Tecnicatura Superior en Análisis de Sistemas

Programación I

"Trabajando con el Compilador GNU C/C++ GCC"
Versión 2.0 Marzo 2008

Lic. Guillermo R. Cherencio

INDICE

Introducción

La programación requiere de orden. Debemos organizar nuestro trabajo. Para trabajar con el lenguaje C/C++, debemos:

1. Instalar el compilador
2. Crear nuestro ambiente de trabajo
3. Utilizar el ambiente de trabajo

Acerca de las Librerías y Bibliotecas

Debemos definir lo que entendemos por función, en C todo el código que escribimos son funciones. Una función contiene al menos un bloque de código (conjunto de instrucciones C encerradas entre llaves, ejemplo: { a = 4; b = 1; }). Una función puede recibir 0, 1 o más argumentos. Los argumentos son información adicional que le proveemos a la función desde el exterior de la misma. Una función puede devolver un valor. Ejemplos de declaración de funciones:

```
void funcion1();    // función sin argumentos y sin retorno
int sumar(int, int);// función con dos argumentos enteros que retorna
                    // un entero
```

La declaración de una función se refiere a la “forma” o “prototipo” de la función. El lenguaje C nos obliga a que declaremos todas las funciones antes de usarlas. Los archivos .h (headers, cabeceras) contienen declaraciones de funciones para ayudarnos a no tener que tipear las declaraciones de funciones en forma repetitiva para cada una de las librerías que usamos en nuestros proyectos. En vez de lo anterior, podemos usar la siguiente instrucción:

```
#include “milibreria.h”
```

En donde, milibreria.h es un archivo que contiene las declaraciones de las funciones funcion1() y sumar(), evitando de esta forma, tener que tipearlas por cada vez que quiera usarlas.

De esto se deduce, que las funciones están vinculadas con la reutilización de código (usar más de una vez un mismo código, para aumentar la productividad y no volver a realizar una tarea ya hecha).

Si una librería o biblioteca es una entidad “separada” de las aplicaciones que la utilizan, esto implica que podemos pensar en que las aplicaciones se conforman por una serie de librerías más el código propio de la aplicación. Ello nos permite pensar en dividir nuestro trabajo en módulos, en donde, una librería podría ser un módulo de la aplicación. Ello permitiría hacer cambios y mejoras en un módulo haciendo que todas las aplicaciones se beneficien de ello. También puede ocurrir lo contrario, si cometemos un error, el mismo se propagará a todas las aplicaciones que usen la librería. Por lo tanto, las funciones están vinculadas al concepto de modularidad. El código de las funciones de una librería debe estar muy bien probado para evitar propagar errores en múltiples aplicaciones. Otra cuestión es que una librería o biblioteca debe estar correctamente diseñada y pensada para lograr el máximo de modularidad y reutilización.

Desarrollar una librería o biblioteca no implica realizar un trabajo “desde cero”, la idea es continuar reutilizando –a su vez- otras librerías o bibliotecas, incluso más allá de las provistas por el lenguaje C. Una librería o biblioteca se conforma con código proveniente de otras librerías más nuestro propio código. Esto se deduce que pueden existir dependencias entre librerías o bibliotecas: si pretendo desarrollar la biblioteca A que hace uso de la biblioteca B, por lo tanto, A depende de B, si B no esta instalada o no existe, A nunca podrá ser compilada.

La definición o implementación de una función se refiere al bloque de código de la misma y no sólo a su declaración. La definición obliga al nombrado de los argumentos (darle un nombre a cada argumento recibido). Ejemplo de definiciones o implementaciones de funciones:

```
void funcion1() {
   printf(“hola, estoy ejecutando funcion1()\n”);
}
int sumar(int a,int b) {
   return a+b;
}
```

Cuando nos referimos al término librería o biblioteca, hacemos referencia a un conjunto de funciones que están agrupadas acorde con su funcionalidad. Por ejemplo, es posible crear una librería o biblioteca para el manejo de archivos, para el manejo de la impresión, para el manejo de determinado dispositivo, etc. Un conjunto de funciones que no están relacionadas entre sí no forman una librería o biblioteca.

