


| | | |
|--|-------------------------|--|
|  | Práctica | |
| Facultad de Ingeniería | Laboratorio de docencia | |

Laboratorios de computación salas A y B

| | |
|---|---|
| <i>Profesor:</i> | Alejandro Pimentel |
| <i>Asignatura:</i> | Fundamentos de Programación |
| <i>Grupo:</i> | Grupo 3 Bloque 135 |
| <i>No de Práctica(s):</i> | Practica #9 |
| <i>Integrante(s):</i> | Muñoz Reyes Laura Vanessa-2823 Gutierrez Ramos Alitzel Anaid-9370 |
| <i>No. de Equipo de cómputo empleado:</i> | Luxemburgo 10 |
| <i>No. de Lista o Brigada:</i> | No. de Cuenta Muñoz Reyes 317752 2823 Gutierrez Ramos 31704 9370 |
| <i>Semestre:</i> | 2020-01 |
| <i>Fecha de entrega:</i> | 14 de octubre del 2019 |
| <i>Observaciones:</i> | Bastante bien |

CALIFICACIÓN: 10

Practica #9.

Estructuras de Repetición

INTRODUCCION:

Todos los lenguajes de programación modernos tienen estructuras de control similares. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis; cada lenguaje tiene una sintaxis propia para expresar la estructura.

Las estructuras de repetición o bucles hacen posible la ejecución repetida de una o más instrucciones mientras que la expresión lógica a evaluar se cumpla (sea verdadera).

En el lenguaje C encontramos tres estructuras de repetición: *while*, *do-while* y *for*.

OBJETIVO:

Elaborar programas en C para la resolución de problemas básicos que incluyan las estructuras de repetición y la directiva *define*.

MARCO REFERENCIAL:

Las estructuras de control repetitivas, son aquellas que permiten ejecutar un conjunto de instrucciones varias veces, de acuerdo al valor que genere la expresión relacional y/o lógica. Esto significa que una instrucción repetitiva permite saltar a una instrucción anterior para volver a ejecutarla.

A estas estructuras también se les conoce como cíclicas, iterativas o bucles. Principalmente son 3: *while*, *do-while*, *for*.

Las tres instrucciones tienen el mismo fin, y difieren únicamente en su sintaxis, siendo posible sustituir una solución en la que se utiliza "*while*", por una en la que se utiliza "*do-while*" o "*for*".

Estructura de control de repetición *for*

La declaración *for* se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Permite realizarse cuando se conoce el número de repeticiones

La declaración *for* tiene tres partes separadas por (;). La inicialización de la variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

```
for (inicialización ; expresión_lógica ; operaciones por iteración) {  
    /*  
        Bloque de código  
        a ejecutar  
    */  
}
```

Estructura de control de repetición *while*

La estructura repetitiva *while* primero valida la expresión lógica y si ésta se cumple (es verdadera) procede a ejecutar el bloque de instrucciones de la estructura, el cual está delimitado por las llaves {}. Si la condición no se cumple se continúa el flujo normal del programa sin ejecutar el bloque de la estructura.

La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor.

```
while (expresión_lógica) {  
    // Bloque de código a repetir  
    // mientras que la expresión  
    // lógica sea verdadera.  
}
```

Estructura de control de repetición Do-While

Do-while es una estructura cíclica que ejecuta el bloque de código que se encuentra dentro de las llaves y después valida la condición, es decir, el bloque de código se ejecuta de una a las veces que se desee. Es decir la diferencia entre *while* y *do-while*, es que en *do-while* la condición se prueba a final del bucle, por lo que el bucle se ejecutara al menos una vez.

Si el bloque de código a repetir consta de una sola sentencia, entonces se pueden omitir las llaves. Esta estructura de control siempre termina con el signo de puntuación ';'.

```
do {  
    /*  
    Bloque de código que se ejecuta  
    por lo menos una vez y se repite  
    mientras la expresión lógica sea  
    verdadera.  
    */  
} while (expresión_lógica);
```

Define

En lenguaje C, una constante puede ser de tipo entero, real, carácter o de cadena. Todas se pueden expresar de dos formas diferentes:

1. Por su valor.
2. Con un nombre (identificador).

Para expresar una constante con un nombre, la constante debe ser declarada previamente. En C, para representar a las constantes, se utilizan constantes simbólicas. Una constante simbólica representa (sustituye) a una secuencia de caracteres, en vez de representar a un valor (dato almacenado en memoria).

