UnADM
UNIVERSIDAD ABIERTA Y A DISTANCIA DE MÉXICO

Programa de la asignatura:

Programación

U3 Estructuras de control

DCSBA

BIOTECNOLOGÍA

Índice

Presentación de la unidad

En la primer unidad, mediante el mundo de la ardilla, aprendiste que cualquier algoritmo puede ser escrito utilizando únicamente tres tipos de instrucciones, conocidas como estructuras de control: secuenciales (cuando se ejecutan una tras otra), selectivas (cuando se ejecutan dependiendo de una condición) y repetitivas (que se ejecutan varias veces en función de una condición); su objetivo es controlar el flujo de ejecución de un programa, es decir, el orden en que se ejecutan las instrucciones.

Considerando que en la unidad anterior diseñaste algoritmos secuenciales y los codificaste en lenguaje C, para obtener el programa deseado que diera solución al problema en cuestión, podemos decir que solamente te falta saber cómo funcionan y cómo se codifican en lenguaje C las otras dos estructuras para poder diseñar algoritmos estructurados. Así que esto es justamente el tema de esta unidad, aquí estudiarás con más detalle los tipos y funcionamiento de las estructuras selectivas y repetitivas, introducidas en la unidad 1.

Para su mejor comprensión, esta unidad está dividida en dos partes: en la primera revisarás algunos problemas donde la solución implica tener que elegir el camino que se debe seguir para llegar al resultado deseado, los cuales se solucionan utilizando estructuras selectivas, por lo cual se analizarás a mayor profundidad el significado (semántica) de cada estructura y verás la forma de codificarla (sintaxis) en lenguaje C. La segunda parte está dedicada a las estructuras repetitivas, para las cuales se sigue la misma estrategia, verás cómo se pueden solucionar problemas utilizando este tipo de estructuras y también analizarás su semántica y aprenderás su sintaxis en lenguaje C. De esta manera, al finalizar la unidad podrás construir programas que incluyan cualquier tipo de estructura de control, así como poder trabajar con variables tipo arreglo.

Propósitos

En esta Unidad:

- Construirás expresiones booleanas para modelar situaciones reales.
- Diseñarás algoritmos para resolver problemas que impliquen la toma de decisiones, utilizando estructuras selectivas.
- Diseñarás algoritmos para resolver problemas que realicen una misma tarea varias veces usando estructuras repetitivas.
- Codificarás en lenguaje C algoritmos estructurados.

Competencia específica

Utilizar estructuras de control selectivas y repetitivas para resolver problemas simples a través del desarrollo de programas en lenguaje C.

3.1. Estructuras selectivas

Para diseñar programas capaces de tomar decisiones se requiere de las estructuras de control selectivas, también llamadas condicionales. Éstas llevan a cabo su función (controlar el flujo del programa) mediante una condición que se representa utilizando expresiones booleanas, de tal manera que cuando la condición se cumple (es verdadera) se ejecuta un conjunto de instrucciones definidas para este caso y, dependiendo del tipo de estructura, es posible que se ejecute otro conjunto de instrucciones distinto para el caso contrario (cuando la condición es falsa); e incluso, es posible definir diferentes conjuntos de instrucciones para valores distintos que pudiera tomar una variable. Es así que dependiendo de su estructura se han definido tres tipos: simples, dobles y múltiples.

Para el estudio de cada estructura selectiva, a continuación, se dedican tres subsecciones, una para cada una, en las cuales entenderás cómo funcionan y la forman en que se codifican en lenguaje C.

3.1.1. Estructura selectiva simple (*if*)

La estructura de decisión simple, como su nombre lo indica, permite decidir entre ejecutar o no un bloque de acciones; en pseudocódigo se propuso la palabra reservada Si para su representación y en lenguaje C esta estructura se codifica mediante la sentencia de control if, tal como se muestra en la siguiente tabla.

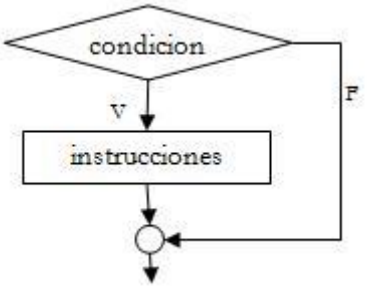
Pseudocódigo	Diagrama de Flujo	Código en C
Si <condición> entonces <instrucciones> Fin Si		**if(**<condición>**)** <instrucciones>

Tabla 3.1: Representaciones de la estructura condicional simple

La <condición> puede ser cualquier expresión booleana y las <instrucciones>, llamadas cuerpo del *Si* (if), bien puede ser una sola instrucción o un bloque de instrucciones.

La manera en la que se ejecuta una instrucción *Si* (if) es la siguiente: se evalúa la condición que aparece entre paréntesis y si es verdadera (tiene un valor diferente de cero) entonces se ejecutan las instrucciones del cuerpo del *Si* (if), en caso de no serlo no se ejecuta y continúa el flujo de ejecución.

NOTA: En lenguaje C, cuando el cuerpo de una estructura tiene más de una instrucción éstas deben ir encerradas entre llaves.

Para ilustrar las representaciones y el funcionamiento de la estructura selectiva simple se presenta el siguiente problema, con el algoritmo en pseudocódigo y programa en lenguaje C.

Problema 3.1: Se requiere un programa que lea un valor entre 0 y 360 y determine el tipo de ángulo, considerando que:

- Angulo agudo: Mayor a cero y menor de 90 grados
- Angulo reto: Es igual a 90 grados
- Angulo obtuso: Es mayor que 90 pero menor a 180 grados
- Angulo llano: Es igual a 180 grados
- Angulo cóncavo: Es mayor a 180 pero menor a 360 grados

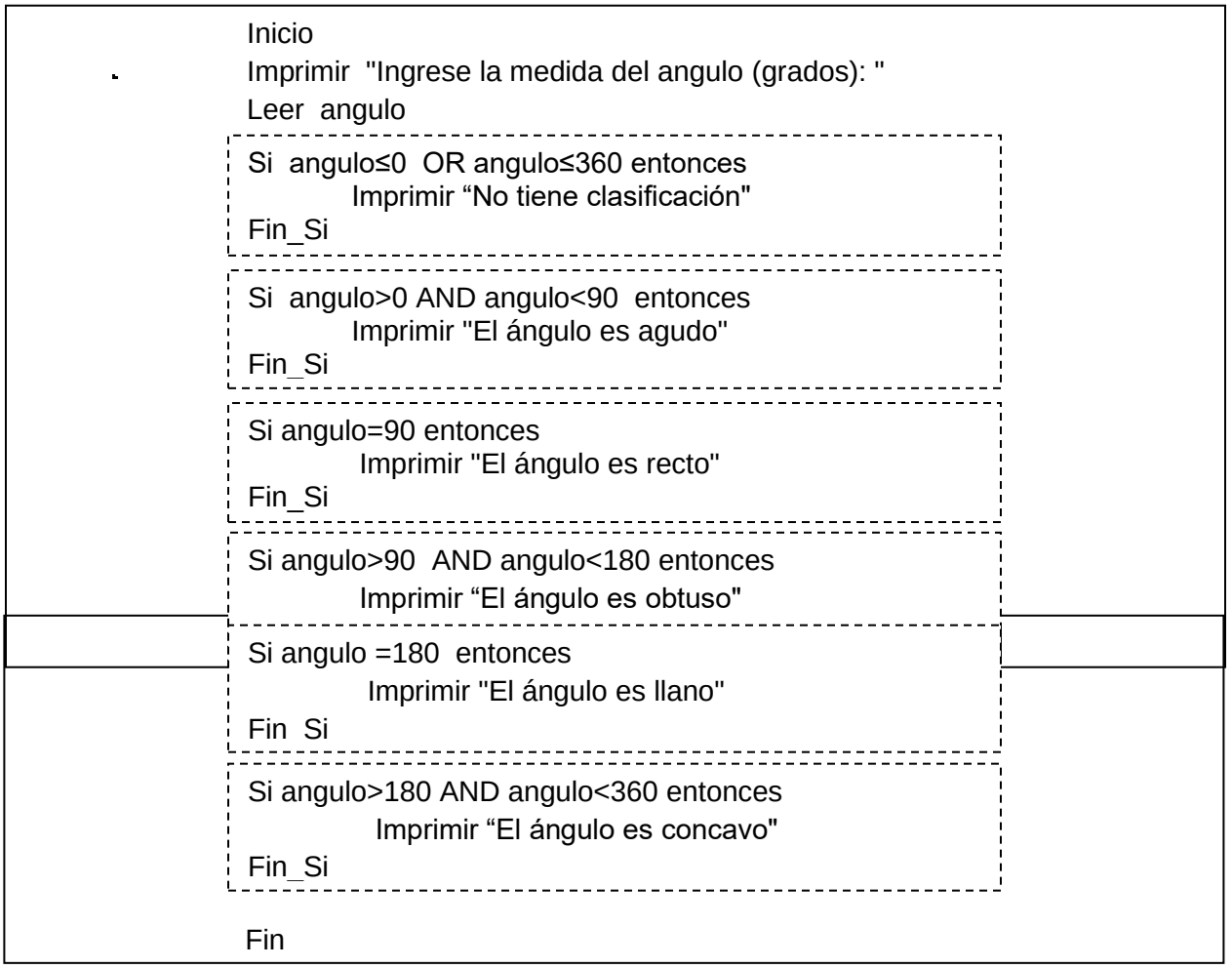
El análisis del problema se resume en la siguiente tabla.

Análisis del problema

Datos de entada: *ángulo*	**Salida:** Mensaje 1: “Agudo” Mensaje 2: “Recto” Mensaje 3: “Obtuso” Mensaje 4: “Llano” Mensaje 5: “Cóncavo”	**Método:** Realizar comparaciones utilizando la estructura de selección simple para determinar el tipo de ángulo, se requiere una por cada tipo

Tabla 3.2: Análisis del problema 3.1

Lo primero que se requiere es leer el valor del ángulo, posteriormente, verificar de qué tipo es para imprimir el mensaje indicado. A continuación, se muestra el algoritmo, en pseudocódigo:



```
Inicio
Imprimir  "Ingrese la medida del angulo (grados): "
Leer  angulo
Si  angulo≤0  OR angulo≤360 entonces
        Imprimir “No tiene clasificación"
Fin_Si
Si  angulo>0 AND angulo<90  entonces
        Imprimir "El ángulo es agudo"
Fin_Si
Si angulo=90 entonces
        Imprimir "El ángulo es recto"
Fin_Si
Si angulo>90  AND angulo<180 entonces
        Imprimir “El ángulo es obtuso"
Si angulo =180  entonces
        Imprimir "El ángulo es llano"
Fin  Si
Si angulo>180 AND angulo<360 entonces
        Imprimir “El ángulo es concavo"
Fin_Si

Fin
```

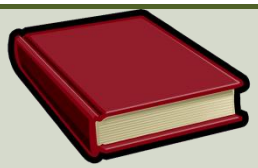
Algoritmo 3.1.a: Tipo de ángulo - pseudocódigo

Observa que, para hacer más legible el algoritmo en pseudocódigo, se han dejado sangrías para indicar qué instrucciones forman el cuerpo de cada una de las estructuras *Si* y se han encerrado con un rectángulo, esto se adoptará para cualquier bloque de instrucciones que corresponda al cuerpo de una estructura.

