



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación  
salas A y B

*Profesor:* Alejandro Esteban Pimentel Alarcon

*Asignatura:* Fundamentos de programación.

*Grupo:* 3

*No de Práctica(s):* 9

*Integrante(s):* Ortiz Garcia Cesar Alan

*No. de Equipo de cómputo  
empleado:* 21

*No. de Lista o Brigada:* 9070

*Semestre:* 20-21

*Fecha de entrega:* 14/oct/19

*Observaciones:*

CALIFICACIÓN: \_\_\_\_\_

# Bucle while

El **bucle while** o **bucle mientras** es un ciclo repetitivo basado en los resultados de una expresión lógica; se encuentra en la mayoría de los lenguajes de programación estructurados. El propósito es repetir un bloque de código mientras una condición se mantenga verdadera.

## Condición.

La condición ha de ser una sentencia que devuelva un valor booleano, y esta puede ser el valor booleano sí, verdadero (true) si la condición se cumple, o falso si esta no se cumple (false). También puede contener el nombre de una variable booleana, y el valor de la expresión dependerá de su contenido. Se debe tener en cuenta que además de las variables también puede haber llamadas a funciones que devuelvan un valor.

## Sentencias comparativas

La forma más obvia tal vez, y la más usada sin duda, son las sentencias comparativas, que usan los operandos igual, diferente, menor o igual, mayor o igual, menor y mayor. En el caso del lenguaje C, se utilizan los siguientes símbolos para representar las comparaciones anteriores: ==, !=, <=, >=, <, >.

## Particularidades de lenguajes

En algunos lenguajes, se pueden utilizar variables no booleanas en la comparación. Por ejemplo, si la variable vale 0 será como si la condición no se cumpliera, y siempre que sea diferente de 0, se considerará que la condición se cumple.

## Ejemplos (usando sintaxis de C)

```
#include <stdio.h>
int main() {
    int tecla = 0;
    while (tecla == 0) {
        cout << mostrar_letra << tecla << endl; /* Leemos el numero ingresado */
    }
}
```

En este ejemplo el programa va a leer la pulsación de una tecla mientras su valor sea igual a 0. En el momento en que se pulse una tecla distinta se detendrá.

## Ciclo **do-while** en C++. Estructura, sintaxis y diferencias con el while. Cómo y cuándo usar un ciclo do-while en C++

Los ciclos do-while son una *estructura de control cíclica*, los cuales nos permiten ejecutar una o varias líneas de código de forma repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuando se va a dar el valor final, hasta aquí son similares a los ciclos while, sin embargo el ciclo do-while nos permite añadir cierta ventaja adicional y esta consiste que nos da la posibilidad de ejecutar primero el bloque de instrucciones antes de evaluar la condición necesaria, de este modo los ciclos do-while, son más efectivos para algunas situaciones específicas. En resumen un ciclo do-while, es una estructura de control cíclica que permite ejecutar de manera repetitiva un bloque

de instrucciones sin evaluar de forma inmediata una condición específica, sino evaluándola justo después de ejecutar por primera vez el bloque de instrucciones

¿Cómo funciona un Ciclo Do-While?

Para comprender mejor el funcionamiento del ciclo while, usemos de nuevo el ejemplo de la sección anterior sobre el ciclo while. Imaginemos entonces que por algún motivo, queremos pedirle a un usuario una serie de números cualquiera y que solo dejaremos de hacerlo cuando el usuario ingrese un número mayor a 100. Como vimos anteriormente, esto se puede hacer por medio de un ciclo while, pero vamos ahora a ver como lo podemos hacer usando un ciclo do-while mejorando así un poco nuestro algoritmo, evitando ciertos comandos, tal como se dijo con el ciclo while, en efecto aquí estamos en la situación de no tener ni idea de cuándo al usuario se le va a ocurrir ingresar un número mayor que 100, pues es algo indeterminado para nosotros, sin embargo el ciclo while y en efecto el do-while nos permite ejecutar cierta acción de forma infinita hasta que se cumpla alguna condición específica, en nuestro caso sería que el número ingresado sea mayor a 100. De modo que si el usuario nos ingresa de manera sucesiva los siguientes números 1, 50, 99, 49, 21, 30, 100 ..., nuestro programa no finalizará, pues ninguno de estos números es mayor que 100, sin embargo si nos ingresara el número 300, el programa finalizaría inmediatamente.