El lenguaje C ya viene con un grupo importante de bibliotecas, tales como la biblioteca standard stdlib, la cual provee un conjunto de funciones que sirven

para realizar tareas elementales que toda aplicación necesitará. También es posible comprar licencias de uso de bibliotecas hechas por terceros. También es posible incorporar a nuestros proyectos bibliotecas bajo licencias de uso de tipo GNU (software libre, gratuito), existen muchas y para distintos propósitos.

Existe en C una función especial llamada `main()` la cual es la función principal de la aplicación, ya que será la función que se pondrá automáticamente en ejecución cuando el usuario utilice la aplicación. El punto de ejecución de la aplicación comienza con la primer línea de código de la función main() . Existen varios prototipos posibles para main():

```
main();
void main();
int main();
int main(int, char *[]);
```

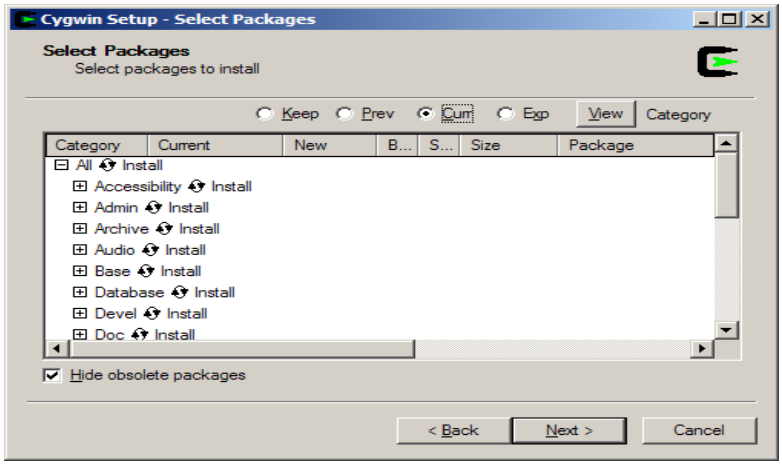
El último prototipo de main() permite hacer referencia a los parámetros indicados en la línea de comando por el usuario en la ejecución de aplicación. El primer argumento se refiere a la cantidad de parámetros y el segundo son los parámetros indicados por el usuario.

Instalación del compilador GNU C/C++ GCC

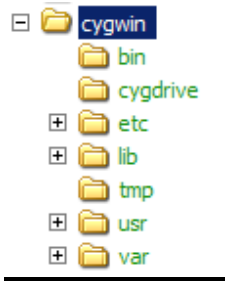
Si estamos utilizando Linux, no debemos hacer nada, pues este compilador ya se encuentra instalado.

Si estamos utilizando windows (en cualquiera de sus versiones) debemos instalar una versión del compilador para windows, existen varias, las más conocidas son: cygwin http://www.cygwin.com y minGW http://www.mingw.org . Arbitrariamente, hemos optado por cygwin, el cual puede instalarse de dos formas:

1. Instalación “on-line”: se debe ingresar a la página de cygwin, bajar el programa setup.exe, ejecutarlo, seguir los pasos que indica el programa y los paquetes del compilador (packages, librerias o bibliotecas de funciones) se irán descargando desde la web e instalándose localmente en nuestro computador.
2. Instalación “off-line”: se puede bajar el programa setup.exe de la página de cygwin (406KB), luego bajar también los paquetes del compilador (90MB aprox.), guardar toda la información en un directorio y luego hacer una instalación desde allí sin necesidad de mantener una conexión abierta. Se debe ejecutar Setup.exe e indicarle “Install from local directory”, el “root directory” se refiere al directorio en donde quedará instalado el compilador, el “Local package directory” se refiere al directorio en donde se encuentran los paquetes del compilador (bajados previamente de internet). El instalador permite instalar, reinstalar, quitar, etc. los paquetes por categorías, usar la categoría “All” (todas), seleccionar “Install” (dando doble click sobre la misma) para instalar todos los paquetes, Ejemplo:



Una vez terminada la instalación, el “root directory” tendrá el siguiente aspecto:



Observe el contenido de los directorios: /cygwin/bin y /usr/include .