Continuemos con la codificación. Observa que el cuerpo de cada una de las estructuras consta de una instrucción por lo tanto no es necesario encerrarla entre llaves {}.

```c
/* Programa: tipoAngulo.c
Descripción: Determina el tipo de angulo (agudo, recto, obtuso, llano
o cóncavo) */
#include<stdio.h>
#include<stdlib.h>
/* Función Principal*/
main () { /*Inicio de la función Principal*/
    /*Declaración de variables */
    int angulo;
    /*Mensaje de bienvenida*/
    printf ("\nEste programa determina de que tipo es el angulo dado.");
    /*Instrucciones */
    printf ("\n\nIngrese la medida del angulo (grados): ");
    scanf ("%d",&angulo);
    if (angulo<=0 || angulo>=360)
       printf ("\n No tiene clasificación");

    if (angulo>0 &&angulo<90)
       printf ("\n El angulo es agudo");

    if (angulo==90)
       printf ("\n El angulo es recto");

    if (angulo>90 &&angulo<180)
       printf ("\nElangulo es obtuso");

    if (angulo ==180)
       printf ("\n El angulo es llano");

    if (angulo>180 &&angulo<360)
       printf ("\nElangulo es concavo");

    printf ("\n\n\t");
    system ("pause");

} /*Fin de la función Principal*/
```

Programa 3.1: tipoAngulo.c



Video 3.1: Consulta el video de desarrollo de la solución anterior, también podrás encontrar el programa fuente c en el material adicional de la unidad.

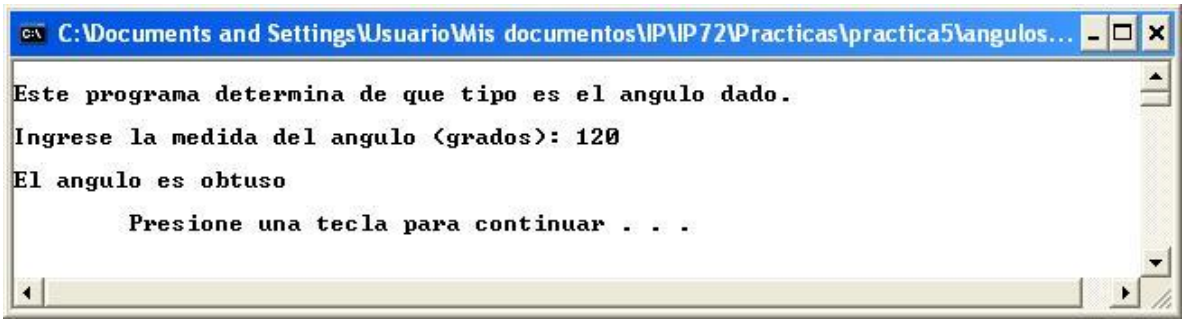
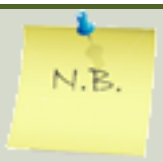


Figura 3.1: Ejecución del programa tipoAngulo.c

En la presente unidad, se sugiere el uso de herramientas (*applets*) en línea que podrás consultar en el sitio: http://www.pcg.ull.es/edapplets/en donde existen distintas actividades para el estudio de las estructuras de control, así como para algunos otros temas de programación.

Se sugiere consultes el video que contiene las bases de las estructuras de control como los que podrías aplicar a lo largo de la unidad: https://www.youtube.com/watch?v=3oK6SIIMnGk y https://www.youtube.com/watch?v=txJnxfs10wk



A lo largo del texto se harán algunas sugerencias de información adicional que corresponde a la sección *Para saber más,* pero serán insertadas a lo largo de la unidad para complementar el contenido.

3.1.2. Estructura selectiva doble (*if-else*)

Las *estructuras selectivas dobles* nos permiten elegir alguna de dos posibles acciones a realizar dependiendo de la condición. En pseudocódigo se propone usar las palabras reservadas *Si-Sino* y en C se codifican mediante la sentencia `if-else`, tal como se muestra en la siguiente tabla.

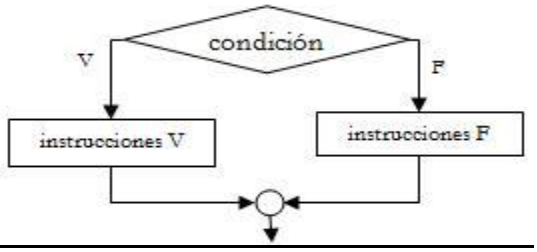
Pseudocódigo	Diagrama de Flujo	Lenguaje C
Si (<condición>) entonces <instruccionesV> sino <instruccionesF> Fin Si-Sino		

Tabla 3.3: Representaciones de la estructura condicional doble

Al igual que en la estructura selectiva simple, la <condición> representa una expresión booleana y, las <instruccionesV> y <instruccionesF> puede ser una o varias, a las primeras se les llama *cuerpo del Si* (`if`) y las segundas son el *cuerpo del Sino* (`else`).

Esta estructura de control ejecuta sólo uno de los dos cuerpos de instrucciones: cuando la condición es verdadera se ejecutan las <instrucciuonesV> y en caso contrario se ejecutan las <instruccionesF>.
En el desarrollo del siguiente problema se ejemplifican las representaciones y el funcionamiento de esta estructura.
Problema 3.2: Se requiere un programa que solicite el año de nacimiento de un votante, determine si su edad es válida para poder emitir su voto, si cumple con la condición y su edad es mayor o igual a los 70 años indicar al que pase a una casilla especial, en caso contrario indicar solo un mensaje de bienvenida.

El algoritmo en pseudocódigo es el siguiente:

Inicio
Imprimir “Año de nacimiento: ”
Leer nac
edad = anio_actual – nac

```
si (edad>=18) entonces
si (edad>=70) entonces
            imprimir “Pase a casilla especial”
      si no
            imprimir “Bienvenido”
      fin si_no
si no
      imprimir “Edad no valida para votar”
fin si_no
```

Fin

Algoritmo 3.2.a: Valida voto

Para la codificación en lenguaje C se debe notar que el cuerpo del *Si* (`if`) tiene un bloque de más instrucciones, por lo que deberán ir encerradas entre llaves `{}`.

```c
/* Programa: validavoto.c
Descripción: Determina si la edad de una persona es valida para votar
si cumple con esta condicion y es mayor a 70 años indicar que pase
a una casilla especial */
#include<stdio.h>
#include<stdlib.h>
/* Función Principal*/
main () { /*Inicio de la función Principal*/
    /*Declaración de variables */
    int nac,edad;
    /*Mensaje de bienvenida*/
    printf ("\nEste programa determina si el usuario puede votar.");
    /*Instrucciones */
    printf ("\n\nIngrese su fecha de nacimiento: ");
    scanf ("%d",&nac);

    edad=2015-nac;
    if (edad>=18){
       if (edad>=70){
          printf ("\n Pase a la casilla especial");
       }else{
          printf ("\ Bienvenido pase a emitir su voto");
       }
    }else{
          printf ("\n Edad no valida para votar");
    }

    printf ("\n\n\t");
    system ("pause");

} /*Fin de la función Principal*/
```

Programa 3.2: validavoto.c

En la siguiente figura se muestra la ejecución de este programa 3.2

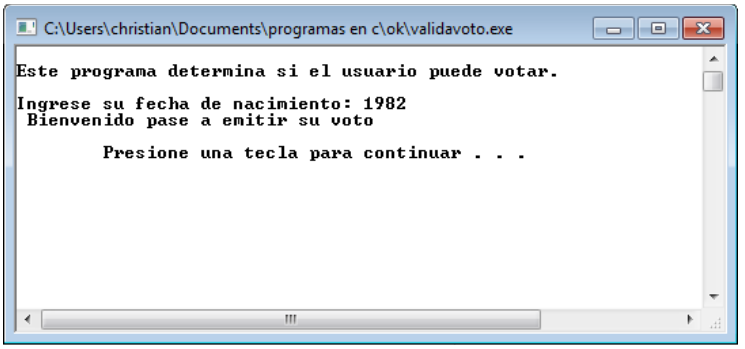
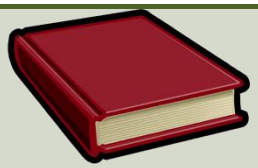


Figura 3.2: Ejecución del programa validavoto.c



Video 3.2: Consulta el video de desarrollo de la solución anterior, también podrás encontrar el programa fuente c en el material adicional de la unidad.

3.1.3. Estructura selectiva múltiple (*switch-case*)

Las *estructuras selectivas múltiples* permiten escoger uno de varios caminos posibles. Para la estructura condicional múltiple se proponen las palabras clave *Seleccionar-caso* en pseudocódigo, misma que se implementa en lenguaje C utilizando las palabras reservadas `switch-case`. Esta secuencia se utiliza cuando existen múltiples posibilidades para la evaluación de una expresión matemática (generalmente una variable), pues de acuerdo con el valor que tome la expresión será el conjunto de instrucciones que se ejecute.

Pseudocódigo	**Lenguaje**

Casos para<expresión>
caso<valor1>:
<instruccionesCaso1>
caso<valor2>:
<instruccionesCaso2>
…
otros casos:
<instruccionesOtros>
Fin_Casos
	switch (<expresión>)
	{
	case<valor1>:
	<instrucciones1>;
	break;
	case<valor2>:
	<instrucciones2>;
	break;
	default:
	<instruccionesOtras>
	}

Diagrama de Flujo	

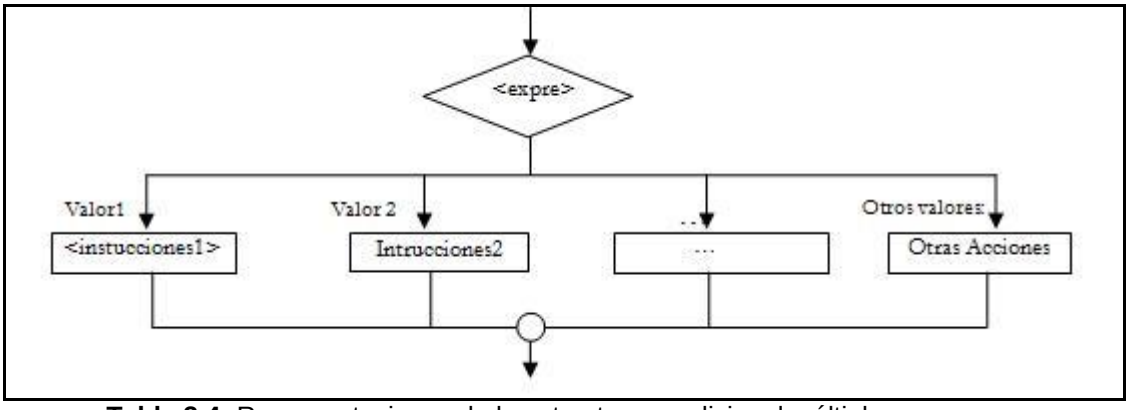


Tabla 3.4: Representaciones de la estructura condicional múltiple

En este caso la <expresión> no es booleana sino aritmética y de tipo entero, así cada caso corresponde a un valor que puede resultar de su evaluación. De esta forma, el flujo de control que sigue la ejecución de una instrucción *Seleccionar-casos* (`switch-case`) es el siguiente: se evalúa la <expresión> y si el valor corresponde al valor de un caso, es decir a un <valor$_i$>, se ejecuta el bloque de <instrucciones$_i$> hasta encontrar el final de la instrucción, que en el caso de C está representado por la palabra reservada `break`, terminando ahí la ejecución de la instrucción. Cuando el valor no corresponde a ningún caso se ejecuta el bloque de instrucciones correspondiente a otros casos (`default`). El conjunto de todos los casos, incluyendo el `default`, conforman el *cuerpo de la estructura Seleccionar-casos* (`switch-case`).

