



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

*Profesor:* Alejandro Esteban Pimentel Alarcón

*Asignatura:* Fundamentos de Programación

*Grupo:* 135

*No de Práctica(s):* 9

*Integrante(s):* Torres Alcántara Alan Eliezer

*No. de Equipo de  
cómputo empleado:* Somalia 42

*No. de Lista o* 9032

*Semestre:* 2020 - 1

*Fecha de entrega:* 14/10/2019

*Observaciones:*

**CALIFICACIÓN:** \_\_\_\_\_

# Estructuras de repetición

Alejandro Pimentel  
[profpimentel9@gmail.com](mailto:profpimentel9@gmail.com)

Facultad de Ingeniería  
UNAM  
<https://qr.go.page.link/aeaVL>



## Objetivo

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

## ***Introducción***

Las estructuras de repetición son las llamadas estructuras cíclicas, iterativas o de bucles. Permiten ejecutar un conjunto de instrucciones de manera repetida (o cíclica) mientras que la expresión lógica a evaluar se cumpla (sea verdadera). En lenguaje C existen tres estructuras de repetición: while, do-while y for. Las estructuras while y do-while son estructuras repetitivas de propósito general.

### ***Estructura de control repetitiva while***

La estructura repetitiva (o iterativa) 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, es decir, el bloque se puede ejecutar de cero a ene veces.

Si el bloque de código a repetir consta de una sola sentencia, entonces se pueden omitir las llaves.

## WHILE

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

## ***Estructura de control repetitiva 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 *n* veces. 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-WHILE

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

## ***Estructura de control de repetición for***

La estructura for ejecuta 3 acciones básicas antes o después de ejecutar el bloque de código. La primera acción es la inicialización, en la cual se pueden definir variables e inicializar sus valores; esta parte solo se ejecuta una vez cuando se ingresa al ciclo y es opcional.

La segunda acción consta de una expresión lógica, la cual se evalúa y, si ésta es verdadera, ejecuta el bloque de código, si no se cumple se continúa la ejecución del programa; esta parte es opcional.

La tercera parte consta de un conjunto de operaciones que se realizan cada vez que termina de ejecutarse el bloque de código y antes de volver a validar la expresión lógica; esta parte también es opcional.

# FOR

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

## ***Define***

Las líneas de código que empiezan con # son directivas del preprocesador, el cual se encarga de realizar modificaciones en el texto del código fuente, como reemplazar un símbolo definido con #define por un parámetro o texto, o incluir un archivo en otro archivo con #include. define permite definir constantes o literales; se les nombra también como constantes simbólicas.

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.

