



Universidad Nacional Autónoma de México Facultad de Ingeniería



Fundamentos de Programación (1122)

Laboratorios de computación
salas A y B

*Profesor: M.I. Marco Antonio Martínez Quintana
Semestre 2021-1*

Practica No. 9

Estructuras de Repetición

Grupo: 1129

No. de Equipo de cómputo empleado: No aplica

No. de Lista o Brigada: No aplica

No. de Lista: 42

Nombre: Adolfo Román Jiménez

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, nos permiten ejecutar un conjunto de instrucciones de manera repetida las veces que se requiera, mientras que la expresión lógica a evaluar se cumpla o sea verdadera.

En lenguaje C existen tres estructuras de repetición:

-While: Esta estructura valida la condición lógica dada y en caso de ser verdadera ejecutará el bloque de instrucciones contenido en ella, por el contrario, si la condición es falsa seguirá el flujo del programa. Las instrucciones dentro de la estructura while dejarán de ser ejecutadas cuando la condición sea falsa.

-do-while: A diferencia del while, esta estructura ejecutará una vez el conjunto de instrucciones y luego validará la condición lógica, si es falsa continuará ejecutando las instrucciones hasta que la condición se vuelva falsa.

-for: La estructura for consta de tres partes, en una de ellas se declaran variables, en la segunda se valida una condición lógica, la última parte son instrucciones que se realizan una vez se termina de ejecutar el conjunto de instrucciones, como dato curioso, la letra i es utilizada dentro de estas estructuras como referencia a iteración, pudiendo ser llamada variable de iteración.

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 *n* veces. Su sintaxis es la siguiente:

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

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

- Código (estructura de repetición while)

```
practica9.c x +
1 #include <stdio.h>
2
3 /*
4 Este programa genera la tabla de multiplicar de un número dado.
5 El número se lee desde la entrada estándar (teclado).
6 */
7
8 int main()
9 {
10     int num, cont = 0;
11
12     printf("\a----- Tabla de multiplicar \n");
13     printf("Ingrese un número: \n"); scanf("%d", &num);
14
15     printf("La tabla de multiplicar del %d es:\n", num);
16
17     while (++cont <= 10)
18
19         printf("%d x %d = %d\n", num, cont, num*cont);
20
21     return 0;
22 }
```

```
~/
x +
~/ $ gcc -o while practica9.c
~/ $ ./while
----- Tabla de multiplicar
Ingrese un número:
5
La tabla de multiplicar del 5 es:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

- Código (estructura de repetición while)

```
practica9.c x +
1 #include <stdio.h>
2 /*
3 Este programa genera un ciclo infinito.
4 */
5 int main()
6 {
7     // Al igual que en la estructura if-else
8     // 0 -> falso
9     // diferente de 0 -> verdadero
10
11     // El siguiente es un ciclo infinito
12     // porque la condición siempre es verdadera.
13     // Así mismo, debido a que el ciclo consta de una sola línea, las
14     // llaves { } son opcionales.
15
16     while (1) {
17         printf("Ciclo infinito.\nPara terminar el ciclo presione ctrl + c.\n");
18     }
19
20     return 0;
21 }
22
```

```
~/ x +
Ciclo infinito.
Para terminar el ciclo presione ctrl + c.
Ciclo infinito.
Para terminar el ciclo presione ctrl + c.
Ciclo infinito.
Para terminar el ciclo presione ctrl + c.
Ciclo infinito.
Para terminar el ciclo presione ctrl + c.
Ciclo infinito.
Para terminar el ciclo presione ctrl + c.
Ciclo infinito.
Para terminar el ciclo presione ctrl + c.
Ciclo inf^C
~/ $ ^C
```

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 muchas veces. Su sintaxis es la siguiente:

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

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
,',
,'

- Código (estructura de repetición do -while)