Para declarar una constante simbólica, en lenguaje C, se utiliza una nueva directiva del preprocesador:

#define <nombre> <valor>

Al definir la constante simbólica con #define, se emplea un nombre y un valor. Cada vez que aparezca el nombre en el programa se cambiará por el valor definido. El valor puede ser numérico o puede ser texto.

Es decir, el define es una palabra clave que se utiliza para declarar un nombre especial con un significado. Es muy parecido a una variable, con la diferencia de que no se puede cambiar a lo largo del programa.

#define MAX 5

ACTIVIDADES:

Para cada uno de los siguientes problemas elegir un tipo de ciclo y resolverlo. Al final deben usar los tres tipos de ciclos y el define por lo menos una vez.

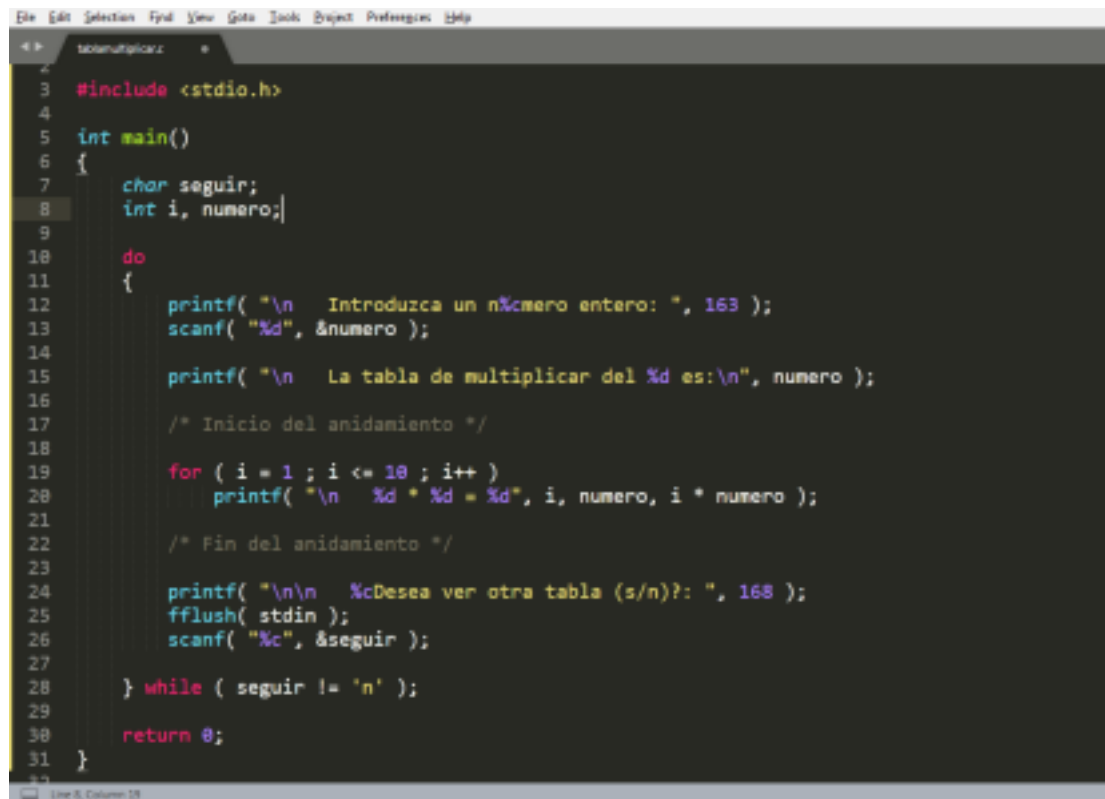
- Hacer un programa que pida un número y muestre su tabla de multiplicar (Hasta el 10)
- Hacer un programa que pida y lea 10 números y muestre su suma y su promedio.
- Hacer un programa que pida un numero e indique si es número primo o no.