Problema 3.3: Se requiere un programa que dada una calificación con número despliegue un mensaje, de acuerdo con la siguiente información:

0-6: Reprobado
7: Suficiente, Aprobado
8: Bien, Aprobado
9: Notable, Aprobado
10: Sobresaliente, Aprobado

En este caso es conveniente utilizar una estructura selectiva múltiple, en donde la expresión que se evalúe sea la calificación del estudiante y se defina un caso por cada una de las calificaciones posibles. Es claro, que la entrada únicamente es la calificación y la salida es el mensaje correspondiente. De lo anterior el algoritmo en pseudocódigo y diagrama de flujo quedaría de la siguiente forma.

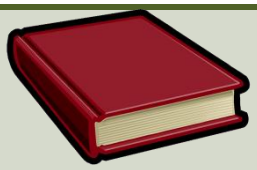
```
Inicio
        Imprimir “ Inserte una calificación: "
    Leer nota Seleccionar (nota)
    caso 0: caso 1: caso2: caso 3: caso 4: caso 5: caso 6:
        Imprimir “Reprobado"
    caso 7:
        Imprimir "Suficiente, Aprobado"
    caso 8:
        Imprimir "Bien, Aprobado"
    caso 9:
        Imprimir “Notable, Aprobado”
    caso 10:
        Imprimir “Sobresaliente, Aprobado”
    otros casos:
        Imprimir "Esa nota es incorrecta" Fin_Casos
Fin
```

Algoritmo 3.3.a: Conversión de calificación numérica a letra - pseudocódigo

Es importante señalar que, a diferencia de las estructuras anteriores, el cuerpo de una estructura selectiva múltiple siempre debe ir encerrado entre llaves {} cuando se codifica en C, más no así las instrucciones que se definen para cada caso, ya que estás se acotan por las palabra reservadas case y break**,** por tal motivo no debes olvidar escribir el break al final de cada caso de lo contrario también se ejecutarán las instrucciones de los casos que aparezcan después.

```c
/* Programa: calificacion.c
*   Descripción: Dada una calificación con número despliega un mensaje
*   0,1,2,3,4,5,6 - Reprobado
*   7 - Suficiente, Aprobado
*   8 - Bien, Aprobado
*   9 - Notable, Aprobado
*   10 - Sobresaliente, Aprobado*/
#include<stdio.h>
#include<stdlib.h>
main(){/*Función principal*/
    int nota; /*Declaración de variables*/
    /*Mensaje de bienvenida */
    printf("\nEl siguiente programa lee una calificacion con número, \n
    determina que tipo de calificacion es\n");
    /*Datos de entrada*/
    printf("\nInserte una calificacion numérica: "); scanf("%d",&nota);
    /*Comparación*/
    switch(nota){
                case 0: case 1: case 2: case 3: case 4: case 5: case 6:
                     printf("\n\n\t\"Reprobado\"");
                     break;
                case 7:
                     printf("\n\n\t\"Suficiente, Aprobado\"");
                     break;
                case 8:
                     printf("\n\n\t\"Bien, Aprobado\"");
                     break;
                case 9:
                     printf("\n\n\t\"Notable, Aprobado\"");
                     break;
                case 10:
                     printf("\n\n\t\"Sobresaliente, Aprobado\"");
                     break;
                default:
                printf("\n\n\t\"Esa nota es incorrecta\"");
    }
    printf ("\n\n\t\t");
    system ("pause");
}
```

Programa 3.3: calificacion.c



Video 3.3: Consulta el video de desarrollo de la solución anterior, también podrás encontrar el programa fuente c en el material adicional de la unidad.

En la siguiente figura se muestra la ejecución de este programa con el valor de entrada igual a 8.

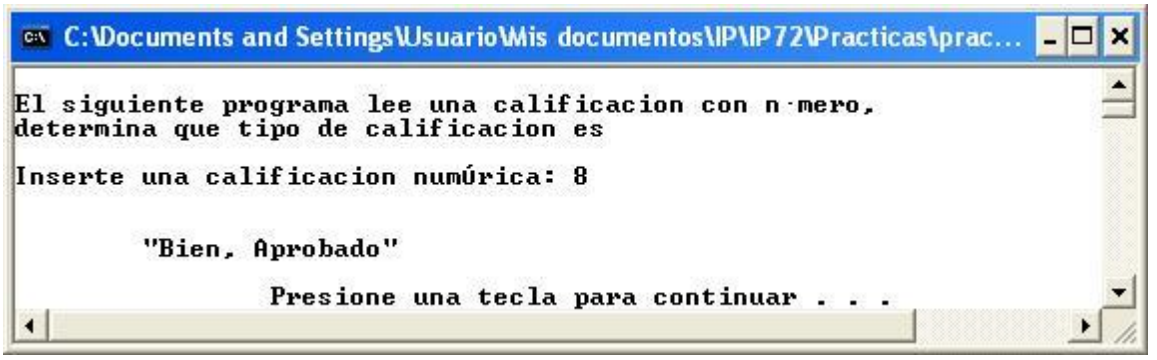


Figura 3.3: Ejecución del programa calificacion.c

A lo largo de esta sección has estudiado los tres tipos de estructuras selectivas y por medio de los ejemplos presentados te has dado cuenta de la importancia y utilidad de estas estructuras, sin ellas sería imposible construir programas que implicaran la toma de decisiones. Sin embargo, todavía existen problemas que requieren de otro tipo de estructuras que permitan repetir una tarea un número determinado de veces, la siguiente sección está dedicada a este tema.

3.2. Estructuras repetitivas

En la mayor parte del diseño o implementación de las soluciones que se plantea a problemas específicos nos encontramos con instrucciones que deben ejecutarse un número determinado de veces, si hacemos un análisis más profundo de estas situaciones, en la mayoría de las ocasiones nos encontramos que las instrucciones son las mismas, pero que los datos varían, esto se hace posible utilizando las *estructuras repetitivas*, generalmente llamadas *ciclos.*

Existen varias estructuras de repetición implementadas por los diferentes lenguajes de programación, todas con la misma idea: repetir un conjunto de instrucciones, llamadas *cuerpo del ciclo*, dependiendo de condición. En la mayoría de los ciclos el cuerpo se repite siempre y cuando la condición se cumpla, sin embargo, también existe una estructura repetitiva que se repite en tanto que no se cumple la condición. En esta sección sólo nos enfocaremos en las primeras que son las que están definidas en el lenguaje C y en la mayoría de los lenguajes estructurados y orientados a objetos actuales. Cabe mencionar que a cada una de las veces que se repite el ciclo se le conoce como *iteración***.**

Cuando se utilizan ciclos dentro de un programa, te puedes enfrentar a dos posibles situaciones:

- Que conozcas desde el diseño cuántas veces deben repetirse las instrucciones (repetición definida),

- Que el número de veces que se deban repetir las instrucciones dependa de un valor que se conoce hasta el momento de la ejecución del ciclo (repetición indefinida).

En el primer caso se necesitará una variable que funja como un *contador*, en la cual se registre el número de iteraciones que se vayan ejecutando. En cambio, en las repeticiones indefinidas generalmente se controlan mediante *interruptores* o *banderas*, o bien, con *valores centinela.*

Con los anterior puedes darte cuenta de que para las estructuras de control repetitivas es muy importante el uso de variables auxiliares y que por la frecuencia con la que se utilizan dentro de un algoritmo y por la función que realizan dentro del mismo toman un nombre especial: *contadores*, *acumuladores* e *interruptores*.

Un *contador* es una variable comúnmente de tipo entero destinada a almacenar un valor que se irá incrementando o decrementando en una cantidad constante. Se suelen utilizar mucho en procesos repetitivos definidos, para contabilizar el número de veces que se repite un conjunto de acciones o eventos, es decir en los cuerpos de las instrucciones repetitivas.

Sobre una variable contadora se realizan dos operaciones básicas: inicialización e incremento o decremento, según sea el caso. Todo contador se debe inicializar con un valor inicial (0, 1...)
contador = Valor_Inicial

Cada vez que aparezca el evento a contar se ha de incrementar o decrementar en una cantidad fija (I, D respectivamente) el valor del contador.
contador = contador+ I;

contador = contador- D;

Los contadores más utilizados tienen incrementos o decrementos de uno en uno, es por ello que la simplificación de dichas expresiones es:

Expresión	Expresión Simplificada en lenguaje C
contador=contador +1;	contador++
contador=contador -1;	contador--

Es importante señalar que en lenguaje C hay tres diferentes estructuras repetitivas: `while` (Mientras-hacer), `for` (Desde-mientras) y `do-while` (Hacer-mientras), con todas ellas es posible modelar ciclos definidos o indefinidos, pues las tres son equivalentes, es decir, cualquiera de ellas se puede expresar en términos de las otras.

3.2.1.Estructura Mientras (*while*)

La estructura repetitiva *Mientras,* codificada en lenguaje C con la palabra reservada `while`, controla las repeticiones a partir de una condición que se evalúa al inicio del ciclo, de esta manera en cada iteración primero se evaluará la condición y mientras resulte verdadera se repetirá el ciclo. En la siguiente tabla se muestran las representaciones del ciclo Mientras (while).

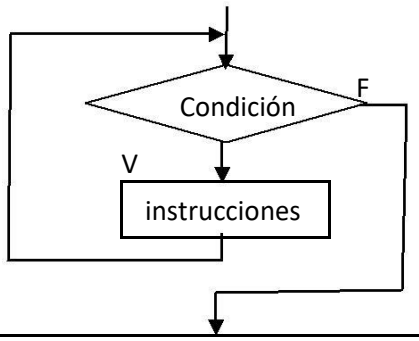
Pseudocódigo	Diagrama de Flujo	Lenguaje C
Mientras <condición>**hacer** <instrucciones> **Fin mientras**		`while` *(<condición>)* *<instrucciones>;*

Tabla 3.5: Representaciones de la estructura repetitiva Mientras (while)

La manera en la que se ejecuta una instrucción Mientras (`while`) es la siguiente: las `<instrucciones>` del cuerpo del ciclo se ejecutan mientras la `<condición>` es verdadera, cuando esto no se cumple se termina el ciclo; de esta forma, si la primera vez que se evalúa la condición esta es falsa, el cuerpo del ciclo no se ejecuta ni una sola vez.