```
1 #include <stdio.h>  
2 /*  
3 Este programa obtiene el promedio de calificaciones ingresadas por  
4 el usuario. Las calificaciones se leen desde la entrada estándar (teclado).  
5 La inserción de calificaciones termina cuando el usuario presiona una tecla  
6 diferente de 'S' o 's'.  
7 */  
8 int main () {  
9     char op = 'n';  
10    double sum = 0, calif = 0; int veces = 0;  
11  
12    do {  
13        printf("\tSuma de calificaciones\n");  
14        printf("Ingrese la calificación:\n");  
15  
16        scanf("%lf", &calif);  
17  
18        veces++;  
19  
20        sum = sum + calif;  
21  
22        printf("¿Desea sumar otra? S/N\n");  
23  
24        setbuf(stdin, NULL); // limpia el buffer del teclado scanf("%c",&op);  
25  
26        getchar();  
27    }  
28    while (op == 'S' || op == 's');  
29  
30    printf("El promedio de las calificaciones ingresadas es: %lf\n", sum/veces);  
31  
32    return 0;  
33 }  
34
```

```
~/ $ gcc -o test practica9.c
~/ $ ./test
    Suma de calificaciones
Ingrese la calificación:
10
¿Desea sumar otra? S/N
S
    Suma de calificaciones
Ingrese la calificación:
9
¿Desea sumar otra? S/N
N
El promedio de las calificaciones ingresadas es: 9.500000
~/ $
```

- Código (estructura de repetición do -while)

```
practica9.c
1 #include <stdio.h>
2
3 /* Este programa genera una calculadora básica. */
4
5 int main ()
6 {
7     int op, uno, dos;
8
9     do
10     {
11         printf(" --- Calculadora ---\n");
12         printf("\n;Qué desea hacer\n");
13         printf("1) Sumar\n");
14         printf("2) Restar\n");
15         printf("3) Multiplicar\n");
16         printf("4) Dividir\n");
17         printf("5) Salir\n");
18
19         scanf("%d",&op);
20
21         switch(op){
22             case 1:
23                 printf("\tSumar\n");
24                 printf("Introduzca los números a sumar separados por comas\n");
25                 scanf("%d, %d",&uno, &dos);
26                 printf("%d + %d = %d\n", uno, dos, (uno + dos));
27                 break;
28
29             case 2:
30                 printf("\tRestar\n");
31                 printf("Introduzca los números a restar separados por comas\n");
32                 scanf("%d, %d",&uno, &dos);
33                 printf("%d - %d = %d\n", uno, dos, (uno - dos));
34                 break;
35
36             case 3:
37                 printf("\tMultiplicar\n");
38                 printf("Introduzca los números a multiplicar separados por comas\n");
39                 scanf("%d, %d",&uno, &dos);
40                 printf("%d * %d = %d\n", uno, dos, (uno * dos));
41                 break;
42
43             case 4:
44                 printf("\tDividir\n");
45                 printf("Introduzca los números a dividir separados por comas\n");
46                 scanf("%d, %d",&uno, &dos);
47                 printf("%d / %d = %.2lf\n", uno, dos, ((double)uno / dos));
48                 break;
49
50             case 5:
51                 printf("\tSalir\n"); break;
52
53             default:
54                 printf("\tOpción inválida.\n");
55         }
56     }
57     while (op != 5);
58
59     return 0;
60 }
```



```
~/ $ gcc -o test practica9.c
~/ $ ./test
--- Calculadora ---

¿Qué desea hacer
1) Sumar
2) Restar
3) Multiplicar
4) Dividir
5) Salir
3
    Multiplicar
Introduzca los números a multiplicar separados por comas
5,9
5 * 9 = 45
--- Calculadora ---

¿Qué desea hacer
1) Sumar
2) Restar
3) Multiplicar
4) Dividir
5) Salir
5
    Salir
~/ $
```

Estructura de control de repetición for

Lenguaje C posee la estructura de repetición for la cual permite realizar repeticiones cuando se conoce el número de elementos que se quiere recorrer. La sintaxis que generalmente se usa es la siguiente:

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

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.

- Código (estructura de repetición for)

