Compiladores 2025-1

Profesora: Ariel Adara Mercado Martínez
Ayudante: Carlos Gerardo Acosta Herández

Práctica 0: Sistema de Procesamiento de Lenguaje.

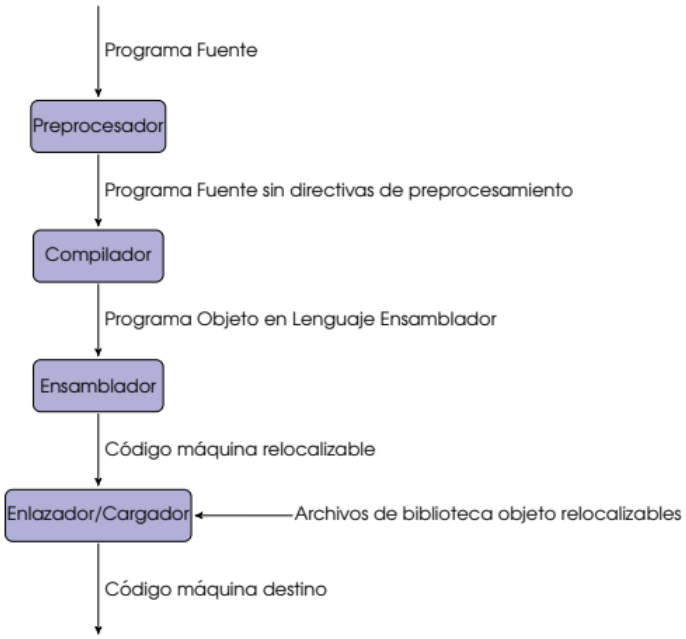
Nombre	**No. de Cuenta**
Vázquez Torrijos Damián	318309877

22 Agosto del 2024

Introducción

Un compilador es un programa que traduce un programa fuente escrito en un lenguaje de alto nivel a un programa en lenguaje objeto, usualmente de bajo nivel. Este proceso no se realiza de manera aislada; el compilador colabora con otros programas como el preprocesador, ensamblador y enlazador.

El preprocesador recopila y expande macros y otros fragmentos de código abreviado en el programa fuente. Luego, el compilador transforma el código preprocesado en un programa objeto en lenguaje ensamblador, que es posteriormente convertido en código máquina por el ensamblador. Finalmente, el enlazador combina los archivos de código máquina y bibliotecas necesarias para producir un programa ejecutable. El cargador lleva este programa ejecutable a la memoria para su ejecución.



Desarrollo

Se instaló el **compilador gcc** mediante la terminal de Ubuntu con ayuda del comando: **sudo apt install build-essential**, el paquete instalado incluye el compilador anteriormente mencionado.

Una vez instalado, se prosiguió a escribir el programa en lenguaje C y se le asignó el nombre de 'programa.c'

```c
#include <stdio.h>
#include <stdlib.h>
//#define PI 3.1415926535897
#ifdef PI
#define area(r) (PI*r*r)
#else
#define area(r) (3.1416*r*r)
#endif
/*
* Este es un programa de prueba, para verificar
* el funcionamiento del sistema de procesamiento de lenguaje
*/
int main(void){
    printf("Hola Mundo!\n");    // Funcion para imprimir hola mundo
    float mi_area = area(3);    //Soy un comentario ...
    printf("Resultado: %f\n", mi_area);
    return 0;
}
```

Se usó el comando **cpp programa.c > programa.i** para ejecutar el preprocesador; se pudo localizar el archivo 'stdio.h' con la ayuda del comando **find**.

```
fritush@DESKTOP-ASC7QFC:~$ find /usr/ -name stdio.h
/usr/include/c++/11/tr1/stdio.h
/usr/include/x86_64-linux-gnu/bits/stdio.h
/usr/include/stdio.h
```

Para comparar el contenido de 'programa.i' con 'stdio.h', se utilizó el comando **cat**; por el lado de 'stdio.h', se pudo ver la definición de varias funciones para posibles estados en el preprocesador, mientras que en 'programa.i' solo tenia la opción de llamar a dichas funciones en caso de necesitarlas, de hecho, hace uso de la posición en 'stdio.h' y la ruta '/usr/include/stdio.h' después de llamar a dichas funciones.

