

Una introducción al compilador C de GNU

Héctor Adrián Valdecantos

Departamento de Ciencias de Computación
Facultad de Ciencias Exactas y Tecnología - UNT

Materia:
Programación

Carrera:
Programador Universitario
Licenciatura en Informática

...

27 de agosto de 2010

1. Introducción

GNU es un proyecto iniciado en 1984 por Richard Stallman para crear un sistema operativo libre parecido a Unix. La idea original de este proyecto es promocionar libertad y cooperación entre usuarios de computadoras y programadores. El acrónimo GNU es recursivo y viene de “*GNU is Not Unix*”, y el compilador para C del proyecto GNU se llama “*gcc*”, y su nombre proviene de “*GNU C Compiler*”¹

Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser simplemente texto. Este proceso de traducción se conoce como compilación.[wik]

En este artículo se tratará el proceso de compilación completo, en todas sus etapas, para traducir archivos de código fuente escritos en el lenguaje de programación C usando el programa compilador *gcc* del proyecto GNU.

¹Hoy en día *gcc* también quiere decir “*GNU Collection Compiler*” porque también representa una colección de compiladores.

2. Escribiendo un programa en C

El compilador `gcc` es capaz de compilar cualquier programa en el lenguaje C escrito en un archivo de texto convencional. Por lo tanto es posible usar cualquier editor de texto para crear y editar un programa en C como el que se muestra a continuación:

```
#include <stdio.h>
int main (void)
{
    printf ("Hola, \nmundo!\n");
    return 0;
}
```

Código 1: Programa en C.

El código fuente está escrito en un archivo de texto, estos archivos llevan el sufijo `.c` para identificar que su contenido corresponde al código de un programa en el lenguaje C. Por ejemplo, el código anterior podría estar almacenado en un archivo de nombre `hola.c`.

Por supuesto también existen herramientas más especializadas para escribir y editar código fuente de una manera más eficiente.

3. Compilando un programa en C

Cuando invocamos al compilador mediante el comando `gcc` es posible enviar múltiples opciones y múltiples nombres de archivos como argumentos. Asumiendo que el código fuente se encuentra en un archivo llamado `hola.c`, para compilarlo con `gcc` usamos el siguiente comando:

```
$ gcc -Wall hola.c -o hola
```

Código 2: Comando para compilar.

El comando anterior compila el código fuente de `hola.c` a código máquina y lo almacena en el archivo ejecutable `hola`.

Se pueden ver las siguientes opciones y archivos involucrados en la invocación al compilador:

- `-Wall`
Opción para activar las advertencias² del compilador más comúnmente usadas. Se recomienda que siempre se use esta opción.

²El compilador genera advertencias sobre los elementos del código fuente que pueden llegar a ser errores en la codificación. También son conocidas como *warnings*.

- `-o`
Opción para especificar el archivo de salida. Esta es usualmente la última opción en la línea de comando. Si se omite esta opción, el archivo de salida por defecto es ‘a.out’
- `hola.c`
Nombre del archivo fuente de entrada para ser compilado.
- `hola`
Nombre del archivo de salida. Es el archivo ejecutable.

4. Etapas de la compilación

Cuando invocamos el comando `gcc`, normalmente se realiza un preprocesamiento, compilación, ensamblado, y enlazado³. Estas son las cuatro etapas involucradas en la compilación o traducción de un archivo *fuentes* a un archivo *objeto* ejecutable.

Cada etapa tiene un código *fuentes* como entrada y un código *objeto* como resultado, que en las etapas intermedias sirve de fuente para la etapa siguiente.

Para pasar de un programa fuente escrito por un humano a un archivo ejecutable es necesario realizar estas cuatro etapas en forma sucesiva.[Bar] El comando `gcc` es capaz de realizar todo el proceso de una sola vez, como se vio en el comando anterior (Código 2).

4.1. Preprocesamiento

El preprocesamiento es realizado por el preprocesador o macro-procesador. Es la primera etapa y traduce el archivo fuente que es una forma ampliada del lenguaje, al mismo lenguaje pero en una forma estandarizada para dejar listo el código para la siguiente etapa.

En el siguiente código fuente se calcula el área de un círculo de un radio de 10 cm. Se define en la directiva de preprocesador `#define` la constante `PI` para poder calcular el área.

```
#include <stdio.h>
#define PI 3.1416

int main (void)
{
    float area, radio;
    radio = 10;
    area = PI * (radio * radio);
    printf ("Area_Circulo_=%f\n", area);
    return 0;
}
```

Código 3: circulo.c

³Del inglés: *preprocessing*, *compilation*, *assembly*, y *linking* respectivamente.

