccionar un elemento predeterminado(stock item) para el botón. Use gtk_button_set_text() para establecer un elemento predeterminado.

Parámetros:

- **button** : Una instancia de GtkButton.
- **use_stock** : TRUE si el botón deberá mostrar elementos predeterminados (stock item).

```
gboolean gtk_button_get_use_stock (GtkButton *button);
```

Descripción: Determina si la instancia del botón muestra elementos predeterminados (stock item).

Parámetros:

- **button** : Una instancia de GtkButton.

Valor de retorno: TRUE si el botón despliega elementos predeterminados.

```
void gtk_button_set_use_underline (GtkButton *button,
gboolean use_underline);
```

Descripción: Si esta propiedad se establece a verdadero entonces cualquier letra que venga precedida de un guión bajo ("_"), aparecerá como texto subrayado. La primera letra que sea precedida con un guión bajo se convierte en el acelerador del botón. Use gtk_button_set_text() para establecer el texto subrayado y/o aceleradores.

Parámetros:

- **button** : Una instancia de GtkButton.
- **use_stock** : TRUE si el botón deberá subrayar elementos y generar mnemónicos.

```
gboolean gtk_button_get_use_underline (GtkButton *button);
```

Descripción: Determina si la instancia del botón subraya caracteres y genera mnemónicos y atajos de teclado.

Parámetros:

- **button** : Una instancia de GtkButton.

Valor de retorno: TRUE si el botón subraya caracteres y genera mnemónicos.

CAPÍTULO 9

Señales y eventos

9.1 La señal «clicked»

```
void retrollamada (GtkButton *button,
gpointer user_data);
```

Descripción: Esta señal se emite cuando se ha activado el botón. Lo anterior implica dos eventos: el usuario presiona el botón y lo libera (button-press-event y button-release-event). Lo anterior es importante debido a la confusión que ocasiona la sutil diferencia entre señales y eventos (Consulte el capítulo 3.4, Teoría de señales y retrollamadas). Como condición para emitir la señal «clicked», el usuario debe presionar el botón y al liberarlo el cursor del ratón debe permanecer en el botón.

Parámetros:

- **button** : La instancia de GtkButton que recibe la señal.
- **user_data** : Datos extras que se registran cuando se conecta la señal a esta retrollamada.

9.2 Ejemplos

El primer ejemplo tendrá como objetivo mostrar el producto de los 4 constructores de clase de GtkButton.

(Listado de Programa 3.9.1)

```
/*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      button1.c
* Descripcion:            Ejemplo basico de uso de GtkButton
* Widgets usados:         GtkButton, GtkVBox y GtkWindow
*  Comentarios:            Este ejemplo muestra el producto de los cuatro
*                          diferentes constructores de clase de GtkButton.
*
```

(continué en la próxima página)

(proviene de la página anterior)

```
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*****************************************************************************/
#include <gtk/gtk.h>
int main( int   argc, char *argv[] )
{
GtkWidget *window;
GtkWidget *button;
GtkWidget *box;


/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "button1.c");
gtk_widget_set_size_request(window,200,200);
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),
    NULL);
/*Creamos una caja vertical con espaciado homogeneo y 5 pixels entre cada
elemento*/
box = gtk_vbox_new(TRUE,5);
gtk_container_add(GTK_CONTAINER(window),box);
/*Un boton sin nada adentro*/
button = gtk_button_new();
gtk_box_pack_start_defaults(GTK_BOX(box),button);

/*Un boton con una etiqueta*/
button = gtk_button_new_with_label("Electronica");
gtk_box_pack_start_defaults(GTK_BOX(box),button);
/*Un boton con un mnemonico*/
button = gtk_button_new_with_mnemonic("_Encender motor");
gtk_box_pack_start_defaults(GTK_BOX(box),button);
/*Un boton con elemento predeterminado*/
button = gtk_button_new_from_stock(GTK_STOCK_CONNECT);
gtk_box_pack_start_defaults(GTK_BOX(box),button);

gtk_widget_show_all (window);
gtk_main ();

return 0;
}
```

El programa anterior crea una ventana y una caja vertical donde se insertan cuatro botones (cada uno instanciado con un constructor de clase diferente).

En la Figura 3.6.10 se muestra el resultado de nuestro programa. Recordemos que GTK+ es una librería que soporta varios idiomas. Cuando el entorno GNOME o GTK+ están configurados para el idioma inglés (por ejemplo), los elementos predeterminados del último botón se traducen automáticamente, de ahí la importancia de usar elementos predeterminados (stock items), cada vez que se tenga la oportunidad.

El segundo ejemplo se vuelve un poco mas complicado pues se comienza a usar las retrollamadas. En este caso hacemos uso de la señal «clicked» para implementar una pequeña máquina de estados que nos ayude a mostrar el efecto de los diferentes métodos de clase de GtkButton.

(Listado de Programa 3.9.2)

```
/*************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      button2.c
*  Descripcion:            Metodos de clase GtkButton
*  Widgets usados:         GtkButton, GtkVBox, GtkLabel, GtkWindow
*  Comentarios:            Este ejemplo muestra el producto de los cuatro
*                          diferentes constructores de clase de GtkButton.
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*************************************************************************/


Figura 3.9.4: Cuatro botones creados con cuatro constructores diferente. Ponga
atención en el último botón de ambas ventanas.
#include <gtk/gtk.h>
void retrollamada (GtkWidget *button, gpointer datos_extra){
GtkLabel *label = GTK_LABEL(datos_extra);
static gint contador =0;
//Si, es una maquina de estados.
switch (contador){
case 0:
//Lo convertiremos en un boton con una etiqueta
gtk_label_set_markup(label,
"<b>Ahora es un botón con una etiqueta.\n"
"Presione el botón para activar \nla opcion de subrayado</b>");
gtk_button_set_label(GTK_BUTTON(button),"_Siguiente");
```

(continué en la próxima página)

(proviene de la página anterior)

```
break;
case 1:
//Ahora sera un boton con un mnemonico
gtk_label_set_markup(label,
"<b>Ahora es un botón con un mnemónico.\n"
"Pruebe presionando las teclas <u>Alt</u> y <u>S</u> o\n"
"presionando el botón para convertirlo\n"
"en un botón con un elemento\npredeterminado</b>\n");
gtk_button_set_use_underline(GTK_BUTTON(button), TRUE);
gtk_button_set_label(GTK_BUTTON(button),"_Siguiente");
break;
case 2:
gtk_label_set_markup(label,
"<span color='blue'>Fin de la demostracion.</span>\n");
gtk_button_set_use_stock(GTK_BUTTON(button), TRUE);
gtk_button_set_label (GTK_BUTTON(button),GTK_STOCK_CLOSE);
break;
default :
gtk_main_quit();
}
contador++;
}
int main( int   argc, char *argv[] )
{
GtkWidget *window;
GtkWidget *button;
GtkWidget *box;
GtkWidget *label;
/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "button2.c");
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),

    NULL);
/*Creamos una caja vertical sin espaciado homogeneo y 5 pixels entre cada
elemento*/
box = gtk_vbox_new(FALSE,5);
gtk_container_add(GTK_CONTAINER(window),box);

/*En la caja insertamos una nueva etiqueta */
label = gtk_label_new("<b>Este es un botón vacío.\n"
"Presione el botón para convertirlo en un botón con etiqueta</b>");
gtk_label_set_use_markup(GTK_LABEL(label),TRUE);
gtk_label_set_line_wrap(GTK_LABEL(label),TRUE);
gtk_widget_set_size_request(label,200,200);
gtk_box_pack_start_defaults(GTK_BOX(box),label);

/*Tambien se añade un boton sin nada adentro*/
button = gtk_button_new();
gtk_widget_set_size_request(button,200,40);
gtk_box_pack_start_defaults(GTK_BOX(box),button);

/*Ahora conectamos la señal "clicked" a la funcion retrollamada*/
g_signal_connect(G_OBJECT(button),"clicked",
G_CALLBACK (retrollamada),
```

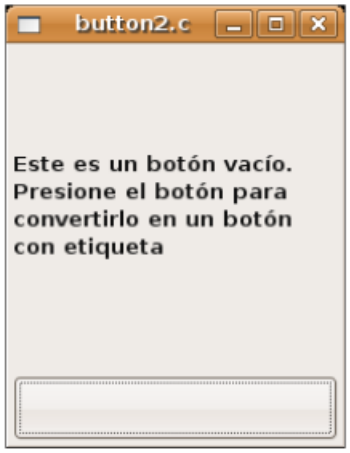
(continué en la próxima página)

(proviene de la página anterior)

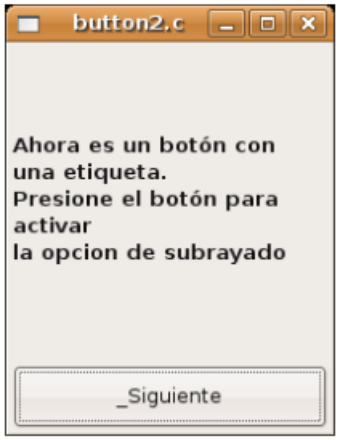
```
label);
/*Atencion: ahora hemos enviado la etiqueta label
como un parametro extra (en lugar de NULL)*/

gtk_widget_show_all (window);
gtk_main ();

return 0;
}
```

Comencemos por la estructura de la aplicación: En una ventana se inserta una caja vertical, una etiqueta y un botón. Para mejorar la presentación visual de la aplicación (algo muy importante), los mensajes que se usen en la etiqueta usarán el lenguaje de marcado de Pango. Debido al comportamiento de la caja vertical (que intentará cambiar el tamaño de los widgets), se ha usado gtk_widget_set_size_request() en el botón y la etiqueta para fijar el tamaño de ambos. Como ha sido usual hasta ahora se conecta el evento «delete-event» de la ventana principalcon la función gtk_main_quit(), esto ocasiona que cuando se presione el botón de cerrar en la ventana el programa termine.

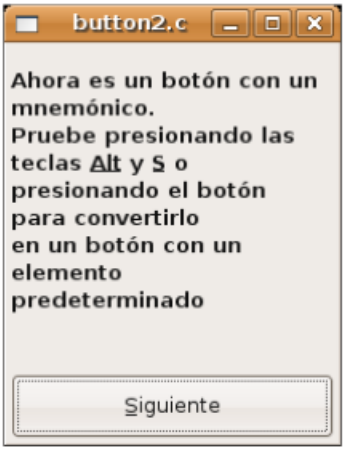
En este ejemplo hemos hecho uso de la señal «clicked». Esta señal se conectó a la función retrollamada(). Cuando el usuario presione el botón se llamará a esta función. Por primera vez hemos usado el último parámetro de la función g_signal_connect(). Casi siempre hemos utilizado la macro NULL en este espacio, indicándole a glib que no deseamos enviar ningún parámetro extra a la función retrollamada(), como fue en el caso de gtk_main_quit() la cual no toma parámetros. Ahora haremos uso de ese espacio enviándole el puntero de la etiqueta que usamos en la ventana a la función retrollamada(): Cada vez que esta función se ejecute tendremos disponible una referencia al botón y a la etiqueta sin la necesidad de usar variables globales, pues estas están dentro de main() y no son visibles desde dentro de la función. Dentro de la función retrollamada() se ha implementado una pequeña maquina de estados: cada vez que presionemos el botón este cambiará de aspecto usando los métodos de clase que hemos discutido aquí. Al iniciar la aplicación, esta tendrá un aspecto parecido al de la Figura 3.9.5.



