

Práctica 0 — Etapas de Compilación

Compiladores — Semestre 2026-2

Hector Valdes

1. Introducción

En esta práctica se exploran las etapas del proceso de compilación de un programa escrito en lenguaje C, desde el preprocesamiento hasta el enlazado. Se utiliza cada herramienta de forma individual para observar las transformaciones que sufre el código fuente en cada fase.

2. Desarrollo

2.1. Preprocesamiento (cpp)

Se ejecutó el siguiente comando para generar el archivo preprocesado:

```
cpp programa.c programa.i
```

a) ¿Qué ocurre cuando se invoca el comando `cpp` con esos argumentos?

`cpp` es un procesador de macros, particularmente para lenguajes de programación de la familia de C. Este programa, nos ayuda a interpretar directivos de procesamiento, por ejemplo **#include**, **#define**, **#ifdef**, **etc..** Por lo tanto, cuando ejecutamos el comando `cpp`. Este, lee el documento de entrada (*programa.c*) y los documentos que hayan sido incluidos por el directivo **#include** y los escribe con la información extra generada por los directivos en el programa de salida, *programa.i*.

b) ¿Qué similitudes encuentra entre los archivos `programa.c` y `programa.i`?

En el archivo, `programa.i`, al final, se puede ver el siguiente código: En la figura 1, podemos ver que las sentencias han sido limpiadas de comentarios, y también el macro **area** ha sido remplazado con su definición. Pero ambos son programas que saludan al mundo e imprimen el área de un círculo con radio 3.

```
int main ( void ) {  
    printf ("Hola Mundo !\\n");  
    float mi_area = (r) (3.1416 * r * r) (3) ;  
    printf ("Resultado : %f\\n", mi_area);  
    return 0;  
}
```

Figura 1: *Parte final del archivo programa.i*

```

int main ( void ) {
    printf ("Hola Mundo !\n"); //Funcion para imprimir hola mundo
    float mi_area = area (3) ; //soy un comentario... hasta donde
        llegare ?
    printf ("Resultado : %f\n", mi_area);
    return 0;
}

```

Figura 2: Estracto de *programa.c*

```

# ifdef PI
# define area (r) (PI * r * r)
# else
# define area (r) (3.1416 * r * r)
# endif

```

Figura 3: Definicion de Area en *programa.c*

- c) ¿Qué pasa con las macros y los comentarios del código fuente original?

Los macros, son remplazados de manera textual en todas las llamadas de este. Por ejemplo, recordemos la definicion de area: Asi, notemos que como no tenemos un directivo que defina PI, se toma la definicion la segunda definicion, la cual podemos ver que es la usada en la Figura 1. Y los comentarios son eliminados, dado que no son necesarios para la etapa de compilacion, solo para el programador.

- d) Compare el contenido de **programa.i** con el de **stdio.h** e indique de forma general las similitudes entre ambos archivos.

Debido a que hay muchas funciones de **stdio.h** en **programa.i** sin directivos de compilacion condicional, ni otro directivo en general. Pienso que *cpp* hace un preprocesado en **stdio.h** y la salida de este preprocesado solo es remplazado con el directivo de **#include**. Por lo tanto las similitudes, son las definiciones de ciertas funciones.

2.2. Compilación (gcc -S)

Se ejecutó la siguiente instrucción:

```
gcc -Wall -S programa.i
```

- a) ¿Para qué sirve la opción **-Wall**?

Para que el sistema de procesamiento *gcc*, muestre todas las advertencias(Warnings) al construir el codigo objeto.

- b) ¿Qué le indica a *gcc* la opción **-S**?

Que detenga su ejecucion despues de haber compilado, antes de ensamblar.

- c) ¿Qué contiene el archivo de salida y cuál es su extensión?

Su extension es **.s** y contiene instrucciones en lenguaje ensamblador generadas con ayuda del archivo *programa.i*.

2.3. Ensamblado (as)

Se ejecutó la siguiente instrucción:

```
as programa.s -o programa.o
```

- a) Antes de revisarlo, indique cuál es su hipótesis sobre lo que debe contener el archivo con extensión `.o`.

Yo pienso que nos dará el código máquina en bytes de nuestro programa escrito en ensamblador.

- b) Diga de forma general qué contiene el archivo `programa.o` y por qué se visualiza de esa manera.

Contiene lo que yo creo son bytes y strings que parecen ser etiquetas de lenguaje ensamblador. Estas etiquetas son llamadas directivos e indican instrucciones al momento de el ensamblado.

- c) ¿Qué programa se invoca con `as`? El ensamblador de GNU. Este programa permite ensamblar código ensamblador de diferentes arquitecturas en una pasada.

2.4. Enlazado (ld)

Se buscó la ruta de los siguientes archivos en el equipo de trabajo:

- `ld-linux-x86-64.so.2`
- `Scrt1.o` (o bien, `crt1.o`)
- `crti.o`
- `crtbeginS.o`
- `crtendS.o`
- `crtn.o`

Se ejecutó el siguiente comando:

```
ld -o ejecutable -dynamic-linker /lib/ld-linux-x86-64.so.2 \
/usr/lib/Scrt1.o /usr/lib/crti.o programa.o \
-lc /usr/lib/crtn.o
```

- a) En caso de que el comando `ld` mande errores, investigue cómo enlazar un programa utilizando `ld` y proponga una posible solución para llevar a cabo este proceso con éxito.

- b) Describa el resultado obtenido al ejecutar el comando anterior.

Generamos el código en lenguaje máquina, finalmente. El argumento `-o` indica el nombre que le daremos al archivo de salida en este caso **ejecutable**. El siguiente argumento, `-dynamic-linker` es una flag que indica que enlazador usaremos, en este caso `/lib/ld-linux-x86-64.so.2`. Los siguientes argumentos son archivos objeto. `Scrt1.o` `crti.o` `crtn.o` son encargados de crear rutinas para la apertura y el cierre del ejecutable. Después, viene el archivo objeto creado por nosotros. En esta parte, también vemos la bandera `-lc` la cual nos ayuda a enlazar **libc**, la librería estándar de C.

2.5. Ejecución

```
./ejecutable
```

Figura 4: Comando para ejecutar el programa ejecutable

```
Hola Mundo !  
Resultado : 28.274401
```

Figura 5: Salida de el programa ejecutable

2.6. Modificación de la macro `#define` PI

Se quitó el comentario de la macro `#define` PI en el código fuente original y se realizó lo siguiente:

- a) Se generó nuevamente el archivo `.i` con un nuevo nombre.
- b) ¿Cambia en algo la ejecución final?

2.7. Segundo programa con directivas del preprocesador

Se escribió un segundo programa en lenguaje C que incluye 4 directivas del preprocesador distintas entre sí y diferentes de las utilizadas en el primer programa.¹

- a) Explique la utilidad general de cada directiva y su función particular en el programa.

3. Resultados y Conclusiones

¹Puede consultarse la lista de directivas en la documentación en línea: <https://gcc.gnu.org/onlinedocs/cpp/Index-of-Directives.html>. También se puede revisar la entrada para el preprocesador con `man cpp`.