Debería contar con un icono en su escritorio que le permita ejecutar el archivo batch (archivo de comandos Windows/dos) cygwin.bat que se encuentra en el “root directory”:

El archivo cygwin.bat

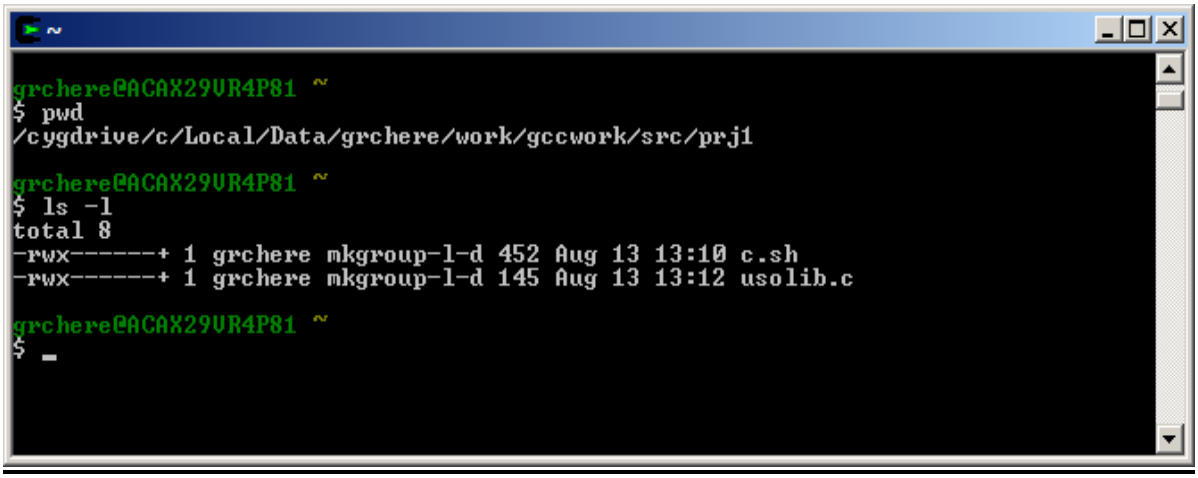
El archivo cygwin.bat permite ejecutar el programa bash.exe (Bourne Again Shell) que permite emular una consola Linux sobre Windows (sólo el shell, no implica que se haya instalado un Linux sobre su Windows).

Contenido del archivo cygwin.bat

```
@echo off
rem
rem cygwin.bat
rem     Archivo batch para acceso a cygwin
rem     permite ingresar a un shell bash en ambiente windows
rem     para utilizar el compilador GNU C/C++ GCC
rem Atte. Guillermo Cherencio
rem       ISFDYT Nº 189
C:
cd C:\Local\Data\grchere\work\cygwin\bin
SET CYGWIN=tty notitle glob
SET
PATH=/cygdrive/c/Local/Data/grchere/work/cygwin/bin:/cygdrive/c/windows:/cygdrive/c/windo
ws/system32
```

```
SET
LD_LIBRARY_PATH=/cygdrive/c/Local/Data/grchere/work/cygwin/bin:/cygdrive/c/Local/Data/grc
here/work/cygwin/lib:/cygdrive/c/Local/Data/grchere/work/cygwin/usr/include:/cygdrive/c/L
ocal/Data/grchere/work/gccwork/lib
SET INCLUDE=c:\Local\Data\grchere\work\cygwin\usr\include
SET
LIB=/cygdrive/c/Local/Data/grchere/work/cygwin/lib:/cygdrive/c/Local/Data/grchere/work/gt
k/lib:/cygdrive/c/Local/Data/grchere/work/gccwork/lib
SET HOME=/cygdrive/c/Local/Data/grchere/work/gccwork/src/prj1
bash --login -i
echo on
```

