

Software en formato fuente

Eugenia Damonte, Ariel Fideleff y Martín Goñi

Índice

1. Configuración previa	1
2. Uso básico de gcc	2

1. Configuración previa

Antes de comenzar a resolver los ejercicios configuramos `vim` para editar archivos en C. Para hacer esto abrimos el archivo `~/.vimrc` (en todo caso de no existir, hay que crearlo) que es el archivo de configuración de `vim`. Estaba vacío por lo que le le añadimos dos líneas: `set nocp` y `filetype plugin on`. Lo que hace el primer comando es desactivar el modo de compatibilidad. Éste hace que algunas de las funciones de `vim` sean deshabilitadas o modificadas para que se comporte de manera similar a `vi`, el antecesor de `vim`. La segunda permite utilizar el plugin `filetype`.

Luego para asegurarnos de tener todos los paquetes de `vim` utilizamos el comando `sudo apt-get install vim-gui-common vim-runtime`. El primer paquete tuvo que instalarse demorando varios minutos por la velocidad de descarga abismal de los repositorios. El segundo, por el otro lado ya estaba instalado en nuestro caso.

Finalmente creamos el archivo de configuración para los archivos con extensión `.c`, llamado `c.vim`. Para poder crearlo primero tuvimos que crear la carpeta `~/.vim/ftplugin`, que es donde se ponen los archivos de configuración. Luego abrimos el mismo con `vim` y escribimos las configuraciones que queríamos usar.

```
setlocal number
syntax on
colorscheme desert
setlocal tabstop=4
setlocal shiftwidth=1
setlocal expandtab
setlocal softtabstop=4
setlocal autoindent
setlocal smartindent
```

Las configuraciones para los archivos `.c`.

2. Uso básico de gcc

Antes de comenzar con el proyecto en si decidimos asegurarnos de que gcc funcionaba correctamente y que sabíamos usarlo. Para hacer esto copiamos el programa de ejemplo, `circulo.c`, que se encuentra en el apunte provisto.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #define PI 3.1416
5
6 int main(){
7     float area, radio;
8
9     radio = 10;
10    area = PI * (radio * radio);
11    printf("Circulo.\n");
12    printf("%s%f\n\n", "Area de circulo de radio 10: ", area);_
13
14    return 0;
15 }
```

El programa de ejemplo `circulo.c`.

2.1. Compilación directa

Una vez copiado el programa realizamos una compilación directa para asegurarnos de que el programa funcionase correctamente. Para hacer esto usamos el comando `gcc -o circulo circulo.c`. Lo que hace el argumento `-o` es especificar como se debe llamar el archivo de salida, si no se usa el archivo se nombra `a.out`.

```
martin@DebianPC:~/C$ gcc -o circulo circulo.c
martin@DebianPC:~/C$ ./circulo
Circulo.
Area de circulo de radio 10: 314.160004

martin@DebianPC:~/C$ rm circulo
```

Muestra de funcionamiento de `circulo.c`.

2.2. Compilación compleja

Habiendo comprobado que gcc funcionaba correctamente decidimos intentar compilar el mismo archivo, `circulo.c`, de manera compleja. Es decir

haciendo cada uno de los pasos que realiza el compilador a la hora de transformar un archivo en C en un programa ejecutable.

2.3. Compilación compleja - Compilación

El primer paso es la compilación donde se transforma el código en C a assembler del procesador de la computadora. Para hacer esto usamos el comando `gcc -S circulo.c`. Luego verificamos que haya funcionado mostrando las primeras líneas del archivo `circulo.s`, que es donde `gcc` almacena el archivo en assembler.

```
martin@DebianPC:~/C$ gcc -S circulo.c
martin@DebianPC:~/C$ head circulo.s
.file "circulo.c"
.section .rodata
.LC2:
.string "Circulo."
.LC3:
.string "Area de circulo de radio 10: "
.LC4:
.string "%s%f\n\n"
.text
.globl main
```

El primer paso, la compilación

2.4. Compilación compleja - Ensamblado

Una vez compilado procedimos a ensamblar el archivo. Es decir transformar el archivo en assembler a código objeto, un archivo binario en lenguaje máquina. Hicimos esto con el comando `as -o circulo.o circulo.s`. Luego verificamos que haya funcionado revisando que tipo de archivo era `circulo.o`.

```
martin@DebianPC:~/C$ as -o circulo.o circulo.s
martin@DebianPC:~/C$ file circulo.o
circulo.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
```

El segundo paso, el ensamblado.

2.5. Compilación compleja - Enlazado

Finalmente enlazamos el archivo, en este paso es donde nos encontramos con problemas. El comando que se da en el apunte no funciona. Al usarlo da varios errores indicando que las librerías usadas como argumentos no existen, así como algunas de las opciones.