Cuando se presiona el botón se llama a la función retrollamada(). La máquina de estados reconoce que es la primera vez que se entra a la función (el contador es 0), así que cambia el mensaje que despliega la etiqueta y usa el método gtk_button_set_label() el cual, en este específico caso, inserta una etiqueta en el botón con un mensaje(Figura 3.9.6)



En el siguiente estado de la máquina (cuando el contador es 1) se activará la propiedad «use-underline» mediante el método gtk_button_set_use_underline().



Cuando el contador llega a 2, la maquina de estados insertará un elemento predeterminado, lo cual implica establecer la propiedad «use-stock» a TRUE utilizando gtk_button_set_use_underline(). Vea la siguiente figura.

Por último, presionando el botón se termina la aplicación. En este último ejemplo nosotros hemos aprendido a utilizar los diferentes métodos de clase de GtkButton. También hemos aprendido a usar la señal clicked e implementar acciones con ella. Por último hemos aprendido una lección importante: las interfases gráficas diseñadas con GTK+ no son

estáticas, si no dinámicas y pueden cambiar dependiendo de las necesidades de la aplicación y del usuario.

CAPÍTULO 10

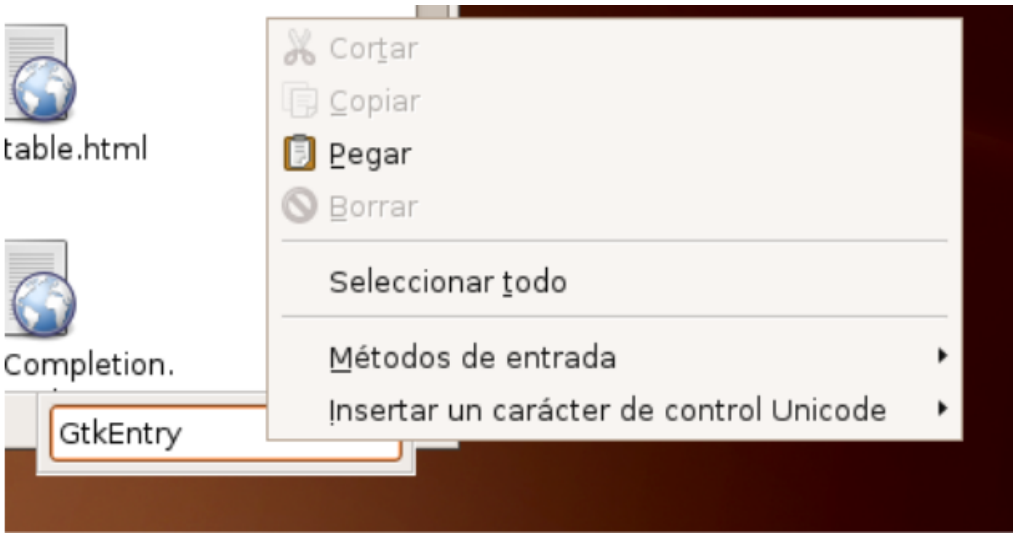
Cajas de texto

GtkEntry es un widget de entrada de texto. Puede almacenar sólo una cantidad limitada de información debido a que sólo despliega una linea de texto. Si el texto que se introduce es más largo del que se puede mostrar, entonces el contenido de la caja de texto se irá desplazando de tal manera que se pueda visualizar lo que se esta escribiendo.

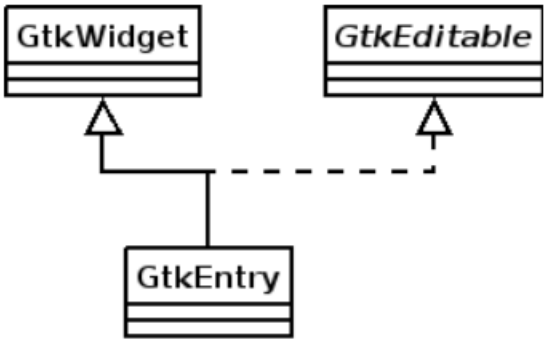
La mayoría de los atajos del teclado, comunes en cualquier aplicación, se encuentran disponibles (ver la Tabla 6). Además de lo anterior también soporta arrastrar y soltar(drag & drop). Con la integración de Pango como componente base de GTK+, todos los objetos de texto de GTK+ tienen la posibilidad de desplegar textos en otros alfabetos diferentes y soportar métodos de escritura diferentes al nuestro (por ejemplo, chino, hindú o ruso).

Tabla 1: Atajos de teclado disponible en cajas de texto

Atajo	Descrpción
Flechas de dirección (←↑→↓)	Navegar en el texto, carácter por carácter.
Ctrl + Flechas de dirección (←↑→↓)	Navegar en el texto, palabra por palabra.
Ctrl + Inicio (Home) ó Fin (End)	Ir al inicio o al fin del texto.
↑Shift + Flechas de dirección (←↑→↓)	Selección de texto, palabra por palabra.
↑Shift + Ctrl + Inicio (Home) ó Fin (End)	Selecciona el texto, desde el cursor hasta el inicio o fin.
Ctrl + C ó Ctrl + Ins (Insert)	Copiar el texto seleccionado.
Ctrl + P ó ↑Shift + Ins (Insert)	Pegar el texto en la posición del cursor.
Ctrl + X ó ↑Shift + Supr (Delete)	Cortar el texto seleccionado.
↑Shift + F10	Desplegar el menú de contexto.



10.1 Constructor de clase



Sólo existe un constructor de clase. En las primeras versiones de GTK+ existieron 2 constructores, sin embargo uno de ellos ha caído en desuso.

```
GtkWidget* gtk_entry_new (void);
```

Descripción: Crea una nueva instancia de una caja de texto GtkEntry.

Valor de retorno: una nueva instancia de GtkEntry.

10.2 Métodos de clase

Algunos de los métodos de clase, anteriormente disponibles para GtkEntry, ahora han caído en desuso en favor de la interfaz GtkEditable. Esta interfaz provee funcionalidad muy similar para todos los widgets de texto (no solamente GtkEntry). En este capítulo solamente discutiremos los métodos de clase propios de GtkEntry.

```
void gtk_entry_set_text (GtkEntry *entry,
const gchar *text);
```

Descripción: Establece el contenido de la caja de texto. Reemplaza cualquier contenido anterior.

Parámetros:

- **entry**: Una instancia de GtkEntry.
- **text**: Un puntero a una cadena que contiene el texto que desplegara la caja de texto. Si especifica NULL equivale a limpiar la caja de texto.

Valor de retorno: Un puntero a una cadena con el contenido de la caja de texto. La instancia de GtkEntry es dueña de la cadena y por tanto la esta no debe ser modificada.

```
const gchar* gtk_entry_get_text (GtkEntry *entry);
```

Descripción: Devuelve el contenido de la caja de texto.

Parámetros:

- **entry** : Una instancia de GtkEntry.

Valor de retorno: Un puntero a una cadena con el contenido de la caja de texto. La instancia de GtkEntry es dueña de la cadena y por tanto la esta no debe ser modificada.

```
void gtk_entry_set_visibility (GtkEntry *entry,
gboolean visible);
```

Algunos de los métodos de clase, anteriormente disponibles para GtkEntry, ahora han caído en desuso en favor de la interfaz GtkEditable. Esta interfaz provee funcionalidad muy similar para todos los widgets de texto (no solamente GtkEntry). En este capítulo solamente discutiremos los métodos de clase propios de GtkEntry.

```
void gtk_entry_set_text (GtkEntry *entry,
const gchar *text);
```

Descripción: Establece el contenido de la caja de texto. Reemplaza cualquier contenido anterior.

Parámetros:

- **entry**: Una instancia de GtkEntry.
- **text**: Un puntero a una cadena que contiene el texto que desplegara la caja de texto. Si especifica NULL equivale a limpiar la caja de texto.

Valor de retorno: Un puntero a una cadena con el contenido de la caja de texto. La instancia de GtkEntry es dueña de la cadena y por tanto la esta no debe ser modificada.

```
const gchar* gtk_entry_get_text (GtkEntry *entry);
```

Descripción: Devuelve el contenido de la caja de texto.

Parámetros:

- **entry** : Una instancia de GtkEntry.

Valor de retorno: Un puntero a una cadena con el contenido de la caja de texto. La instancia de GtkEntry es dueña de la cadena y por tanto la esta no debe ser modificada.

```
void gtk_entry_set_visibility (GtkEntry *entry,
gboolean visible);
```

Parámetros:

- **entry**: Una instancia de GtkEntry.

Valor de retorno: Regresa el número máximo de caracteres . Si es 0 entonces no hay un límite más allá que el valor máximo de gint.

10.3 Señales

GtkEntry tiene una lista de 10 señales diferentes.

Una discusión detallada de las 10 señales diferentes seria una tarea larga. En lugar de hacer una lista detallada aprovecharemos una característica útil de GTK+: cada señal define el tipo de retrollamada que quiere usar. Afortunadamente la mayoría de las señales no necesitan retrollamadas complejas y utilizan el mismo prototipo. Este es el caso de cinco señales de más usuales de GtkEntry. En la Tabla 7 hacemos una relación de esas 5 señales y su descripción. Todas esta señales usan el mismo prototipo de función retrollamada el cual resulta ser muy parecido al de la señal «clicked» de GtkButton. El prototipo genérico es:

```
void funcion_retrollamada ( GtkWidget *widget, gpointer datos);
```

Aunque es común encontrarlo en esta forma:

```
void funcion_retrollamada ( GtkEntry *entry, gpointer datos);
```

La diferencia entre ambas es que si usamos la primera podemos conectar esa retrollamada a casi cualquier señal de cualquier widget. La segunda tiene la ventaja de habernos hecho el moldeado del widget a GtkEntry. Su desventaja radica en que solo puede ser usada para las señales emitidas por GtkEntry.

Tabla 2: Atajos de teclado disponible en cajas de texto

Señal	Causas de la emision de la señal
«activate»	Se ha presionado la tecla enter.
«backspace»	Todas las veces que se ha presionado la tecla de retroceso
«copy-clipboard»	Cuando el usuario ha copiado texto de la caja, ya sea usando un menú contextual o cualquiera de los atajos del teclado.
«cut-clipboard»	Cuando el usuario ha cortado texto de la caja, ya sea usando un menú contextual o cualquiera de los atajos del teclado.
«paste-clipboard»	Cuando el usuario ha pegado texto en la caja, ya sea usando un menú contextual o cualquiera de los atajos del teclado.

La decisión de usar una u otra forma la toma el programador de acuerdo a su conveniencia.

10.4 Ejemplos

Ha llegado el tiempo de mostrar lo que podemos hacer con GtkEntry. El primer ejemplo mostrará como conectar una sola retrollamada a las 6 señales descritas en la Tabla 7 y a un botón.

