	Depuración de programas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M. I. Marco Antonio Martínez Quintana

Asignatura: Fundamentos de Programación

Grupo: 3

No. de Práctica(s): 10

Integrante(s): Sánchez García Rocío

No. de Equipo de cómputo empleado: No aplica

No. de Lista: 47

Semestre: 2021-1

Fecha de entrega: Domingo 13 de diciembre de 2020

Observaciones:

CALIFICACIÓN: _____

Práctica 10

Depuración de programas

Objetivo

Aprender las técnicas básicas de depuración de programas en C para revisar de manera precisa el flujo de ejecución de un programa y el valor de las variables; en su caso, corregir posibles errores.

Actividades

- Revisar, a través de un depurador, los valores que va tomando una variable en un programa escrito en C, al momento de ejecutarse.
- Utilizando un depurador, revisar el flujo de instrucciones que se están ejecutando en un programa en C, cuando el flujo depende de los datos de entrada.

Introducción

Depurar un programa significa someterlo a un ambiente de ejecución controlado por medio de herramientas dedicadas a ello. Este ambiente permite conocer exactamente el flujo de ejecución del programa, el valor que las variables adquieren, la pila de llamadas a funciones, entre otros aspectos. Es importante poder compilar el programa sin errores antes de depurarlo.

Antes de continuar, es necesario conocer las siguientes definiciones (extraídas del Glosario IEEE610) ya que son parte latente del proceso de Desarrollo de Software:

Error. Se refiere a una acción humana que produce o genera un resultado incorrecto.

Defecto (Fault). Es la manifestación de un error en el software. Un defecto es encontrado porque causa una Falla (failure).

Falla (failure). Es una desviación del servicio o resultado esperado.

La depuración de un programa es útil cuando:

- Se desea optimizar el programa: no basta que el programa se pueda compilar y se someta a pruebas que demuestren que funciona correctamente. Debe realizarse un análisis exhaustivo del mismo en ejecución para averiguar cuál es su flujo de operación y encontrar formas de mejorarlo (reducir el código, utilizar menos recursos llegando a los mismos resultados, hacer menos rebuscado al algoritmo), o bien, encontrar puntos donde puede fallar con ciertos tipos de entrada de datos.

- El programa tiene algún fallo: el programa no muestra los resultados que se esperan para cierta entrada de datos debido a que el programador cometió algún error durante el proceso de diseño. Muchas veces encontrar este tipo de fallos suele ser difícil, ya sea porque la percepción del programador no permite

encontrar la falla en su diseño o porque la errata es muy pequeña, pero crucial. En este caso es de mucha utilidad conocer paso a paso cómo se ejecutan las estructuras de control, qué valor adquieren las variables, etc.

- El programa tiene un error de ejecución o defecto: cuando el programa está ejecutándose, éste se detiene inesperadamente. Suele ocurrir por error en el diseño o implementación del programa en las que no se contemplan las limitaciones del lenguaje de programación o el equipo donde el programa se ejecuta. Como el programa se detiene inesperadamente, no se conoce la parte del programa donde se provoca el defecto, teniendo que recurrir a la depuración para encontrarlo. El más común de este tipo de defecto es la “violación de segmento”.

Algunas funciones básicas que tienen en común la mayoría de los depuradores son las siguientes:

- Ejecutar el programa: se procede a ejecutar el programa en la herramienta de depuración ofreciendo diversas opciones para ello.

- Mostrar el código fuente del programa: muestra cuál fue el código fuente del programa con el número de línea con el fin de emular la ejecución del programa sobre éste, es decir, se indica qué parte del código fuente se está ejecutando a la hora de correr el programa.

- Punto de ruptura: también conocido por su traducción al inglés *breakpoint*, sirve para detener la ejecución del programa en algún punto indicado previamente por medio del número de línea. Como la ejecución del programa es más rápida de lo que podemos visualizar y entender, se suelen poner puntos de ruptura para conocer ciertos parámetros de la ejecución como el valor de las variables en determinados puntos del programa. También sirve para verificar hasta qué punto el programa se ejecuta sin problemas y en qué parte podría existir el error, esto es especialmente útil cuando existe un error de ejecución.