PROCEDIMIENTO / RESULTADO:

1. Realiza los siguientes programas con un tipo de ciclo.

1.1 Tabla de Multiplicar

Programa en Sublime Text

A screenshot of the Sublime Text editor showing a C program. The editor has a dark theme and a menu bar at the top with options like File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The code is written in a file named 'tablas_multiplicar.c'. It includes the standard input/output header <stdio.h>. The main function starts with a loop that prompts the user to enter an integer, reads it, and then prints the multiplication table for that number. The table is generated using a for loop from 1 to 10. After printing the table, the program asks if the user wants to see another table and continues the loop if the answer is 's' (si/sí). The program ends by returning 0.

```
1  <
2
3  #include <stdio.h>
4
5  int main()
6  {
7      char seguir;
8      int i, numero;
9
10     do
11     {
12         printf( "\n  Introduzca un n%mero entero: ", 163 );
13         scanf( "%d", &numero );
14
15         printf( "\n  La tabla de multiplicar del %d es:\n", numero );
16
17         /* Inicio del anidamiento */
18
19         for ( i = 1 ; i <= 10 ; i++ )
20             printf( "\n    %d * %d = %d", i, numero, i * numero );
21
22         /* Fin del anidamiento */
23
24         printf( "\n\n  %cDesea ver otra tabla (s/n)? ", 168 );
25         fflush( stdin );
26         scanf( "%c", &seguir );
27
28     } while ( seguir != 'n' );
29
30     return 0;
31 }
```

Programa Compilado y Corrido

Nota: Se compilo en Windows usando el programa alterno MinGW y teniendo un archivo de salida.exe

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\usuario>cd MinGW\bin

C:\Users\usuario\MinGW\bin>gcc -o C:\Users\usuario\tablamultiplicar.exe C:\Users\usuario\tablamultiplicar.c

C:\Users\usuario\MinGW\bin>C:\Users\usuario\tablamultiplicar.exe

Introduzca un número entero: 7

La tabla de multiplicar del 7 es:

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

¿Desea ver otra tabla (s/n)? : s

Introduzca un número entero: 5

La tabla de multiplicar del 5 es:

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

¿Desea ver otra tabla (s/n)? : n

C:\Users\usuario\MinGW\bin>_
```

1.2 Programa que pida y lea 10 números y muestre su suma y su promedio.

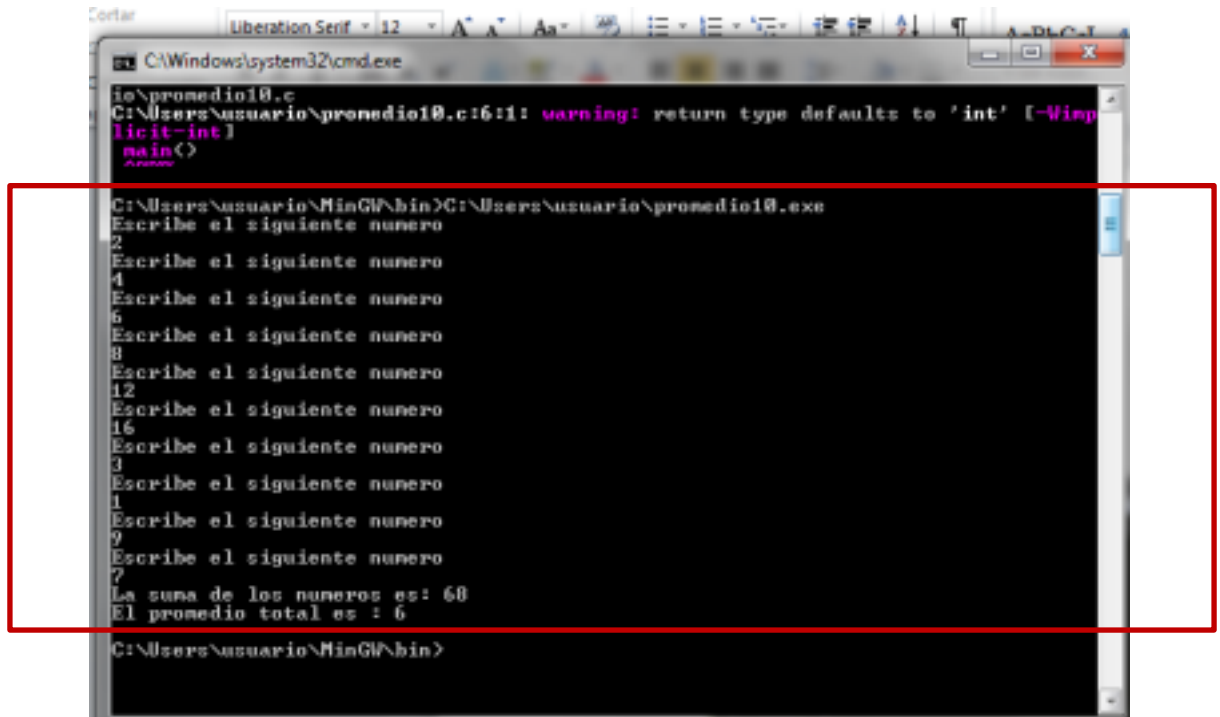
Código en Sublime Text