Para ejemplificar cómo se construye un ciclo indefinido utilizando un *valor centinela*, se propone el siguiente problema.

Problema 3.4: Se requiere un programa que calcule el promedio de temperaturas que registra una ciudad, las temperaturas se introducirán en grados Fahrenheit °F y no se conoce de antemano el número de temperaturas que el usuario introducirá.

Para resolver el problema planteado se podría pedir el número de temperaturas que se desean registrar para calcular el promedio, pero esto equivale a una estructura de repetición definida, si decidiéramos dejar abierto este dato hasta el momento de la ejecución del programa, tendríamos que construir una condición que haga que el ciclo se repita mientras que el usuario desea ingresar temperaturas. Pero ¿cómo se puede resolver esto? En casos como este se propone utilizar un valor centinela que indique el fin de la captura de datos. Claramente el valor centinela debe ser seleccionado de tal forma que no se confunda con algún valor de entrada aceptable, por ejemplo podríamos considerar que dado que existe un límite mínimo de temperaturas en grados Fahrenheit, a saber -460°F, el valor centinela sería cualquier número inferior a éste, es claro que no existe una

temperatura más baja, sin embargo el límite máximo es difícil de definir ya que en forma experimental se obtienen en los laboratorios temperaturas de miles de grados, mientras que en una explosión atómica se alcanzan temperaturas de millones de grados. Se supone que la temperatura en el Sol alcanza los mil millones de grados (Pérez, 1992, pág. 325).

Para calcular el promedio, debemos realizar la suma de todas las temperaturas y dividirlas entre el número total de temperaturas ($tempF1$+$tempF2$+ …,+ $tempFn$) que se hayan leído, digamos n. Lo anterior se expresa con la siguiente fórmula.

$$promT \leftarrow \frac{\sum tempF_i}{n}$$

Así que en este caso se usará un ciclo que vaya leyendo una a una las temperaturas (almacenándolas en la variable) y acumulando la suma en la variable, estas acciones se repetirán hasta que el usuario introduzca un número menor a -460. De esta manera, la condición de término es: ≥ −460; por lo que antes de iniciar el ciclo se debe pedir la primera temperatura, para que se compare con la condición y si es mayor a -460 se acumule en la suma. Además, se utilizará un contador para registrar el número de temperaturas que se lean. Finalmente, cuando se termina el ciclo se divide el resultado de la suma de las temperaturas entre el valor del contador. Lo anterior se expresa en el siguiente pseudocódigo.

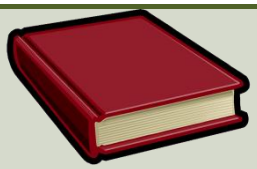
```
Inicio

        c←0, sumaF←0
        Imprimir "Ingrese la primer temperatura registrada en grados Fahrenheit:" Leer tempF
        Mientras (tempF≥-460) c←c+1 sumaF=sumaF+tempF
        Imprimir "Ingrese la siguiente temperatura en grados Fahrenheit (un número mayor a -
        460) para calcular el promedio"
        Leer tempF Fin Mientras
        promF←sumaF/c
        Imprimir “El promedio de las temperaturas es” promF

Fin
```

Algoritmo 3.4.a: Promedio temperaturas - pseudocódigo

El siguiente paso es la codificación, para la cual se determinó utilizar una variable para representar el valor centinela que controla el ciclo.

```c
/* Programa: promTemp.c
Descripción: Calcula el promedio de las temperaturas que el usuario ingresa.
*/
#include<stdio.h>
#include<stdlib.h>
#define centinela -460
main(){ /* Función principal */
    float tempF,promF, sumaF=0; /*Declaración de acumuladores y contadores*/
    int c=0;
    /* Lectura de la primera temperatura */
    printf ("Programa que calcula el promedio de temperaturas en grados Fahrenheit\n\n\n");
    printf ("\n Ingrese la primer temperatura registrada:");
    scanf ("%f",&tempF);
    /* Codificación del ciclo */
    while (tempF>= centinela){
          /* Se registra la temperatura que se leyó */
          c = c + 1;
          /* Se acumulala temperatura en la suma */
          sumaF=sumaF+tempF;
          /* Se lee la siguiente temperatura */
          printf ("\n\nIngrese la siguiente temperatura
          (si desea terminar ingrese un número menor a %d): ", centinela);
          scanf ("%f",&tempF);
    }
    /* Promedio de Temperaturas Fahrenheit */
    promF=sumaF/c;
    printf ("\nPromedio de temperaturas Celsius=%.2f\n", promF);
    system ("pause");

}
```

Programa 3.4: promTemp.c



Video 3.4: Consulta el video de desarrollo de la solución anterior, también podrás encontrar el programa fuente c en el material adicional de la unidad.

Por último, en la siguiente figura se muestra la ejecución del programa.

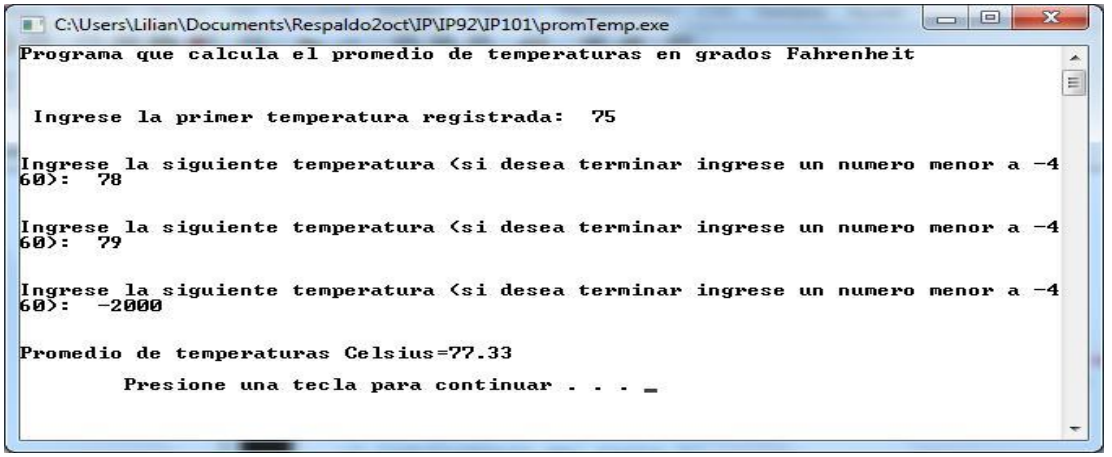


Figura 3.4: Ejecución del programa promTemp.c

3.2.2.Estructura Desde-mientras (*for*)

El ciclo *Desde-mientras*, en inglés y lenguaje C `for`, evaluará una condición y mientras ésta sea verdadera se ejecutará el conjunto de instrucciones definidas en el cuerpo de la estructura, generalmente las repeticiones se controlan por un contador, ya que como parte de su sintaxis tiene la opción de inicializar una variable (el contador) e incrementarlo o decrementarlo. Este tipo de estructura es conveniente utilizarla cuando se conoce de antemano el número de veces que se debe repetir el ciclo (*ciclos definidos*). Sus representaciones se muestran en la siguiente tabla.

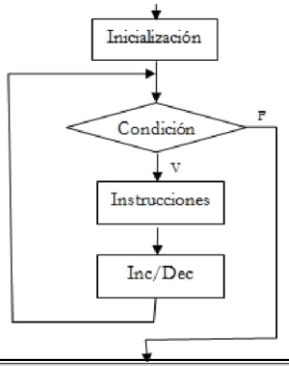
Pseudocódigo	Diagrama de Flujo
Desde <inicialización>**Mientras**<condición>, <incr/decr> <Instrucciones> **Fin desde**	
Lenguaje C	
`for (<inicialización>;<condición>; <inc/dec>)` `<instrucciones>`	

Tabla 3.6: Representaciones de la estructura repetitiva Desde-mientras (for)

En este caso, primero se realiza la <inicialización> que corresponde a la asignación de un valor inicial de una variable (el contador), posteriormente se evalúa la <condición> si es verdadera se ejecutan las <instrucciones> del cuerpo del ciclo y, posteriormente, se incrementa o decrementa el contador, según sea el caso, para después volver a repetir el ciclo, excepto por la <inicialización> que sólo se ejecuta una vez.

Para ejemplificar las representaciones, codificación y funcionamiento de esta estructura se presenta el siguiente problema desarrollado.

Problema 3.5: Se requiere un programa que calcule el total de la nómina de los trabajadores de una empresa.

El problema es similar al que se presentó en la sección anterior, se debe leer el pago de cada trabajador y realizar la suma de cada uno de éstos, para lo cual se puede utilizar un acumulador. La diferencia es que en este caso no se utilizará un valor centinela para terminar la lectura de los pagos, pues se preguntará al usuario al inicio del programa cuántos trabajadores hay, así el número de iteraciones quedará determinado antes de iniciar el ciclo. De lo anterior tenemos que si el número de empleados es *n* entonces el ciclo debe repetirse *n*-veces, para lo cual se utilizará un contador *c* que debe tomar los valores *1,2,…, n,* así que el ciclo debe repetirse siempre que $c \leq n$. En cuanto a la suma de los pagos, se utilizará un acumulador, al cual llamaremos *nom*, que se inicializará en cero dado que se trata de una suma. Observa la solución del problema.

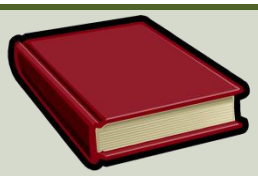
```
Inicio

        Imprimir "Ingrese el total de empleados: " Leer n

        Desde c=1 , nom=0,   Mientras (c<=n), c=c+1

        Imprimir “Ingresa el salario del trabajador”, c

            Leer sal nom=nom+sal

Fin desde

        Imprimir “La nómina a pagar es en total $”, nom

Fin
```

Algoritmo 3.5.a: Nómina - pseudocódigo

Por lo tanto, la salida del algoritmo es: "*La nómina a pagar es $45".* La codificación sería la siguiente.

```
/* Programa: nomina.c
Descripción: calcula la nómina a pagar de n trabajadores
*/
/*directivas de preprocesador*/
#include<stdio.h>
#include<stdlib.h>
main (){ /*Función Principal*/
int n,c;/* Declaración de las variables */
float nom,sal;
/* Lectura del número de empleados */
printf ("Calculo de la Nomina\n\n ");
printf ("Ingrese el total de empleados: ");
scanf ("%d",&n);
/*Ciclo definido de 1 hasta el número de empleados ingresados*/
for (nom=0,c=1;c<=n;c=c+1){
    printf ("\nIngresa el salario del trabajador %d: ", c);
    scanf ("%f",&sal);
    /*Acumulador de salários*/
    nom=nom+sal;
}
printf("\n La nomina a pagar es $&.2f", nom);
}
```

Programa 3.5: nomina.c



Video 3.5: Consulta el video de desarrollo de la solución anterior, también podrás encontrar el programa fuente c en el material adicional de la unidad.

En la siguiente figura se muestra la ejecución del programa.

