Universidad Nacional Autónoma de México

Facultad de Ciencias

Compiladores. 2025-1 | 7008

Práctica 0: Sistemas de procesamiento de lenguaje.

Kevin Steve Quezada Ordoñez

stevequezada@ciencias.unam.mx

22/08/2024

1. Instale el compilador `gcc` y asegurese de trabajar en un entorno Linux.

```
sudo dnf install gcc
```

2. Escriba el siguiente programa en lenguaje `C` y guárdelo como `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;
}
```

3. Use el siguiente comando para ejecutar el preprocesador: `cpp programa.c > programa.i`

   (*a*) Localice el archivo `stdio.h` en su sistema y examine su contenido.

```
sudo find / -name "stdio.h"
```

```
/usr/include/stdio.h
```

   (*b*) Compare `programa.i` con `stdio.h`. Describa las similitudes mas notables entre ambos archivos.

   Usando el programa `Meld` para analizar las diferencias de los archivos se obtuvo que el contenido de `stdio.h` se encuentra en `programa.i`, este muestra el resultado de las directivas de preprocesador, en donde se incluyen todas las funciones, macros y declaraciones.

- `stdio.h`

```
extern int vsnprintf (char *__restrict __s, size_t __maxlen,
                      const char *__restrict __format,
                          __gnuc_va_list __arg)__THROWNL __attribute__
                              ((__format__ (__printf__, 3, 0)));}
```

- `programa.i`

```
extern int vsnprintf (char *__restrict __s, size_t __maxlen,}
                      const char *__restrict __format, __gnuc_va_list __arg)
                          __attribute__ ((__nothrow__)) __attribute__
                              ((__format__ (__printf__, 3, 0)));
```

(*c*) Explique que ocurre durante la ejecución del comando `cpp`.

El preprocesador realiza la expansión de macros, la inclusión del contenido de archivos de cabecera, la eliminación de comentarios y el procesamiento de directivas condicionales. Al final del proceso se obtiene un archivo que contiene el código fuente para compilar.

4. Compile el archivo preprocesado usando el comando: `gcc -Wall -S programa.i`

(*a*) Describa la función de la opción `-Wall`.

Activa las advertencias del compilador, como variables sin inicializar o funciones declaradas pero no utilizadas.

(*b*) Explique el propósito de la opción `-S`.

Indica que el proceso de compilación debe detenerse después de generar el código ensamblador.

(*c*) Indique que contiene el archivo de salida y cuál es su extensión.

El archivo de salida contiene el código ensamblador correspondiente al código fuente en `C`. La extensión del archivo es `.s`.

(*d*) Identifique que programa es invocado por `gcc` al usar la opción `-S`.

Invoca su compilador interno, que traduce el código `C` en instrucciones de ensamblador.

5. Ensamble el archivo usando: `as programa.s -o programa.o`

   (*a*) Explique que se espera que contenga el archivo. `o`.

   Un `archivo .o` es un archivo objeto generado por el ensamblador que contiene el código máquina del código ensamblador que estaba en `programa.s`. Este archivo es el producto intermedio entre el código fuente y el ejecutable final.

   (*b*) Describa de manera general el contenido del archivo `.o` y por que es importante.

   El Código máquina, la información sobre las funciones y variables definidas y referenciadas en el código y los bloques de memoria

   Son importantes porque permiten una compilación eficiente y modular, facilitan el proceso de vinculación y optimización.

   (*c*) Identifique que programa se invoca con el comando `as`.

   Invoca el ensamblador de `GNU`, convierte el código fuente escrito en lenguaje ensamblador en código máquina específico para esa arquitectura de procesador.

6. Encuentre las rutas en su sistema para los siguientes archivos:

   - crt1.o

     ```
     sudo find / -name crt1.o
     ```

     ```
     /usr/lib64/crt1.o
     ```

   - crti.o

     ```
     sudo find / -name crti.o
     ```

     ```
     /usr/lib64/crti.o
     ```

   - crtbegin.o

     ```
     sudo find / -name crtbegin.o
     ```

     ```
     /usr/lib/gcc/x86_64-redhat-linux/14/crtbegin.o
     ```

   - crtend.o

     ```
     sudo find / -name crtend.o
     ```

     ```
     /usr/lib/gcc/x86_64-redhat-linux/14/crtend.o
     ```

   - crtn.o

     ```
     sudo find / -name crtn.o
     ```

     ```
     /usr/lib64/crtn.o
     ```

7. Enlazar el programa ejecutando el siguiente comando, reemplazando las rutas:

8. Ejecute el siguiente comando, sustituyendo las rutas que encontro en el paso anterior:

```
ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie
    /ruta/para/Scrt1.o
        /ruta/para/crti.o
            /ruta/para/crtbe- ginS.o -L/usr/lib/gcc/x86 64-pc-linux-gnu/7.3.1 -L
            /usr/lib -L/lib -L/usr/lib -L/usr/lib programa.o -lgcc -as- needed
            -lgcc s -no-as-needed -lc -lgcc -as-needed -lgcc s -no-as-needed
                /ruta/para/crtendS.o /ru- ta/para/crtn.o -o ejecutable
```

o bien

```
ld -o programa -dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie /usr/lib/Scrt1.o
    /ruta/para/crti.o
        /ruta/para/crt- beginS.o programa.o -lc
            /ruta/para/crtendS.o
                /ruta/para/crtn.o
```

(*a*) Si el comando ld arroja errores, investigue como enlazar un programa utilizando `ld`.

Tuve un problema al usar la opción `-pie`, usando `no-pie`, creando así un ejecutable con una ubicación fija en memoria.