Vamos a ver ahora como es la sintaxis de un ciclo do-while en C++, así estaremos listos para usarlos en nuestros programas de ahora en adelante cada vez que lo necesitemos

Sintaxis del Ciclo Do-While en C++:

La sintaxis de un ciclo do-while es un tanto más larga que la del ciclo while en C++, sin embargo no se hace más complicado, de hecho con tan solo tener bien clara una condición de finalización para el ciclo tendremos prácticamente todo terminado.

```
do
```

```
{
```

```
    ....
```

```
    ....
```

```
    Bloque de Instrucciones....
```

```
    ....
```

```
    ....
```

```
}
```

```
while(condición de finalización); //por ejemplo numero != 23
```

# Ciclo **for** en C++. Estructura, sintaxis y uso de un ciclo for en C++

Los ciclos for son lo que se conoce como estructuras de control de flujo cíclicas o simplemente estructuras cíclicas, estos ciclos, como su nombre lo sugiere, nos permiten ejecutar una o varias líneas de código de forma iterativa, conociendo un valor específico inicial y otro valor final, además nos permiten determinar el tamaño del paso entre cada "giro" o iteración del ciclo.

En resumen, un ciclo for es una estructura de control iterativa, que nos permite ejecutar de manera repetitiva un bloque de instrucciones, conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo.

¿Cómo funciona un Ciclo For?

Para comprender mejor el funcionamiento del ciclo for, pongamos un ejemplo, supongamos que queremos mostrar los números pares entre el 50 y el 100, si imaginamos un poco como sería esto, podremos darnos cuenta que nuestro ciclo deberá mostrar una serie de números como la siguiente: 50 52 54 56 58 60 ... 96 98 100. Como podemos verificar, tenemos entonces los componentes necesarios para nuestro ciclo for, tenemos un valor inicial que sería el 50, tenemos también un valor final que sería el 100 y tenemos un tamaño de paso que es 2 (los números pares). Estamos ahora en capacidad de determinar los componentes esenciales para un ciclo for.

Vamos a ver ahora como es la sintaxis de un ciclo for en C++, así estaremos listos para usarlos en nuestros programas de ahora en adelante

Sintaxis del Ciclo For en C++:

La sintaxis de un ciclo for es simple en C++, en realidad en la mayoría de los lenguajes de alto nivel es incluso muy similar, de hecho, con tan solo tener bien claros los 3 componentes del ciclo for (inicio, final y tamaño de paso) tenemos prácticamente todo hecho

```
for(int i = valor inicial; i <= valor final; i = i + paso)
```

```
{
```

```
....
```

```
....
```

```
Bloque de Instrucciones....
```

```
....
```

```
....
```

```
}
```

# Fflush

Lo que hace `fflush(stdin)` es "limpiar" el buffer de entrada estándar del teclado.

La función `fflush` solamente está definida para flujos de datos donde no se produzcan operaciones de entrada. Es decir, `fflush` no está definida para streams de la entrada estándar, también hay que hacer notar que `fflush` está especificado para volcar los datos de los tampones a fichero.

`fflush(stdin)` es una "extensión" de algunos compiladores -especialmente de Windows- como ser Turbo C++, Dev-C++, Visual C++, o Borland C++, y no forma parte del estándar de C. Por lo tanto el comportamiento de esta función en compiladores de Unix/Linux/Mac es incierto, indeterminado, nulo, o peligroso.

Es por ello que es recomendable reemplazarla por algún bloque de código que cumpla la misma función de limpiar el buffer de la entrada estándar (teclado).

## ¿Para que sirve y como se limpia el buffer?

Supones que tienes una aplicación que, **a medida que pasan las cosas va haciendo preguntas al usuario...**

**En un caso así**, sería importante limpiar el buffer de entrada, pues durante la espera, el usuario pudo tocar el teclado sin la intención, y este toque accidental ha quedado en el buffer hasta tanto sea leída la entrada standard (cin).

Es una decisión de diseño, cuando quieres estar (algo mas) seguro que es una decisión consiente del usuario y por ello no aceptas los datos del buffer.