# DEFINE

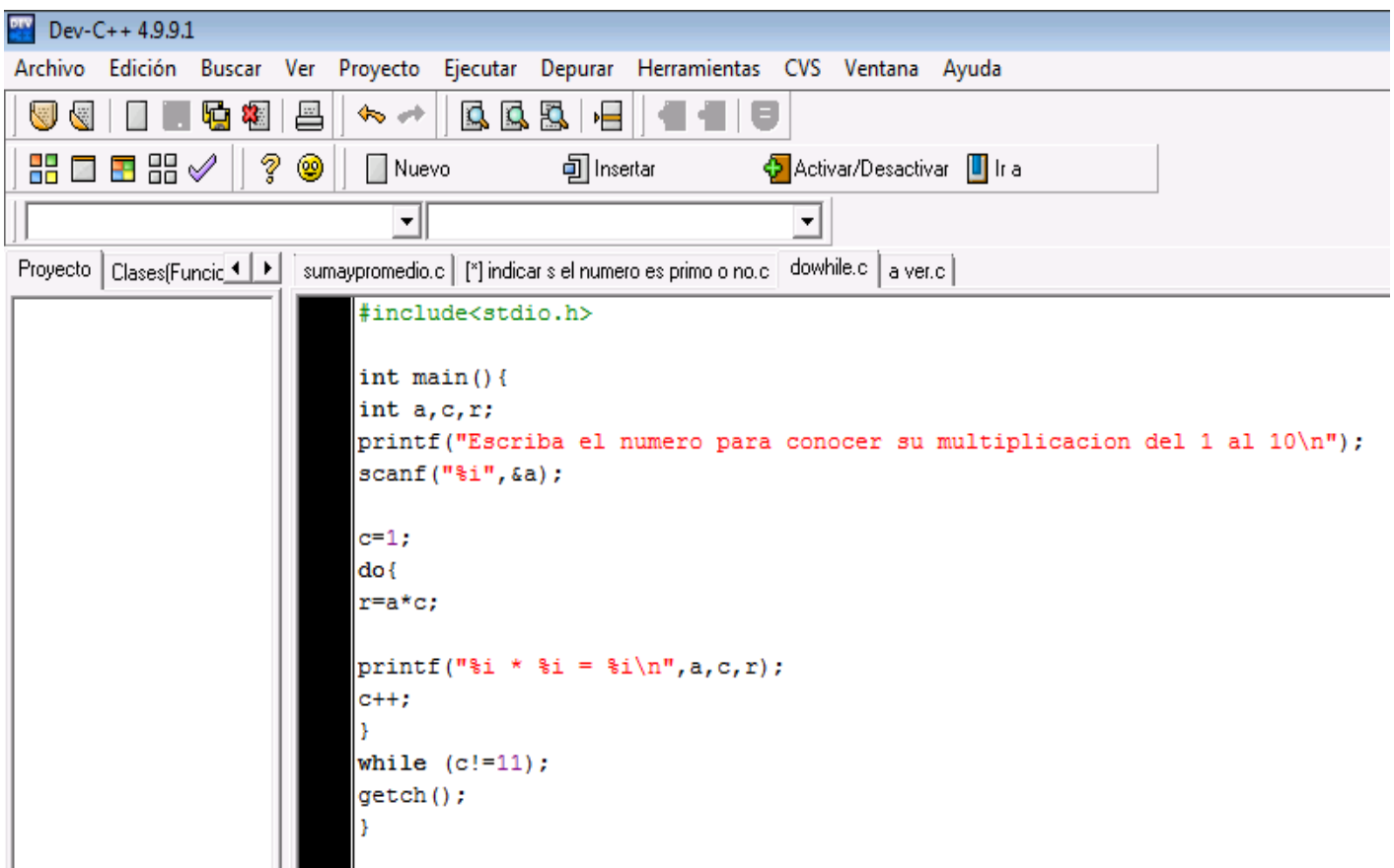
El *define* es una palabra clave que se utiliza para declarar un nombre especial con un significado. Es muy parecido a una variable, con la diferencia de que no se puede cambiar a lo largo del programa.

```
#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 usar *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 número e indique si es primo o no.