El shell bash tomará en cuenta el contenido de algunas variables de entorno tal como PATH, HOME, etc. ; en este ejemplo, el “root directory” de cygwin es C:\Local\Data\grchere\work\cygwin ; este directorio, expresado en bash sería /cygdrive/c/Local/Data/grchere/work/cygwin . No obstante, puede indicarse en la variable de una u otra forma y bash lo comprenderá. Puede utilizar este archivo para configurar bash (ver manual cygwin). Una vez dentro de bash, se soportan algunos comandos Linux:



También se soportan comandos Windows, por ejemplo, podemos tipear “Notepad miprog.c” y se ejecutará el “Block de Notas” de Windows, editando el archivo “miprog.c”.

Nuestro ambiente de trabajo

Los entornos integrados de desarrollo (IDE, integrated development environment) proveen de una aplicación con la cual interactúa el programador para llevar adelante todo el proceso de desarrollo de software, en algunos casos, cubren el ciclo de vida completo de un sistema. Estas herramientas pueden ser desde simples editores hasta aplicaciones muy sofisticadas. Esta guía pretende montar un ambiente básico y elemental de desarrollo de aplicaciones en lenguaje C sin utilizar ningún IDE, simplemente haciendo uso del compilador GNU C/C++ GCC.

Existen metodologías y normas que guían este tópico, aquí proponemos una forma muy básica y elemental de organizar el trabajo. Toda metodología implica seguir algunas normas:

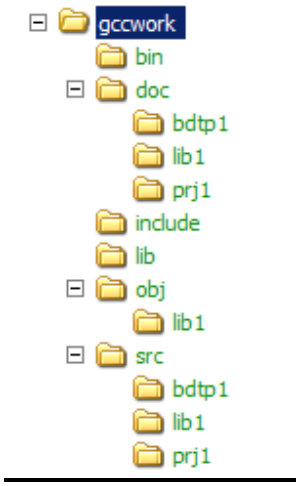
- Todo proyecto podrá ser una aplicación o librería.
- Toda aplicación tendrá un directorio propio, único e irrepetible (ejemplo: prj1) dentro de los distintos directorios de nuestro ambiente de trabajo (debe coincidir su nombre en /doc, /src, /obj). Cabe utilizar alguna norma para el nombrado de los mismos, separando los proyectos que consideramos aplicaciones de aquellos que son librerías para ser reutilizadas en n aplicaciones. Por ejemplo, llamar appXXXX a las aplicaciones (reemplazar XXXX por lo que corresponda) y libXXXX a las librerías. Evitar usar espacios intermedios.
- Toda aplicación tendrá un script (programa del shell, similar a un archivo batch windows/dos) llamado “c.sh” que realizará la compilación y linkedición del proyecto.

Primero debemos separar nuestro trabajo del compilador en sí mismo, no podemos guardar nuestros programas dentro del directorio /cygwin/bin . Por lo tanto, se creará un directorio separado para nuestro trabajo (/gccwork), el cual será nuestro directorio “root” (raíz). Dentro de dicho directorio, tendremos –al menos- los siguientes directorios:

Directorio	Propósito
/bin	Ejecutable de las distintas aplicaciones que desarrollamos.
/doc	Directorio “root” de la documentación de cada una de aplicaciones.
/include	Cabeceras (headers, archivos *.h) de cada una de las librerías propias o de terceros que requieran nuestros proyectos.
/lib	Librerías en código nativo de la plataforma (ya sean propias o de terceros) para ser utilizadas en la la etapa de linkedición.
/obj	Directorio “root” de objetos compilados. Generalmente se almacenarán

	los objetos compilados de las librerías propias o de terceros (que los provean); ya que los mismos se combinarán para formar el o los archivos de código nativo en /lib.
/src	Directorio “root” de programas fuentes.