```
practica9.c x +
1 #include <stdio.h>
2 /*
3  *Este programa genera un arreglo unidimensional de 5 elementos y
4  *accede a cada elemento del arreglo a través de un ciclo for.
5  */
6
7 int main ()
8 {
9     int enteroNumAlumnos = 5;
10
11     float realCalif = 0.0, realPromedio = 0.0;
12
13     printf("\tPromedio de calificaciones\n");
14
15     for (int indice = 0 ; indice < enteroNumAlumnos ; indice++)
16     {
17         printf("\nIngrese la calificación del alumno: %d\n", indice+1);
18         scanf("%f",&realCalif);
19         realPromedio += realCalif;
20     }
21
22     printf("\nEl promedio de las calificaciones ingresadas es: %f\n", realPromedio/enteroNumAlumnos);
23
24     return 0;
25 }
26
```

```
~/ x +
~/ $ gcc -o test practica9.c
~/ $ ./test
    Promedio de calificaciones

Ingrese la calificación del alumno: 1
9

Ingrese la calificación del alumno: 2
3

Ingrese la calificación del alumno: 3
10

Ingrese la calificación del alumno: 4
8

Ingrese la calificación del alumno: 5
7

El promedio de las calificaciones ingresadas es: 7.400000
```

Define

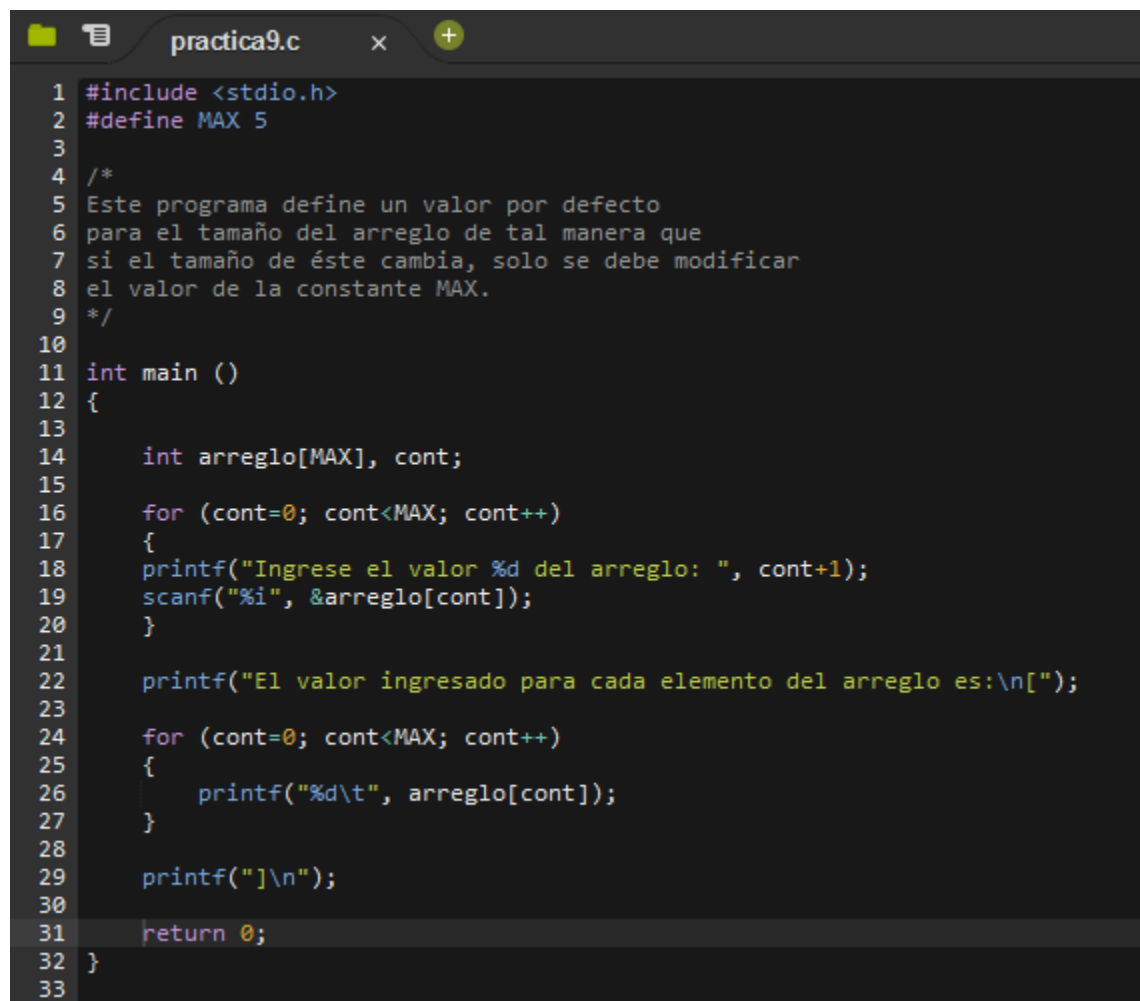
Las líneas de código que empiezan con # son directivas del preprocesador, el cual se encarga de realizar modificaciones en el texto del código fuente, como reemplazar un símbolo definido con #define por un parámetro o texto, o incluir un archivo en otro archivo con #include.

define permite definir constantes o literales; se les nombra también como constantes simbólicas. Su sintaxis es la siguiente:

```
#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.

- Código (define)



```
practica9.c x +
1 #include <stdio.h>
2 #define MAX 5
3
4 /*
5 Este programa define un valor por defecto
6 para el tamaño del arreglo de tal manera que
7 si el tamaño de éste cambia, solo se debe modificar
8 el valor de la constante MAX.
9 */
10
11 int main ()
12 {
13
14     int arreglo[MAX], cont;
15
16     for (cont=0; cont<MAX; cont++)
17     {
18         printf("Ingrese el valor %d del arreglo: ", cont+1);
19         scanf("%i", &arreglo[cont]);
20     }
21
22     printf("El valor ingresado para cada elemento del arreglo es:\n");
23
24     for (cont=0; cont<MAX; cont++)
25     {
26         printf("%d\t", arreglo[cont]);
27     }
28
29     printf("]\n");
30
31     return 0;
32 }
33
```

Cuando se compila el programa, se reemplazan la palabra MAX por el valor definido para la misma. Esto permite que, si el tamaño del arreglo cambia, solo se tiene que modificar el valor definido para MAX y en automático todos los arreglos y el recorrido de los mismos adquieren el nuevo valor (Mientras se use MAX para definir el o los arreglos y para realizar los recorridos).

```
~/ $ gcc -o test practica9.c
~/ $ ./test
Ingrese el valor 1 del arreglo: 5
Ingrese el valor 2 del arreglo: 3
Ingrese el valor 3 del arreglo: 8
Ingrese el valor 4 del arreglo: 4
Ingrese el valor 5 del arreglo: 9
El valor ingresado para cada elemento del arreglo es:
[5    3    8    4    9    ]
~/ $
```

Break

Algunas veces es conveniente tener la posibilidad de abandonar un ciclo. La proposición **break** proporciona una salida anticipada dentro de una estructura de repetición, tal como lo hace en un switch. Un *break* provoca que el ciclo que lo encierra termine inmediatamente.

- Código (Break)

```
practica9.c
1  #include <stdio.h>
2
3  /*
4   * Este programa hace una suma de números. Si la suma rebasa la cantidad
5   * de 50 el programa se detiene.
6   */
7  #define VALOR_MAX 5
8
9  int main ()
10 {
11     int enteroSuma = 0;
12     int enteroNumero = 0;
13     int enteroContador = 0;
14
15     while (enteroContador < VALOR_MAX)
16     {
17         printf("Ingrese un número:");
18         scanf("%d", &enteroNumero);
19         enteroSuma += enteroNumero;
20         enteroContador++;
21
22         if (enteroSuma > 50)
23         {
24             printf("Se rebasó la cantidad límite.\n");
25             break;
26         }
27     }
28 }
29
```

Cuando se compila el programa, MAX se sustituye por 5.

```
~/ $ gcc -o test practica9.c
~/ $ ./test
Ingrese un número:5
Ingrese un número:9
Ingrese un número:1
Ingrese un número:8
Ingrese un número:6
~/ $ ./test
Ingrese un número:10
Ingrese un número:20
Ingrese un número:40
Se rebasó la cantidad límite.
~/ $
```

Continue

La proposición **continue** provoca que inicie la siguiente iteración del ciclo de repetición que la contiene.