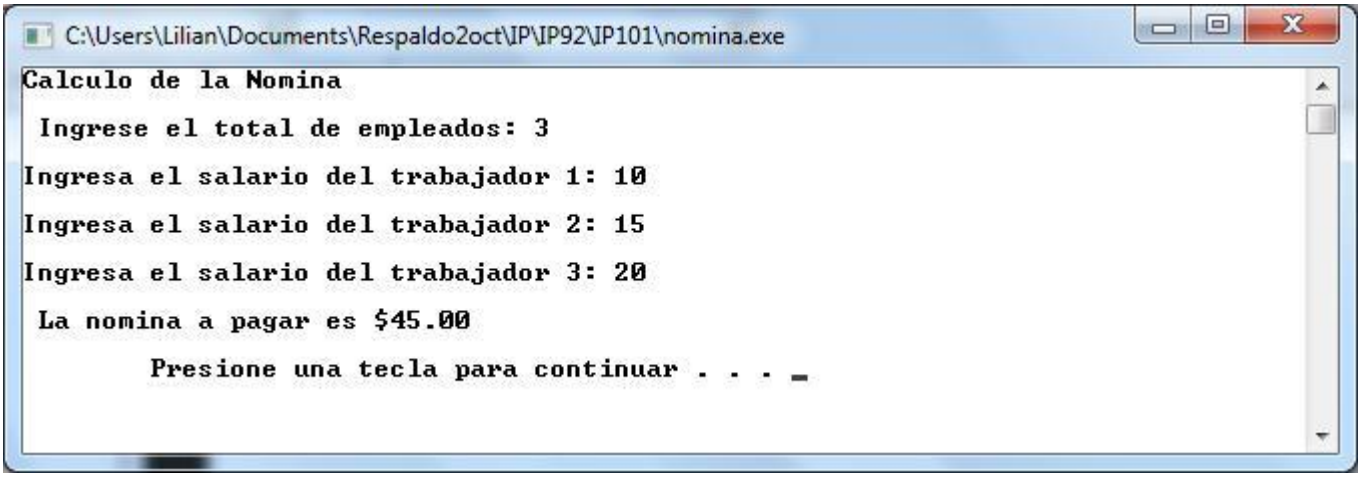


Figura 3.5: Ejecución del programa nomina.c

3.2.3.Estructura Hacer-mientras (*do-while*)

A diferencia de las estructuras repetitivas anteriores, en las cuales las condiciones se evalúan al principio del ciclo, por lo que las instrucciones que se repiten se ejecuten de 0 hasta N veces, en la *estructura Hacer-mientras* (`do-while`) la evaluación se lleva acabo al final, esto implica que el conjunto de instrucciones que se repite se ejecuta al menos una vez.

Pseudocódigo	Diagrama de Flujo	Lenguaje C

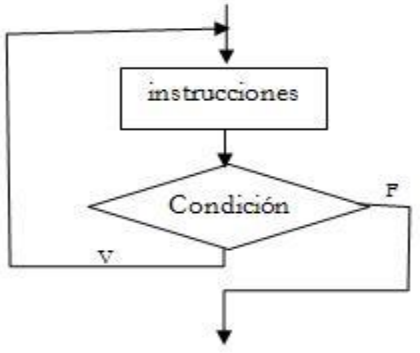
Hacer
<instrucciones>
 	
Mientras <condición>**Fin**
		`do`
		`<instrucciones>;`
		`while(<condición>);`

Tabla 3.7: Representaciones de la estructura repetitiva Hacer-mientras (do-while)

Observa que, en el código en C, la única estructura de control, de todas las que hemos visto, que tiene punto y coma después de la expresión o condición es el `do-while`.

Por el funcionamiento de la estructura, el caso típico del uso del `do-while` son los menús. Para ejemplificar lo anterior se propone el siguiente problema.

Problema 3.6: Se requiere un programa que imprima un menú con las siguientes opciones, el cual se repita en tanto no se elige la opción d (Salir).

a. Calcular la fuerza
b. Calcular la aceleración
c. Calcular la masa
d. Salir

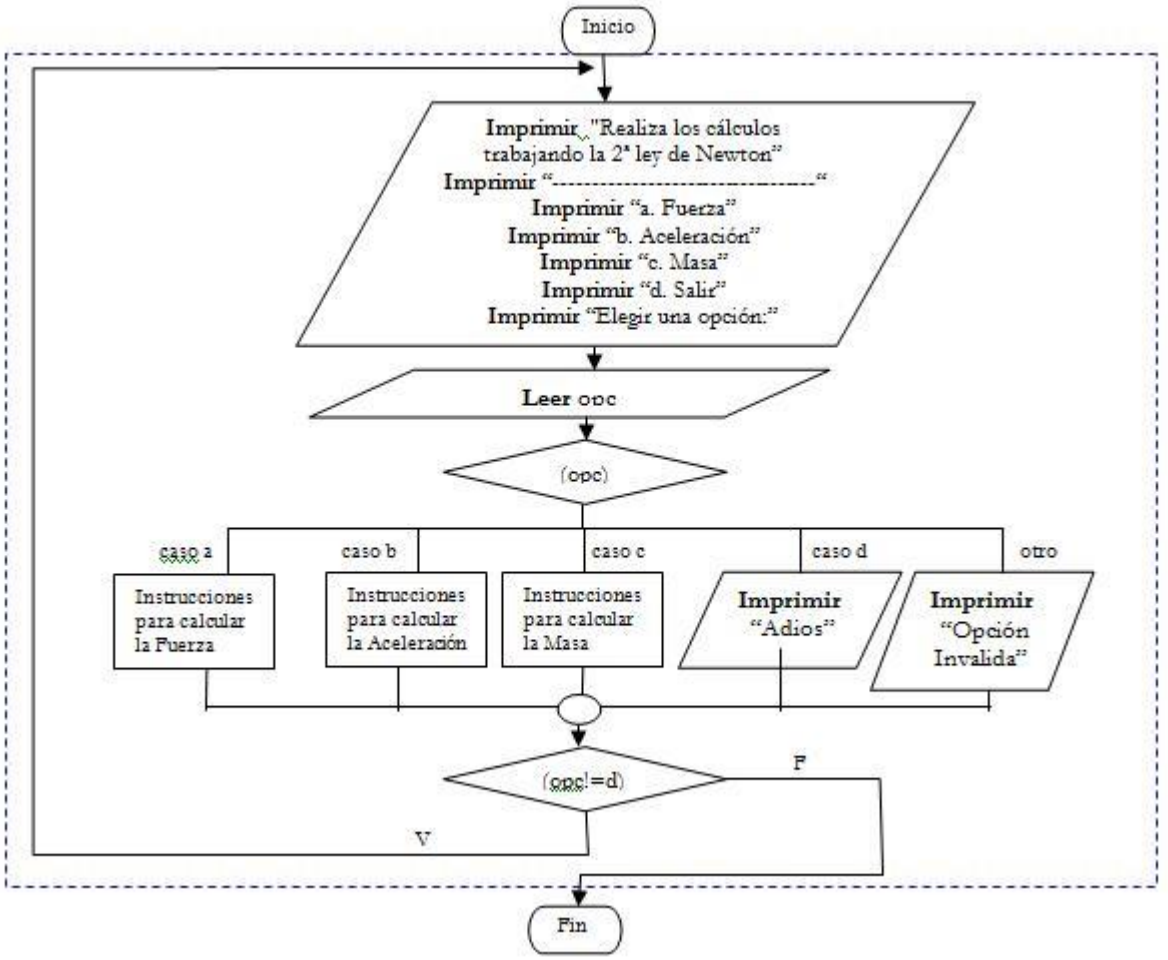
Además, dependiendo de la opción que elija el usuario se deberá realizar la tarea indicada utilizando la segunda ley de Newton que dicta: “La aceleración que un cuerpo adquiere es directamente proporcional a la resultante de las fuerzas que actúan en él, y tiene la misma dirección en el sentido que en dicha resultante”

En este caso, para resolver la parte del menú se utilizará un `switch-case`, en el que cada opción del menú corresponda a un caso, así las instrucciones que lo forman deben ser: la lectura de los datos correspondientes y la operación apropiada (que se define despejando la variable en cuestión de la fórmula dada). Para que el menú se repita se plantea un ciclo `while` que se ejecute mientras la opción sea distinta de 4 (Salir). De esta forma el algoritmo se presenta a continuación en sus dos representaciones.

```
Inicio
        Hacer
        Imprimir "Realiza Cálculos trabajando la 2a. Ley de Newton"
        Imprimir " ------------------------------------------------"
        Imprimir " a. Fuerza." Imprimir " b. Aceleración."
        Imprimir " c. Masa." Imprimir " d. Salir."
        Imprimir " Elegir una Opción: "
        Leeropc Selecciona (opc)
                Caso 1:
                        Imprimir "Ingresa La masa:” Leer m
                        Imprimir “Ingresa la aceleración:”
                        Leer a
                        f = m*a
                        Imprimir “Fuerza = ”, f

                Caso 2:
                        Imprimir "Ingresa la fuerza:” Leer f
                        Imprimir “Ingresa la masa:”
                        Leer m a = f/m
                        Imprimir “Aceleración = ”, a

                Caso 3:
                        Imprimir "Ingresa la fuerza:” Leer f
                        Imprimir “Ingresa la aceleración:”
                        Leer a
                        m = f/a
                        Imprimir “Masa = ”, m
                Caso 4:
                        Imprimir "Adios"
                Otro:
                        Imprimir " Opción inválida"
        Fin_Selecciona
```

Algoritmo 3.6.a: Segunda ley de Newton - pseudocódigo

A continuación, se presenta el diagrama de flujo, salvo que únicamente se ha dejado indicado en donde van las instrucciones de los tres primeros casos, ya definidas en el pseudocódigo.



Algoritmo 3.6.b: Segunda ley de Newton – diagrama de flujo

En el diagrama de flujo se puede observar claramente que el ciclo se ejecutará mientras el usuario no elija la opción d, que corresponde a salir. Por lo que no se deja como ejercicio al lector la validación del algoritmo.