Ejemplo:



Ejemplo de archivos contenidos en ambiente de trabajo:

```
grchere@ACAX29VR4P81 /cygdrive/c/Local/Data/grchere/work/gccwork
$ pwd
/cygdrive/c/Local/Data/grchere/work/gccwork

grchere@ACAX29VR4P81 /cygdrive/c/Local/Data/grchere/work/gccwork
$ ls -lR
.:
total 4
-rwx------+ 1 grchere ???????? 616 Aug 10 15:50 Cygwin.bat
drwx------+ 2 grchere ????????   0 Aug 21 14:57 bin
drwx------+ 5 grchere ????????   0 Aug 22 15:24 doc
drwx------+ 2 grchere ????????   0 Jul 25 16:29 include
drwx------+ 2 grchere ????????   0 Aug 13 13:12 lib
drwx------+ 3 grchere ????????   0 Aug 13 13:04 obj
drwx------+ 5 grchere ????????   0 Aug 13 13:19 src

./bin:
total 1976
-rwxrwxrwx  1 grchere mkgroup-l-d   34547 Aug 17 11:10 bdtp11.exe
-rwxrwxrwx  1 grchere mkgroup-l-d   38235 Aug 17 11:16 bdtp12.exe
-rwxrwxrwx  1 grchere mkgroup-l-d   41701 Aug 21 14:57 bdtp13.exe
-rwx------+ 1 grchere ????????    1873811 Jan 31  2007 cygwin1.dll
-rwxrwxrwx  1 grchere mkgroup-l-d   21423 Aug 13 13:12 usolib.exe

./doc:
total 0
drwx------+ 2 grchere ???????? 0 Aug 22 15:24 bdtp1
drwx------+ 2 grchere ???????? 0 Aug 22 15:24 lib1
drwx------+ 2 grchere ???????? 0 Aug 22 15:24 prj1

./doc/bdtp1:
total 0
```

```
./doc/lib1:
total 0

./doc/prj1:
total 0

./include:
total 4
-rwx------+ 1 grchere mkgroup-l-d 75 Jul 25 16:13 milib01.h

./lib:
total 16
-rw-rw-rw- 1 grchere mkgroup-l-d 12310 Aug 13 13:12 libmilib01.a

./obj:
total 0
drwx------+ 2 grchere ???????? 0 Aug 13 13:12 lib1

./obj/lib1:
total 12
-rw-rw-rw- 1 grchere mkgroup-l-d 12164 Aug 13 13:12 libmilib01.o

./src:
total 0
drwxrwxrwx+ 2 grchere mkgroup-l-d 0 Aug 21 17:03 bdtp1
drwx------+ 2 grchere ????????    0 Aug 13 13:12 lib1
drwx------+ 2 grchere ????????    0 Aug 13 13:11 prj1

./src/bdtp1:
total 336
-rwxrwxrwx 1 grchere mkgroup-l-d  9062 Aug 17 11:10 bdtp11.c
-rwxrwxrwx 1 grchere mkgroup-l-d 16883 Aug 17 11:16 bdtp12.c
-rw-r--r-- 1 grchere mkgroup-l-d 15395 Aug 21 14:55 bdtp13.c
-rwx------ 1 grchere mkgroup-l-d   281 Aug 15 10:06 c.sh
-rwx------ 1 grchere mkgroup-l-d   281 Aug 15 10:32 c2.sh
-rwx------ 1 grchere mkgroup-l-d   281 Aug 16 11:13 c3.sh

./src/lib1:
total 8
-rwx------+ 1 grchere mkgroup-l-d 572 Aug 13 13:10 c.sh
-rwx------+ 1 grchere mkgroup-l-d 170 Jul 25 16:13 milib01.c

./src/prj1:
total 8
-rwx------+ 1 grchere mkgroup-l-d 452 Aug 13 13:10 c.sh
-rwx------+ 1 grchere mkgroup-l-d 145 Aug 13 13:12 usolib.c
```