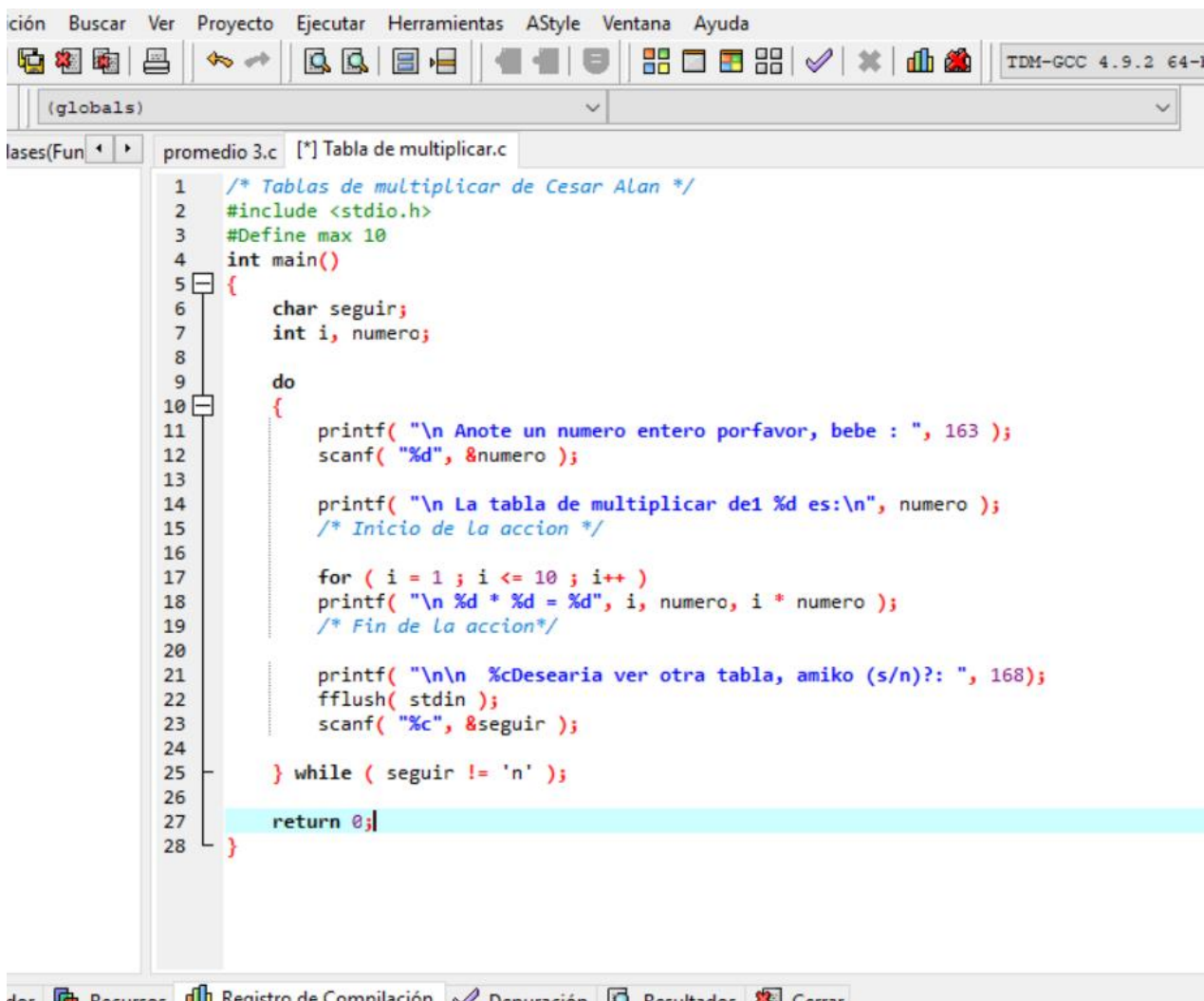
Entonces antes de `cin >> opcion` pones `cin.ignore(N)`, para descartar N bytes del buffer de entrada, forzadamente.