- Código (continue)

```
practica9.c
1 #include <stdio.h>
2
3 /*
4  * Este programa obtiene la suma de un LIMITE de números pares ingresados
5  */
6
7 #define LIMITE 5
8
9 int main ()
10 {
11     int enteroContador = 1;
12     int enteroNumero = 0;
13     int enteroSuma = 0;
14
15     while (enteroContador <= LIMITE)
16     {
17         printf("Ingrese número par %d:", enteroContador);
18         scanf("%d",&enteroNumero);
19
20         if (enteroNumero%2 != 0)
21         {
22             printf("El número insertado no es par.\n"); continue;
23         }
24         enteroSuma += enteroNumero; enteroContador++;
25     }
26
27     printf("La suma de los números es: %d\n", enteroSuma);
28
29     return 0;
30 }
31
32
33
```

```
~/ $ gcc -o test practica9.c
~/ $ ./test
Ingrese número par 1:8
Ingrese número par 2:2
Ingrese número par 3:6
Ingrese número par 4:10
Ingrese número par 5:20
La suma de los números es: 46
~/ $ ./test
Ingrese número par 1:5
El número insertado no es par.
Ingrese número par 1:9
El número insertado no es par.
Ingrese número par 1:^C
```

Actividades:

- Elaborar un programa que utilice la estructura *while* en la solución de un problema
- Elaborar un programa que requiera el uso de la estructura *do-while* para resolver un problema. Hacer la comparación con el programa anterior para distinguir las diferencias de operación entre *while* y *do-while*.
- Resolver un problema dado por el profesor que utilice la estructura *for* en lugar de la estructura *while*.
- Usar la directiva *define* para elaboración de código versátil.

mario.c

El siguiente programa fue hecho por mi como parte del programa de introducción a la ciencia computacional (CS50) del curso en línea de la universidad de Harvard.

Este programa busca recrear una media pirámide doble hecha con hashes como las que se encuentran en Super Mario Brothers, existe un ejercicio mas simple que solo hace una pirámide, en este que es mas complicado se pide hacer las dos.



El programa utiliza un *do-while* loop para pedir al usuario un valor de altura que no debe de ser menor a 1 y tampoco mayor a 8, una vez que se ingresa el valor aceptado, el primer *for*-loop contiene a otros 2 indentados, el primero se encarga de la fila de la pirámide mientras que los otros dos se encargan de las columnas.

Escribí en ingles el programa ya que es el idioma de la clase y le adapte un par de directivas *define* para los valores constantes de altura máxima y mínima.

Le pide al usuario una altura para la pirámide que no deberá de ser menor a 1 ni tampoco mayor a 8, estos valores están definidos como constantes.

```

1  #include <stdio.h>
2  #include <cs50.h>
3
4  #define MIN 1
5  #define MAX 8
6
7  int main(void)
8  {
9
10     //Declare variables
11     int height, length, i, j;
12
13     //Asks user for pyramid height
14     do
15     {
16         height = get_int("Height: ");
17     }
18     while (height < MIN || height > MAX);
19

```

El primer loop toma como valor principal la altura dada por el usuario en height, despues el segundo loop se encarga de imprimir el valor especificado a traves de un operador ternario, una vez que termina, se imprimen los espacios que esta entra cada una de las medias piramides y por ultimo el segundo loop imprime los hashes correspondientes a la segunda mitad.