Se pudo ver que, además de pasar lo que se tenía en programa.c a programa.i, durante la ejecución del comando **cpp**, el preprocesador busca en las rutas estándar de búsqueda, todas las funciones, definiciones y archivos de cabecera incluidos en el código fuente en 'stdio.h' para poder controlar todas las excepciones que puedan pasar durante la marcha y las pone en nuestro archivo 'programa.i'.

Se compiló el archivo preprocesado 'programa.i' usando el comando **gcc -Wall -S programa.i**; la función de la opción **-Wall** es pedirle al compilador que sea estricto, es decir, que ayude a detectar todo tipo de errores y excepciones en el código. Por el lado de la opción **-S**, se tiene que su propósito es general código en lenguaje ensamblador. Por lo que el archivo de salida contiene instrucciones en lenguaje ensamblador y su extensión es .s; el programa invocado por **gcc** al usar la opción **-S** es el compilador.

```
.LC1:
    .long   1105342969
    .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long   1f - 0f
    .long   4f - 1f
    .long   5
```

Se ensambló el archivo 'programa.s' usando el comando **as programa.s -o programa.o**; basándose en la introducción de la práctica, se espera que el archivo 'programa.o' contenga código máquina. En cuanto al contenido del archivo 'programa.o', se puede decir que en efecto, es código máquina, ya que es algo que un humano puede leer pero es difícil o incluso imposible de interpretar; es importante porque serán las instrucciones para la recta final.

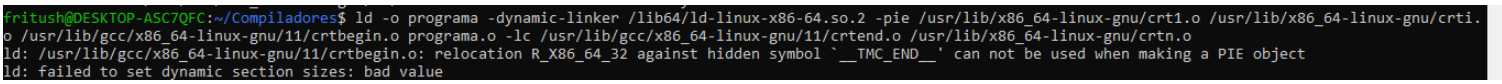
El programa al que se está llamando con el comando **as** es el ensamblador.

Se encontraron las rutas en el sistema para los siguiente archivos, de nuevo, usando el comando **find**

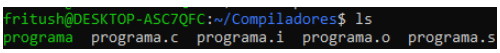
- crt1.o - '/usr/lib/x86_64-linux-gnu/crt1.o'
- crti.o - '/usr/lib/x86_64-linux-gnu/crt1.o'
- crtbegin.o - '/usr/lib/gcc/x86_64-linux-gnu/crt1.o'
- crtend.o - '/usr/lib/gcc/x86_64-linux-gnu/crt1.o'
- crtn.o - '/usr/lib/x86_64-linux-gnu/crt1.o'

Usando las rutas anteriormente mencionadas, se enlazó el programa ejecutando el siguiente comando: **ld -o programa -dynamic-linker /lib64/ld-linux-x86-64.so.2 -no-pie /usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/11/crtbegin.o programa2.o -lc /usr/lib/gcc/x86_64-linux-gnu/11/crtend.o /usr/lib/x86_64-linux-gnu/crtn.o**

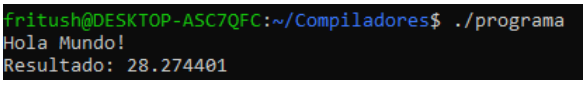
Originalmente se mantendría la estructura proporcionada, pero el comando **ld** arrojó un error del tipo **bad value**:



Por lo que después de investigarlo en foros, el problema se solucionó cambiando el comando **-pie** por **-no-pie**. Obtuvimos literalmente un **programa**, sin extensión alguna, pero en la terminal se mostraba con un color diferente.



Una vez que se enlazó el código máquina relocalizable, se pudo ejecutar el programa con la instrucción en la terminal **./programa**



Después de quitar el comentario en el macro #**define PI**, se generó el archivo 'programa2.i':

programa.c

```c
#include <stdio.h>
#include <stdlib.h>
#define PI 3.1415926535897
#ifdef PI
#define area(r) (PI*r*r)
#else
#define area(r) (3.1416*r*r)
#endif
/*
* Este es un programa de prueba, para verificar
* el funcionamiento del sistema de procesamiento de lenguaje
*/
int main(void){
    printf("Hola Mundo!\n");    // Funcion para imprimir hola mundo
    float mi_area = area(3);    //Soy un comentario ...
    printf("Resultado: %f\n", mi_area);
    return 0;
}
```