Restricciones

Toda metodología impone reglas, pero también existen restricciones. Algunas son obvias, derivadas de las propias reglas, por ejemplo:

- Los nombres de los proyectos son únicos, puesto que todos sus ejecutables se guardan en un mismo lugar: /bin
- Las librerías propias o de terceros no pueden tener igual nombre de archivo con código nativo, pues se guardan en un mismo lugar: /lib

- Idem con los archivos de cabecera de las librerias, se guardan en /include
- El directorio de ejecución, para probar la aplicación está dentro del mismo directorio de los programas fuentes (si bien se ejecutan de /bin, estamos posicionados en /src/<proyecto>/). Esto habitualmente no es aceptable debido a que un programa que trabaja con archivos accidentalmente podría destruir información dentro del directorio y en tal caso, perderíamos los programas fuentes. Se recomienda la creación de otro directorio root llamado /test que sea utilizado para ejecución y prueba de programas; esto ha sido omitido por razones de simplicidad.
- No se contemplan distintas versiones de un mismo proyecto, se deberían crear directorios nuevos y únicos para diferenciarlos, una forma posible sería appXXXXversión (Ej. appPrj1.1, appPrj1.2, etc.)
- La actual descripción de ambiente corresponde a un ambiente de desarrollo. En una ambiente más profesional, existen al menos 4 ambientes: desarrollo, test, aceptación y producción. Serían ambientes muy similares al descripto aquí, pero con distintos objetivos y roles. El ambiente de test sería utilizado por analistas para probar el trabajo de los programadores usando datos de prueba. El ambiente de aceptación sería utilizado por los usuarios finales con copias de datos reales para aceptar su correcto funcionamiento. El ambiente de producción sería el utilizado por los usuarios para la ejecución real de la aplicación. Los analistas y programadores –a lo sumo- tendrán acceso de sólo lectura a los ambientes de aceptación y producción.

Compilación y Linkedición de Aplicaciones

El compilador GCC nos permite realizar con un solo comando, en un solo paso ambos procesos. Veamos un ejemplo:

Contenido del archivo prj2.c de la carpeta /src/prj2

```
/*
   programa muy simple prj2.c
*/

#include <stdio.h>
void main() {
   printf("Hola Mundo\n");
}
```

En este caso, se trata de un simple programa que sólo hace uso de la librería standard de C stdlib (la cual esta incluida en GCC), no hace uso de ninguna librería propia o de terceros, por lo tanto, no necesita buscar nada en /include, /lib, /obj de nuestro ambiente. Un script de compilación y linkedición posible para este programa sería:

Contenido del archivo c.sh de la carpeta /src/prj2

```
#!/bin/sh
#
# script para compilar aplicacion prj2.c
#
#                Atte. Guillermo Cherencio
#                ISFDyT Nº 189
#
echo compilo y enlazo
gcc -Wall -g prj2.c -o ../../bin/prj2.exe
echo ejecuto
../../bin/prj2.exe
```

El "switch" (opción) –Wall indica a GCC que active el aviso de todos los mensajes de observación (warnings); en este caso, emitirá el siguiente mensaje:

```
prj2.c:6: warning: return type of 'main' is not `int'
```

el mensaje nos dice que es habitual que main() retorne un int que le indica al sistema operativo cómo ha terminado el proceso, cosa que hemos omitido, no implica un error, pero si una observación, ya que esto no es habitual.

El "switch" –g implica incorporar al código nativo toda información de debug para este programa, esto permite que nuestro programa –en caso de errores en tiempo de ejecución- pueda ser tomado por una herramienta de "debug" (depuración) que nos ayude a encontrar el problema. Este switch es útil en tiempo de desarrollo, una vez terminado nuestro trabajo, habiendo probado el programa, conviene sacarlo para que el programa ejecutable tenga un tamaño menor y mayor performance.

El switch –o sirve para indicar el destino de esta compilación y linkedición (también llamado enlace (link), ya que "enlaza" nuestro código con el código de las librerías que usamos), en este caso el programa ejecutable se llamará prj2.exe y deberá estar en el directorio /bin, acorde a nuestro ambiente.