- Continuar: continúa con la ejecución del programa después del punto de ruptura.

- Ejecutar la siguiente instrucción: cuando la ejecución del programa se ha detenido por medio del depurador, esta función permite ejecutar una instrucción más y detener el programa de nuevo. Esto es útil cuando se desea estudiar detalladamente una pequeña sección del programa. Si en la ejecución existe una llamada a función se ingresará a ella.

- Ejecutar la siguiente línea: es muy similar a la función anterior, pero realizará todas las instrucciones necesarias hasta llegar a la siguiente línea de código. Si en la ejecución existe una llamada a función se ignorará.

- ❖ Ejecutar la instrucción o línea anterior: deshace el efecto provocado por alguna de las funciones anteriores para volver a repetir una sección del programa.

- ❖ Visualizar el valor de las variables: permite conocer el valor de alguna o varias variables.

Dependiendo de la herramienta usada para compilar el programa, si es de consola o de terminal, su uso y las funciones disponibles variarán.

En las IDE (Entornos de Desarrollo Interactivo), suelen existir herramientas de depuración integradas de manera gráfica. Es muy común que existan dos modos de desarrollar un programa y producir el archivo ejecutable que son “Debug” y “Release”. El primer modo se recomienda exclusivamente durante el desarrollo del programa para poder depurarlo continuamente durante cualquier prueba de ejecución. El segundo modo se establece cuando el programa ha sido terminado y totalmente probado.

Depuración de programas escritos en C con GCC y GDB

Ejemplo:

Para compilar un programa llamado *calculadora.c* con GCC con información de depuración, debe realizarse en una terminal con el siguiente comando:

```
gcc -g -o calculadora calculadora.c
```

El parámetro *-g* es quien indica que el ejecutable debe producirse con información de depuración.

Una vez hecho el paso anterior, debe usarse la herramienta GDB, la cual, es el depurador para cualquier programa ejecutable realizado por GCC.
Para depurar un ejecutable debe invocarse a GDB en la terminal indicando cuál es el programa ejecutable a depurar, por ejemplo, para depurar *calculadora*:

```
gdb ./calculadora
```

Al correr GDB se entra a una línea de comandos. De acuerdo con el comando es posible realizar distintas funciones de depuración:

- *list* o *l:* Permite listar diez líneas del código fuente del programa, si se desea visualizar todo el código fuente debe invocarse varias veces este comando para mostrar de diez en diez líneas. Se puede optar por colocar un número separado por un espacio para indicar a partir de qué línea desea mostrarse el programa. También es posible mostrar un rango de líneas introduciendo el comando y de qué línea a qué línea separadas por una coma. Ejemplo: *list 4,6*
- *b:* Establece un punto de ruptura para lo cual debe indicarse en qué línea se desea establecer o bien también acepta el nombre de la función donde se desea realizar dicho paso. Ejemplo: *b 5*

- *d* o *delete:* Elimina un punto de ruptura, indicando cuál es el que debe eliminarse
- usando el número de línea. Ejemplo: *d 5*
- *clear:* Elimina todos los puntos de ruptura. Ejemplo: *clear*
- *info line:* Permite mostrar información relativa a la línea que se indique después del comando. Ejemplo: i*nfo line 8*
- *run* o *r:* Ejecuta el programa en cuestión. Si el programa tiene un punto de ruptura se ejecutará hasta dicho punto, de lo contrario se ejecutará todo el programa.
- *c:* Continúa con la ejecución del programa después de un punto de ruptura.
- *s:* Continúa con la siguiente instrucción después de un punto de ruptura.
- *n:* Salta hasta la siguiente línea de código después de un punto de ruptura.
- *p* o *print:* Muestra el valor de una variable, para ello debe escribirse el comando y elnombre de la variable separados por un espacio. Ejemplo: *p suma_acumulada*
- *ignore:* Ignora un determinado punto de ruptura indicándolo con el número de línea de código. Ejemplo: *ignore 5*
- *q* o *quit:* Termina la ejecución de GDB.