Al final, se nota un cambio en la ejecución final:

```
fritush@DESKTOP-ASC7QFC:~/Compiladores$ ./programa
Hola Mundo!
Resultado: 28.274334
```

En la segunda linea, de generar 'Resultado: 28.274401' pasó a generar 'Resultado: 28.274334'
Se escribió un programa que genero un número "aleatorio" dentro de un rango definido:

punto11.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100
#ifndef MIN
#define MIN 1
#endif
#if MAX > 50
#define RANGO "amplio"
#else
#define RANGO "estrecho"
#endif
#line 13 "punto11.c"

/*
* Este programa genera un número aleatorio dentro de un rango
* segun el tamaño del rango, menciona si es un rango amplio o estrecho
*/
int main(void) {
    // Inicializar la semilla para generar números aleatorios
    srand(time(NULL));

    // Generar un número aleatorio entre MIN y MAX
    int num_aleatorio = MIN + rand() % (MAX - MIN + 1);

    printf("Número aleatorio entre %d y %d: %d\n", MIN, MAX, num_aleatorio);
    printf("Rango de números aleatorios: %s\n", RANGO);
    return 0;
}
```

```
fritush@DESKTOP-ASC7QFC:~$ cd Compiladores/
fritush@DESKTOP-ASC7QFC:~/Compiladores$ cpp punto11.c > punto11.i
fritush@DESKTOP-ASC7QFC:~/Compiladores$ gcc -Wall -S punto11.i
fritush@DESKTOP-ASC7QFC:~/Compiladores$ as punto11.s -o punto11.o
fritush@DESKTOP-ASC7QFC:~/Compiladores$ ld -o punto11 -dynamic-linker /lib64/ld-linux-x86-64.so.2 -no-pie /usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crt
i.o /usr/lib/gcc/x86_64-linux-gnu/11/crtbegin.o punto11.o -lc /usr/lib/gcc/x86_64-linux-gnu/11/crtend.o /usr/lib/x86_64-linux-gnu/crtn.o
fritush@DESKTOP-ASC7QFC:~/Compiladores$ ls
programa    programa.i  programa.s   programa2.o  punto11    punto11.i  punto11.s
programa.c  programa.o  programa2.i  programa2.s  punto11.c  punto11.o  stdio.tex
fritush@DESKTOP-ASC7QFC:~/Compiladores$ ./punto11
Número aleatorio entre 1 y 100: 92
Rango de números aleatorios: amplio
fritush@DESKTOP-ASC7QFC:~/Compiladores$ ./punto11
Número aleatorio entre 1 y 100: 44
Rango de números aleatorios: amplio
fritush@DESKTOP-ASC7QFC:~/Compiladores$ ./punto11
Número aleatorio entre 1 y 100: 29
Rango de números aleatorios: amplio
fritush@DESKTOP-ASC7QFC:~/Compiladores$
```

1. En este caso usamos las directivas:

   - #define - Define un valor constante o una macro que se reemplaza en el código fuente antes de la compilación.
   - #ifndef - Verifica si una constante o una macro está definida, y si no lo está, define una nueva constante o macro.
   - #if - Evalúa una condición y compila código según sea necesario.
   - #line - Especifica el número de línea y el archivo de origen para el código subsecuente.

   Dentro del programa, se hace uso de '#define MAX 100' para definir una constante MAX que será la cota superior del número que se generará aleatoriamente, '#ifndef MIN' verifica si la constante MIN está definida, si no lo está, define un valor constante MIN con un valor de 1, '#if MAX $>$ 50' evalúa la condición y define una constante RANGO con un valor de "amplio" si se cumple la condición, si no, se define con un valor de "estrecho", '#line 13 "punto11.c"' especifica el número de la línea y el archivo de origen para el código restante, lo que puede ser útil en la depuración y seguimientos de errores.

Conclusión

Además de aprender a moverse en la terminal de Ubuntu, se pudo comprender y analizar el funcionamiento de los diferentes programas que intervienen en el proceso de traducción de un programa fuente a un programa ejecutable. Al final del día pareciera un proceso rápido y un tanto sencillo, pero hay muchas cosas atrás, es más de lo que uno ve.