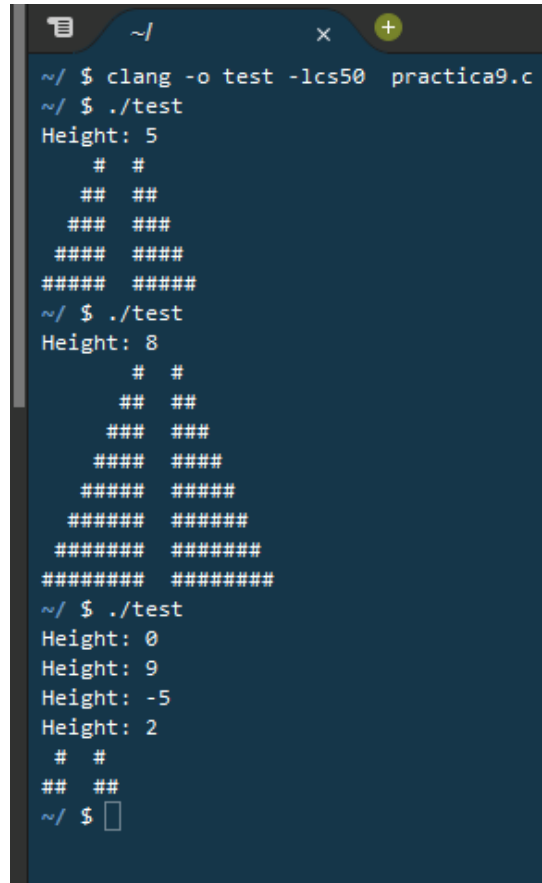
```

20     //Sets number of spaces that will be later printed
21     length = height - MIN;
22
23     //Creates a loop for each row
24     for (i = 0; i < height; i++)
25     {
26         //Creates a loop for each cell printed
27         for (j = 0; j < height; j++)
28         {
29             //Prints space //Prints hash
30             j < length ? printf(" ") : printf("#");
31         }
32
33         printf(" "); //Prints space in between
34
35         for (j = 0; j <= i; j++) //Creates loop for second half of pyramid
36         {
37             printf("#"); //Prints hash
38         }
39
40         printf("\n"); //Moves to the next line
41
42         length--;
43     }
44
45 }

```


Funcionamiento:

De esta forma el programa funciona como debe, notese que en la compilacion utilice un `-lcs50` precisamente para incluir en la compilacion la librería `cs50.h` que contiene la funcion `get_int` que fue disenada por ellos y que se pide usar en el programa.



```
~/ $ clang -o test -lcs50 practica9.c
~/ $ ./test
Height: 5
# #
## ##
### ###
#### ####
##### #####
~/ $ ./test
Height: 8
# #
## ##
### ###
#### ####
##### #####
##### #####
##### #####
##### #####
~/ $ ./test
Height: 0
Height: 9
Height: -5
Height: 2
# #
## ##
~/ $
```

Credit.c

El siguiente programa de igual forma requiere una solución con loops, es la implementación del algoritmo de Luhn, un ingeniero de IBM quien inventó este algoritmo que las tarjetas de crédito o débito usan para validar si el número de la tarjeta es correcto o si es una tarjeta falsa.

El algoritmo de Luhn es simple, dado el número de la tarjeta, de derecha a izquierda se multiplica y suma cada número par por dos, una vez que se terminan los dígitos, se suman los restantes, por ejemplo:

Para el número de tarjeta Visa: 4003600000000014.

Se multiplican los dígitos que están subrayados

4003600000000014

$$1 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 0 \cdot 2 + 6 \cdot 2 + 0 \cdot 2 + 4 \cdot 2$$

Y después se suman:

$$2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13$$

Si la multiplicación en la multiplicación un resultado es mayor que 10, entonces se suman ambos dígitos para que de una cifra solo en unidades.

$$\text{Ej. } 6 \cdot 2 = 12 \rightarrow 1 + 2 = 3$$

Paso siguiente, se suma ese resultado más el de los dígitos que no se multiplicaron:

$$13 + 4 + 0 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20$$

Si el resultado de esta operación es una cifra con un 0 al final, entonces quiere decir que el número, es un número de tarjeta potencialmente válido.

Para lograr la implementación el algoritmo únicamente a través de operadores matemáticos, usamos el módulo para ir descomponiendo el número original que es un número entero de al menos 16 cifras.