```
C:\Users\usuario\Documents\promedio00.c - Sublime Text [UNREGISTERED]
File Edit Selection Find View Goto Tools Project Preferences Help

tablamultiplicar.c  promedio00.c  promoci  promedio

1
2 #include <stdio.h>
3
4 #define x 10
5
6 main()
7 {
8     int total = 0, contador = 0, prom, cali;
9
10    while (contador != 10){
11        printf ("Escribe el siguiente numero \n");
12        scanf ("%d", &cali);
13        total += cali;
14        contador += 1;
15    }
16
17    if (contador != 0){
18        prom = total/x;
19        printf ("La suma de los numeros es: %d\n", total);
20        printf ("El promedio total es : %d\n", prom);
21    }
22
23    return 0;
24 }
25
26
27
```

Compilado y Corrido

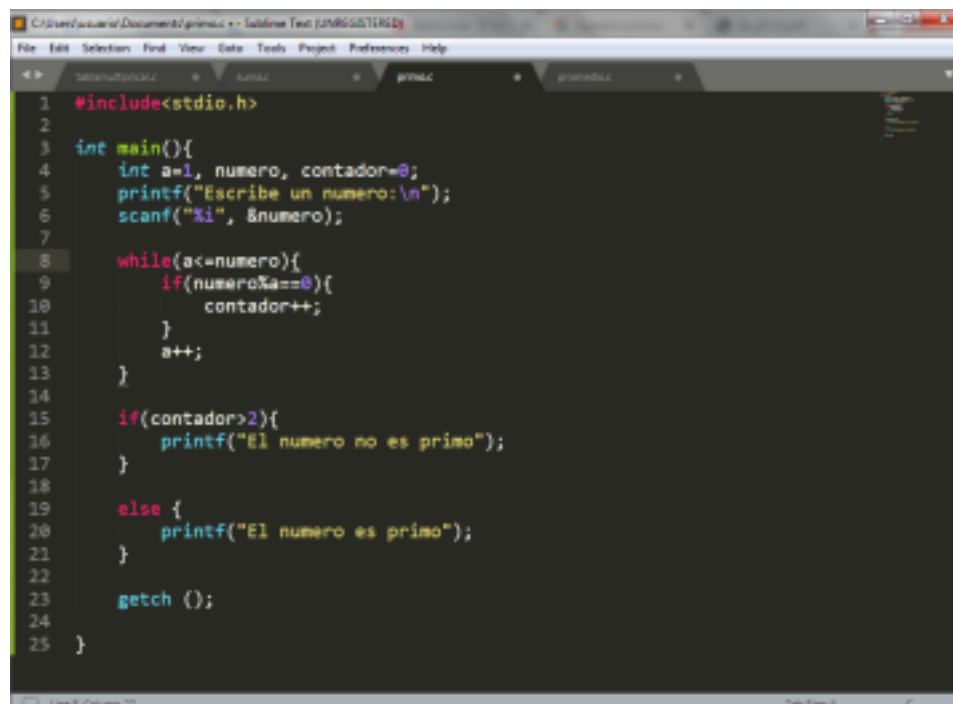


```
C:\Windows\system32\cmd.exe
C:\Users\usuario\promedio10.c:16:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
{
    ...