(Listado de Programa 3.10.1)

```
/*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      entry1.c
*  Descripcion:            Señales de GtkEntry
*  Widgets usados:         GtkEntry, GtkButton, GtkVBox, GtkLabel, GtkWindow
*  Comentarios:            Se mostrará como conectar una sola retrollamada
```

(continué en la próxima página)

(proviene de la página anterior)

```
*                          a las 6 señales descritas en la Tabla 7 y a un
*                          botón.
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*****************************************************************************/
#include <gtk/gtk.h>
void callback (GtkWidget *widget, gpointer datos_extra){
gint i = GPOINTER_TO_UINT(datos_extra);


if (i == 5) {
g_print("El botón ha generado la señal \"clicked\"\n");
return;
}
g_print("La caja de texto ha generado la señal ");
switch (i){
case 0: g_print("\"activate\"\n"); break;
case 1: g_print("\"backspace\"\n"); break;
case 2: g_print("\"copy-clipboard\"\n"); break;
case 3: g_print("\"cut-clipboard\"\n"); break;
case 4: g_print("\"paste-clipboard\"\n"); break;
}
}
int main( int   argc, char *argv[] )
{
GtkWidget *window;
GtkWidget *widget;
GtkWidget *box;
gchar *signals[] = {"activate", "backspace", "copy-clipboard",
"cut-clipboard", "paste-clipboard"};
gint i;
/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "entry1.c");
g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (gtk_main_quit),
    NULL);
/*Creamos una caja vertical sin espaciado homogeneo y 5 pixels entre cada
elemento*/
box = gtk_vbox_new(FALSE,5);
gtk_container_add(GTK_CONTAINER(window),box);

/*En la caja insertamos una nueva etiqueta */
widget = gtk_label_new("Presione el botón y/o \nescriba en la caja de texto");
gtk_box_pack_start_defaults(GTK_BOX(box),widget);

/*Ahora instanciamos una caja de texto y la insertamos en la ventana*/
/* Note que estamos reusando el puntero "widget"*/
widget = gtk_entry_new();
gtk_box_pack_start_defaults(GTK_BOX(box),widget);
/*Ahora conectaremos las 5 diferentes señales de GtkEntry*/
for (i=0; i<5; i++)
g_signal_connect(G_OBJECT(widget),signals[i],
```

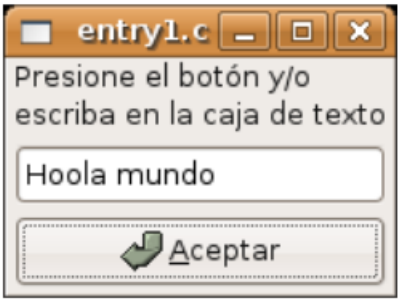
(continué en la próxima página)

(proviene de la página anterior)

```
G_CALLBACK (callback),GINT_TO_POINTER(i));
/*Atencion: ahora hemos enviado un número entero
como un parametro extra (en lugar de NULL)*/


/*Tambien se añade un boton con contenido predeterminado*/

widget = gtk_button_new_from_stock(GTK_STOCK_OK);
gtk_box_pack_start_defaults(GTK_BOX(box),widget);

/*Ahora conectamos la señal "clicked" a la funcion retrollamada*/
g_signal_connect(G_OBJECT(widget),"clicked",
G_CALLBACK (callback),GINT_TO_POINTER(i));
gtk_widget_show_all (window);


gtk_main ();

return 0;
}
```

La estructura de la aplicación es una ventana con una etiqueta, una caja de texto y un botón (obviando la caja vertical donde se empacaron todos los widgets).

Preste atención al arreglo de cadenas signals. Cuando llega el momento de conectar todas las señales de GtkEntry, se hace mediante un ciclo (for ...) enviando como datos extras el índice del ciclo. Usamos la macro de conversión de tipos GINT_TO_POINTER() para empacar el índice en el puntero (Ver la sección 2.2.6). También se hace lo mismo cuando se conecta la señal de GtkButton.



Cuando ambos widgets generen alguna señal esta será atendida en la función retrollamada callback(). El primer parámetro de esta función no nos sirve de mucho en esta ocasión. El segundo parámetro es de más utilidad pues ahí viene empacado un número que indica quien generó la señal y qué señal fue. Un par de lineas de condiciones nos permitirán imprimir esa información a la consola.

El siguiente ejemplo mostrará como usar GtkEntry para crear un diálogo de autenticación de usuarios.

(Listado de Programa 3.10.2)

```
/*****************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:      entry2.c
*  Descripcion:            Autenticación de usuarios
*  Widgets usados:         GtkEntry, GtkTable, GtkLabel, GtkWindow
*  Comentarios:            Se mostrará como usar GtkEntry para autenticar
```

(continué en la próxima página)

(proviene de la página anterior)

```
*                          un usuario.
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Autor: Noe Misael Nieto Arroyo tzicatl@gmail.com
*
*************************************************************************/
#include <gtk/gtk.h>
#include <glib.h>
gboolean ignorar (GtkWidget *window, GdkEvent *event, gpointer *data){
/*Si regresamos TRUE, entonces GTK considera que este evento ya no
debe propagarse, que es lo que deseamos*/
return TRUE;
}
void auth_cb (GtkEntry *entry2, GtkEntry *entry1){
const gchar *usuario="hildebrando117";
const gchar *password="6002perp";
if (g_str_equal(gtk_entry_get_text(entry1), usuario) &&
g_str_equal(gtk_entry_get_text(entry2), password))
gtk_main_quit();
}
int main( int   argc, char *argv[] )
{
GtkWidget *window;
GtkWidget *table;
GtkWidget *widget;
GtkWidget *entry1, *entry2;
/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "entry1.c");
123
g_signal_connect (G_OBJECT (window), "delete-event",
    G_CALLBACK (ignorar), NULL);
/*Creamos una tabla de tres filas, dos columnas y elementos no
homogeneos*/
table = gtk_table_new(3,2,FALSE);
gtk_container_add(GTK_CONTAINER(window),table);

/*En la tabla insertamos una nueva etiqueta */
widget = gtk_label_new("");
gtk_label_set_use_markup(GTK_LABEL(widget),TRUE);
gtk_label_set_markup(GTK_LABEL(widget),
"<span color=\"blue\">Bienvenido a la fortaleza secreta</span>\n"
"<span color=\"red\">¡¡¡ IDENTIFÍQUESE !!!</span>\n");
/*Insertamos la etiqueta en la parte superior de la tabla y usará
dos celdas*/
gtk_table_attach_defaults(GTK_TABLE(table),widget,0,2,0,1);
/* Insertamos otra etiqueta ...*/
widget = gtk_label_new("Usuario   : ");
gtk_table_attach_defaults(GTK_TABLE(table),widget,0,1,1,2);
/* ... y aun lado, una caja de texto*/
entry1 = gtk_entry_new();
gtk_table_attach_defaults(GTK_TABLE(table),entry1,1,2,1,2);
/*De igual manera, pero en el siguiente renglón*/
widget = gtk_label_new("Contraseña: ");
gtk_table_attach_defaults(GTK_TABLE(table),widget,0,1,2,3);
```

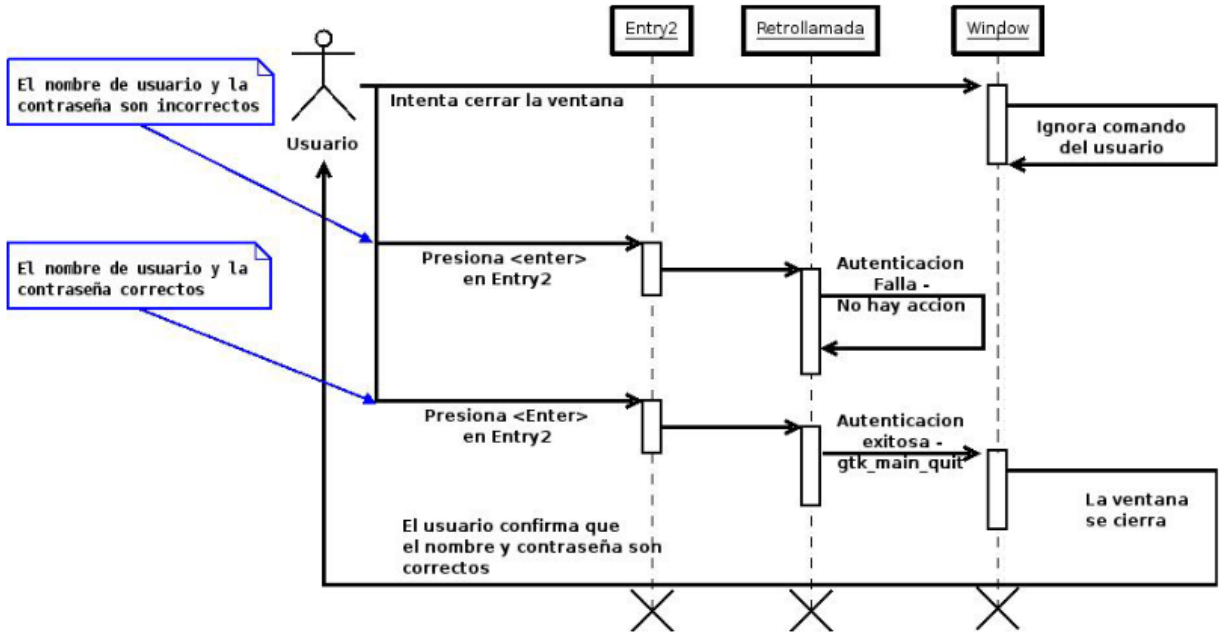
(continué en la próxima página)

(proviene de la página anterior)

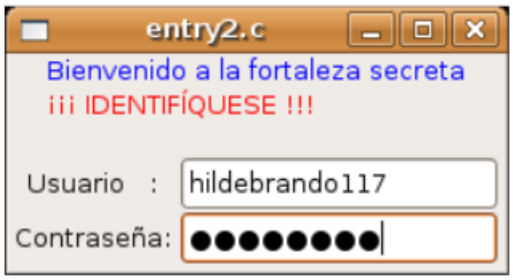
```
/* Esta es la caja de texto para la contraseña*/
entry2 = gtk_entry_new();
gtk_entry_set_visibility(GTK_ENTRY(entry2),FALSE);
/* Presionando <enter> en esta caja de texto activa  la autenticacion*/
g_signal_connect(G_OBJECT(entry2),"activate",G_CALLBACK(auth_cb),entry1);
gtk_table_attach_defaults(GTK_TABLE(table),entry2,1,2,2,3);


gtk_widget_show_all (window);


gtk_main ();

return 0;
}
```

Comencemos por la estructura de la aplicación. Una ventana contiene ahora una tabla con tres filas y dos columnas. En la primera fila, ocupando ambas columnas se encuentra una etiqueta con un mensaje de bienvenida. La primera columna de las filas restantes contienen sendas etiquetas especificando el tipo de información que se requiere. La segunda columna de las filas 2 y 3 contienen dos cajas de texto diferentes. La primera caja de texto (segunda columna) alojará el nombre de usuario mientras que la ultima caja servirá para escribir la contraseña.

Comencemos por la estructura de la aplicación. Una ventana contiene ahora una tabla con tres filas y dos columnas. En la primera fila, ocupando ambas columnas se encuentra una etiqueta con un mensaje de bienvenida. La primera columna de las filas restantes contienen sendas etiquetas especificando el tipo de información que se requiere. La segunda columna de las filas 2 y 3 contienen dos cajas de texto diferentes. La primera caja de texto (segunda columna) alojará el nombre de usuario mientras que la ultima caja servirá para escribir la contraseña



Las Figura 3.10.5 muestra el aspecto de la aplicación.


entry2.c
Bienvenido a la fortaleza secreta
¡¡¡ IDENTIFÍQUESE !!!
Usuario :
hildebrando117
Contraseña:

CAPÍTULO 11

Introducción

Glade es una aplicación que nos permite diseñar la interfaz gráfica de nuestro programa, en una forma visual. Las herramientas de desarrollo visual proveen un enorme ahorro de tiempo y trabajo al evitar tener que codificar la GUI nosotros mismos y el tiempo restante se puede aprovechar para mejorar la lógica del programa. Glade guarda el diseño de la interfaz gráfica en un archivo XML. Glade construye la aplicación a partir del modelo descrito en el archivo XML usando la librería libglade.

Nosotros podemos hacer uso de Glade para dibujar la GUI. Una vez guardada la descripción de la GUI podremos reconstruirla con muy pocas líneas de código.