## 1.\_ Hacer un programa que pida un número y muestre su tabla de multiplicar (hasta el 10).

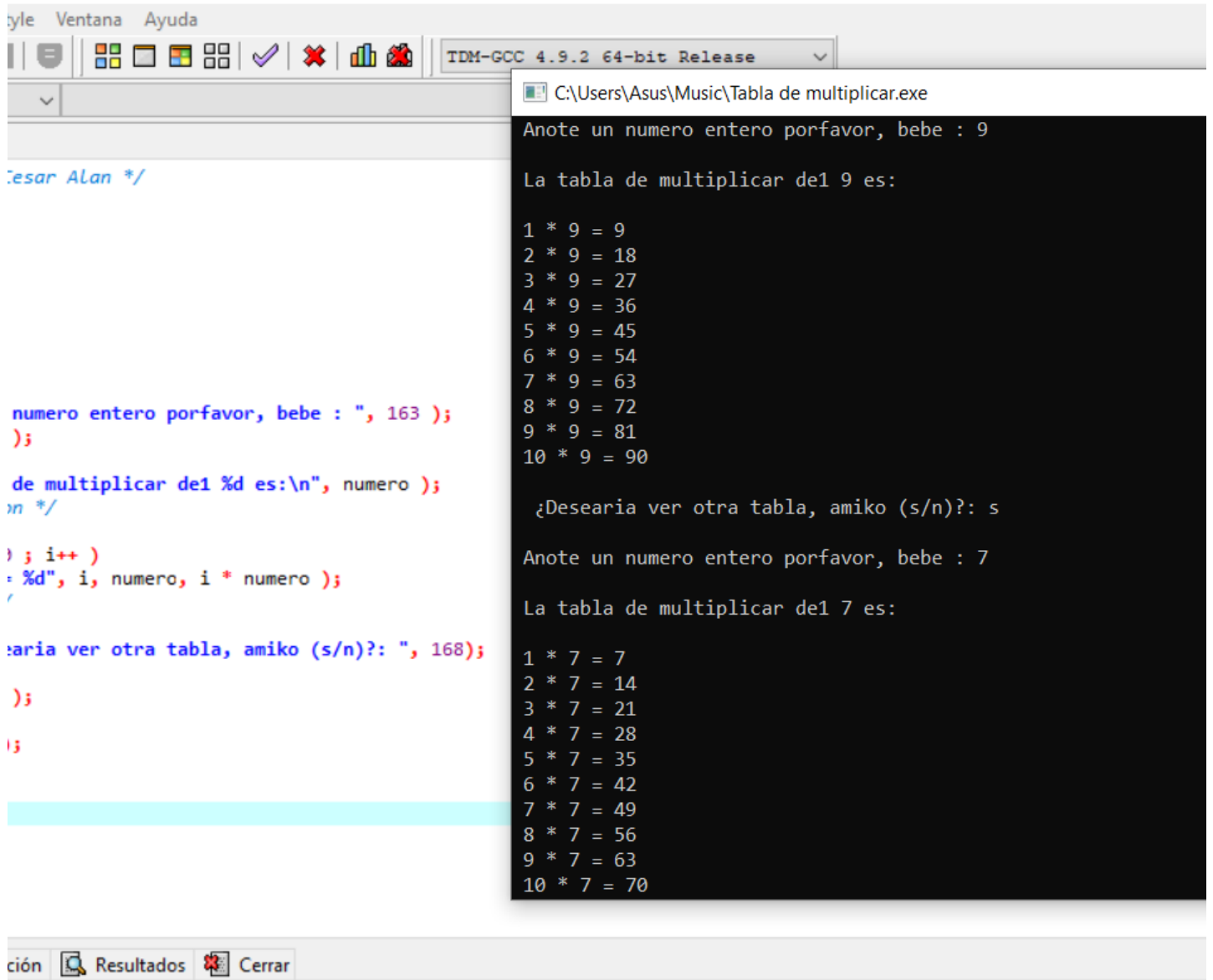
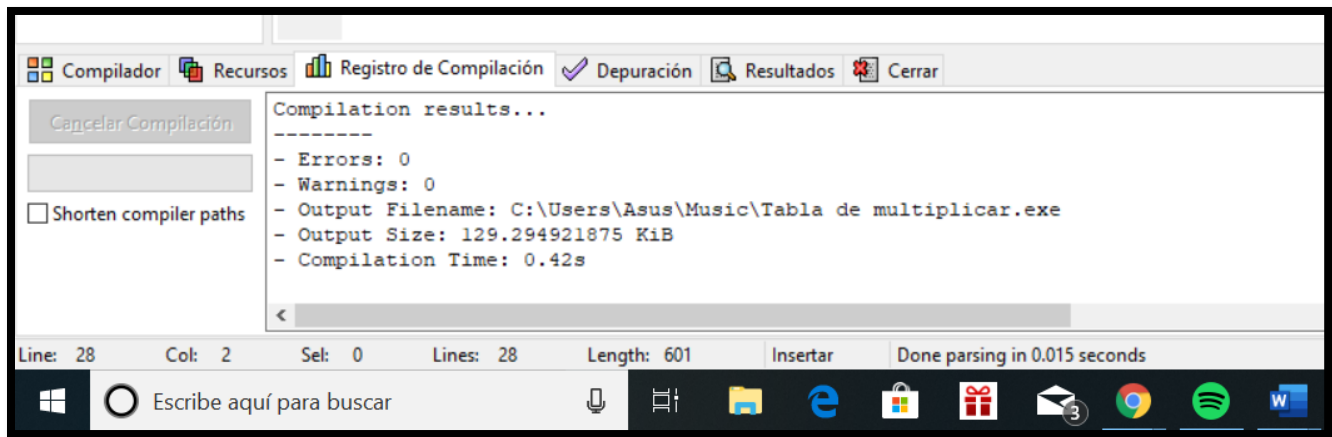
Para que se muestre en la pantalla la tabla de multiplicar del número introducido por nosotros, usé el bucle **for**, ya que se debe de rehacer 10 veces.