Para mayor información sobre otros switchs disponibles en GCC tipear el comando: `man gcc`

Luego se debe indicar el o los programas a compilar y linkeditar separados por uno o más espacios, en este caso, prj2.exe está sólo compuesto por prj2.c .

Utilizamos el comando: `./c.sh` para lanzar la compilación, linkedición y ejecución del programa (tener en cuenta que si falla alguna etapa del proceso de

compilación, linkedición estaremos ejecutando la última versión del programa compilada y linkeditada satisfactoriamente).

Compilación y Catalogación de Librerías

Con el término catalogación nos referimos a la instalación de una librería o aplicación en un lugar determinado que permita su uso y normal funcionamiento. Por ejemplo, una vez terminado nuestro trabajo en este ambiente, debemos "catalogar" la aplicación en un lugar en donde el usuario pueda utilizarla.

Existen distintos tipos de librerías o bibliotecas, aquí sólo tendremos en cuenta el caso de las denominadas "librerías estáticas", librerías con código nativo que será incorporado a nuestro ejecutable. También existen otras librerías: las compartidas (shared) , las dinámicas (las famosas DLL's dynamic loaded libraries), etc.

Una librería o biblioteca debe compilarse –al igual que una aplicación- para obtener un código objeto compilado, ese código –junto con otros objetos compilados- debe guardarse en un archivo de código nativo de la biblioteca –el cual no necesariamente lleva el mismo nombre que el programa fuente- y por último, debe catalogarse en el directorio /lib para que este disponible para todas las aplicaciones. Veamos un ejemplo, creamos en nuestro ambiente el proyecto "lib1", es una librería que solo posee una única función (mensaje1()) que será reutilizada en nuestro proyecto "prj1". El código de la librería se encuentra en /lib1/milib01.c ; debemos poner la declaración de la función dentro de /include/milib01.h (por lo que ya explicamos antes en cuanto a la declaración de funciones):

Contenido del archivo milib01.c de la carpeta /src/lib1

```
/*
milib01.c libreria c a ser reutilizada

*/

#include <stdio.h>
#include "milib01.h"

void mensaje1() {
   printf("ejecute mensaje1() de libreria.c\n");
}
```

Obsérvese que esta librería no se hizo “desde cero”, de hecho, esta reutilizando una librería standard de C llamada stdio (standard input – output, entrada – salida standard).

Contenido del archivo milib01.h de la carpeta /include

```
/*
Prototipos de funciones de libreria milib01.c
*/

void mensaje1();
```

Debemos compilar milib01.c indicándole a GCC que busque en /include el archivo de cabecera (header) milib01.h, para ello se utiliza el switch –I (letra “i” mayúscula) que indica el directorio de “includes”.

Para indicarle a GCC que sólo compile y no realice la linkedición, debemos usar el switch –c (compile, compilar).

El código compilado de la librería quedará en /obj/lib1.

Luego de compilado debemos “ensamblar” la librería en un único archivo (para ello usamos el comando `ar` (archive, archivar)) que finalmente quedará en el directorio /lib . Veamos el script de compilación – ensamblado y catalogación de la librería:

Contenido del archivo c.sh de la carpeta /src/lib1

```
#!/bin/sh
#
# script para compilar libreria milib01.c
# si la libreria se llama "milib01" entonces el archivo
# que la contiene se llama "libmilib01.a"
#
#                Atte. Guillermo Cherencio
#                ISFDyT Nº 189
#
echo compilo libreria
gcc -Wall -g -c -o../../obj/lib1/libmilib01.o -I../../include milib01.c
#ensamblo objetos compilados: creo libreria
echo ensamblo libreria
ar rcs libmilib01.a ../../obj/lib1/libmilib01.o
#catalogo libreria: muevo libreria a directorio lib
echo catalogo libreria
mv libmilib01.a ../../lib/libmilib01.a
```

El comando `ar` (archive, archivar) genera en el directorio actual el archivo libmilib01.a que será el archivo de la librería (si quisiéramos vender nuestra librería, deberíamos distribuir este archivo). El nombre no es arbitrario, debemos anteponer el prefijo “lib” al nombre de nuestra libreria, ya que GCC tiene el switch –l<nombre de librería> (“ele” minúscula) el cual buscará a “lib<nombre de librería>” en los directorios indicados por los switches –L<directorio libreria>.