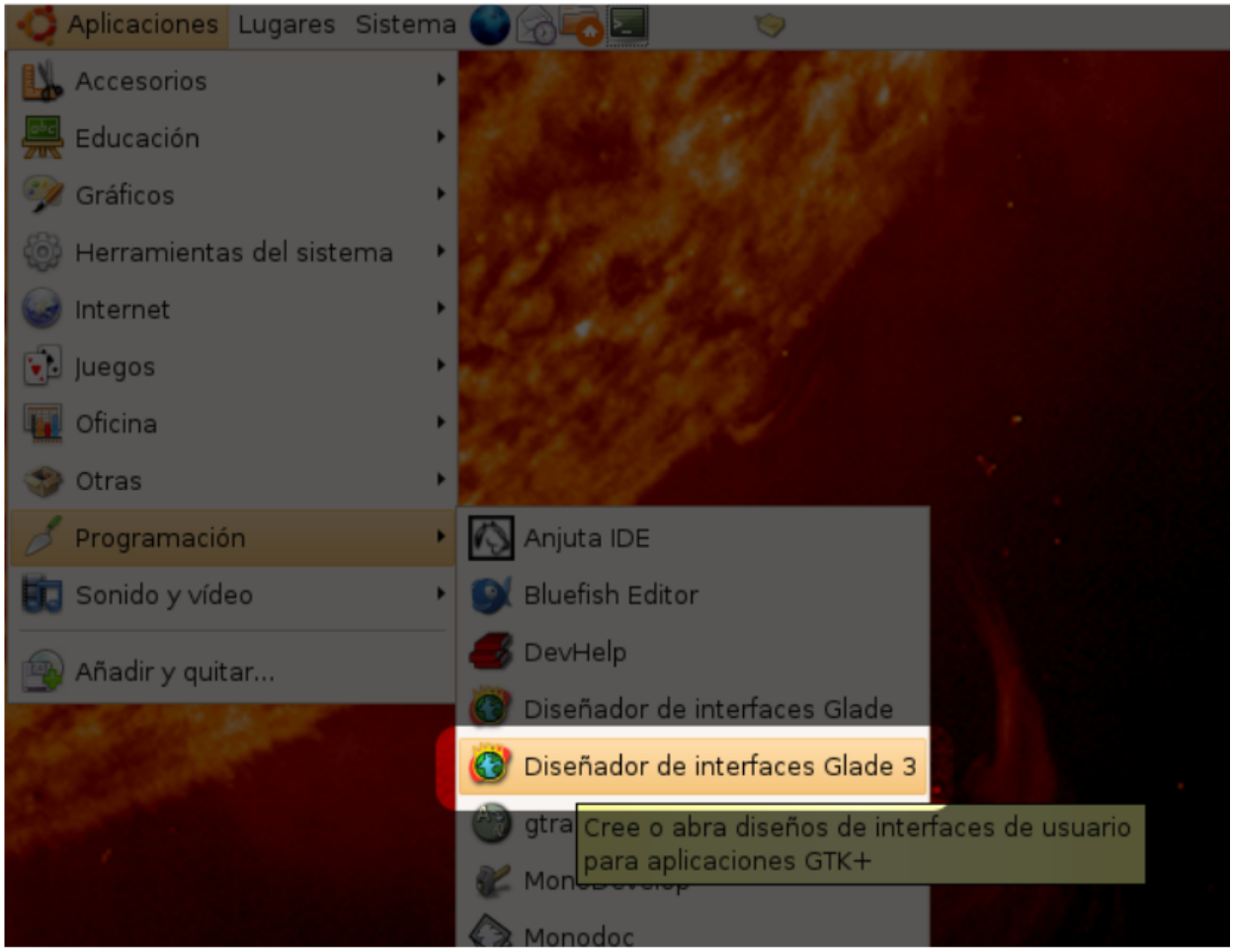
A continuación proporcionaremos una breve guía de uso de Glade para dibujar nuestras aplicaciones. Al momento de redactar este documento existían dos versiones operativas: glade2 y glade3. Glade3 es la versión de prueba que ha sido rediseñada por completo. Se han removido características obsoletas, mejorado el uso y la presentación del programa. También esta en planes la integración de Glade3 con el sistema de documentación DevHelp, el entorno integrado de desarrollo Anjuta. Sin duda, en los próximos meses se prevee una mejora significativa en los entornos de desarrollo en Linux.

Glade3 madura día a día y Glade2 va quedando relegada en el olvido, es por eso que las siguientes secciones se basarán en el Glade3 versión 3.0

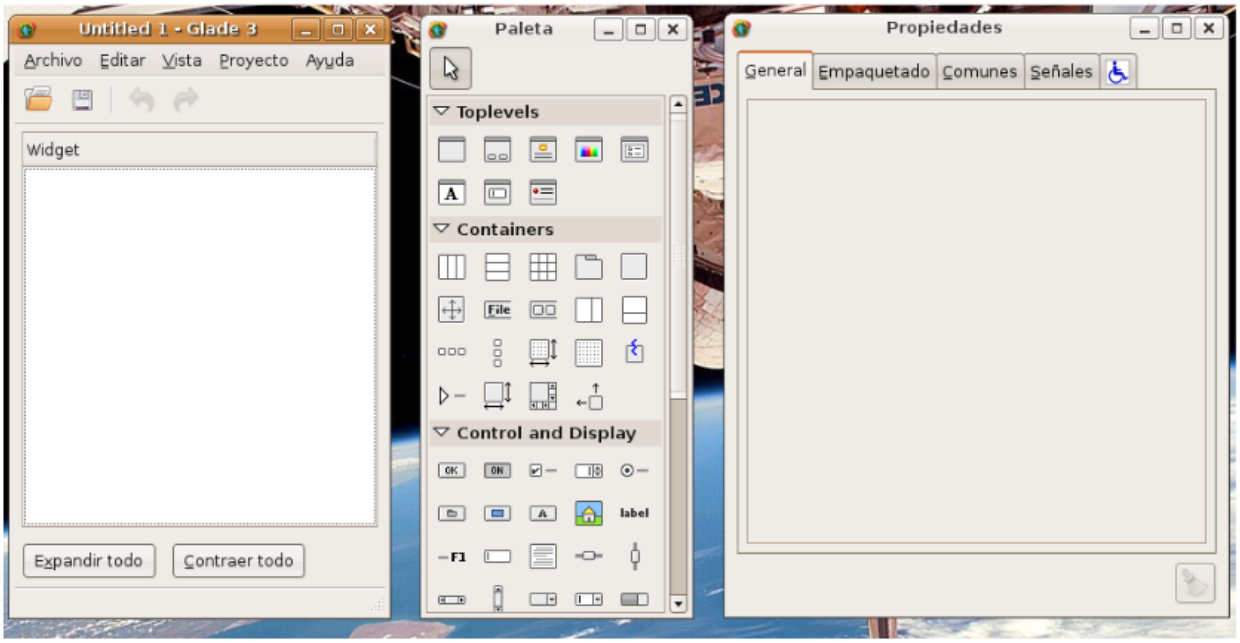
CAPÍTULO 12

Conociendo Glade

Glade se integra limpiamente al escritorio GNOME. Para iniciar la aplicación seleccione del menú **Aplicaciones**, el sub-menú **Programación** y por último **"Diseñador de interfases Glade 3"**



Al iniciar la aplicación, aparecerán tres ventanas: La ventana de proyecto de Glade, la paleta de **widgets*** y la ventana de propiedades. Vea la Figura 4.1.2.



La ventana principal esta compuesta por la barra de menús, la barra de herramientas y una sección en blanco que contendrá una representación de la estructura de la aplicación. Los últimos dos botones permitirán controlar la manera en que se visualiza la representación de la estructura de nuestra aplicación.

La paleta contiene colecciones de **widgets** agrupados en las siguientes categorias:

- Toplevels - Contiene ventanas y cuadros de dialogo.
- Containers - Contenedores de todo tipo, como cajas y tablas.
- Control and Display – **Widgets** de control y despliegue de datos como botones, etiquetas y entradas de texto.
- GTK+ Obsolete – Este grupo contiene **widgets** que han sido reemplazados por otros mas flexibles u otros que son de uso muy poco común, como GtkCurve.
- GNOME User Interface – **Widgets** diseñados para ser usados con el entorno GNOME.
- GNOME UI Obsolete – **Widgets** diseñados para usarse con el entorno GNOME pero que han sido reemplazados con otros más flexibles o ya son de uso poco común. Algunos **widgets** ya han sido implementados directamente en GTK+.
- GNOME Canvas – **Widgets** de dibujo, especialmente diseñados para aplicaciones con dibujos vectoriales o elementos gráficos interactivos.

La ventana de propiedades contiene cinco pestañas, cada una mostrando información de configuración para cada **widget**.

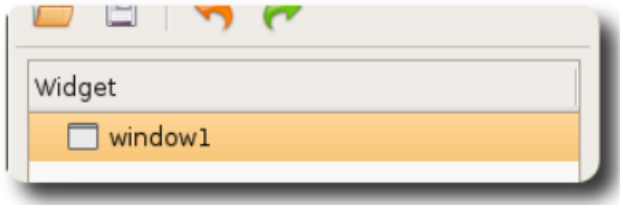
- General - En esta pestaña se ajustan propiedades generales de un **widget** como su nombre o el ancho de borde.
- Empaquetado - En esta pestaña se encuentran los ajustes para aquellos **widgets** que se encuentren insertos en en contenedor (Consulte el capitulo 3.5 para una referencia completa del sistema de empaquetado de GTK+).
- Comunes - Esta pestaña muestra una colección de propiedades ajustables de cualquier **widget**. Estas operaciones tienen como contra parte algún método de la clase GtkWidget.
- Señales - Esta es una lista, agrupada por clase, de todas las señales que puede atrapar un *widget*. A la izquierda se muestra el nombre de la señal y a la derecha el nombre de la función retrollamada.
- Tecnología de asistencia - GTK+ toma en cuenta a las personas con capacidades diferentes. En esta pestaña permite añadir soporte de tecnología de asistencia a una aplicación de GTK+.

Una vez que hemos revisado de manera general Glade3 podemos comenzar un proyecto.

12.1 Trabajando con un proyecto

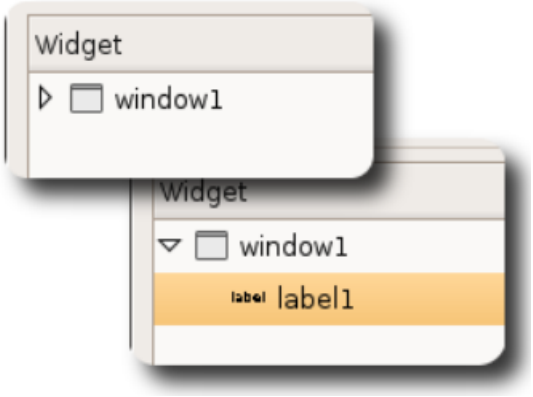
A manera de comparación, recrearemos el ejemplo visto en la sección 3.8.

1. En el grupo Toplevels de la paleta, presione en el primer icono. Una ventana aparecerá con un fondo gris: Glade3 acaba de crear una instancia de GtkWindow para usted. También note que en la ventana principal de Glade3 ha aparecido la representación de la instancia de GtkWindow que acaba de ser creada.



2. El siguiente paso es insertar una etiqueta. El icono se encuentra en el grupo «Control and Display» de la paleta. Haga **click** en el icono y visualice la ventana que se acaba de crear. Cuando el cursor del ratón circula por el área gris de la ventana, el puntero del ratón cambia por una cruz. Haga **click** en cualquier área gris de la ventana. El área gris desaparecerá y en su lugar aparecerá una instancia de GtkLabel (ver Figura 4.2.3).