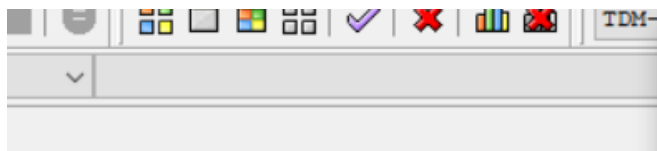
También utilicé el bucle **do...while** ya que el bloque de instrucciones que contiene debe de hacerse una vez.

También otro recurso que usé para aprovechar los bucles fue **Fflush**, ya que ayuda a hacer una pregunta de que si se quiere realizar otra tabla con otro número (s/n), y si la respuesta es sí, se hace otra tabla y si no, no.



```
1  /* Tablas de multiplicar de Cesar Alan */
2  #include <stdio.h>
3  #Define max 10
4  int main()
5  {
6      char seguir;
7      int i, numero;
8
9      do
10     {
11         printf( "\n Anote un numero entero porfavor, bebe : ", 163 );
12         scanf( "%d", &numero );
13
14         printf( "\n La tabla de multiplicar de1 %d es:\n", numero );
15         /* Inicio de la accion */
16
17         for ( i = 1 ; i <= 10 ; i++ )
18             printf( "\n %d * %d = %d", i, numero, i * numero );
19         /* Fin de la accion*/
20
21         printf( "\n\n %cDesearia ver otra tabla, amiko (s/n)? : ", 168);
22         fflush( stdin );
23         scanf( "%c", &seguir );
24
25     } while ( seguir != 'n' );
26
27     return 0;
28 }
```





Cesar Alan \*/

```
1 numero entero porfavor, bebe : ", 163 );  
2 );  
3 de multiplicar de1 %d es:\n", numero );  
4 ion */  
5  
6 10 ; i++ )  
7 = %d", i, numero, i * numero );  
8 */  
9  
10 searia ver otra tabla, amiko (s/n)? : ", 168);  
11  
12 ~ );  
13  
14 );
```

C:\Users\Asus\Music\Tabla de multiplicar.exe

Anote un numero entero porfavor, bebe : 10000000

La tabla de multiplicar de1 10000000 es:

```
1 * 10000000 = 10000000  
2 * 10000000 = 20000000  
3 * 10000000 = 30000000  
4 * 10000000 = 40000000  
5 * 10000000 = 50000000  
6 * 10000000 = 60000000  
7 * 10000000 = 70000000  
8 * 10000000 = 80000000  
9 * 10000000 = 90000000  
10 * 10000000 = 100000000
```

¿Desearia ver otra tabla, amiko (s/n)? : s

Anote un numero entero porfavor, bebe : 464524

La tabla de multiplicar de1 464524 es:

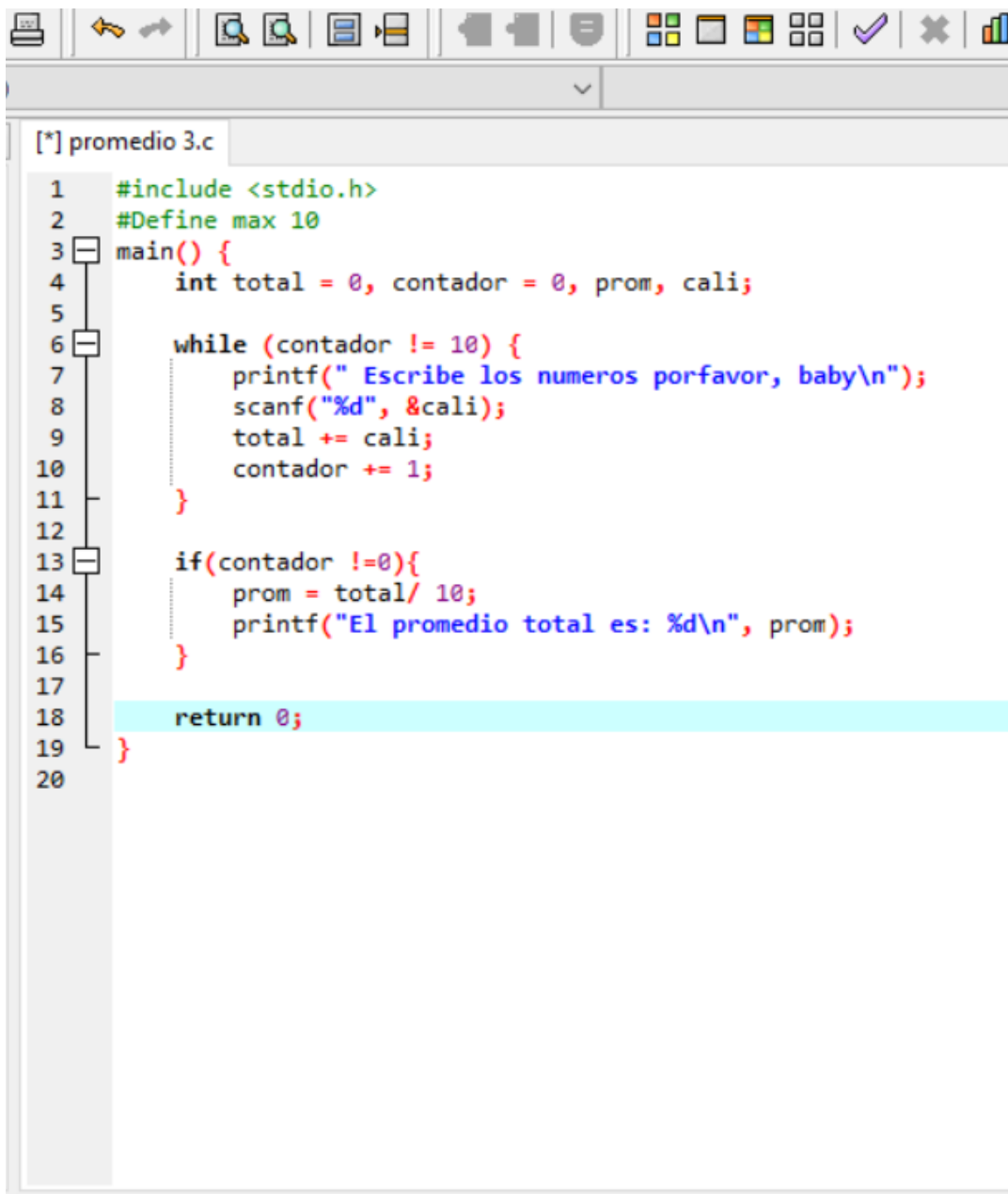
```
1 * 464524 = 464524  
2 * 464524 = 929048  
3 * 464524 = 1393572  
4 * 464524 = 1858096  
5 * 464524 = 2322620  
6 * 464524 = 2787144  
7 * 464524 = 3251668  
8 * 464524 = 3716192  
9 * 464524 = 4180716  
10 * 464524 = 4645240
```



## 2.\_Hacer un programa que pida y lea 10 números y muestre su promedio.

En este trabajo se usó el while, if condición ya que los ciclos son una estructura de control esencial, ya que permiten ejecutar una o varias líneas del código de forma repetitiva sin tener un valor inicial y a veces sin conocer cuando se va a dar el valor final.

de igual forma con Max. Para obtener el promedio total de la suma de 10 números que escribimos.



```
1  #include <stdio.h>
2  #Define max 10
3  main() {
4      int total = 0, contador = 0, prom, cali;
5
6      while (contador != 10) {
7          printf(" Escribe los numeros porfavor, baby\n");
8          scanf("%d", &cali);
9          total += cali;
10         contador += 1;
11     }
12
13     if(contador !=0){
14         prom = total/ 10;
15         printf("El promedio total es: %d\n", prom);
16     }
17
18     return 0;
19 }
20
```



ana Ayuda



TDM-GCC 4.9.2 64-bit Release

C:\Users\Asus\Music\promedio 3.exe

```
r, cali;
```

```
porfavor, baby\n");
```

```
: %d\n", prom);
```

Escribe los numeros porfavor, baby

8

Escribe los numeros porfavor, baby

9

Escribe los numeros porfavor, baby

6

Escribe los numeros porfavor, baby

7

Escribe los numeros porfavor, baby

5

Escribe los numeros porfavor, baby

7

Escribe los numeros porfavor, baby

8

Escribe los numeros porfavor, baby

5

Escribe los numeros porfavor, baby

7

Escribe los numeros porfavor, baby

6

El promedio total es: 6

-----  
Process exited after 11.57 seconds with return value 0  
Presione una tecla para continuar . . .

