Desarrollo de Compilación en C con GCC en Linux

Márquez Corona Danna Lizette

1 Preprocesamiento

Ejecute el siguiente comando:

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

Use el siguiente comando: cpp programa.c programa.i

- ¿Qué ocurre cuando se invoca el comando `cpp` con esos argumentos? Se crea "programa.i" de donde toma el codigo de "programa.c" y procesa los macros, además quita algunas cosas no útiles como los comentarios.

- ¿Qué similitudes encuentra entre los archivos `programa.c` y `programa.i`? En general comparten el mismo codidgo pero "programa.i" tiene más cosas y a la vez, como mencione anteriormente, quita cosas no utiles como los comentarios. En lo que se parecen es que en la ultima sección de "programa.i" hay una parte de codigo muy parecida a la de "programa.c".

- ¿Qué pasa con las macros y los comentarios del código fuente original en `programa.i`? En "programa.i" los macros se procesan y se eliminan comentarios de "programa.c"

- Compare el contenido de `programa.i` con el de `stdio.h` e indique de forma general las similitudes entre ambos archivos. Podemos ver que todo stdio.h es un subconjunto de "programa.i", ya que "programa.i" importo stdio.h y todo su contenido lo pego en ese programa.

- ¿A qué etapa corresponde este proceso? Preprocesamiento

2 Compilación a Ensamblador

Ejecute:

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

Responda:

- ¿Para qué sirve la opción `-Wall`? Habilita todas las advertencias comunes durante la compilación.
- ¿Qué le indica a `gcc` la opción `-S`? Hace que solo haga la fase de compilación y se detenga antes del ensamblaje.
- ¿Qué contiene el archivo de salida y cuál es su extensión? el "programa.s" contiene el "programa.c" pero en ensamblador, ya que ya se tradujo.
- ¿A qué etapa corresponde este comando? compilación

3 Generación de Código Máquina

Ejecute:

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

Responda:

- ¿Cuál es su hipótesis sobre el contenido del archivo `.o`? Contiene código máquina relocalizable, aun falta especificar sus direcciones especificas de memoria y enlazar sus librerias requeridas.
- ¿Qué contiene el archivo `programa.o` y por qué se visualiza así? Afirmativamente contiene código maquina y se ve así porque es un archivo binario, por lo que hay simbolos no legibles.
- ¿Qué programa se invoca con `as`? Ensamblador
- ¿A qué etapa corresponde este proceso? Ensamblaje

4 Enlazado

Encuentre la ruta de los siguientes archivos en su sistema:

`Scrt1.o, crti.o, crtbeginS.o, crtendS.o, crtn.o`

- `scrt1.o`: `/usr/lib/x86_64-linux-gnu/Scrt1.o`
- `crti.o`: `/usr/lib/x86_64-linux-gnu/crti.o`
- `crtbeginS.o`: `/usr/lib/gcc/x86_64-linux-gnu/13/crtbeginS.o`
- `crtendS.o`: `/usr/lib/gcc/x86_64-linux-gnu/13/crtendS.o`
- `crtn.o`: `/usr/lib/x86_64-linux-gnu/crtn.o`

Ejecute el siguiente comando:

```
ld -o ejecutable -dynamic-linker /usr/lib/x86_64-linux-gnu/ld-linux-x8
/usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o progr
```

Si el comando falla, investigue cómo enlazar correctamente con `ld` y proponga una solución. Describa el resultado obtenido.

5 Ejecución

Una vez enlazado, ejecute:

```
./programa
```

Nos da: Hola Mundo ! Resultado : 28.274401

6 Modificación del Código

Quite el comentario de la macro `#define PI` y genere nuevamente `programa.i`. Responda:

- ¿Cambia en algo la ejecución final? Si, ahora nos da un resultado diferente: Hola Mundo! Resultado : 28.274334

7 Segundo Programa

Escriba otro programa en C con cuatro directivas del preprocesador distintas de las anteriores. Explique su utilidad y función dentro del programa.

```
#include <stdio.h>
#include <stdlib.h>
#define CUBO 3
#undef CUBO
#ifndef CUBO
```

```
    #define CUBO 3
#endif
#if CUBO < 3
    #error "Queremos que el cubo sea 3"
#elif CUBO > 3
    #error "Tampoco queremos que sea mayor a 3"
#endif
#define cubo(r) ((r)*(r)*(r))

/**
* Compiladores 2025-1
*
*/
int main ( void ) {
printf ("Adios Mundo !\n"); //Función para imprimir hola mundo
float mi_cubo = cubo(3) ;
printf ("Resultado : %f\n", mi_cubo);
return 0;
}
```

Calcula el cubo de 3 y tenemos las directivas `#undef`, `#ifndef`, `#error` y `#elif`, y lo que hace es que `#undef` elimina la definición de una macro, `#ifndef` verifica si una macro no está definida. `#ifndef` / `#elif` / `#error` realiza una verificación condicional y puede generar un error si la condición se cumple, y finalmente `#define cubo(r) ((r)*(r)*(r))`, la cual define una macro para calcular el cubo de un número.
Su salida es `Adios Mundo!  Resultado:  27.000000`.

8 Conclusión

Redacte un informe detallado con los resultados obtenidos y las conclusiones derivadas del proceso.
A lo largo de este experimento, exploramos en detalle las distintas fases del proceso de compilación en C utilizando gcc. Comenzamos con el preprocesamiento, donde se procesan las macros y se eliminan comentarios, luego analizamos la compilación a ensamblador, obteniendo un archivo con instrucciones en lenguaje ensamblador.
Luego transformamos el código ensamblador en código máquina lo que nos da una representación binaria de las instrucciones. Luego, en el enlazado, integramos este archivo con bibliotecas y otros archivos necesarios.
Finalmente lo ejecutamos.

Este proceso nos permitió entender la importancia de cada una de las etapas de compilación, además nos ayudo a aprender el uso de herramientas como cpp, gcc, as y ld, esenciales para la generación de programas en C.