3. Observe que la ventana principal de Glade3 ha actualizado la representación de la aplicación que estamos diseñando. Se ha vuelto un árbol de donde se desprenden los widgets hijos de cada ventana de nivel superior (ver Figura 4.2.4).

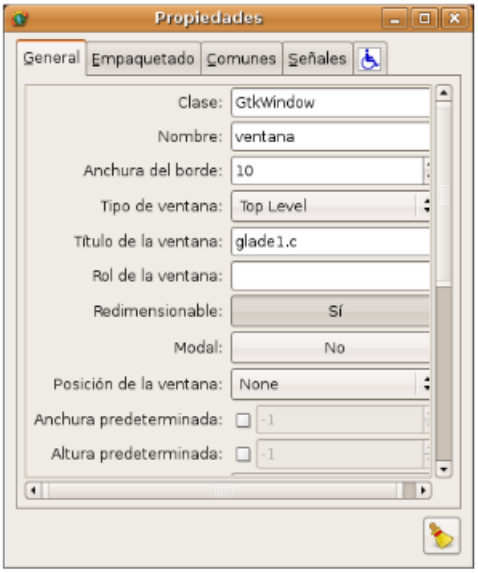


4. Haciendo **click** en cualquier elemento de la representación provoca que ese widget obtenga el foco; cuando un widget obtiene el foco es posible configurarlo usando la ventana de propiedades.

Haga **click** en el elemento que representa la instancia de GtkWindow.

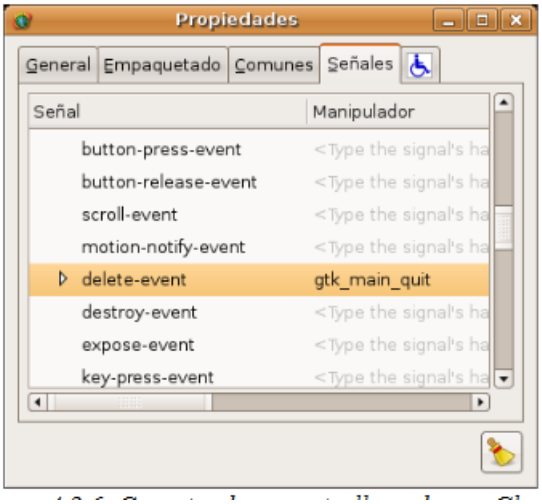
5. Centre ahora su atención en la ventana de propiedades (ver Figura 4.2.5); ahí puede cambiar el nombre de la instancia de GtkWindow, que en este ejemplo se llamará ventana.

6. Cambie el título de la ventana a «glade1.c». Esto equivale a llamar al método gtk_window_set_title().

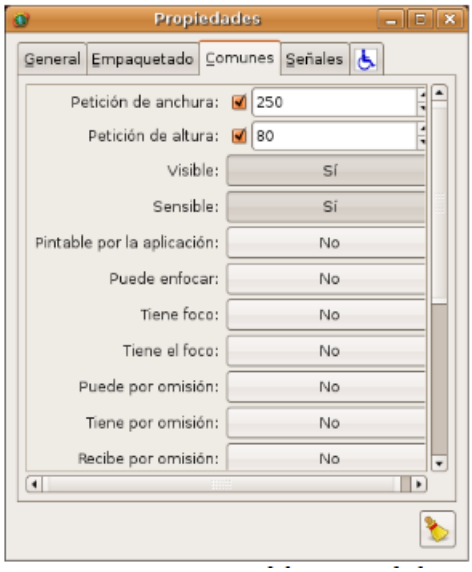


7. Seleccione la pestaña «Señales» y busque la señal «delete-event» dentro del grupo de señales pertenecientes a GtkWidget.

8. Haga doble **click** en la columna de la derecha y escriba «gtk_main_quit». Esto conectará la señal «delete-event» de la instancia de GtkWindow con la retrollamada gtk_main_quit().



9. En la pestaña Comunes, de la ventana de propiedades establezca la petición de anchura en 260 y la petición de altura en 60. Esto equivale a usar el método gtk_widget_set_size_request().



10. Dentro de la misma pestaña asegúrese que la propiedad visible este ajustada a Si. Esta propiedad instruye a libglade a que llame el método gtk_widget_show() para el **widget** que se esta configurando.

11. Seleccione ahora la etiqueta y cambie el texto a «INSTITUTO TECNOLÓGICO DE PUEBLA». Ajuste la propiedad visible a Si.

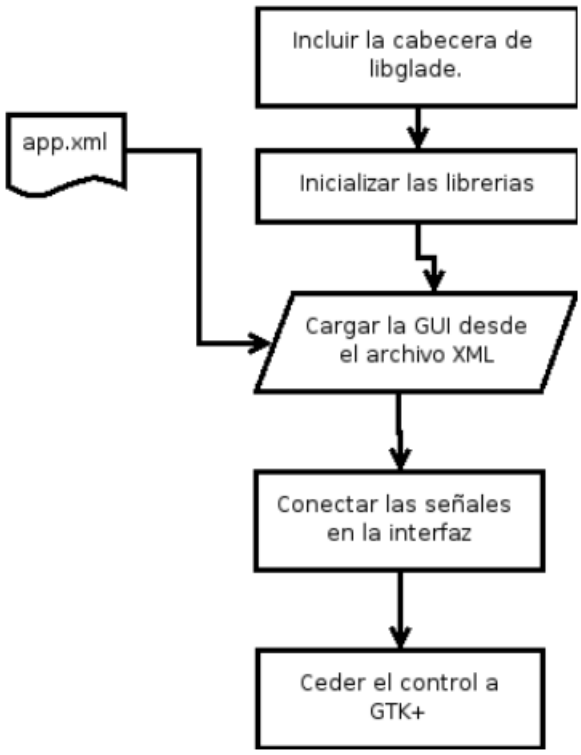
El ejemplo está preparado ahora. Guarde el ejemplo con el nombre «glade1.glade» y prosiga con la siguiente sección para aprender el uso de la librería libglade.

12.2 Introducción a libglade

Libglade es la librería de soporte de Glade3. Permite construir y modificar la estructura de una GUI mediante un archivo XML Esta forma de programación de interfases gráficas de usuario permite la separación del código del programa del código de la interfaz gráfica. El tiempo de ejecución de la librería ha sido optimizado para ser extremadamente rápida. Una vez que la aplicación ha sido construida, esta funcionará a su máxima velocidad pues libglade esta implementada en el lenguaje C.

12.3 Proceso de creación de una aplicación con libglade

El uso de libglade en nuestras aplicaciones permite la separación de la parte lógica de la aplicación de la parte gráfica. Lo anterior quiere decir que nosotros dibujaremos la aplicación a nuestro gusto usando Glade3 y posteriormente escribiremos la lógica de comportamiento de la aplicación en cualquier lenguaje como Python o C. Una vez guardada la descripción de la GUI en un archivo XML el proceso para reconstruir la GUI se muestra a continuación.



Del manual de referencia de libglade copiamos un programa básico de libglade en C.

(Listado de Programa 4.3.1)

```
/*************************************************************************
* Programacion de interfases graficas de usuario con GTK
*
*  Nombre de archivo:
*  Descripcion:            Ejemplo de una aplicación básica con libglade.
*  Widgets usados:
*  Comentarios:            Ejemplo tomado del manual de referencia de
*                          libglade
*
*  TESIS PROFESIONAL       INSTITUTO TECNOLOGICO DE PUEBLA
*                          INGENIERIA ELECTRONICA
*  Fuente: Manual de referencia de libglade versión 2.5.0
*
*************************************************************************/
#include <gtk/gtk.h>
/*Incluir la cabecera de libglade*/
#include <glade/glade.h>
void func_retrollamada(GtkWidget *widget, gpointer datos_extra) {
/* Hacer algo util aqui */
}

Figura 4.3.1: Proceso de creación de una GUI con
libglade.
int main(int argc, char *argv[]) {
    /* Este es un puntero a un objeto de tipo GladeXML*/
    GladeXML *xml;
    /*Inicializar las librerias*/
```

(continué en la próxima página)

(proviene de la página anterior)

```
    gtk_init(&argc, &argv);
    /* cargar la GUI desde el archivo XML */
    xml = glade_xml_new("app.xml", NULL, NULL);
    /* Conectar las señales en la GUI */
    glade_xml_signal_autoconnect(xml);
    /* Ceder  el control a GTK+*/
    gtk_main();
    return 0;
}
```

La notable simplicidad de libglade es donde radica su capacidad. Si compiláramos el ejemplo anterior tendríamos la posibilidad de construir aplicaciones muy sencillas como una ventana con un botón hasta aplicaciones complejas como un editor de texto. Lo único que tendríamos que hacer es intercambiar la descripción XML de la GUI y liblgade hará el trabajo por nosotros. Aunque libglade nos libera de las tareas tediosas de crear GUIs usando el API de GTK+, aún así debemos conocer la manera de cómo interactuar con la interfaz gráfica generada: tenemos que crear retrollamadas e interactuar con el usuario.

12.4 Constructor de clase

La librería libglade utiliza el estilo de programación de GTK+ y GNOME, así que podremos seguir aplicando la metodología aprendida hasta ahora.

La construcción e interacción con una GUI creada con libglade se hace mediante el objeto GladeXML. Ya que libglade se implementa usando el modelo de GTK+ y Glib, podemos esperar un funcionamiento similar a los **widgets** de GTK+.

El objeto GladeXML representa una instancia de la GUI creada a partir de una descripción en formato XML. Cuando se crea una instancia de la clase GladeXML, la descripción se lee desde un archivo y se crea la GUI.

Una vez instanciada la clase GaldeXML, esta provee una serie de útiles métodos para acceder a los **widgets** de la GUI por medio de una referencia o nombre dentro de la descripción XML. La clase GladeXML también provee métodos para conectar cualquier retrollamada que haya sido asociada con alguna señal o evento dentro de la descripción XML.

Por último, libglade provee métodos que buscan nombres de manejadores de señal en la tabla de símbolos de la aplicación y automáticamente conectar tantas retrollamadas como pueda.

```
GladeXML* glade_xml_new (const char *fname,
const char *root,
const char *domain);
```

Descripción: Crea una nueva instancia del objeto GladeXML a partir de un archivo de descripción en formato XML. Opcionalmente se puede comenzar a construir la interfaz a partir de un widget. Es útil si se desea construir solamente una barra de menú y no toda la aplicación en la que esta contenida. La descripción en XML se cachea para acelerar futuras operaciones.

Parámetros:

- **fname** : Nombre del archivo que contiene la descripción XML de la GUI.
- **root** : El nodo desde donde se desea comenzar a construir. NULL si desea construir.
- **domain** : Dominio de transición XML.

Valor de retorno: Una nueva instancia de la clase GladeXML que describe una interfaz gráfica de usuario. Regresa NULL si la operación ha fallado.

12.5 Métodos de clase

```
GtkWidget* glade_xml_get_widget (GladeXML *self,
const char *name);
```

Descripción: Regresa el puntero del widget con el nombre especificado. Esta función permite el acceso a componentes individuales de una GUI después de que ha sido construida.

Parámetros:

- **self** : Una instancia de GladeXML.
- **name** : El nombre del widget.

Valor de retorno: El puntero del widget cuyo nombre coincida con el especificado. Regresa NULL si el widget no existe.

```
void glade_xml_signal_connect (GladeXML *self,
const char *handlername,
GCallback func);
```

Descripción: Dentro de la descripción XML de una GUI, las funciones retrollamada se especifican usando el nombre de la función y no un puntero a ella. Esta función permite conectar una función a todas aquellas señales que hayan especificado esta función como función retrollamada.