El preprocesado del archivo fuente puede realizarse con el siguiente comando:

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

Código 4: Comando para preprocesar un archivo de código fuente.

- **-E**
Opción para detener el proceso de compilación luego de realizado el preprocesamiento. La salida es en la forma de un archivo preprocesado. Los archivos que no requieren preprocesamiento son ignorados.
- **-o**
Opción para especificar el archivo de salida.
- **circulo.c**
Nombre del archivo de entrada para ser preprocesado.
- **circulo.i**
Nombre del archivo de salida. Es el archivo objeto preprocesado, se utiliza el sufijo `‘.i’` para identificar a los archivos preprocesados.

4.2. Compilación

La compilación transforma el código C preprocesado en el lenguaje ensamblador propio del procesador de nuestra máquina.

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

Código 5: Comando para traducir un código C a código assembler.

Con este comando se realiza las dos primeras etapas, creando el archivo `circulo.s` que contiene el programa en lenguaje ensamblador. Es posible realizar sólo la etapa de compilación si el archivo fuente de la línea de comando anterior es un archivo ya preprocesado con sufijo `‘.i’`.

- **-S**
Opción para detener luego de la etapa de compilación, no realiza la etapa de ensamblado. La salida será en la forma de código *assembly*⁴, por defecto, el nombre del archivo de salida será del mismo nombre que el archivo fuente, pero con sufijo `‘.s’`. Los archivos que no requieren compilación son ignorados.

⁴Los lenguaje ensambladores son un tipo de lenguaje de programación de bajo nivel.

- `circulo.c`
Nombre del archivo de entrada para ser compilado.

4.3. Ensamblado

En la etapa de ensamblado se traduce el programa escrito en lenguaje ensamblador de la etapa anterior a código binario en lenguaje de máquina entendible por el procesador.

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

Código 6: Comando para ensamblar.

Con este comando se realiza las tres primeras etapas, creando el archivo `circulo.o`. Este archivo contiene código binario listo para ser enlazado. Es posible realizar sólo la etapa de ensamblado si el archivo fuente de la línea de comando es un archivo ya compilado con sufijo `‘.s’`.

- `-c`
Opción para detener luego de la etapa de ensamblado, no realiza la etapa de enlace. La salida será en la forma de código objeto binario. Por defecto, el nombre del archivo objeto será del mismo nombre del archivo fuente, pero con el sufijo `‘.o’`.
- `circulo.c`
Nombre del archivo de entrada para ser ensamblado.

4.4. Enlazado

Las funciones de C que se usan en el código fuente, como `printf`, se encuentran compiladas y ensambladas en librerías existentes. Para que el archivo resultado del proceso de compilación completo sea ejecutable es necesario incorporar el código binario de estas funciones en el ejecutable final. De esto se trata la etapa de enlace o *linking*.

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

Código 7: Comando para enlazar.

Con este comando se realiza el proceso de compilación completo, creando el archivo `circulo` ejecutable. Este archivo contiene código binario enlazado con las funciones de librería que utiliza, listo para ser ejecutado. Es posible realizar sólo la última etapa si pasamos un archivo con código binario ensamblado con sufijo `‘.o’`.

Esta es la línea de comando que comúnmente se usará para compilar un archivo en C para obtener un archivo ejecutable, agregándole la opción `-Wall` como se vio en la Sección 3.

5. Corrección y pruebas

Las advertencias o *warnings* del compilador `gcc` son de ayuda esencial cuando programamos. Se mostrará esto a través de un programa en C con errores:

```
1 #include <stdio.h>
2 int
3 main (void)
4 {
5     int suma 4;
6     printf ("Dos más dos es %f\n", suma);
7     return 0;
8 }
```

Código 8: Programa en C con errores (mensaje.c).

Al compilar el archivo `mensaje.c` se puede ver que el resultado no es el archivo ejecutable que esperábamos, sino una lista de errores y advertencias:

```
$ gcc -Wall mensaje.c -o mensaje
mensaje.c: In function 'main':
mensaje.c:5: error: expected '=', ',', ';', 'asm' or '__attribute__' before
numeric constant
mensaje.c:5: warning: statement with no effect
mensaje.c:6: error: 'suma' undeclared (first use in this function)
mensaje.c:6: error: (Each undeclared identifier is reported only once
mensaje.c:6: error: for each function it appears in.)
```

Código 9: Compilando un programa con errores.