Implementacion:

```
1 #include <stdio.h>
2 #include <cs50.h>
3
4 int main(void)
5 {
6     long number = get_long("Number: "); //Pide el numero de la tarjeta
7     int sum = 0; //La suma principal que usamos para saber si la tarjeta es valida
8     int count = 0; //Lleva el conteo de posicion de las cifras del numero de tarjeta
9     string id = 0; //Identificador para saber si la tarjeta es MC, VISA o AMEX
10 }
```

Este que es el while loop principal, genera todo el algoritmo de Luhn y da valores a las variables:

```
11 while (number) //Mientras el numero de tarjeta tiene un valor mayor a 0, se ejecuta la funcion.
12 {
13     if (count % 2 == 0) //Si el valor de count es par
14     {
15         sum += number % 10; //Se calcula el residuo del numero mod 10 y se suma a la principal
16     }
17     else if (number % 10 * 2 >= 10) //Si la multiplicacion es mayor a 10
18     {
19         sum += number % 10 * 2 - 9; //Se le resta 9 al producto y se suma
20     }
21     else
22     {
23         sum += number % 10 * 2; // Si la multiplicacion es menor a 10, solo se suma el residuo
24     }
25
26     number = (number - (number % 10)) / 10; //Se le extrae el ultimo dígito al numero de tarjeta
27
28     if (number > 33 && number < 56) //Cuando solo quedan dos digitos en la tarjeta, se evalua su procedencia
29     {
30         if (number > 50) //Del 51 al 56 es MasterCard
31         {
32             id = "MASTERCARD";
33         }
34         else if (number > 39 && number < 50) //Del 40 al 49 es VISA
35         {
36             id = "VISA";
37         }
38         else if (number == 34 || number == 37) //34 o 37 es American Express
39         {
40             id = "AMEX";
41         }
42     }
43
44     count++; //Se suma una unidad al contador de lugar de digitos de la tarjeta
45
46 };
```

Se evalúan finalmente los resultados, que la longitud del número equivalga a dígitos de una tarjeta válida (count), que el resultado de la suma principal sea una decena entera y que el ID tenga un valor válido.

Si alguno falla, la salida muestra un resultado de inválido, de lo contrario imprime la compañía a la que la tarjeta pertenece.

```

47 //Si el numero ingresado no corresponde al de una tarjeta de debito, entonces es invalida.
48 if (((count < 13 || count > 16) || count == 14) || sum % 10 != 0) || id == 0)
49 {
50     printf("INVALID\n");
51 }
52 else //Si si corresponde, entonces se imprime su procedencia.
53 {
54     printf("%s\n", id);
55 }
56 }

```

Funcionamiento:

Tomamos algunos números de tarjetas que PayPal recomienda para poder hacer pruebas.

American Express	378282246310005
American Express	371449635398431
Mastercard	5555555555554444
Mastercard	5105105105105100
Visa	4111111111111111
Visa	4012888888881881

```

~/ $ clang -o test -lcs50 practica9.c
~/ $ ./test
Number: 378282246310005
AMEX
~/ $ ./test
Number: 371449635398431
AMEX
~/ $ ./test
Number: 5105105105105100
MASTERCARD
~/ $ ./test
Number: 5555555555554444
MASTERCARD
~/ $ ./test
Number: 4012888888881881
VISA
~/ $ ./test
Number: 4111111111111111
VISA
~/ $ ./test
Number: 651651654544
INVALID
~/ $ ./test
Number: 58
INVALID
~/ $ ./test
Number: 4848456451151561515448464654545611
Number: 561561111561
INVALID
~/ $ 

```

Conclusión:

Me ha ayudado mucho conocer como implementar correctamente las estructuras de repetición y facilitan mucho el trabajo cuando saben usarse correctamente. A pesar de que la mayoría pueden parecer iguales, en realidad tienen características propias que cada programador debe aprender a explotar para hacer que su código sea mas eficiente a través de usar las herramientas correctas.

El uso de las estructuras de repetición me parece una herramienta demasiado útil para elaborar software eficiente y seguro.

Referencias:

- Facultad de ingeniería - UNAM. (2018, 6 abril). Manual de prácticas del laboratorio de Fundamentos de programación. Recuperado 03 de diciembre de 2020, de: http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf
- Cursos en Línea. (s. f.). Soluciones MyL. Recuperado 03 de diciembre de 2020, de http://solucionesmyl.com/cursos/lenguaje_c/scanf.html
- Introduction to Computer Science, Harvard's CS50 Week 1: C, <https://cs50.harvard.edu/x/2021/psets/1/>
- Pay Pal Test Payflow Transactions: <https://developer.paypal.com/docs/payflow/payflow-pro/payflow-pro-testing/>