Parámetros:

- **self**: Una instancia de GladeXML.
- **handlername**: El nombre de la función retrollamada.
- **func**: Un puntero a la función retrollamada. Use la macro G_CALLBACK() para moldear el puntero de la función al tipo adecuado.

```
void glade_xml_signal_connect_data (GladeXML *self,
const char *handlername,
GCallback func,
gpointer user_data);
```

Descripción: La diferencia entre este método y glade_xml_signal_connect() es que esta permite pasar el parámetro extra que se acostumbra en g_signal_connect().

Parámetros:

- **self** : Una instancia de GladeXML.
- **handlername** : El nombre de la función retrollamada.
- **handlername** : Un puntero a la función retrollamada. Use la macro G_CALLBACK() para moldear el puntero de la función al tipo adecuado.
- **user_data** : Datos extra que se pasarán a la función retrollamada.

```
void glade_xml_signal_autoconnect (GladeXML *self);
```

Descripción: Este método permite conectar automáticamente todas las retrollamadas que hayan sido descritas en la descripción XML de la GUI.

Parámetros:

- **self** : Una instancia de GladeXML.

Nota: Nota: glade_xml_signal_autoconnect() usa la tabla de símbolos de la aplicación para tratar de encontrar las funciones retrollamadas. Si desea auto conectar retrollamadas que se hayan declarado y definido en el ejecutable principal de la aplicación (main.c para muchos casos), será necesario pasar alguna directiva al enlazador de la aplicación para que también exporte los símbolos de la aplicación principal. Para un entorno de desarrollo GNU se debe agregar la directiva -export-dynamic a la orden de compilación. Los Makefiles de los ejemplos incluidos en este documento ya están preparados para compilar adecuadamente.

CAPÍTULO 13

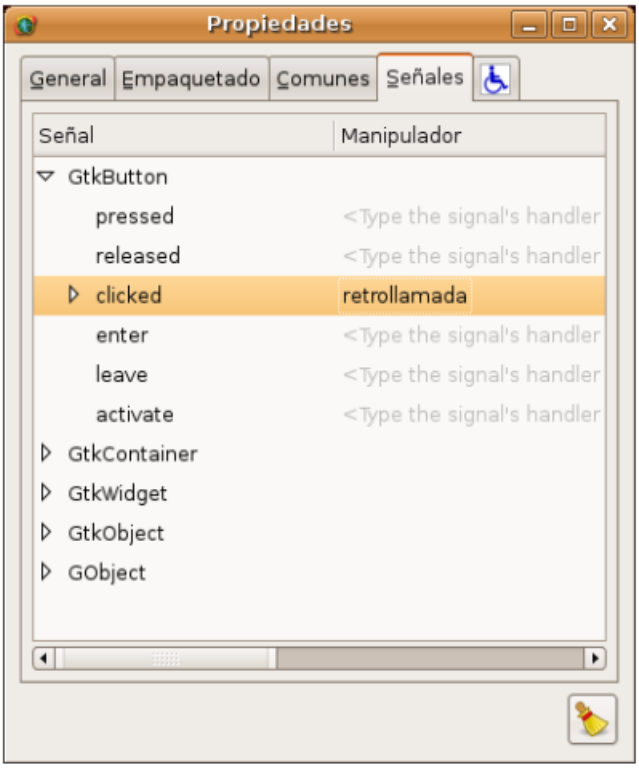
Ejemplos.

13.1 Ejemplo 1 – Ciclo de vida de una aplicación con libglade

En este primer ejemplo mostraremos el ciclo de vida de una aplicación con libglade. Se construirán dos interfases diferentes y mostraremos que con sólo cambiar el archivo XML podremos cambiar completamente la GUI sin cambiar una sola línea de código. La primera GUI se retomará del ejemplo que se construyó en los capítulos 4.3 y 4.3.1: Una ventana con una etiqueta adentro.

La segunda GUI será una ventana con un botón adentro. Con respecto a esta última debemos de asegurarnos que:

- La instancia de la ventana deberá conectar la señal «delete-event» con el método delete_event() de GTK+.
- La instancia del botón deberá conectar la señal **clicked** con el método retrollamada() que proveerá nuestra aplicación (ver Figura 4.6.1).
- Que tanto la ventana como el botón tengan activada la propiedad visible (en la pestaña Comunes de la ventana de propiedades).
- El botón deberá tener un ancho de 260 **pixeles** y una altura de 60 **pixeles** (en la pestaña Comunes de la ventana de propiedades).
- La ventana deberá un ancho de borde de 10 **pixeles** (en la pestaña Generales de la ventana de propiedades).
- Guarde el archivo XML como glade2.xml.



El código fuente de la aplicación estará basado en el mostrado en los capítulos 4.3 y 4.3.1.

(Listado de Programa 4.6.1)

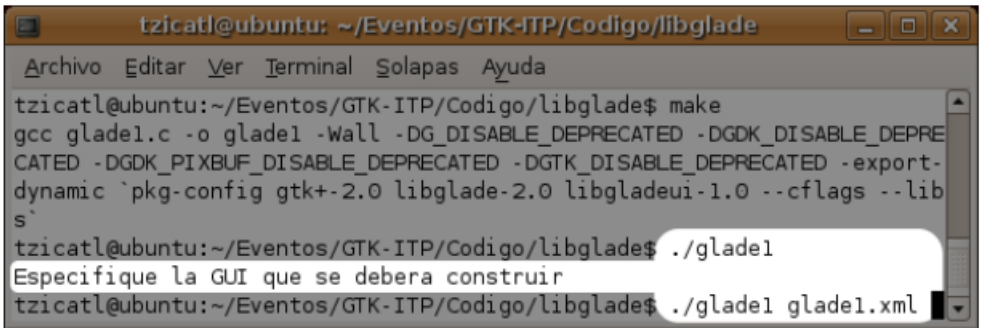
```
/*****************************************************************************
*   Programacion de interfases graficas de usuario con GTK
*
* Nombre de archivo:        glade1.c
* Descripcion:              Ejemplo del ciclo de vida de una aplicación
*                           con libglade.
* Widgets usados:           GtkWidget
* Objetos usados:           GladeXML
* Comentarios:              Ejemplo basado del manual de referencia de
*                           libglade
*
* TESIS PROFESIONAL         INSTITUTO TECNOLOGICO DE PUEBLA
*                           INGENIERIA ELECTRONICA
* Fuente: Manual de referencia de libglade versión 2.5.0
*
*****************************************************************************/
#include <gtk/gtk.h>
/*Incluir la cabecera de libglade*/
#include <glade/glade.h>
/*Incluir stdlib para usar la función exit()*/
#include <stdlib.h>
void retrollamada(GtkWidget *widget, gpointer datos_extra) {
g_print("Funcion retrollamada\n");
}
int main(int argc, char *argv[]) {
    GladeXML *xml;
//GtkWidget *ventana;
```

(continué en la próxima página)

(proviene de la página anterior)

```
/* Inicializar la libreria GTK */
gtk_init (&argc, &argv);
if (!argv[1]){
g_print("Especifique la GUI que se debera construir\n");
exit(1);
}
g_print("Construyendo GUI del archivo %s\n",argv[1]);
    /* cargar la GUI desde el archivo XML */
    xml = glade_xml_new(argv[1], NULL, NULL);
//ventana =  glade_xml_get_widget(xml,"ventana");
    /* Conectar las señales en la GUI */
    glade_xml_signal_autoconnect(xml);
//gtk_widget_show_all(ventana);
    /* Ceder el control a GTK+*/
    gtk_main();
    return 0;
}
```

Este ejemplo, aunque es una GUI, debe de llamarse desde la línea de comandos y requiere de un parámetro para funcionar: el nombre del archivo XML que contiene la descripción de la GUI. En este caso puede ser glade1.xml o glade2.xml. En caso de que no se le suministre ningún nombre de archivo el programa imprimirá un mensaje informativo y terminará inmediatamente.

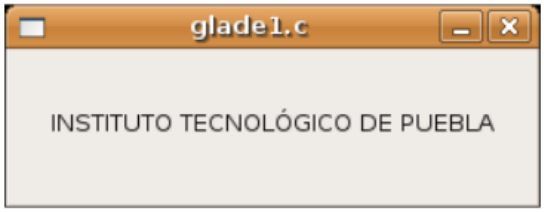


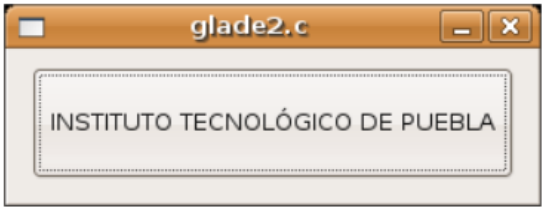
La GUI que se muestra en la Figura 4.6.3 ha sido llamada mediante el siguiente comando:

```
./glade1 glade1.xml
```

La GUI que se muestra en la Figura 4.6.4 ha sido llamada mediante este otro comando.

```
./glade1 glade1.xml
```





Ambas ventanas se cierran automáticamente pulsando el botón de cerrar. La segunda aplicación imprime un mensaje en la consola cuando se presiona el botón.

Se ha demostrado que con el mismo código se pueden construir dos interfases diferentes usando libglade. Se ha demostrado, también, como conectar señales automáticamente usando libglade. Por último se ha demostrado el ciclo de vida básico de una aplicación que usa libglade

CAPÍTULO 14

Bibliografía

[1] «X Window System». .Wikipedia, The Free Encyclopedia. Disponible al 1 Enero 2006 en la URL http://en.wikipedia.org/wiki/X_Window_System [2] «Windows GDI». .Microsoft Corporation. Disponible al 1 de Enero de 2006 en la URL http://msdn.microsoft.com/library/default.asp?url= [3] «Quartz Extreme, Faster graphics». . Apple Computer, Inc.. Disponible al 1 de Enero de 2006 en la URL http://www.apple.com/macosx/features/quartzextreme [4] «The Pango connection, (01 Mar 2001)». Tony Graham.IBM Corporation. Disponible al 1 de Enero de 2006 en la URL http://www-128.ibm.com/developerworks/library/l-u- [5] Brian Kernighan, Dennis Ritchie, The C Programming Languaje (Second Edition), 1988 [6]Noe Nieto, Christian Alarcon, Sotero I. Fuentes, Micro Laboratorio Virtual, 2004 [7] «Linked List Basics». Nick Parlante.Stanford CS Education Library. Disponible al en la URL [8] «GNOME Programming Guidelines». Federico Mena Quintero, Miguel de Icaza. Morten Welinder.. Disponible al 2 de Febrero de 2006 en la URL http://developer.gnome.org/doc/guides/programming-guidelines/book1.html

CAPÍTULO 15

ANEXO 4.6.1.1 : El COMPILADOR GCC

15.1 Introducción

Esta es una revisión rápida de los comandos necesarios para hacer funcionar el compilador de GNU C, el cual es de libre distribución y prácticamente se le puede encontrar en cualquier maquina Linux. GCC significa "GNU Compiler Collection". Esto significa que la suite de compilación GCC no solamente ofrece soporte para el lenguaje C, si no que además contiene soporte para Java, C++, Ada, Objective C y Fortran.

15.2 Sintaxis

El compilador GNU C es una herramienta de línea de comandos y puede aceptar gran cantidad de instrucciones que le indican la manera de comportarse. La lista de todas las opciones, modificadores y conmutadores es muy extensa, sin embargo todas obedecen simples reglas.

