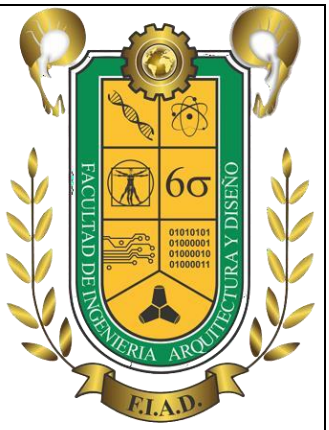

Universidad Autónoma deBaja California

Facultad de Ingeniería, Arquitectura yDiseño

Ingeniero en Computación

Asignatura:

Programación Estructurada

Actividad 5:

Estructuras de control Repetitivas

Funciones

Brayan Arturo Rocha Meneses

Matricula:

371049

Ensenada Baja California 12 de Septiembre del 2023

Introducción:

En esta práctica principalmente se aborda el tema de estructuras de control repetitivas, también conocidas como bucles, son herramientas que permiten ejecutar un bloque de código varias veces de manera iterativa. Estos bucles son esenciales cuando se necesita realizar una tarea repetitiva, como procesar elementos de una lista o llevar a cabo cálculos iterativos. Hay diferentes tipos de bucles, como el bucle while, el bucle for, y el bucle do-while, cada uno adecuado para situaciones específicas.

Estas estructuras aseguran que una porción de código se repita mientras se cumpla una condición particular, lo que facilita la automatización de tareas repetitivas y mejora la eficiencia del programa.

.

Competencia:

Aplicar la abstracción procedimental, al analizar las generalidades en las estrategias de solución de problemas complejos, para simplificar el proceso de resolución, con actitud creativa y organizada.

.

Fundamentos:

En este capítulo se revisan los distintos métodos con los que C controla el flujo lógico de un programa.

Los operadores relaciones binarios que se usan son:

< Menor que

<= Menor Igualque

> Mayor que

>= Mayor igual que

== Exactamente Igual

!= Diferente

Los operadores lógicos binarios:

|| OR

&& AND

y el operador lógico unario de negación !, que sólo toma un argumento.

! NOT

Los operadores anterior son usados con las siguientes estructuras que se muestran.

La sentencia if

Las tres formas como se puede emplear la sentencia if son:

if (condicion)

sentencia;

...o

if (condicion)

sentencia1;

else

sentencia2;

...o

```
if (condicion1)



sentencia1;

else

if (condicion2)

sentencia2;

...

else

sentencian;
```

El flujo lógico de esta estructura es de arriba hacia abajo. La primera sentencia se ejecutará y se saldrá de la estructura if si la primera condición es verdadera. Si la primera condición fue falsa, y existe otra condición, se evalúa, y si la condición es verdadera, entonces se ejecuta la sentencia asociada.

Si existen más condiciones dentro de la estructura if, se van evaluando éstas, siempre y cuando las condiciones que le preceden sean falsas.

La sentencia que está asociada a la palabra reservada else, se ejecuta si todas las condiciones de la estructura if fueron falsas.

Por ejemplo:

```
int main()

{

int x, y, w;

.....

if (x>0)

{

z=w;

.....
```

```
}
else
{
z=y;
.....
}
}
```

La sentencia switch

Aunque con la estructura if ... else if se pueden realizar comprobaciones múltiples, en ocasiones no es muy elegante, ya que el código puede ser difícil de seguir y puede confundir incluso al autor transcurrido un tiempo.

Por lo anterior, C tiene incorporada una sentencia de bifurcación múltiple llamada switch. Con esta sentencia, la computadora comprueba una variable sucesivamente frente a una lista de constantes enteras o de carácter. Después de encontrar una coincidencia, la computadora ejecuta la sentencia o bloque de sentencias que se asocian con la constante.

La forma general de la sentencia switch es:

```
switch (variable) {
case constante1:

secuencia de sentencias
break;
case constante2:

secuencia de sentencias
break;
case constante3:
```

secuencia de sentencias

break;

...

default:

secuencia de sentencias

}

Donde la computadora ejecuta la sentencia default si no coincide ninguna constante con la variable, esta última es opcional. Cuando se encuentra una coincidencia, la computadora ejecuta las sentencias asociadas con el case hasta encontrar la sentencia break con lo que sale de la estructura switch.

Las limitaciones que tiene la sentencia switch ... case respecto a la estructura if son:

● Sólo se tiene posibilidad de revisar una sola variable.

● Con switch sólo se puede comprobar por igualdad, mientras que con if puede ser con cualquier operador relacional.

● No se puede probar más de una constante por case.

La forma como se puede simular el último punto, es no teniendo sentencias asociados a un case, es decir, teniendo

una sentencia nula donde sólo se pone el caso, con lo que se permite que el flujo del programa caiga al omitir las

Sentencias, como se muestra a continuación:

Switch (letra)

{

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

```
numvocales++;
break;
case ' ':
numesp++;
break;
default:
numotras++;
```

Procedimiento:

Las funciones son bloques de código independientes y reutilizables que realizan tareas específicas.

Las funciones permiten dividir un programa en partes más pequeñas y manejables, lo que simplifica la programación y facilita la depuración y el mantenimiento. Una función puede aceptar argumentos (datos de entrada), realizar una serie de operaciones y devolver un resultado.

Además, las funciones pueden ser llamadas desde diferentes partes de un programa, lo que fomenta la modularidad y la reutilización de código. Esto significa que una vez que se ha definido una función, puede ser utilizada una y otra vez sin necesidad de volver a escribir su lógica.

REALIZA LOS SIGUIENTES EJERCICIOS EN C SUBIR UN PROGRAMA QUE LLAME LOS 4 EJERCICIOS

1.- Función en C que pida al usuario el valor de n, y desplegar todos los números enteros positivos menores de n en orden descendente.

2.- Función en "C" que genere 40 números aleatorios entre el 0 y 200, desplegar los números y la leyenda de cada número si es par o impar , la cantidad de los números pares e impares así como la suma de los números pares o impares.

3.- Función en "C" que genere N (35) cantidad de números (100 -200), desplegar al final el número mayor y el número menor.

4.- Función en "C" que despliegue la tabla de multiplicar de un número dado (número entre el 1 y 20).

Tabla del 5

5 * 1 = 5

5* 2 = 10

5*10=50

Conclusiones:

La combinación adecuada de estructuras de control repetitivas y funciones permite a los programadores escribir programas más legibles, mantenibles y escalables. Esto se traduce en un código más limpio y menos propenso a errores, así como en la capacidad de abordar problemas complejos de manera más eficiente.

Además, la reutilización de código a través de funciones reduce la duplicación y facilita las actualizaciones y modificaciones en el software.

Anexo:

https://drive.google.com/file/d/1oJ2Ej4dxNRtJjlxM3mBeeNT9j2qq0lei/view?usp=sharing