Véase el script de compilación de la aplicación que hace uso de esta librería. Esto es muy importante, porque sino no se podrá ubicar el archivo de librería y la linkedición fallará.

El comando `mv` (move, mover) simplemente cataloga el archivo de librería con código nativo al directorio /lib con el nombre que corresponde, para dejarla lista para ser usada.

De este proceso se deduce la existencia de los directorios /obj en donde se almacenan el código compilado de las librerías, lo cual nos permite volver a construir el archivo de código nativo de la librería sin necesidad de contar con los programas fuentes de la librería.

Compilación y Linkedición de Aplicaciones que usan Librerías

Para usar una librería debemos indicar –al menos- una sentencia #include indicando el nombre del archivo de cabecera (header) de la librería:

```
#include "milib01.h"
```

Esto se aplica a todas las librerías, incluso a las standards del lenguaje C, por ejemplo:

```
#include <stdlib.h>
```

Pero observe la sutil diferencia de ambas instrucciones, ello se debe a que los archivos de cabecera standards se buscan automáticamente –estando correctamente instalado el compilador-; mientras que los otros deben indicársele a GCC en el script de compilación usando los switches[1] –I<directorio includes> (i mayúscula). Por lo tanto, nuestro script de compilación deberá incluir –al menos un switch –I apuntando al directorio /gccwork/include, lugar donde se encuentra milib01.h .

Algo similar sucede con el código nativo de las librerías, en la linkedición se incluyen automáticamente el código nativo de las librerias standards, pero debemos indicar el o los directorios en donde buscar el código nativo de las librerías propias o de terceros; para ello se utiliza el switch –L<directorio librerías>. Además de ello, debemos también indicar el nombre de la librería (puesto que en los directorios de librerías podría haber muchísimos archivos de librerías y examinarlos a todos haría muy lento el proceso de linkedición) utilizando el o los switch –l<librería> ("ele" minúscula). Veamos el código del

[1] Se habla de "switches" porque el el switch –I puede repetirse n veces. Idem para el switch -L

programa usolib.c del proyecto /src/prj1 y su script de compilación – linkedición - ejecución:

Contenido del archivo usolib.c. de la carpeta /src/prj1

```
/*
programa para usar la libreria milib01.c

*/

#include <stdio.h>
#include "milib01.h"

int main() {
   mensaje1();
   return 1;
}
```

Contenido del archivo c.sh de la carpeta /src/prj1

```
#!/bin/sh
#
# script para compilar aplicacion usolib.c que usa libreria de usuario milib01
# (libreria incluida en archivo ../../lib/libmilib01.a)
# ejecutable=objetos compilados+librerias (propias, de gcc y de terceros)
#
#                Atte. Guillermo Cherencio
#                ISFDyT Nº 189
#
echo compilo y enlazo
gcc -Wall -g usolib.c -o ../../bin/usolib.exe -I../../include -L../../lib -lmilib01
echo ejecuto
../../bin/usolib.exe
```

Como nuestro ambiente irá acumulando los archivos de cabecera (headers, *.h) en el directorio /include y el código nativo de las librerías en /lib, a medida que nuestros proyectos sean más complejos y reutilicen más librerías propias o de terceros, sólo deberemos ir agregando switches –l (“ele” minúscula) a los scripts de compilación – linkedición – ejecución de las aplicaciones que reutilicen dicho código.

Espero que esta guía básica de compilación, linkedición y desarrollo de aplicaciones usando GCC haya sido de vuestro agrado.