La sinopsis de uso de gcc es:

```
gcc [ opciones | archivos ] ...
```

- Las opciones generalmente van precedidas de un guión (estilo UNIX), o un doble guión (estilo GNU). Las opciones estilo UNIX pueden agruparse y constar de varias letras; Las opciones estilo GNU deberán indicarse por separado, en forma de una palabra completa.
- Algunas opciones requerirán de algún parámetro como un número, un directorio,un archivo, una cadena o una frase.
- La orden gcc se puede utilizar indistintamente, no importando el lenguaje usado: Apegándose a unas sencillas reglas, el compilador será capaz de determinar la acción a ejecutar dependiendo de la extensión de los archivos.
- Finalmente, todo lo que no se reconozca como parámetro u opción, será tratado como archivo y, dependiendo de su extensión, éste será procesado como código fuente o código objeto.

Tabla 1: Extensiones de archivo y su significado para GCC.

Extensión de Archivo	Descripción
.c	Código fuente en C.
.C .cc .cpp .c++ .cp .cxx	Código fuente en C++. Se recomienda usar extensión .cpp
.m	Código fuente en Objective C. Un programa hecho en Objective C debe ser ligado a la librería libobjc.a para que pueda funcionar correctamente.
.i	C preprocesado
.ii	C++ preprocesad
.s	Código fuente en lenguaje ensamblador
.o	Código objeto.
.h	Archivo para preprocesador (encabezados), no suele figurar en la línea de comandos de GCC.
OTRO	Cualquier otro parámetro que no sea archivo de los arriba expuesto o un parámetro válido, será tomado como si fuera un archivo objeto

Ejemplos Compila el programa hola.c y genera un archivo ejecutable que se llame a.out:

```
gcc hola.c
```

Compila el programa hola.c y genera un archivo ejecutable que se llame hola:

```
gcc -o hola hola.c
```

Compila el programa hola.cpp y genera un archivo ejecutable que se llame hola:

```
gcc -o hola hola.cpp
```

Los siguientes tres ejemplos compilan el programa hola.c y hola.cpp En este caso ninguno de los tres ejemplos generarán un ejecutable, si no solo el código objeto (hola.o):

```
gcc -c hola.c
gcc -c hola.c -o hola.o
gcc -c hola.cpp
```

Generar el ejecutable hola en el directorio bin, dentro del directorio propio del usuario:

```
gcc -o ~/bin/hola hola.c
```

Compilar el programa quetal.c, pero ahora indicarle a GCC dónde buscar las bibliotecas que usa quetal.c. Usaremos la opción -L; podemos repetirla cuantas veces necesitemos para indicar todos los directorios de librerías que necesitemos. El orden de búsqueda de las librerías es el mismo orden en el que se especificaron en la linea de comandos.

```
gcc -L /lib -L/usr/lib -L~/Librerias -L/opt/ProgramaX/lib quetal.cpp
```

Ahora usaremos -I para indicarle a GCC dónde buscar archivos de cabecera (.h):

```
gcc -I/usr/include/gtk-2.0 -I/opt/ProgramaX/include comoves.c
```

Aunque pueda parecer que GCC es un gran programa que lo hace todo, en realidad es una colección de herramientas pequeñas que hacen una cosa a la vez.

GCC tiene un compilador para cada lenguaje, así si GCC detecta que se desea procesar un programa escrito en C, llamará al compilador de C (gcc), pero si es un programa escrito en C++, llamará al compilador de C++ (g++),

y así de manera consecutiva. El proceso de compilación comprende fases bien definidas. Cuando GCC es invocado, generalmente hace el 1) pre-procesamiento, 2)compilación, 3)ensamblaje y ligado y por último entrega como resultado final un archivo ejecutable.

Una miríada de opciones nos permiten tomar control de cada paso del proceso, por ejemplo, el interruptor -c omite proceso de ligado y entrega solamente el código objeto del programa. Algunas opciones se pasan directamente a alguna de las fases de construcción, algunas otras controlan el pre-procesamiento, otras controlan al ligador o al ensamblador y otras controlan al mismo compilador.

La orden gcc acepta opciones y nombres de archivos como operando. Muchas opciones son operandos de varias letras, así, no es lo mismo especificar la opción -ab que las opciones -a -b. Salvo los anteriores casos, la mayoría de las veces no importa el orden en que se dan los argumentos.

Como ya hemos dicho el proceso de compilación tiene cuatro fases: pre-procesamiento, compilación, ensamblaje y ligado. Todas ellas siempre en el mismo orden. Las primeras tres de ellas trabajan con un solo archivo de código fuente y terminan produciendo un archivo objeto.

El proceso de ligado consiste en combinar a todos los archivos objeto generados, (incluyendo a los que se le han pasado por la linea de comando), resuelve las referencias entre ellos y entrega un archivo ejecutable (código de maquina).

Las opciones más comunes del compilador GCC son:

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -c
```

Compila, ensambla, pero no liga. Como resultado obtenemos un archivo objeto por cada archivo de código fuente. Generalmente se les asigna una extensión .o

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -S
```

Compila pero no ensambla. Se entrega un archivo en ensamblador por cada archivo de código fuente. A los resultados de la salida se les asigna la extensión: .S

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    '-E'
```

Sólo realiza la etapa de preproceso. La salida estará en formato del código fuente, procesado con respectivo compilador.

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -o archivo
```

Se puede especificar el nombre del archivo de salida que generará el compilador. Esto aplica a cualquier forma de salida que se le esté instruyendo al compilador, ya sea, sólo ensamblar, compilar, ligar o todos.

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -Iruta
```

Especifica la ruta hacia el directorio donde se encuentran los archivos marcados para incluir el programa fuente. No debe llevar espacio entre la I y la ruta, así: -I/usr/include.

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -L
```

Especifica una ruta hacia el directorio donde se encuentran los archivos de biblioteca con el código objeto de las funciones que se usan en el programa. No lleva espacio entre la L y la ruta, así: -L/usr/lib

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -Wall
```

Muestra todos los mensajes del compilador(advertencias y errores).

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -g
```

Incluirá en el programa generado, la información necesaria para poder rastrear posibles errores en un programa usando un depurador, tal como GDB (GNU Debugger).

Error in «code-block» directive: 1 argument(s) required, 0 supplied.

```
.. code-block::

    -v
```

Muestra los comandos ejecutados en cada etapa de compilación , así como la versión del compilador. Es un informe muy detallado.

15.3 Etapas de compilación

El proceso de compilación involucra cuatro etapas sucesivas: Pre-procesamiento, compilación, ensamblaje y enlazado. El proceso de conversión de creación de un programa a partir del código fuente exige la ejecución de estas cuatro etapas en forma sucesiva. Los comandos gcc y g++ son capaces de realizar todo el proceso de una sola vez.

15.4 Preprocesamiento

En esta etapa se interpretan las directivas del preprocesador. Entre otras cosas las constantes y macros definidas con #define son sustituidas por su valor en todos los lugares donde aparece su nombre. Usemos como ejemplo este sencillo programa en C.

```
/* Circulo.c: calcula el área de un círculo.
Ejemplo que muestra las etapas de compilación de GCC
*/
#include <stdio.h>
# define PI 3.14159265358979323846264338327950291L /* pi */
main()
{
float area, radio;
radio = 10;
area = PI * (radio * radio);
printf("Circulo.\n");
printf("%s%f\n\n", "Area de circulo radio 10: ", area);
return(0);
}
```

El preprocesado puede pedirse llamando directamente al preprocesador (con la orden cpp), o haciéndolo mediante GCC (con la orden gcc). Los siguientes dos comandos producen una archivo de salida idéntico.

```
$ cpp circulo.c > circulo.i
$ gcc -E circulo.c > circulo.i
```

Si examinamos circulo.pp (observe la extensión y compare con la tabla), podremos observar que la constante PI ha sido substituida por su valor en todos los lugar donde se hacia referencia a ella.

15.5 Compilación

El proceso de compilación transforma el código fuente preprocesado en lenguaje ensamblador, propio para el procesador en el que será usado el programa (típicamente nuestra propia maquina). Por ejemplo..

```
$ gcc -S circulo.c
```

... realiza las primeras dos etapas y crea el archivo circulo.s, si lo examinamos encontraremos código en lenguaje ensamblador.

15.6 Ensamblado

El ensamblaje de nuestra aplicación es el penúltimo paso, transforma el archivo circulo.s o cualquier otro código en ensamblador en lenguaje binario ejecutable por la máquina. El ensamblador de GCC es as, he aquí un ejemplo:

```
$ as -o circulo.o circulo.s
```

as creará el archivo en código de máquina o código objeto (circulo.o) a partir de un código en ensamblador (circulo.s). Es muy infrecuente utilizar ensamblado, preprocesado o compilación por separado, lo usual es realizar todas las etapas anteriores hasta obtener el código objeto:

```
$ gcc -c circulo.c
```

El anterior comando producirá el código objeto y lo guardará en el archivo (circulo.o). A diferencia de las etapas anteriores, en programas muy extensos, donde el programa final se debe partir en diferentes módulos, la práctica común es usar gcc o g++ con la opción -c para compilar cada archivo de código fuente por separado y luego unirlos o enlazarlos para formar el programa final.

15.7 Enlazado

Las funciones de C/C++ incluidas en cualquier programa(printf, por ejemplo), se encuentran ya compiladas y ensambladas en las bibliotecas existentes en el sistema. Es necesario incorporar de algún modo el código binario de estas funciones a nuestro programa ejecutable. En esto consiste la etapa de enlace, donde se reúnen uno o más códigos objeto con el código existente en las bibliotecas del sistema. El enlazador de GCC es la orden ld. A continuación un ejemplo:

```
$ ld -o circulo circulo.o -lc
ld: warning: cannot find entry symbol _start; defaulting to 08048184
El error anterior se debe a la falta de referencias, pues el enlazador no sabe a␣
↪dónde debe buscar
las funciones que el módulo circulo.c esta usando. Para que esto funcione y␣
↪obtengamos un
ejecutable debería ejecutarse una orden como la que sigue:
$ ld -o circulo /usr/lib/gcc-lib/i386-linux/2.95.2/collect2 -m
elf_i386 -dynamic-linker /lib/ld-linux.so.2 -o circulo
/usr/lib/crt1.o /usr/lib/crti.o /usr/lib/gcc-lib/i386-
linux/2.95.2/crtbegin.o -L/usr/lib/gcc-lib/i386-linux/2.95.2
circulo.o -lgcc -lc -lgcc /usr/lib/gcc-lib/i386-linux/2.95.2/crtend.o
/usr/lib/crtn.o
```

Esto es incómodo, es por eso que GCC puede ahorrarnos mucho trabajo si le pasamos el nombre del código objeto (o los nombres) que queremos convertir en ejecutable:

```
$ gcc -o circulo circulo.o
```

Creará el programa ejecutable de una manera sencilla y en un sólo paso. En un programa con un sólo archivo fuente, todo el proceso puede hacerse de una vez por todas:

```
$ gcc -o circulo circulo.c
```

A manera de aprendizaje podríamos activar el interruptor -v de GCC que nos mostrará aspectos del proceso de compilación que normalmente quedan ocultos. Recibiremos un informe detallado de todos los pasos de compilación.

15.8 Enlace dinámico y estático

Existen dos modos de realizar un enlace:

- **Estático**: Los binarios de las funciones se incorporan al código de nuestra aplicación.
- **Dinámico**: El código de las funciones permanece en las bibliotecas del sistema, nuestra aplicación cargará en memoria la librería necesaria y obtendrá de ella las funciones que requiere para trabajar.