Resultados



Cerrar

### 3.\_ Hacer un programa que pida un número e indique si es primo o no.

Este programa es para saber si un número es primo o no, para hacer este programa utilicé el bucle Do-While, ya que permiten ejecutar una o varias líneas del código de forma repetitiva sin tener un valor inicial y a veces sin conocer cuando se va a dar el valor final.

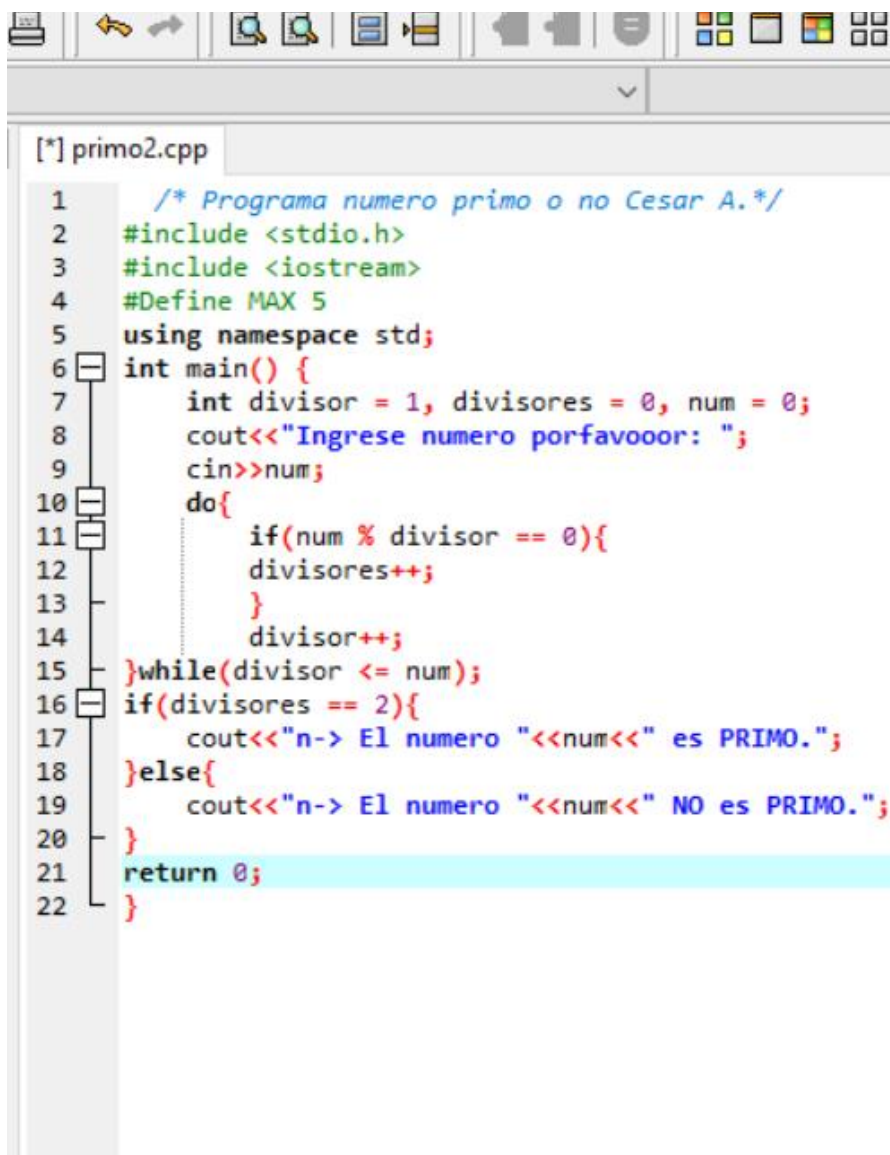
**iostream** es un componente de la biblioteca estándar (STL) del lenguaje de programación C++ que es utilizado para operaciones de entrada/salida. Su nombre es un acrónimo de **Input/Output Stream**. El flujo de entrada y salida de datos en C++ (y su predecesor C) no se encuentra definida dentro de la sintaxis básica y se provee por medio de librerías de funciones especializadas como iostream. iostream define los siguientes objetos:

cin : Flujo de entrada

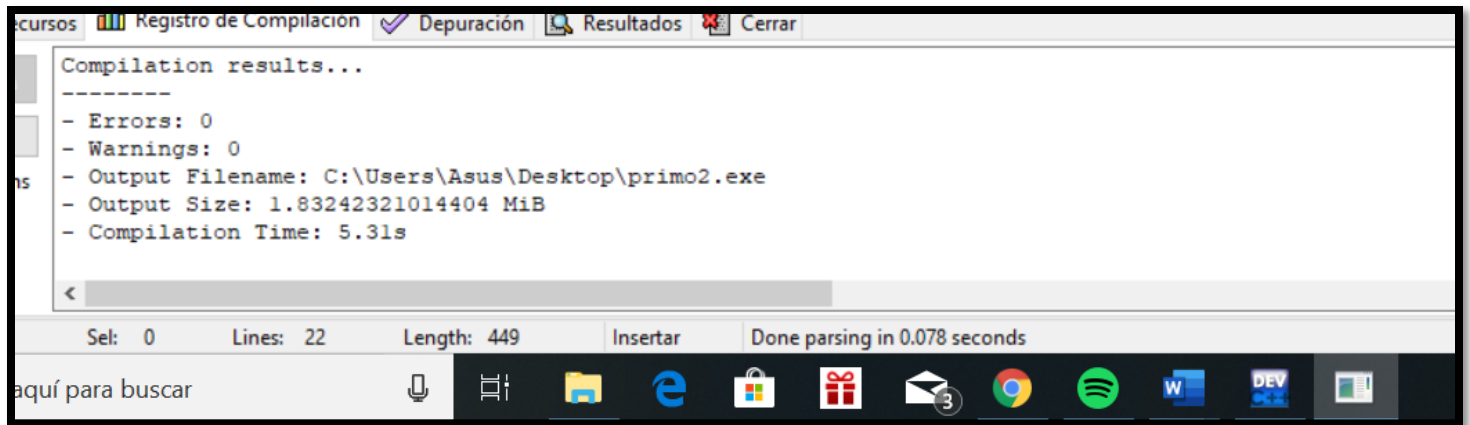
cout : Flujo de salida

cerr : Flujo de error no almacenado.

clog : Flujo de error almacenado.



```
1  /* Programa numero primo o no Cesar A.*/
2  #include <stdio.h>
3  #include <iostream>
4  #Define MAX 5
5  using namespace std;
6  int main() {
7      int divisor = 1, divisores = 0, num = 0;
8      cout<<"Ingrese numero porfavooor: ";
9      cin>>num;
10     do{
11         if(num % divisor == 0){
12             divisores++;
13         }
14         divisor++;
15     }while(divisor <= num);
16     if(divisores == 2){
17         cout<<"n-> El numero "<<num<<" es PRIMO.";
18     }else{
19         cout<<"n-> El numero "<<num<<" NO es PRIMO.";
20     }
21     return 0;
22 }
```



o no Cesar A.\*/

```
res = 0, num = 0;  
porfavoor: ";
```

```
: 0){
```

```
<num<<" es PRIMO.";
```

```
<num<<" NO es PRIMO.";
```

C:\Users\Asus\Desktop\primo2.exe

Ingrese numero porfavoor: 5

n-> El numero 5 es PRIMO.

Process exited after 65.2 seconds with return value 0

Presione una tecla para continuar . . .


Depuración Resultados Cerrar

r A.\* /

```
num = 0;  
: ";
```

```
s PRIMO.";
```

```
0 es PRIMO.";
```

Resultados  C

top\primo2.exe

C:\Users\Asus\Desktop\primo2.exe

Ingrese numero porfavooor: 9

n-> El numero 9 NO es PRIMO.

-----

Process exited after 18.61 seconds with return value 0

Presione una tecla para continuar . . .

# Conclusión.

Los ciclos o también conocidos como bucles, son una estructura de control esencial al momento de programar. Tanto C como C++ y la mayoría de los lenguajes utilizados actualmente, nos permiten hacer uso de estas estructuras. en programación, es una secuencia que ejecuta repetidas veces un trozo de código, hasta que la condición asignada a dicho bucle deja de cumplirse. Los tres bucles más utilizados en programación son el bucle while, el bucle for y el bucle do-while.

El ciclo de vida iterativo está relacionado directamente con la metodología ágil para la ejecución de proyectos. Se trata de ir obteniendo parte del producto por pequeños bloques, a los que se denomina iteraciones o ciclos de desarrollo, dentro del ciclo de vida de un proyecto en su conjunto.

iostream es un componente de la biblioteca estándar (STL) del lenguaje de programación C++ que es utilizado para operaciones de entrada/salida. Su nombre es un acrónimo de Input/Output Stream. El flujo de entrada y salida de datos en C++ (y su predecesor C) no se encuentra definida dentro de la sintaxis básica y se provee por medio de librerías de funciones especializadas como iostream. iostream define los siguientes objetos.