La codificación del algoritmo se muestra enseguida.

```
/* Programa: newton.c
Descripción: Muestra un menú para calcular la aceleración, fuerza o
masa, conforme a la segunda ley de Newton */
/*directivas de preprocUnADMor*/
#include <stdio.h>
```

```
#include <stdlib.h>
#include <conio.h>
/*Función Principal*/
main ()
{
/*Declaración de variables*/
  char opc;
  float f,m,a;
/*Ciclo para repetir el menú mientras que la opción no sea salir*/
do
{
/*Impresión del menú*/
system ("cls");
/*Instrucción para borrar la pantalla*/
printf ("\n Realiza Calculos trabajando la 2a. Ley de Newton");
printf ("\n ------------------------------------------------");
printf ("\n a. Fuerza. \n b. Aceleracion \n c. Masa \n d. Salir");
printf ("\n Elige una opcion: ");
/*Instrucción que lee una variable de tipo carácter*/
opc=getche();
/*Estructura de Sección Múltiple*/
witch (opc)
  {
case 'a':
      printf ("\n\nIngresa la masa: ");
      scanf("%f",&m);
      printf ("\nIngresa la aceleracion: ");
      scanf("%f",&a);
      f=m*a;
      printf("\nLa fuerza es %.2f\n\n\t",f);
      system ("pause");
break;

case 'b':
       printf ("\n\nIngresa la fuerza: ");
       scanf("%f",&f);
       printf ("\nIngresa la masa: ");
       scanf("%f",&m);
       a=f/m;
       printf("\nLaaceleracion es %.2f\n\n\t",f);
       system ("pause");
break;


      case 'c':
             printf ("\n\nIngresa la fuerza: ");
             scanf("%f",&f);
             printf ("\nIngresa la aceleración: ");
```

```
                scanf("%f",&m);
                m=f/a;
                printf("\nLa masa es %.2f\n\n\t",f);
                system ("pause");
        break;

    case 'd':
                printf ("\n\nAdios\n");
                system ("pause");
    break;

    default:
                printf ("\n\n Opcion Invalida");
}
/*Fin dela Selección Múltiple*/
}while (opc!='d'); }/*Fin*/
```

Programa 3.6: newton.c

En la siguiente figura se muestra la ejecución de una iteración del ciclo, en la cual la opción elegida es la primera.

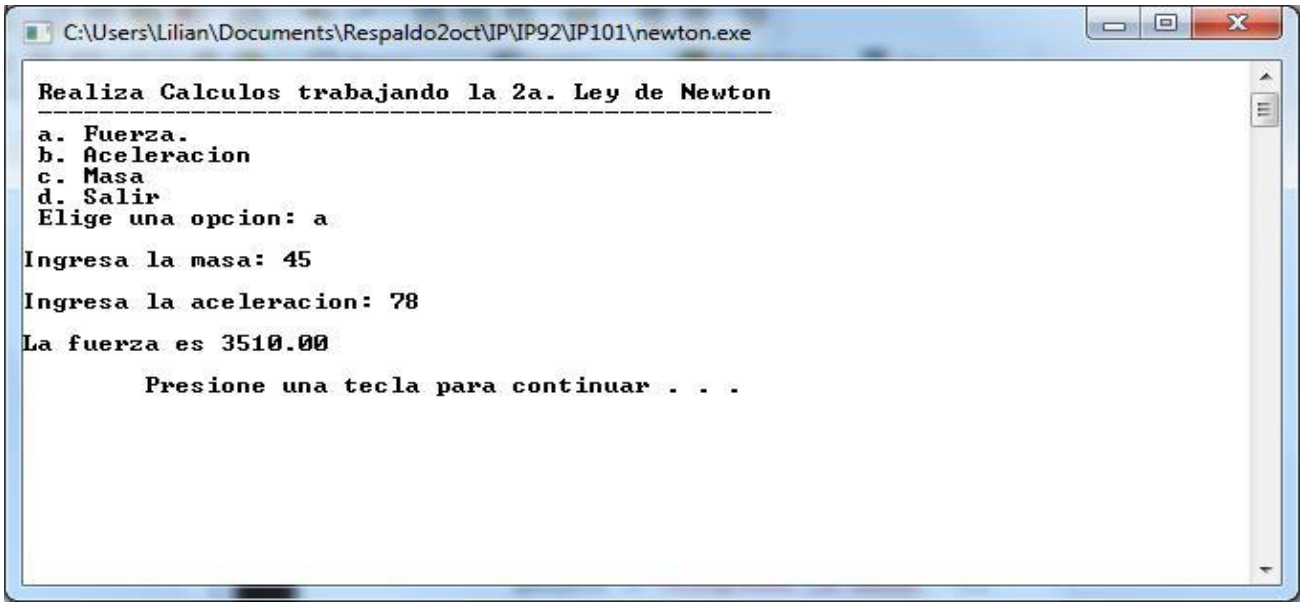


Figura 3.6: Ejecución del programa newton.c

Observa que dentro del cuerpo del ciclo se definió una estructura selectiva, es decir, que las instrucciones del cuerpo de cualquier estructura compuesta, sea repetitiva o selectiva, puede contener a otras. Uno de los casos más utilizados es el anidamiento de los *if*´s, de lo cual hablaremos en la

siguiente sección. Pero antes de dar por terminada esta sección se propone la siguiente actividad.

3.3. Estructuras Anidadas

Las estructuras de control selectivas y repetitivas se consideran compuestas ya que se forman a partir de otras instrucciones que son las que se ejecutaran de acuerdo con una condición dada. Es importante remarcar que las instrucciones que forman el cuerpo de una estructura pueden ser también estructuras compuestas, como se demostró en la solución del último problema visto (ver algoritmo 3.6 y programa 3.6), en el cual un `switch` está dentro de un `while`. Así que es posible anidar cualquier tipo de estructura, sin embargo, lo más común es anidar instrucciones `if`, pues se utilizan cuando se tienen varios casos, por ejemplo, si revisamos nuevamente el problema 3.1, donde se quiere determinar el tipo de ángulo, es mejor solución utilizar `if`-*anidados* para resolverlo porque así no se evalúan condiciones que, una vez que se ha definido el tipo de ángulo, son innecesarias.

Para ilustrar lo anterior, a continuación se muestra el pseudocódigo y su codificación para la solución del mismo problema.

```
Inicio
   Imprimir  "Ingrese la medida del angulo (grados): "
   Leer  angulo

   Si angulo≤0 OR angulo≤360 entonces
                Imprimir “No tiene clasificación"
 Sino Si  angulo<90  entonces
                Imprimir "El ángulo es agudo"
          Sino     Si angulo=90 entonces
                 Imprimir "El angulo es recto"
                   Sino Si  angulo<180 entonces
                Imprimir “El angulo es obtuso"
                      Sino    Si angulo =180  entonces
                 Imprimir "El angulo es llano"
                             Sino
                Imprimir “El angulo es concavo"
Fin_Si-Sino
Fin_Si-Sino
Fin_Si-Sino
Fin_Si-Sino
Fin_Si-Sino
Fin
```

Algoritmo 3.7: Tipo de ángulo (versión 2)- pseudocódigo (Fuente: elaboración propia)

Si realizas la prueba de escritorio con el ángulo igual a 90 grados, podrás darte cuenta de que a diferencia de la primera versión del algoritmo donde se evalúan todas las condiciones, aquí sólo se evalúan las tres primeras, en los dos primeros *Si* es falsa y por lo tanto se ejecutan las instrucciones del *Sino* correspondiente, pero en el tercer *Si* anidado la condición es verdadera y se imprime el tipo de ángulo, posteriormente se acaba el anidamiento.

El programa en C es el siguiente:

```
main ()
{
/*Declaración de variables */ intangulo;

 /*Mensaje de bienvenida*/
printf ("\nEste programa determina de que tipo es el angulo
dado.");

   /*Instrucciones */
printf ("\n\nIngrese la medida del angulo (grados): ");
scanf ("%d",&angulo);
if (angulo<=0 || angulo>=360)
printf ("\n No tiene clasificación");

else if (angulo<90)
printf ("\n El angulo es agudo");

else if (angulo==90)
printf ("\n El angulo es recto");

else if (angulo<180)
printf ("\nElanguloesobtuso");

else if (angulo ==180)
printf ("\n El angulo es llano");

else
printf ("\nElangulo es concavo");


printf
     ("\n\n
     \t");
     system
     ("paus
     e");
}
```

Programa 3.7: tipoAngulo2.c

La ejecución con el ángulo igual a 90 grados se muestra en la siguiente figura.

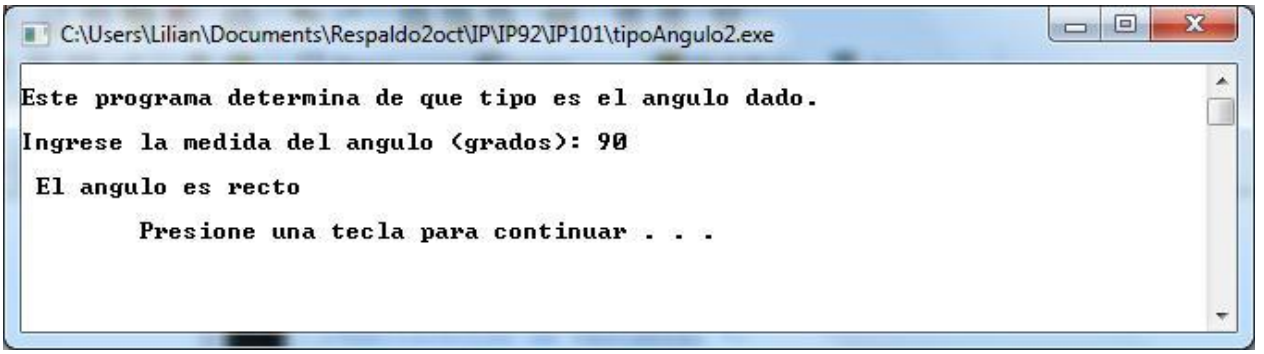


Figura 3.7: Ejecución del programa tipoAngulo2.c

Con este ejemplo se da por terminada esta unidad, ahora ya conoces todas las estructuras y has visto cómo funcionan y qué tipo de situaciones se puede modelar con ellas. Aunque cabe destacar que para solucionar cualquier problema basta con que sepas utilizar el ciclo `while` y la estructura selectiva `if-else`, pues ya se mencionó que todos los ciclos son equivalentes y con la estructura `if-else`, puedes modelar un `switch-case` anidando `if´s`.

3.4. Arreglos

El uso de arreglos facilita y hace más eficiente la declaración y manipulación de una *colección de datos de un mismo tipo* que están relacionados entre sí, como es el caso de las calificaciones en el Problema1, ya que todas las calificaciones se pueden considerar como valores enteros.

Problema 3.7: Se requiere un programa para llevar el registro de calificaciones de un grupo de diez estudiantes y generar reportes que incluyan datos como el promedio del grupo, la calificación máxima, el número de estudiantes que tienen una calificación superior al promedio del grupo, entre otros.

En este caso, a diferencia de los ejemplos anteriores, es claro que las calificaciones de cada estudiante se puede tratar como un dato simple e independiente de los otros, sin embargo las operaciones que se desean realizar serán las mismas para todo el conjunto de calificaciones, de tal forma que habría que escribir una serie de instrucciones secuenciales para ingresar cada dato y procesarlo. Por ejemplo, para ingresar los datos se requiere leer una por una cada calificación, para

obtener el promedio se tendría que hacer la suma de todas y después dividirlas entre 10, hasta aquí no se ha complicado mucho, pero imagina todas las comparaciones que debes hacer para identificar cuál es la calificación mayor. Es claro que este método resulta de lo más ineficiente, y por supuesto si consideramos la posibilidad de modificar el programa para que sea capaz de procesar 60 o más calificaciones, el programa además de extenderse, implica reestructurarlo en su totalidad y que éste sea más complejo que la versión anterior. En cambio si consideramos a todos las calificaciones como un dato estructurado podemos hacer uso de una estructura de dato que nos facilite su manipulación.

