



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: ADRIAN ULISES MERCADO MARTINEZ

Asignatura: FUNDAMENTOS DE PROGRAMACIÓN

Grupo: 15

No de Práctica(s): 10

Integrante(s): MORALES VELASCO BRAYAN, WALLS CHÁVEZ LUIS
FERNANDO

*No. de Equipo de
cómputo empleado:*

No. de Lista o Brigada: 12

Semestre: 2020-1

Fecha de entrega: 24/10/19

Observaciones:

CALIFICACIÓN: _____

INTRODUCCIÓN:

La depuración de un programa significa someter al programa a un ambiente de ejecución controlado por medio de diversas herramientas dedicadas a ello. Este mismo ambiente permite conocer exactamente el flujo de ejecución del programa, el valor que las variables adquieren, la pila de llamadas a funciones , entre otras cosas.

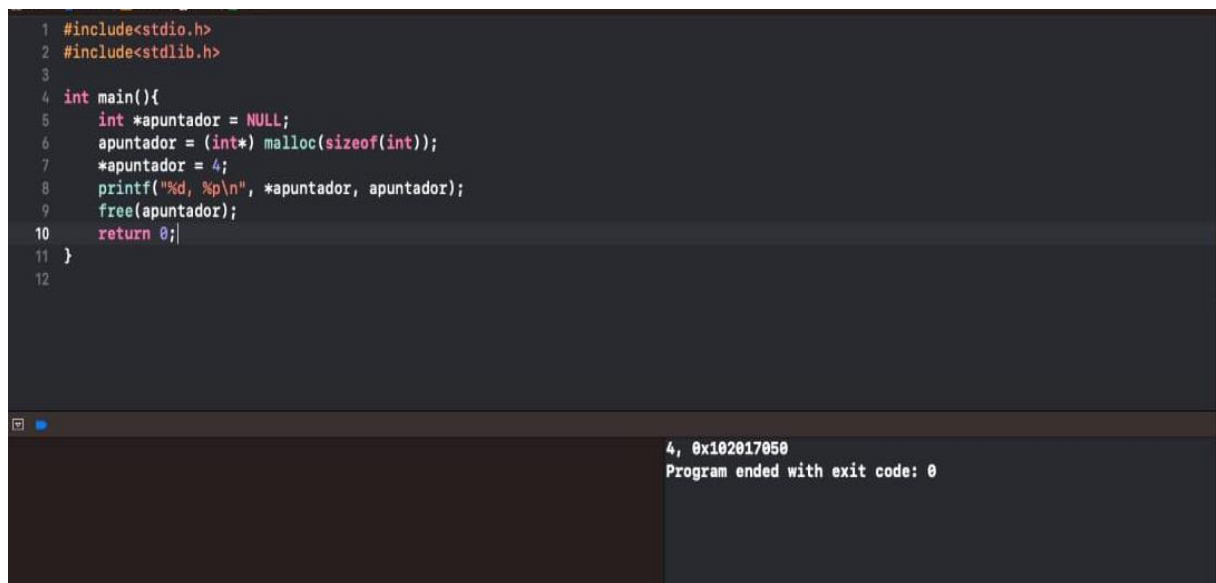
Es importante poder compilar el programa sin errores antes de depurarlo.

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.

En esta ocasión ocupamos la herramienta de XCode disponible en mac, donde copiamos los ejemplos de la práctica para poder encontrar los errores.



```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main(){
5     int *apuntador = NULL;
6     apuntador = (int*) malloc(sizeof(int));
7     *apuntador = 4;
8     printf("%d, %p\n", *apuntador, apuntador);
9     free(apuntador);
10    return 0;|
11 }
12
```

4, 0x102017050
Program ended with exit code: 0

La interfaz de esta aplicación se divide en: En la parte superior se encuentra el código creado y en la parte inferior derecha, el código ejecutado con los resultados obtenidos.

Lo siguiente que hicimos fue marcar ciertos breakpoints para poder detener la ejecución del código en ciertas instrucciones.

Ahora con un código con "switch" ocupamos los mismos breakpoints, donde comprobamos que estos nos sirven para ir verificando paso a paso como se ejecuta el código, y en caso de encontrar algún error, poder identificarlo de manera rápida.

```
1 #include <stdio.h>
2 /* Este programa genera una calculadora básica. */
3 int main () {
4     int op, uno, dos; do {
5         printf(" --- Calculadora ---\n"); printf("\n¿Qué desea hacer\n"); printf("1) Sumar\n");
6         printf("2) Restar\n");
7         printf("3) Multiplicar\n"); printf("4) Dividir\n"); printf("5) Salir\n"); scanf("%d",&op);
8         switch(op){
9             case 1:
10                 printf("\tSumar\n");
11                 printf("Introduzca los números a sumar separados por comas\n"); scanf("%d, %d",&uno, &dos);
12                 printf("%d + %d = %d\n", uno, dos, (uno + dos));
13                 break;
14             case 2:
15                 printf("\tRestar\n");
16                 printf("Introduzca los números a restar separados por comas\n"); scanf("%d, %d",&uno, &dos);
17                 printf("%d - %d = %d\n", uno, dos, (uno - dos));
18                 break;
19             case 3:
20                 printf("\tMultiplicar\n");
21                 printf("Introduzca los números a multiplicar separados por comas\n"); scanf("%d, %d",&uno, &dos);
22                 printf("%d * %d = %d\n", uno, dos, (uno * dos));
23                 break;
24             case 4:
25                 printf("\tDividir\n");
26                 printf("Introduzca los números a dividir separados por comas\n"); scanf("%d, %d",&uno, &dos);
27                 printf("%d / %d = %.2lf\n", uno, dos, ((double)uno / dos)); break;
28             case 5:
29                 printf("\tSalir\n"); break;
30             default:
31                 printf("\tOpción inválida.\n");
32         }
33     } while (op != 5);
34     return 0;
35 }
36
```

Debugger output:

Variable	Value
op = (int)	32622
uno = (int)	32766
dos = (int)	-272632464

Thread 1: 0 main

Calculadora ---

¿Qué desea hacer

1) Sumar

2) Restar

(lldb)

Conclusiones:

En esta práctica se vieron como poder identificar un problema después de haber generado el código, gracias a la herramienta XCode, que en esencia al poner diferentes breakpoints se pueden verificar casi "línea por línea" el funcionamiento del programa para que sean más fáciles de identificar y poder corregir la falla lo antes posible.

Nos mantendrá o experimentará mejor gracias a la prueba y error y así poder mejorar con la practica la elaboración de diferentes códigos.

-Brayan Morales Velasco

Los depuradores son una herramienta esencial en el trabajo de un programador. Como buenos ingenieros en computación que seremos, es indispensable aprender el uso y funcionalidad de estos. El uso del depurador en esta practica me pareció sencillo y concreto, usamos los breakpoints para poder analizar la ejecución del programa paso a paso, y poder identificar los valores tomados por las variables a medida del avance de los programas.

Un depurador es muy útil al momento de llegar a la fase de mantenimiento del ciclo de un programa ya que con estos, aunque sea un poco más duradero, los errores salen a la luz más fácilmente.

-Walls Chávez Luis Fernando