Confrontemos ambos alcances:

Tabla 2: Title

Enlazado Dinámico	Enlazado Estático
El enlazado dinámico permite crear un archivo ejecutable más chico, pero requiere que el acceso a las librerías del sistema siempre este disponible al momento de correr el programa.	El enlazado estático crea un programa autónomo pero el precio a pagar es un mayor tamaño.
gcc -static -o circulo_s circulo.c	*gcc -o circulo_d circulo.c*
7.0kB 475kB	

Como podemos ver, la versión estática del programa no muestra dependencia alguna con las librerías del sistema.

15.9 Resumen

Si desea producir un ejecutable a partir de un solo archivo de código fuente:

```
$ gcc -o circulo circulo.c
```

Para crear un módulo objeto, con el mismo nombre del archivo de código fuente y extensión .o:

```
$ gcc -c circulo.c
```

8 El tamaño de ambos ejecutables varía dependiendo del Sistema Operativo, el compilador, las librerías.

Para enlazar los módulos verde.o, azul.o y rojo.o en un ejecutable llamador colores:

```
$ gcc -o colores verde.o azul.o rojo
```

CAPÍTULO 16

ANEXO 4.6.1.2 : MAKE

16.1 Introducción

Cuando nuestros programas son sencillos (1 archivo de código fuente), el compilar es un proceso rápido, basta con usar gcc:

```
$ gcc -o ejemplo ejemplo.c
```

Sin embargo, si tenemos más de un archivo, la compilación sería más compleja:

```
$ gcc -c modulo1.c
$ gcc -c modulo2.c
$ gcc -o programa modulo1.o modulo2.o
```

Conforme crezca la complejidad de nuestro proyecto así crecerá la dificultad de crear algún entregable tal como una librería o un programa ejecutable.

16.2 La herramienta make

Según se indica en el manual de make, el propósito de esta utilidad es determinar automáticamente quémpiezas de un programa necesitan ser recompiladas y, de acuerdo a un conjunto de reglas, lleva a cabo las tareas necesarias para alcanzar el objetivo definido el cual normalmente es un programa ejecutable. make agiliza el proceso de construcción de proyectos con cientos de archivos de código fuente separados en diferentes directorios. De esta forma y con las configuraciones adecuadas, make compila y enlaza todos los programas. Si alguno de los archivos de código fuente sufre alguna modificación sólo será reconstruido aquel módulo de cuyos componentes haya cambiado. Por supuesto es necesario indicarle a make que módulos u objetivos dependen de qué archivos, este listado se concentra en el archivo Makefile.

16.3 El formato del archivo Makefile

Un archivo Makefile es un archivo de texto en el cual se distinguen cuatro tipos básicos de declaraciones

- Comentarios.
- Variables.
- Reglas explícitas
- Reglas implícitas.

16.4 Comentarios

Al igual que en cualquier lenguaje de programación, los comentarios en los archivos Makefile contribuyen a un mejor entendimiento de las reglas definidas en el archivo. Los comentarios se iniciancon el carácter # y se ignora todo lo que viene después de este carácter hasta el final de línea. Ejemplo: # Este es un comentario.

16.5 Variables

Las variables en un Makefile no están tipeadas (es decir, no es necesario declarar previamente el tipo de valor irán a almacenar), en cambio todas son tratadas como cadenas de texto. Las variables que no están declaradas simplemente se tratan como si no existieran (por ejemplo son cero, o son una cadena vacía). La asignación de valores a una variable se hace de una manera sencilla:

```
nombre = dato
```

De esta forma se simplifica el uso de los archivos Makefile. Para obtener el valor de una variable deberemos encerrar el nombre de la variable entre paréntesis y anteponer el carácter $. En el caso anterior, todas las instancias de $(nombre) serán reemplazadas por dato. Por ejemplo, la

Siguiente regla:

```
SRC = main.c
```

Origina la siguiente línea:

```
gcc $(SRC)
```

Y será interpretada como:

```
$ gcc main.c
```

Sin embargo, una variable puede contener más de un elemento, por ejemplo:

```
objects = modulo_1.o modulo_2.o \
modulo_3.o \
modulo_4.o
programa : $(objects)
gcc -o programa $(objects)
```

Debemos hacer notar que la utilidad make hace distinción entre mayúsculas y minúsculas. Reglas explícitas. Las reglas explícitas le dictan a make qué archivos dependen de otros y los comandos a usar para lograr un objetivo en específico. El formato es:

```
objetivo: requisitos
comando #para lograr el objetivo
```

Esta regla le instruye a make como crear un objetivo a partir de los requisitos utilizando un comando específico. Por ejemplo, para generar un ejecutable que se llame main, escribiremos algo por el estilo:

```
main: main.c main.h
gcc -o main main.c main.h
```

Esto significa que el requisito para poder lograr el objetivo main(un programa), es que existan los archivos main.c y main.h y para lograr el objetivo deberemos utilizar gcc en la forma descrita.

16.6 Reglas implícitas

La reglas implícitas confían a make el trabajo de adivinar qué tipo de archivo queremos procesar (para ello utiliza las extensiones o sufijos del o los archivos). Las reglas implícitas ahorran el trabajo de tener que indicar qué comandos hay que ejecutar para lograr el objetivo, pues esto se infiere a partir de la extensión del archivo a procesar. Por ejemplo:

```
funciones.o : funcion1.c funcion1.c
```

origina la siguiente linea:

```
$(CC) $(CFLAGS) -c funcion1.c funcion2.c
```

Existe un conjunto de variables que ya están predefinidas y se utilizan para las reglas implícitas. De ellas existen dos categorías: (a) aquellas que son nombres de programas (como CC, que invoca al compilador de C), y (b) aquellas que contienen los argumentos para los programas invocados (como CFLAGS, que contiene las opciones que se le pasarán al compilador de C). Todas estas variables ya son provistas y contienen valores predeterminados , sin embargo, pueden ser modificados como se muestra a continuación:

```
CC = gcc
CFLAGS = -g -Wal
```

En el primer caso se indicará que el compilador de C será GNU GCC y el segundo caso activará todo tipo de avisos del compilador y compilará una versión para depurado.

16.7 Un ejemplo de un archivo Makefile

A continuación se muestra el ejemplo de un archivo Makefile completo donde se incluyen todos los tipos de declaraciones. En este ejemplo se utiliza la utilidad make para ayudar a la compilación de los módulos funciones.c y main.c para crear un ejecutable llamado mi_programa.

```
# La siguiente regla implicita instruye a make en como
# procesar los archivos con extensión .c y .o
.c.o:
$(CC) -c $(CFLAGS) $<
# Definición de variables globales.
CC = gcc
CFLAGS = -g -Wall -O2
SRC = main.c funciones.c funciones.h
OBJ = main.o funciones.o
# La regla explicita all indica a make como
```

(continué en la próxima página)

(proviene de la página anterior)

```
# procesar todo el proyecto.
all: $(OBJ)
$(CC) $(CFLAGS) -o main $(OBJ)
# Esta regla indica como limpiar el proyecto de
# archivos temporales.
clean:
$(RM) $(OBJ) main
# Reglas implícitas
funciones.o: funciones.c \
funciones.h
main.o: main.c \
funciones.h
```

En este archivo Makefile se han definido dos reglas explícitas que indican como construir los objetivos all y clean. Para llevar a cabo alguno de los dos objetivos basta ejecutar:

```
$ make
```

... lo cual ejecutará la primera regla que encuentra, es decir all, la cual compilará los programas definidos en la variable $(OBJECT). Si se desea que se ejecuten las tareas de la regla clean, se deberá ejecutar:

```
$ make clean
```

El archivo funciones.h contiene el prototipo de las funciones de las funciones empleadas en el programa main.c y estas, a su vez, se encuentran implementadas en funciones.c. De esta manera, es posible separar en distintos módulos las funciones, objetos, métodos, definiciones y variables que necesitemos en un proyecto determinado.

16.8 Definiendo nuevas reglas

make tiene definido un conjunto de reglas básicas para convertir archivos, típicamente los archivos cuyas extensiones pertenecen a los lenguajes más conocidos como C, C++, Java, Fortran, entre otros. También es posible crear reglas propias para formatos de archivos que no necesariamente han de crear un programa ejecutable.

Por ejemplo, se puede mantener un conjunto de documentos, cuyo fuente se encuentran en formato .lyx y que se desea convertir a otros formatos como PDF, TeX, Postcript, etc y cuyos sufijos son desconocidos por make.

A continuación se describe cómo añadir nuevas reglas con GNU make, el cual puede diferir con versiones antiguas de make. Por compatibilidad, más adelante se explica cómo definirlo de la antigua forma, que GNU también puede interpretar.

La forma de definir una regla que permita convertir un archivos PostScript en formato PDF sería de la siguiente manera:

```
%.pdf: %.ps
ps2pdf $<
```

Se ha indicado que los archivos cuya extensión son .pdf dependen de los archivos .ps, y que se generan utilizando el programa indicado en la linea siguiente(ps2pdf). El parámetro de entrada para el programa será el nombre del archivo con extensión .ps. Sólo falta indicar la regla que archivos se irán a convertir, por ejemplo:

```
all: documento1.pdf documento2.pdf
```

De esta forma, el objetivo de make será construir all, para lo cual debe construir documento1.pdf y documento2.pdf. Para lograr este objetivo, make buscará los archivos documento1.ps y documento2.ps, lo cual se traducirá en los siguientes comandos:

```
ps2pdf documento1.ps
ps2pdf documento2.ps
```

16.9 Mejorando los Makefiles con variables automáticas

Existen algunas variables automáticas que permiten escribir los archivos Makefile de una forma genérica, así, si se requiere modificar el nombre de un archivo o regla que entonces sólo sea necesario realizar los cambios en un solo lugar, o en la menor cantidad de lugares posibles y así evitar errores.

Las variables automáticas más empleadas son:

- **$ <** El nombre del primer requisito.
- **$** En la definición de una regla implícita tiene el valor correspondiente al texto que reemplazará el símbolo %.
- **$?** Es el nombre de todos los prerequisitos.
- **$@** Es el nombre del archivo del objetivo de la regla.

```
%.pdf : %.ps
ps2pdf $ <
%.zip: %.pdf
echo $*.zip $<
PDF = documento1.pdf documento2.pdf
ZIP = documento1.zip documento2.zip
pdf: $(PDF)
tar -zcvf $@.tar.gz $?
zip: $(ZIP)
clean:
rm -f *.pdf *.tar
```

En el ejemplo, se han definido dos reglas implícitas. La primer indica cómo convertir un archivo PostScript a PDF y la segunda dice cómo comprimir un archivo pdf en formato ZIP. También se han definido cuatro reglas, dos de ellas son implícitas (pdf y zip), donde sólo se han indicado sus requisitos de las otras dos (paquete y clean) son explícitas. Cuando se ejecute la regla paquete, make analizará las dependencias, es decir, verificará si existen los correspondientes archivos PDF, si no existieren, los construye para luego ejecutar el comando indicado en la regla. La variable $? será expandida a «documento1.pdf documento2.pdf» y la variable $@ será expandida a «paquete». De esta forma el comando a ejecutar será:

```
tar -zcvf paquete.tar.gz documento1.pdf documento2.pdf
```

En el caso de la regla zip, al resolver las dependencias se ejecutará:

```
zip documento1.zip documento1.pdf
zip documento2.zip documento2.pdf
```

Es decir, el patrón buscado es documento1 y documento2, los cuales corresponden con la expresión %. Dicha operación se realizará para cada archivo .pdf.

CAPÍTULO 17

Índices y Tablas

- genindex
- modindex
- search