Existen diferentes tipos de estructuras de datos, cada una caracterizada por la forma de acceso a sus elementos, y el tipo que estos pueden tener, así tenemos arreglos, listas, colas, tablas, pilas, entre otros. No obstante, para este tema nos centraremos sólo en las estructuras de datos que implementa el lenguaje C de forma directa: los arreglos.

La solución del problema representada en pseudocódigo se muestra en el siguiente algoritmo.

```
inicio
suma ← 0
Desde i ← 0 mientras i<10,  i ← i+1
            Imprimir “Ingresa la calificación” i
            Leer calif[i]
    suma← suma+calif[i]
    Fin Desde
prom ← prom/10
Imprimir  “Las  calificaciones  ingresadas  fueron:”
Desde i ← 0 mientras i<10, i ← i+1
Imprimir “Calificación” i “:” calif[i]
Fin Desde
      Imprimir “Calificación promedio = ” prom
Fin
```

Algoritmo 3.8. Promedio de calificaciones

La codificación del algoritmo anterior es la siguiente:

```
/*Directivas de preprocesador*/
#include <stdio.h>
#include <stdlib.h>
/* Definimos como constante simbólica el tamaño del arreglo*/
#define TAM 10
/* Definición de función principal */
main( )
{
   /*Declaración del arreglo calificaciones*/
int calif[TAM];
   double prom
=           0;
int i;
   printf("*** ");
   printf("* El siguiente programa calcula el promedio de
   *\n");
   printf("* un grupo de diez estudiantes");
   printf("***");
  /*Lectura y suma de las calificaciones*/
for(i=0; i < TAM; i++)
  {
      printf("Proporciona la calificación %d: ",i+1);
      scanf("%d", &calif[i]);
      prom = prom + calif[i];
  }
/*Cálculo e impresión del promedio*/
  prom = prom/TAM;
  /*Impresión de las calificaciones*/
  printf("\nLas calificaciones ingresadas fueron:
\n"); for(i=0; i < TAM; i++)
       printf("\nCalificacion %d: %d",i+1, calif[i]);
  printf("\n\n\tPromedio = %.2f\n\n", prom);
  system("pause");
}
```

Programa 3.8: promCalificaciones.c

En la siguiente figura se muestra una ejecución del programa.

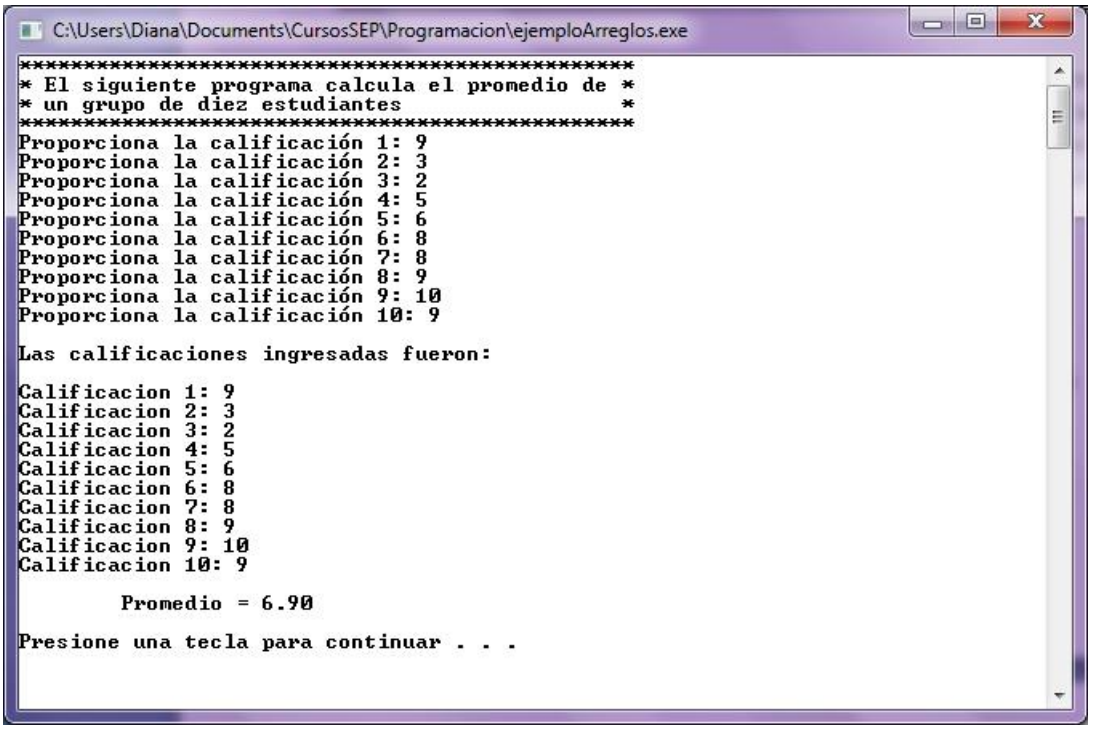


Figura 3.8: Ejecución del programa promCalificaciones.c

Se sugiere consultes el video que contiene las bases de los arreglos como los que podrías aplicar a lo largo de la unidad: https://www.youtube.com/watch?v=gCBpGyKyaGU.

3.4.1. Definición y tipos de arreglos

"Un *arreglo* se define como una colección finita, homogénea y ordenada de elementos. Finita ya que para todo arreglo debe especificarse el número máximo de elementos que podrá contener; la homogeneidad se refiere a que todos los elementos deben ser del mismo tipo, y ordenada porque es posible determinar cuál es el primer elemento, cual el segundo, y así hasta el enésimo elemento" (Cairo Osvaldo, Guardati Buemo Silvia, 1993).

La posición que ocupa un elemento dentro de un arreglo se le denomina formalmente *índice* y siempre es un número entero. El *tamaño* o *longitud* de un arreglo se define como el número de elementos que

lo constituyen. La *dimensión* de un arreglo está relacionada con el número de índices necesarios para especificar un elemento en particular.

Podemos clasificar a los arreglos de acuerdo con su dimensión como *unidimensionales* o *multidimensionales*.

Los arreglos *unidimensionales*(también llamados *lineales*) reciben su nombre debido a que cualquier elemento es referenciado por un único índice, por ejemplo retomando el caso de las calificaciones del problema 3.7*,* éstas pueden ser almacenadas en un arreglo unidimensional como el que se muestra en la Figura 3.9, en donde el nombre del arreglo es lista y los nombres de las variables donde se almacenan las calificaciones son: lista[0], lista[1], lista[2], lista[3], lista[4] …, lista[9]. En este caso el nombre en común es lista y lo único que cambia para cada elemento es el número que le corresponde a cada variable según la posición que ocupa en la lista. Observa que un solo índice es suficiente para diferenciar a un elemento de otro.

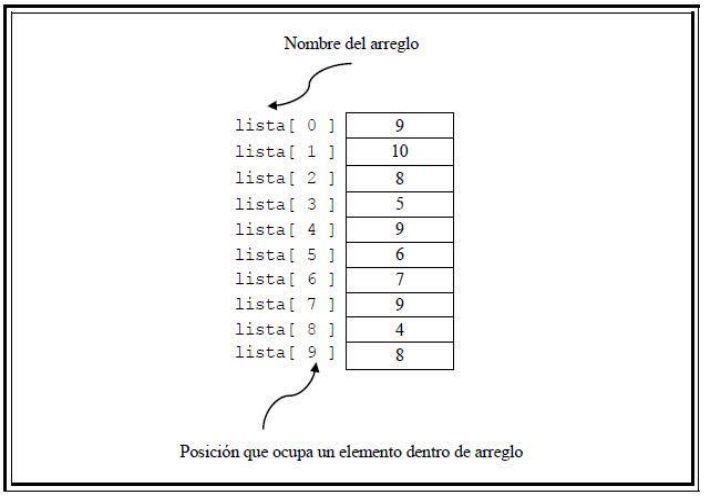


Figura 3.9. Representación gráfica de un arreglo unidimensional

Por otro lado los arreglos *multidimensionales* son aquellos para los cuales un solo índice no es suficiente para poder referenciar a un elemento individual, los arreglos *bidimensionales* son el caso más comúnmente utilizado de arreglos multidimensionales y por tanto los únicos que presentaremos.

“Un arreglo *bidimensional* es un conjunto de datos homogéneos, finito y ordenado, donde se hace referencia a cada elemento por medio de dos índices. El primero de los cuales generalmente se utiliza para indicar renglón y el segundo para indicar columna” (Cairo Osvaldo, Guardati Buemo Silvia, 1993)

Un arreglo bidimensional también puede verse como una tabla de valores, o bien como un arreglo de arreglos, de ahí la necesidad de dos índices, en la Figura 3.10 se muestra un ejemplo gráfico de un

arreglo bidimensional, en la cual del lado derecho podemos ver al arreglo como una tabla y del lado izquierdo representado como un arreglo de arreglos, Observa que cada renglón de la tabla es cada uno de los elementos del arreglo de arreglos. Es claro que con un solo índice no podríamos identificar a un único elemento ya que solo podríamos ubicar toda una columna o todo un renglón, en cambio la combinación de renglón-columna si nos identifica a un elemento en particular.

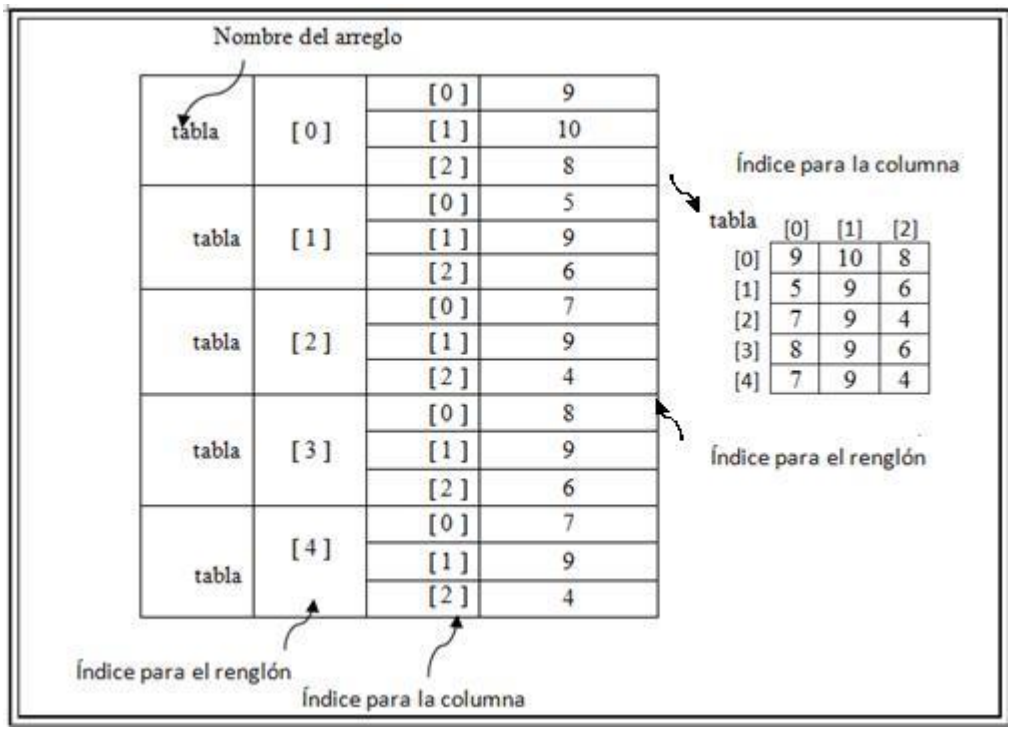


Figura 3.10 Representación gráfica de un arreglo bidimensional

3.4.2.Declaración e inicialización

En lenguaje C los índices de los arreglos siempre empiezan en cero, es decir, al primer elemento del arreglo le corresponde la posición 0, al segundo la posición 1, al tercero la posición 2 y así sucesivamente hasta llegar al elemento `TAM-1`, donde `TAM` corresponde al tamaño del arreglo.