```
martin@DebianPC:~/C$ ld -o circulo /usr/lib/gcc-lib/1386-linux/2.95.2/collect2 -m elf_1386 -dynamic-linker /lib/ld-linux.so.2 -s
circulo /usr/lib/crti.o /usr/lib/crt1.o /usr/lib/gcc-lib/1386-linux/2.95.2/crtbegin.o -L/usr/lib/gcc-lib/1386-linux/2.95.2 circ
ulo.o -lgcc -lc -lgcc /usr/lib/gcc-lib/1386-linux/2.95.2/crtend.o /usr/lib/crtn.o
ld: cannot find /usr/lib/gcc-lib/1386-linux/2.95.2/collect2: No such file or directory
ld: cannot find /usr/lib/crt1.o: No such file or directory
ld: cannot find /usr/lib/gcc-lib/1386-linux/2.95.2/crtbegin.o: No such file or directory
ld: cannot find -lgcc
ld: cannot find -lc
```

El primer intento de usar `ld`, siguiendo el apunte.

Dado todos los errores que había decidimos borrar todas las opciones innecesarias y probar nuevamente. Al hacer esto los errores anteriores desaparecieron pero uno nuevo apareció, este decía “cannot find entry symbol `_start`”. Esto se traduce como “no se encuentra el simbolo de entrada `_start`”. Luego de algo de investigación descubrimos que este error se debe a que el verdadero punto de entrada¹ de un programa es `_start`, no `main`, `_start` simplemente redirige a él. Para solucionar esto usamos el argumento `--entry main` para especificar la función `main` como el punto de entrada del programa.

Una vez hechos estos cambios la función no daba mas errores y el programa parecia estar listo para usar. A la hora de ejecutarlo, sin embargo este no era reconocido como un programa ejecutable. Para asegurarnos de haber hecho todo correctamente revisamos que el archivo existiese así como también sus permisos, siendo estos correctos.

```
martin@DebianPC:~/C$ ld -o circulo circulo.o -lc
ld: warning: cannot find entry symbol _start; defaulting to 00000000080481e0
martin@DebianPC:~/C$ ld -o circulo circulo.o -lc --entry main
martin@DebianPC:~/C$ ./circulo
-bash: ./circulo: No such file or directory
martin@DebianPC:~/C$ ls -l circulo
-rwxr-xr-x 1 martin martin 2245 Jul 29 16:20 circulo
```

El segundo intento de usar `ld`.

Dado que no podíamos ejecutar el programa decidimos intentar volver a añadir algunas de las opciones que no causaban errores.

¹El punto de entrada de un programa es donde se ejecutan las primeras instrucciones y se pasa control al programa.

Primero volvimos a añadir las opciones `-m elf_i386` y `-dynamic-linker /lib/ld-linux.so.2`. La primera opción define el objetivo del compilador². La segunda define la ubicación del enlazador dinámico³ a usar.

Luego de hacer estos cambios logramos ejecutar el programa, que parecía funcionar correctamente. Sin embargo al final de este tuvimos el error **Segmentation fault**. Para tratar de averiguar de donde venía el error decidimos debuggear el programa utilizando `gdb`.

```
martin@DebianPC:~/C$ ld -o circulo -m elf_i386 --dynamic-linker /lib/ld-linux.so.2 circulo.o -lc --entry main
martin@DebianPC:~/C$ ./circulo
Circulo.
Area de circulo de radio 10: 314.160004
Segmentation fault
```

El tercer intento de usar `ld`.

```
martin@DebianPC:~/C$ gdb circulo
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/martin/C/circulo...(no debugging symbols found)...done.
(gdb) r
Starting program: /home/martin/C/circulo
Circulo.
Area de circulo de radio 10: 314.160004

Program received signal SIGSEGV, Segmentation fault.
0x00000001 in ?? ()
```

Nuestro intento de debuggear el programa.

Cuando intentamos debuggear el programa nos encontramos con algo extraño, `gdb` no sabía de que línea provenía el error. Esto nos llevó a creer que provenía de el enlazado del programa, y no del programa en si. Luego de buscar más todavía encontramos el problema, nos faltaba incluir las librerías que regeria el enlazador dinámico. Para entender porque pasa esto hay que entender c'omo funciona el comando.

Lo primero que hace el comando es especificar la ubicación del enlazador dinámico que requieren las demás librerías para acceder a las funciones dinámicas de C. Luego se incluyen otras tres librerías `/usr/lib/i386-linux-gnu/crt1.o`, `/usr/lib/i386-linux-gnu/crti.o`

²El objetivo del compilador es lo que determina que tipo de código objeto debe producir la función.

³Un enlazador dinámico o *dynamic linker* es una forma de enlazar los archivos binarios que se necesitan para que el programa funcione. En este caso el código de las funciones se mantiene en la biblioteca y la hora de ejecutar el programa se cargan en memoria.

y `/usr/lib/i386-linux-gnu/crti.o`. La primera es la librería que tiene referencias a los archivos que requiere el enlazador(`/lib/libc.so.6` y `/usr/lib/libc_nonshared.a`). Las otras dos se encargan de que existan `_init` y `_fini`, que son el código de inicialización y finalización. Algo importante de recordar es que la ubicación de las librerías puede cambiar dependiendo del sistema y la instalación específica. En nuestro caso las encontramos buscando en `/usr/lib` y revisando todas las carpetas que parecían tener alguna relación.

Es importante notar la posición de las librerías, `crti.o` debe ir después de `crt1.o`. Esto es porque este hace referencia al primero. Además ambas deben ir antes de el archivo que se está enlazando. Finalmente `crti.o` va al final del comando, después de todos los demás argumentos.

```
martin@debianPC:~/C$ ld -o circulo -m elf_i386 -dynamic-linker /lib/ld-linux.so.2 /usr/lib/i386-linux-gnu/crti.o circulo.o -lc /usr/lib/i386-linux-gnu/crti.o /usr/lib/i386-linux-gnu/crt1.o
martin@debianPC:~/C$ ./circulo
Circulo.
Area de circulo de radio 10: 314.160004
martin@debianPC:~/C$ _
```

Luego de hacer todo esto el programa finalmente funcionó y se ejecutó de manera correcta y sin errores. Habiendo terminado decidimos que ya teníamos el suficiente conocimiento para intentar compilar un programa usando `make`.