C:\Users\usuario\MinGW\bin>C:\Users\usuario\promedio10.exe
Escribe el siguiente numero
2
Escribe el siguiente numero
4
Escribe el siguiente numero
6
Escribe el siguiente numero
8
Escribe el siguiente numero
12
Escribe el siguiente numero
16
Escribe el siguiente numero
3
Escribe el siguiente numero
1
Escribe el siguiente numero
9
Escribe el siguiente numero
7
La suma de los numeros es: 60
El promedio total es : 6
C:\Users\usuario\MinGW\bin>
```

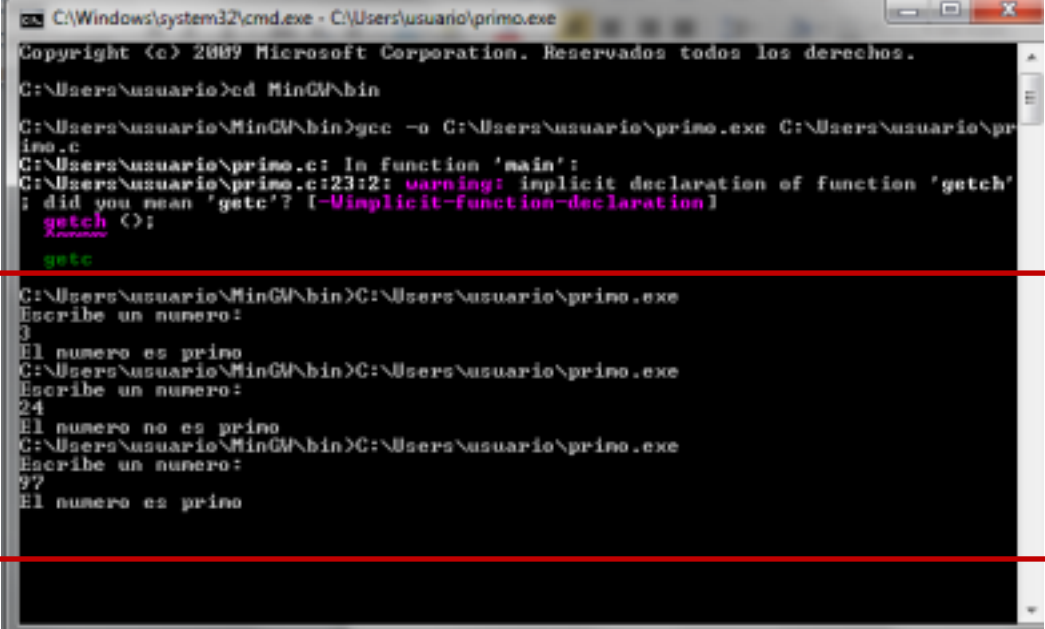
1.3 Numero Primo

Código en Sublime Text



```
1 #include<stdio.h>
2
3 int main(){
4     int a=1, numero, contador=0;
5     printf("Escribe un numero:\n");
6     scanf("%i", &numero);
7
8     while(a<=numero){
9         if(numero%a==0){
10             contador++;
11         }
12         a++;
13     }
14
15     if(contador>2){
16         printf("El numero no es primo");
17     }
18
19     else {
20         printf("El numero es primo");
21     }
22
23     getch ();
24
25 }
```

Compilado y corrido en Windows



```
C:\Windows\system32\cmd.exe - C:\Users\usuario\primo.exe
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\usuario>cd MinGW\bin

C:\Users\usuario\MinGW\bin>gcc -o C:\Users\usuario\primo.exe C:\Users\usuario\primo.c
C:\Users\usuario\primo.c: In function 'main':
C:\Users\usuario\primo.c:23:2: warning: implicit declaration of function 'getch'
; did you mean 'getc'? [-Wimplicit-function-declaration]
    getch ();
    ^~~~~
    getc
    ^~~~

C:\Users\usuario\MinGW\bin>C:\Users\usuario\primo.exe
Escribe un numero:
3
El numero es primo
C:\Users\usuario\MinGW\bin>C:\Users\usuario\primo.exe
Escribe un numero:
24
El numero no es primo
C:\Users\usuario\MinGW\bin>C:\Users\usuario\primo.exe
Escribe un numero:
97
El numero es primo
```

CONCLUSIONES

Como conclusión nos gustó ya que identificamos tipos de ciclos, de repetición y en qué situación usar cada uno. Así como aprender que este tipo de estructuras nos sirven para ejecutar una o más líneas repetidamente y nos ahorra el trabajo de estar haciendo una para cada línea.

En la práctica, la estructura do-while se nos facilitó más, sin embargo en ciertos casos el while constituye una solución más elegante.

FUENTES:

https://drive.google.com/drive/folders/1D7b4sZXX0V_66ZjvOIDd8OEhdCck6vaQ
<https://www.arduino.cc/reference/en/language/structure/control-structure/dowhile/>
http://www.carlospes.com/curso_de_lenguaje_c/01_07_constantes.php
http://www.utn.edu.ec/reduca/programacion/estructuras/sentencias_repetitivas.html