GDB tiene más opciones disponibles que pueden consultarse con comandos como *help* o invocando desde la terminal del sistema *man gdb.*

Desarrollo

- Para el siguiente código fuente, utilizar algún entorno de depuración para encontrar la utilidad del programa y la funcionalidad de los principales comandos de depuración, como puntos de ruptura, ejecución de siguiente línea o instrucción.

```
1  #include <stdio.h>
2  void main()
3  {
4      int N, CONT, AS;
5      AS=0;
6      CONT=1;
7      printf("TECLEA UN NUMERO: ");
8      scanf("%i",&N);
9      while(CONT<=N)
10     {
11         AS=(AS+CONT);
12         CONT=(CONT+2);
13     }
14     printf("\nEL RESULTADO ES %i\n", AS);
15 }
```

```
Reading symbols from C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos\depuracion.exe...done.
(gdb) b 9
Breakpoint 1 at 0x40144e: file depuracion.c, line 9.
(gdb) b 11
Breakpoint 2 at 0x401450: file depuracion.c, line 11.
(gdb) b 12
Breakpoint 3 at 0x401458: file depuracion.c, line 12.
(gdb)  b 14
Breakpoint 4 at 0x401467: file depuracion.c, line 14.
(gdb) run
Starting program: C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos/./depuracion.exe
[New Thread 18912.0xb90]
[New Thread 18912.0x1698]
TECLEA UN NUMERO: 4

Breakpoint 1, main () at depuracion.c:9
9           while(CONT<=N)
(gdb) print cont
No symbol "cont" in current context.
(gdb) print CONT
$1 = 1
(gdb) print N
$2 = 4
(gdb) next

Breakpoint 2, main () at depuracion.c:11
11              AS=(AS+CONT);
(gdb) print AS
$3 = 0
(gdb) print CONT
$4 = 1
(gdb) next
```

Utilidad del programa: el numero que se teclee corresponderá a la cantidad de repeticiones que hará el programa en cada una de las repeticiones ira sumando 2 cifras a partir del número 1 y al final nos mostrara la sumatoria total.

Puntos de ruptura:
Nos dan la oportunidad de parar la ejecución del programa, en consecuencia, podemos ver los valores que toma una variable en cada instrucción que realiza.

Ejecución de la siguiente línea o instrucción:
En este caso nos muestra cual es la instrucción que el programa ejecutara y podremos consultar el valor de las variables nuevamente, en base al punto de ruptura establecido.

- El siguiente programa debe mostrar las tablas de multiplicar desde la del 1 hasta la del 10. En un principio no se mostraba la tabla del 10, luego después de intentar corregirse sin un depurador dejaron de mostrarse el resto de las tablas. Usar un depurador de C para averiguar el funcionamiento del programa y corregir ambos problemas.

```c
#include <stdio.h>
void main()
{
    int i, j;
    for(i=1; i<10; i++)
    {
        printf("\nTabla del %i\n", i);
        for(j=1; j==10; j++)
        {
            printf("%i X %i = %i\n", i, j, i*j);
        }
    }
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc depurador2.c -o depurador2.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>depurador2.exe

Tabla del 1

Tabla del 2

Tabla del 3

Tabla del 4

Tabla del 5

Tabla del 6

Tabla del 7

Tabla del 8

Tabla del 9

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

Funcionalidad del programa: mostrar las tablas del 1 al 10

En este caso el operador empleado para la impresión del mensaje que indica de que tabla se trata, únicamente establece que se deben mostrar los números menores a 10 con excepción de este mismo por lo que lo que la solución a este problema sería cambia al número 11 para que aparezca la tabla del 10.

El segundo problema surge a raíz del operador ya que indica que cuando la variable `j` sea igual a 10 se deben mostrar las tablas, pero estas no se mostraran debido a que para que esto ocurra primero deben mostrarse los números que van desde al el 1 al 9 por lo que se debe usar la misma sentencia que en el caso anterior.