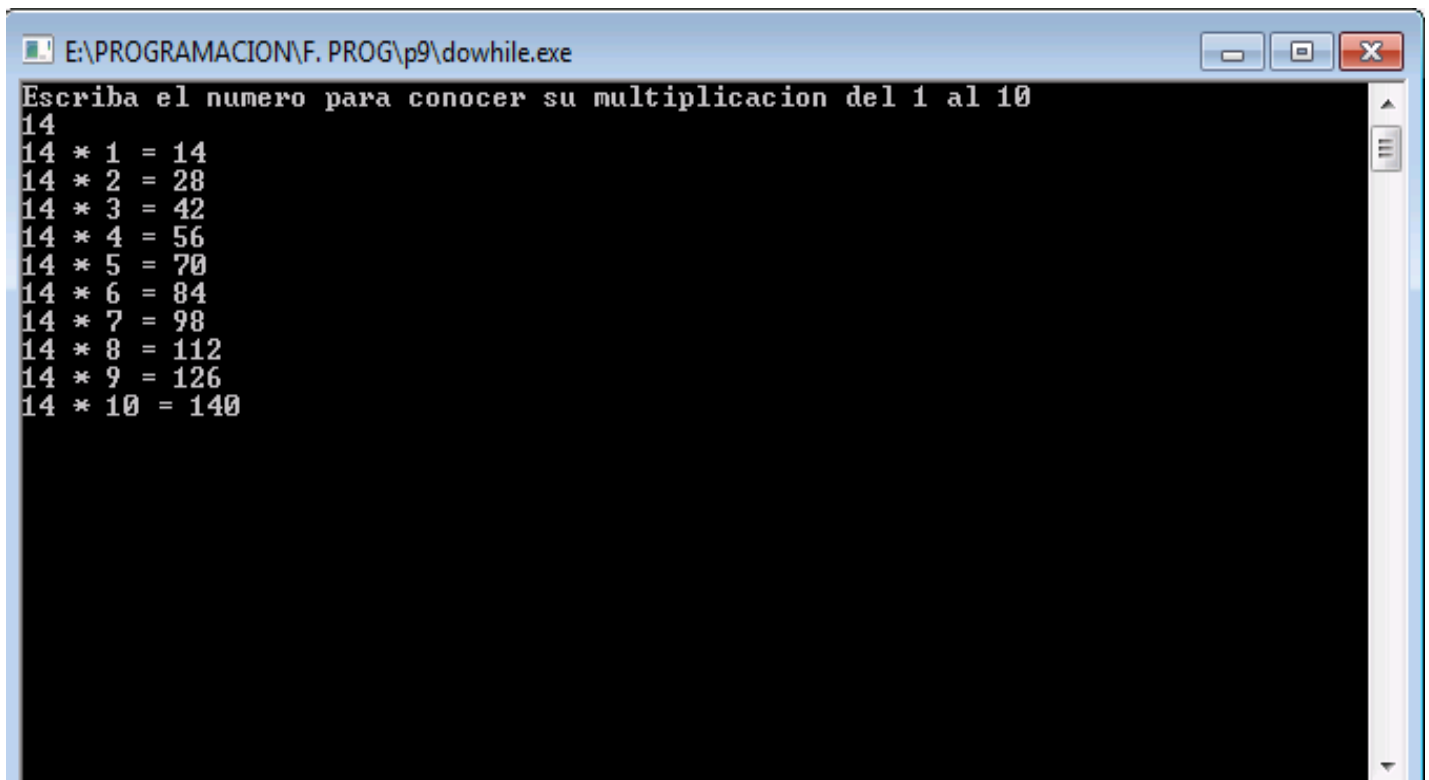
The screenshot shows the Dev-C++ 4.9.9.1 IDE. The menu bar includes Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Depurar, Herramientas, CVS, Ventana, and Ayuda. The toolbar contains various icons for file operations, editing, and execution. The project explorer on the left shows a project named 'sumaypromedio.c'. The main editor window displays the following C code:

```
#include<stdio.h>

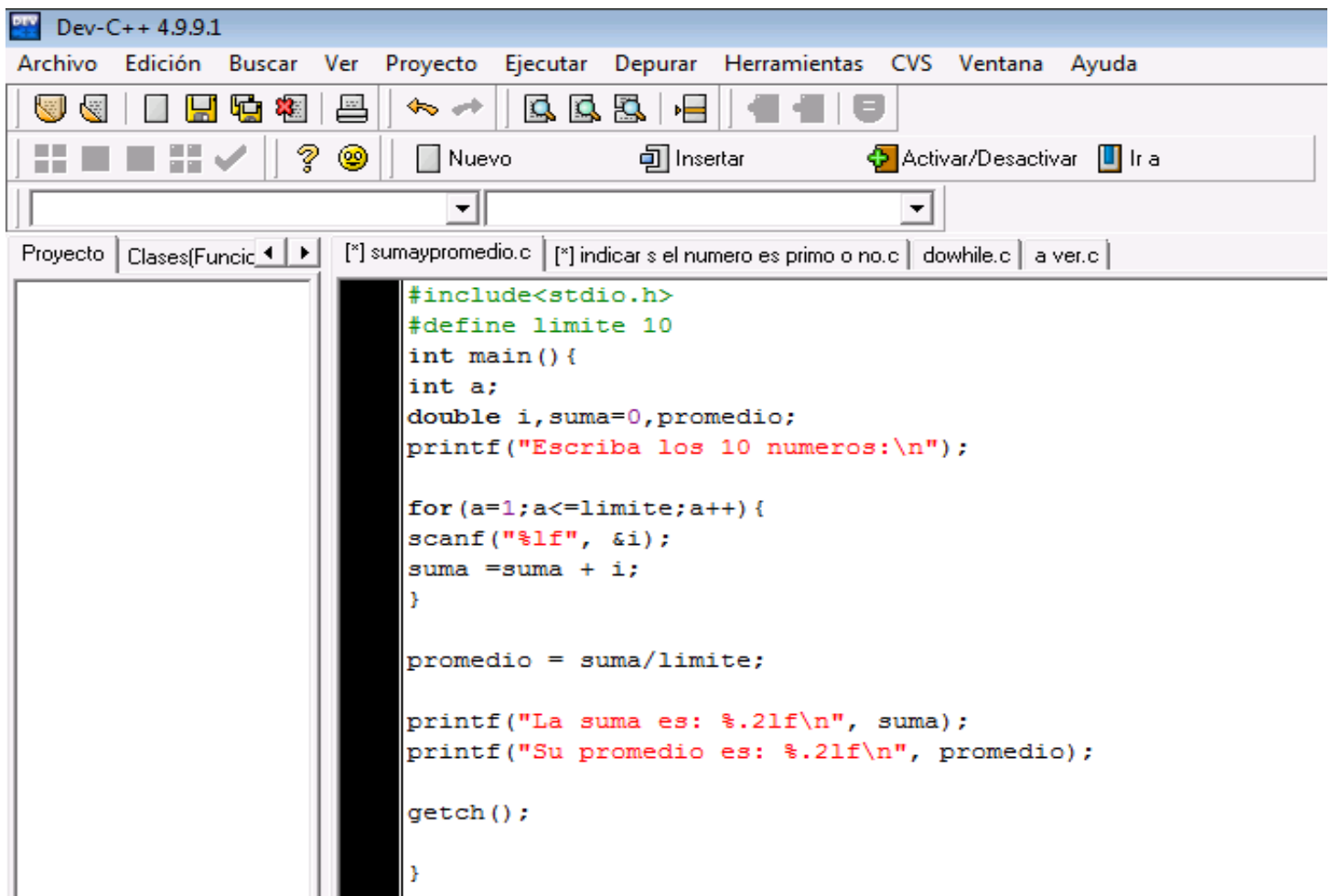
int main() {
    int a,c,r;
    printf("Escriba el numero para conocer su multiplicacion del 1 al 10\n");
    scanf("%i",&a);

    c=1;
    do{
        r=a*c;

        printf("%i * %i = %i\n",a,c,r);
        c++;
    }
    while (c!=11);
    getch();
}
```



```
Escriba el numero para conocer su multiplicacion del 1 al 10
14
14 * 1 = 14
14 * 2 = 28
14 * 3 = 42
14 * 4 = 56
14 * 5 = 70
14 * 6 = 84
14 * 7 = 98
14 * 8 = 112
14 * 9 = 126
14 * 10 = 140
```



```
#include<stdio.h>
#define limite 10
int main(){
    int a;
    double i,suma=0,promedio;
    printf("Escriba los 10 numeros:\n");