La declaración de un arreglo consiste en reservar espacio de memoria suficiente para el conjunto de datos homogéneos. La declaración de *una variable de tipo arreglo* sigue las mismas reglas que para las variables simples; con la diferencia de que ahora será necesario especificar el tamaño del arreglo, esto se hace escribiendo el tamaño del arreglo encerrado entre corchetes `[ TAM ]`, después del identificador.

La sintaxis para la declaración de un arreglo unidimensional en lenguaje C es la siguiente:

```
<tipo><nombre>[<tamaño>];
```

Y para un arreglo bidimensional es:

```
<tipo><nombre>[<tamaño1>] [<tamaño2>];
```

El tipo de dato para los arreglos puede ser cualquier tipo básico, es decir entero, flotante o carácter (en C `int`, `float`, `double` o `char`). De todos ellos los arreglos de tipo carácter (`char`) tienen un tratamiento especial, ya que un arreglo de este tipo se considerara una *cadena*. Debido a la importancia que tienen las cadenas en la programación más adelante los trataremos de manera particular.

Al igual que las variables simples, un arreglo puede inicializarse al momento de ser declarado, para ello se utiliza el operador asignación "=", pero como un arreglo almacena a un conjunto de datos, es necesario inicializarlo con un conjunto de valores, los cuales se indican mediante llaves, separando por comas cada elemento del conjunto de valores iniciales, la sintaxis se muestra a continuación:

```
<tipo><nombre>[<tamaño>]={<valor0>,<valor1>,…,<valorTAM-1>};
```

La asignación de cada valor inicial se hace consecutivamente desde el elemento 0, por tanto no es posible asignar valores a elementos salteados.

Veamos como ejemplo la declaración del arreglo unidimensional lista (Figura 3.9) planteado para las calificaciones del problema 3.7. Inicializando sus elementos en la declaración queda como:

```
int lista[10] = {9,10,8,5,9,6,7,9,4,8};
```

En el caso de los arreglos bidimensionales la sintaxis es la siguiente:

```
<tipo><nombre>[<tamaño1>][<tamaño2>]={ {<valor00>,<valor01>,…,<valor0(TAM21)>},
{<valro10>,<valor11>,…,<valor1(TAM21-1)>},…,
{<valor(TAM1-1)0>,<valor (TAM2-1)1>,…,<elem(TAM1-1)(TAM2-1)>}
};
```

Veamos ahora como queda la declaración del arreglo bidimensional tabla mostrado en la Figura 3.10, inicializando sus valores:

```
int tabla[5][3]={{9,10,8},{5,9,6},{7,9,4},{8,9,6},{7,9,4}};
```

Aunque también es posible declararlo de la siguiente forma:

```
int tabla[5][3]={9,10,8,5,9,6,7,9,4,8,9,6,7,9,4};
```

Esta es debido a que como ya se dijo antes un arreglo bidimensional se pude ver como un arreglo de arreglos.

Por otro lado, en lenguaje C siempre es necesario especificar el tamaño del arreglo al momento de declararlo, sin embargo esto se puede hacer de forma explícita o implícita.

- Explícitamente es cuando se especifica el tamaño dentro de los corchetes que siguen al identificador, como en los ejemplos anteriores.
- De forma implícita se hace cuando el arreglo es inicializado con un conjunto de valores, y se omite el tamaño dentro de los corchetes, entonces el compilador asume el tamaño del arreglo igual al tamaño del conjunto de valores iniciales, de tal forma que la declaración del arreglo lista puede quedar como:

```
int lista[] = {9,10,8,5,9,6,7,9,4,8};
```

Observa que en este caso no se escribe el tamaño dentro de los corchetes, pero como hay 10 elementos en el conjunto de valores iniciales, el compilador de C asume un tamaño 10 para el arreglo.

Para los arreglos bidimensionales, sólo es posible especificar una dimensión de forma implícita, el tamaño de renglones siempre debe hacerse de forma explícita.

La asignación de un conjunto de valores al arreglo, en una sola operación de asignación, únicamente es posible en su declaración, si se intenta realizar en otro momento el compilador marcará un error, ya que en cualquier otra parte del programa sólo se podrán asignar valores simples a cada uno de los elementos por separado.

Es importante señalar que cuando se desea inicializar el arreglo al declararlo, es posible inicializar sólo algunos de sus elementos, pero en este caso se tendría que especificar explícitamente el tamaño, además se debe recordar que la asignación de valores iniciales es consecutiva desde el elemento 0. Los elementos para los cuales no se indique un valor inicial, automáticamente se inicializan en cero. Por ejemplo la declaración

```
int lista[10] = {5};
```

Reservará espacio en memoria para los 10 elementos del arreglo de los cuales al primer elemento se le asignará un 5 y al resto se les asignará un cero.

En el caso de los arreglos bidimensionales es posible declara sólo algunos elementos por renglón, siempre y cuando los elementos sean consecutivos, como en el caso de los unidimensionales. Por ejemplo la siguiente declaración para el arreglo tabla:

```
int tabla[5][3]={{9,10},{5},{7,9,4},{8,9,}};
```

Daría como resultado la siguiente asignación de valores iniciales

	[0]	[1]	[2]
[0]	9	10	0
[1]	5	0	0
[2]	7	9	4
[3]	8	9	0
[4]	0	0	0

En el caso de que la declaración fuera:

```
int tabla[5][3]={9,10,5,7,9,4,8,9,};
```

Entonces la asignación de valores iniciales se haría de la siguiente forma

	[0]	[1]	[2]
[0]	9	10	5
[1]	7	9	4
[2]	8	9	0
[3]	0	0	0
[4]	0	0	0

3.4.3. Acceso a los elementos de un arreglo

Para referirse a un elemento del arreglo es necesario indicar el nombre del arreglo seguido del índice o índices correspondientes al elemento que deseamos acceder. Para ello se debe seguir la siguiente sintaxis.

Elementos de un arreglo unidimensional:

```
<nombre del arreglo>[<índice>];
```

Elementos de un arreglo bidimensional:

```
<nombre del arreglo>[<índice de renglón>][<índice de columna>];
```

Observa que para cada índice se utilizan corchetes separados.

Cada elemento del arreglo se puede tratar igual que a cualquier otra variable, es decir, podemos asignarle un valor, incluir en una expresión algebraica o lógica, imprimir en pantalla su valor, asignarle desde el teclado un valor, etc.

Instrucción	Descripción
`tabla[0][2] = 8;`	Asignar el valor de 8 al tercer elemento del primer renglón de arreglo tabla
`printf("%d",lista[4]);`	Imprimir en pantalla el quinto elemento del arreglo lista
`scanf("%d",&tabla[0][0]);`	Lee un entero desde teclado y asignarlo en la primera posición del arreglo tabla.
`lista[1]++;`	Incrementar en uno el valor del segundo elemento del arreglo lista

Actividades

La elaboración de las actividades estará guiada por tu docente en línea, mismo que te indicará, a través de la Planeación didáctica del docente en línea, la dinámica que tú y tus compañeros (as) llevarán a cabo, así como los envíos que tendrán que realizar.

Para el envío de tus trabajos usarás la siguiente nomenclatura: BPRG_U3_A1_XXYZ, donde BPRG corresponde a las siglas de la asignatura, U3 es la unidad de conocimiento, A1 es el número de actividad, el cual debes sustituir considerando la actividad que se realices, XX son las primeras letras de tu nombre, Y la primera letra de tu apellido paterno y Z la primera letra de tu apellido materno.

Autorreflexiones

Para la parte de **autorreflexiones** debes responder las *Preguntas de Autorreflexión* indicadas por tu docente en línea y enviar tu archivo. Cabe recordar que esta actividad tiene una ponderación del 10% de tu evaluación.

Para el envío de tu autorreflexión utiliza la siguiente nomenclatura:

BPRG_U3_ATR _XXYZ, donde BPRG corresponde a las siglas de la asignatura, U3 es la unidad de conocimiento, XX son las primeras letras de tu nombre, y la primera letra de tu apellido paterno y Z la primera letra de tu apellido materno

Cierre de la unidad

En esta Unidad hemos aprendido a utilizar las estructuras de control y los arreglos elementos importantes en la base del conocimiento de la programación es importante seguir practicando para mantener un nivel de conocimiento base, por otro lado, pueden ir viendo temas como programación orientada a objetos como complemento de la programación.

Para saber más

- Puedes encontrar más información acerca de diagramas de flujos, pseudocódigo y algunos programas para desarrollar en C en los siguientes vínculos:
http://pseint.sourceforge.net/index.php?page=documentacion.php
http://www.c.conclase.net/curso/index.php
http://zinjai.sourceforge.net/index.php?page=documentacion.php

Fuentes de consulta

Fuentes básicas

- Böhm, C., & Jacopini, G. (1966). Flow diagrams, Turing machines, and languages only with two formation rules". *Communications of the ACM, 9* (5), 366-371.
- Cairó, O. (2005). *Metodología de la programación: Algoritmos, diagramas de flujo y programas.* México, D.F.: Alfaomega.
- Guerrero, F. (s.f.). *mailxmail.com*. Recuperado el 15 de 8 de 2010, de https://web.archive.org/web/20221129015722/http://www.mailxmail.com/curso-introduccion-lenguaje-c
- Hernández, María Lourdes (2010), Diseño Estructurado de Algoritmos, Diagramas de Flujos y Pseudocódigos, Documento recopilado de la Universidad de Teuxtepe.
- Joyanes, L., & Zohanero, I. (2005). *Programación en C. Metodología, algoritmos y estructuras de datos.* España: Mc Graw Hill.
- Kernighan, B., & Ritchie, D. (1991). *El lenguaje de programción C.* México: Prentice-Hall Hispanoamericana.
- López, L. (2005). *Programación estructurada en lenguaje C.* México: Alfaomega.
- Reyes, A., & Cruz, D. (2009). *Notas de clase: Introducción a la programación.* México, D.F.: UACM.
- Villela, H. T. (20 de agosto de 2010). Manual de C. Consultado el 25 de marzo de 2020 en: http://diarium.usal.es/mlperez/files/2012/06/lenguajec-unix-gcc.pdf
- Viso, E., & Pelaez, C. (2007). *Introducción a las ciencias de la computación con Java.* México, D.F.: La prensa de ciencias, Facultad de Ciencias, UNAM.