```c
#include <stdio.h>
void main()
{
    int i, j;
    for(i=1; i<11; i++)
    {
        printf("\nTabla del %i\n", i);
        for(j=1; j<11; j++)
        {
            printf("%i X %i = %i\n", i, j, i*j);
        }
    }
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc depurador2.c -o depurador2.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>depurador2.exe

Tabla del 1
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
1 X 4 = 4
1 X 5 = 5
1 X 6 = 6
1 X 7 = 7
1 X 8 = 8
1 X 9 = 9
1 X 10 = 10

Tabla del 2
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20

Tabla del 3
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30

Tabla del 4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40

Tabla del 5
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50

Tabla del 6
6 X 1 = 6
6 X 2 = 12
6 X 3 = 18
6 X 4 = 24
6 X 5 = 30
6 X 6 = 36
6 X 7 = 42
6 X 8 = 48
6 X 9 = 54
6 X 10 = 60

Tabla del 7
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70

Tabla del 8
8 X 1 = 8
8 X 2 = 16
8 X 3 = 24
8 X 4 = 32
8 X 5 = 40
8 X 6 = 48
8 X 7 = 56
8 X 8 = 64
8 X 9 = 72
8 X 10 = 80

Tabla del 9
9 X 1 = 9
9 X 2 = 18
9 X 3 = 27
9 X 4 = 36
9 X 5 = 45
9 X 6 = 54
9 X 7 = 63
9 X 8 = 72
9 X 9 = 81
9 X 10 = 90
```

```
Tabla del 10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

- El siguiente programa muestra una *violación de segmento* durante su ejecución y se interrumpe; usar un depurador para detectar y corregir la falla.

```c
#include <stdio.h>
#include <math.h>
void main()
{
    int K, X, AP, N;
    float AS;
    printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
    printf("\nN=");
    scanf("%d",N);
    printf("X=");
    scanf("%d",X);
    K=0;
    AP=1;
    AS=0;
    while(K<=N)
    {
        AS=AS+pow(X,K)/AP;
        K=K+1;
        AP=AP*K;
    }
    printf("SUM=%le",AS);
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc depurador3.c -o depurador3.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>depurador3.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=3

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

La interrupción ocurría debido a que al nombre de la variable debe antecederle un ampersand (&) para que indique el dato que se guardara en la localidad de memoria asignada para cada variable.

```c
#include <stdio.h>
#include <math.h>
void main()
{
    int K, X, AP, N;
    float AS;
    printf("EL TERMINO GENERICO DE LA SERIE ES: X^K/K!");
    printf("\nN=");
    scanf("%d",&N);
    printf("X=");
    scanf("%d",&X);
    K=0;
    AP=1;
    AS=0;
    while(K<=N)
    {
        AS=AS+pow(X,K)/AP;
        K=K+1;
        AP=AP*K;
    }
    printf("SUM=%le",AS);
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc depurador3.c -o depurador3.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>depurador3.exe
EL TERMINO GENERICO DE LA SERIE ES: X^K/K!
N=3
X=2
SUM=6.333333e+000
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

Conclusiones

- La depuración sirve principalmente para identificar y corregir los errores que haya en el programa.
- Si se conocen y se manejan adecuadamente las herramientas que nos ofrece un depurador será más fácil, rápida y eficiente la tarea de detección de errores.
- Existen una gran variedad de formas en las que se depura un programa, pero esto depende del la IDE en la que fue desarrollado dicho programa.

Bibliografía

Manual de prácticas del Laboratorio de Fundamentos de Programación, Facultad de ingeniería UNAM, recuperada el 7 de diciembre, en http://lcp02.fi-b.unam.mx/