    for(a=1;a<=limite;a++){
        scanf("%lf", &i);
        suma =suma + i;
    }

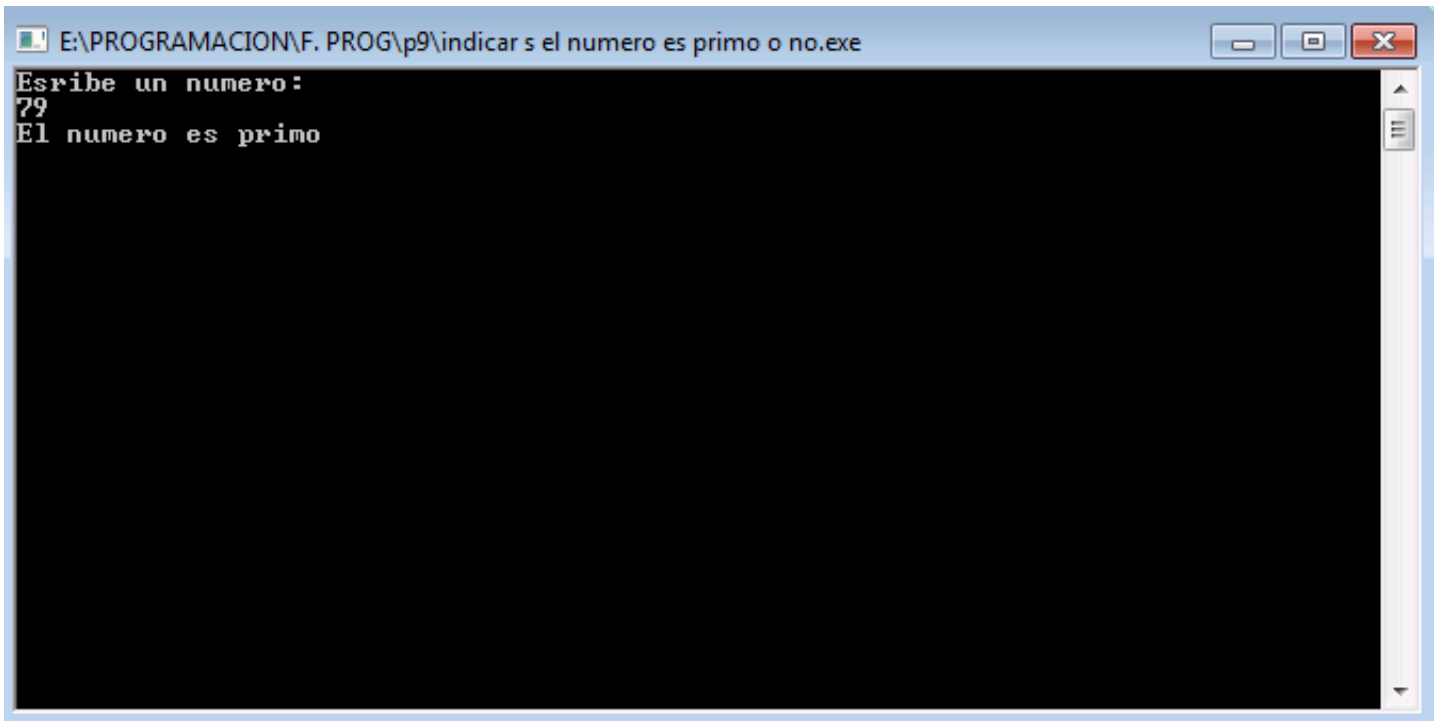
    promedio = suma/limite;

    printf("La suma es: %.2lf\n", suma);
    printf("Su promedio es: %.2lf\n", promedio);

    getch();
}
```







```
E:\PROGRAMACION\F. PROG\p9\indicar s el numero es primo o no.exe
Escribe un numero:
79
El numero es primo
```

## Conclusión

Los ciclos de repetición son muy útiles en muchos sentidos, para empezar nos sirven para hacer operaciones como sumas, divisiones, multiplicaciones, comparación de números, es decir operaciones aritméticas, pero también nos sirven para interactuar con el programa al leer datos de entrada o imprimir valores en pantalla.

Los tres ciclos son muy útiles, pero la importancia de aprender sus características es vital al momento de decidir cuál usar, dependiendo del programa que busquemos realizar ocuparemos uno de los tres ya sea para mejorar el código, minimizarlo u optimizarlo para cualquier caso. Por último, gracias a estas estructuras de repetición nos evitamos la molestia de escribir más líneas de variables y procesos.