El formato del mensaje producido por el compilador `gcc` ante una advertencia o error tiene siempre la forma *archivo:línea:mensaje*.

Podemos ver que existe un error en el archivo `mensaje.c`, en la función `main` de ese archivo, en la línea 5, donde se esperaba un '='... Procedemos a corregirlo:

```
1 #include <stdio.h>
2 int
3 main (void)
4 {
5     int suma = 4;
6     printf ("Dos más dos es %f\n", suma);
7     return 0;
8 }
```

Código 10: Programa en C con errores (Primera corrección).

Corregido el error, procedemos a compilar de nuevo:

```
$ gcc -Wall mensaje.c -o mensaje
mensaje.c: In function 'main':
mensaje.c:6: warning: format '%f' expects type 'double', but argument 2
has type 'int'
```

Código 11: Compilando un programa con errores.

Por desgracia existe un error más en el código. Este error no es tan fácil de ver como el anterior, pero el compilador nos advierte generando un *warning* en el proceso de compilación. Este mensaje indica que el uso de la cadena de formato fue usada incorrectamente en la línea 6.

El compilador distingue entre errores que impide el proceso de compilación se complete, y mensajes de advertencia que indican un posible error pero no detienen la compilación. Si ejecutamos el archivo resultado de la compilación obtendremos resultados inesperados:

```
$ ./mensaje
Dos más dos es 0.000000
```

Código 12: Compilando un programa con errores.

En este caso, el error es que el especificador de formato en el código es el incorrecto, ya que con `%f` se está esperando un valor del tipo `float` y la variable `suma` es del tipo `int`. Números enteros y de punto flotantes son almacenados en diferentes formatos en la memoria, y generalmente ocupan diferentes cantidades de bytes.

```
1 #include <stdio.h>
2 int
3 main (void)
4 {
5     int suma = 4;
6     printf ("Dos más dos es %d\n", suma);
7     return 0;
8 }
```

Código 13: Programa en C (Segunda corrección).

Habiendo corregido el error, podemos volver a compilar el archivo, esta vez exitosamente, sin ningún mensaje de error, ni mensaje de advertencia, para luego proceder a ejecutarlo:

```
$ gcc -Wall mensaje.c -o mensaje
$ ./mensaje
Dos más dos es 4
```

Código 14: Compilando un programa con errores.

Se puede advertir que la ejecución del archivo ya compilado muestra el resultado esperado. Esta es la forma en que comúnmente se probarán los programas para encontrar los errores y poder corregirlos.

6. Estándares del lenguaje C

Por defecto, `gcc` compila programas usando el dialecto GNU del lenguaje C, conocido como GNU C. Este dialecto incorpora el estándar oficial ANSI/ISO para el lenguaje C con algunas extensiones útiles propias de GNU. La gran mayoría de programas escritos con el estándar ANSI/ISO compilarán sin problemas con `gcc`. Aun así, el compilador de GNU dispone de la opción `-ansi` para realizar una compilación teniendo en cuenta únicamente el estándar ANSI/ISO cuando pueda haber alguna incompatibilidad con las extensiones de GNU C.

7. Conclusión

En las últimas etapas del proceso de programación, después del análisis del problema y el diseño algorítmico de la solución, el programador debe manipular los elementos constructivos del lenguaje mediante un editor para escribir su programa. También debe manejar la herramienta fundamental que le permitirá traducir el archivo que almacena el programa escrito en un lenguaje de alto nivel a un archivo de código máquina para que la máquina sea capaz de ejecutarlo.

La etapa de traducción, que en sí es el proceso de compilación, le permitirá al programador realizar las correcciones adecuadas en su programa, y luego, a partir del archivo ejecutable lograr realizar las pruebas y verificación del programa.

Referencias

- [Bar] Víctor A. González Barbone. *El compilador GCC*. Instituto de Ingeniería Eléctrica. Facultad de Ingeniería. Montevideo, Uruguay. Recurso online: <http://iie.fing.edu.uy/vagonbar/gcc-make/gcc.htm>.
- [gnu] *Official GNU website*. Recurso online: <http://www.gnu.org/>.
- [Gou04] Brian Gough. *An Introduction to GCC*. Network Theory Limited, 1st edition, 2004.
- [StDCG] Richard M. Stallman and the Developer Community GCC. *Using the GNU Compiler Collection*. GNU Press.
- [wik] Wikipedia. Recurso online: <http://www.wikipedia.org/>.