```
ld -o programa -dynamic-linker /lib64/ld-linux-x86-64.so.2 -no-pie
    /usr/lib64/crt1.o
        /usr/lib64/crti.o
            /usr/lib/gcc/x86_64-redhat-linux/14/crtbegin.o programa.o -lc
                /usr/lib/gcc/x86_64-redhat-linux/14/crtend.o
                    /usr/lib64/crtn.o
```

(*b*) Describa el resultado obtenido al ejecutar el comando anterior.

Se generó un archivo ejecutable enlazado dinámicamente.

9. Una vez que se enlazo el código máquina relocalizable, podemos ejecutar el programa con la siguiente instrucción en la terminal: ./programa

```
./programa

Hola Mundo!
Resultado: 28.274401
```

10. Quite el comentario de la macro #define PI

```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;
}
```

(*a*) Genere nuevamente el archivo.i. De preferencia asigne un nuevo nombre.

```
cpp programaModificado.c >programaModificado.i

gcc -Wall -S programaModificado.i

as programaModificado.s -o programaModificado.o

ld -o programaModificado -dynamic-linker /lib64/ld-linux-x86-64.so.2 -no-pie
/usr/lib64/crt1.o
 /usr/lib64/crti.o
  /usr/lib/gcc/x86_64-redhat-linux/14/crtbegin.o programaModificado.o -lc
   /usr/lib/gcc/x86_64-redhat-linux/14/crtend.o
    /usr/lib64/crtn.o

./programa

Hola Mundo!
Resultado: 28.274334
```

(*b*) Describa el resultado obtenido al ejecutar el comando anterior.

Se obtuvo una diferencia entre los resultados debida a la precisión del valor de π. En el archivo original, el programa usa una aproximación de 3,1416. Por otro lado, en el archivo modificado, se utiliza un valor más preciso de 3,1415926535897. El segundo archivo tiene una mayor precisión, lo que provoca un un cálculo más exacto.

11. Escribe un segundo programa en lenguaje C

    En el agregue 4 directivas del preprocesador de C (cpp)*. Las directivas elegidas deben jugar algún papel en el significado del programa.

(*a*) Explique la utilidad general de las directivas usadas y su funcion en particular para su programa.

```c
/*
 * Este programa muestra el uso de directivas del preprocesador en C.
 * Se define un arreglo, se llena con valores naturales consecutivos
 * y se multiplican por un valor determinado.
 * Finalmente, se imprimen los valores originales y los multiplicados.
 */

#include <stdio.h>

#define TAMAÑO 5

#ifndef MÍNIMO
#define MÍNIMO 1
#endif

#if TAMAÑO > MÍNIMO
#define MULTIPLICADOR 2
#else
#define MULTIPLICADOR 1
#endif

#if MULTIPLICADOR > 1
#warning "El valor de MULTIPLICADOR es mayor a 1, lo cual puede afectar el resultado."
#endif

#if TAMAÑO <= 0
#error "El tamaño del arreglo debe ser mayor que 0!"
#endif

#pragma GCC optimize ("O2")

int main()
{
    int numeros[TAMAÑO];

    for (int i = 0; i < TAMAÑO; i++)
    {
        numeros[i] = i + MÍNIMO;
    }

    printf("Valores originales y valores multiplicados:\n");
    for (int i = 0; i < TAMAÑO; i++)
    {
        printf("Original: %d, Multiplicado: %d\n", numeros[i], numeros[i] * MULTIPLICADOR);
    }

    return 0;
}
```

- `#define`: Define constantes o macros.
- `#ifndef` Ejecuta código solo si una macro no está definida.
- `#if, #else, #endif`: Condicionales para incluir o excluir partes del código.
- `#warning`: Genera una advertencia durante la compilación.
- `#error`: Detiene la compilación con un mensaje de error.
- `#undef`: Deshace la definición de una macro.
- `#pragma`: Instrucciones específicas para el compilador.

12. Redacte un informe detallado con sus resultados y conclusiones.

    En conclusión, en todo este proceso nos dimos cuenta de lo importante que es el preprocesador. este se encarga de gestionar el flujo de compilación, emitir advertencias y verificar condiciones críticas. Además, al usar herramientas como `gcc` y `ld`, logré entender mejor cómo se transforma el código fuente en un ejecutable.