	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación
salas A y B

Profesor: Alejandro Pimentel

Asignatura: Fundamentos de Programación

Grupo: 3

No de Práctica(s): 10

Integrante(s): Mendoza Hernández Mariana

No. de Equipo de cómputo empleado: 54

No. de Lista o

Semestre: 2020-1

Fecha de entrega: Octubre 14, 2019

Observaciones: Faltó el segundo ejercicio.
Las descripciones se podían hacer directamente en el código, para eso no hace falta captura.

CALIFICACIÓN: **8**

Práctica 10: Depuración de programas.

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

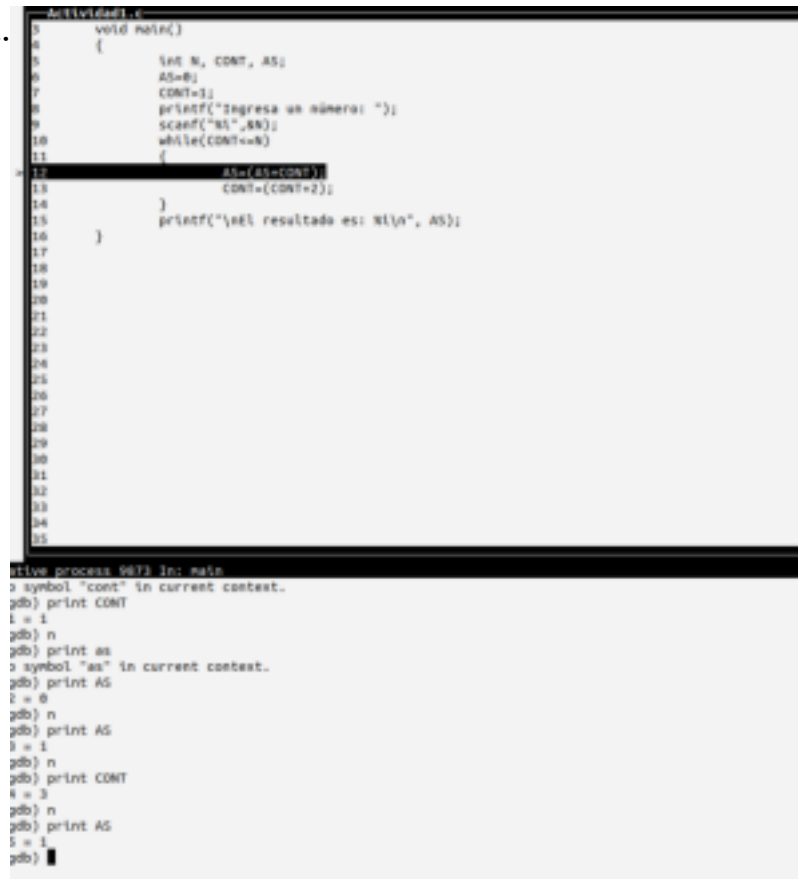
función permite ejecutar una instrucción más y detener el programa de nuevo. Esto es útil cuando se desea estudiar detalladamente una pequeña sección del programa. Si en la ejecución existe una llamada a función se ingresará a ella. Ejecutar la siguiente línea: es muy similar a la función anterior, pero 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.

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.

- Utilizar GDB para encontrar la utilidad del programa y describir su funcionalidad.



- Utilizar GDB para corregir el programa.

```
File  Edit  View  Search  Terminal  Help
No executable file specified.
Use the "file" or "exec-file" command.
luis@pc-lrdo-VirtualBox:~/Desktop/pMarina$ gdb actividad2
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from actividad2...(no debugging symbols found)...done.
luis@pc-lrdo-VirtualBox:~/Desktop/pMarina$ gcc -w Actividad2.c -o actividad2 -lm
luis@pc-lrdo-VirtualBox:~/Desktop/pMarina$ gdb actividad2
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from actividad2...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/luis/Desktop/pMarina/actividad2
Ingrese cuántos términos calcular de la serie: X*X/X!
N=3
X=4
Resultado=23.666667[Inferior 1 (process 11237) exited normally]
(gdb) quit
luis@pc-lrdo-VirtualBox:~/Desktop/pMarina$ echo en este programa era que al momento de pedir los datos de una variable para la serie no se hizo uso del token para el acceso al apuntador de la variable si
no quería modificarse el valor de la localidad de ran en donde estaba almacenada la vlarable por lo que gdb nos arrojaba un error de core dumpo porque estabamos intentanto modificar ran la solución fue po
ner los datos a la variable de manera correcta mediante el token # en scanf
[1] 11264
en este programa era que al momento de pedir los datos de una variable para la serie no se hizo uso del token para el acceso al apuntador de la variable si no quería modificarse el valor de la localidad d
e ran en donde estaba almacenada la vlarable por lo que gdb nos arrojaba un error de core dumpo porque estabamos intentanto modificar ran la solución fue poner los datos a la variable de manera correcta m
ediante el token
en: command not found
[1]+  Done                    echo en este programa era que al momento de pedir los datos de una variable para la serie no se hizo uso del token para el acceso al apuntador de la variable si no quería mod
ificarse el valor de la localidad de ran en donde estaba almacenada la vlarable por lo que gdb nos arrojaba un error de core dumpo porque estabamos intentanto modificar ran la solución fue poner los datos
 a la variable de manera correcta mediante el token
luis@pc-lrdo-VirtualBox:~/Desktop/pMarina$
```

Podías escribir directamente en el documento

- Utilizar GDB para corregir el programa.

```
Actividad3.c
4       {
5               int numero;
6
7               printf("Ingrese un número:\n");
8               scanf("%l",&numero);
9
10              long int resultado = 0;
11              while(numero>=0){
12                      numero--;
13                      resultado *= numero;
14              }
15
16          printf("El factorial de %l es %ll.\n", numero, resultado);
17
18          return 0;
19      }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
native process 11487 In: main
(gdb) start
Temporary breakpoint 1 at 0x772: file Actividad3.c, line 4.
Starting program: /home/luis/Desktop/pMarina/act3

Temporary breakpoint 1, main () at Actividad3.c:4
(gdb) n
(gdb) n
(gdb) 3
Undefined command: "3".  Try "help".
(gdb) n
(gdb) n
(gdb) p numero
$1 = 4
(gdb) n
(gdb) n
(gdb) p numero
$2 = 3
(gdb)
```

Conclusiones.

Se cumplió el objetivo de la práctica ya que se revisaron los programas a través de depuradores y aprendí a corregir los posibles errores, sólo es cuestión de familiarizarse con esto.