Ingeniería en Desarrollo de Software

Primer Semestre

Programa de la asignatura:

Fundamentos de programación

Unidad 4. Estructuras de control

Clave:

TSU	Licenciatura
16141102	15141102

Universidad Abierta y a Distancia de México

Índice

Unidad 4: Estructuras de control

Presentación

En la segunda 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**, las cuales son:

- **Secuenciales** (cuando se ejecutan una tras otra)
- **Selectivas** (cuando se ejecutan dependiendo de una condición)
- **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 tres 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 éste será justamente el tema de esta unidad, donde estudiarás con más detalle los tipos y funcionamiento de las estructuras selectivas y repetitivas, presentadas en la unidad 2.

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, esto se soluciona utilizando estructuras selectivas, por lo cual 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 sintaxis en lenguaje C. De esta manera, al finalizar la unidad podrás construir programas que incluyan cualquier tipo de estructura de control.

Propósitos

- 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.

4.1. 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 sub-secciones, una para cada una, en las cuales entenderás cómo funcionan y la forman en que se codifican en lenguaje C.

4.1.1. 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.

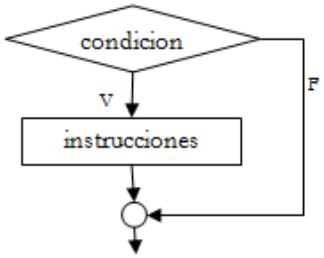
Pseudocódigo	Diagrama de Flujo	Código en C
Si <condición> entonces <instrucciones> Fin Si		**if(**<condición>**)** <instrucciones>

Tabla 4.1: 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 en cuyo caso van entre llaves {}.

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, el diagrama de flujo y el programa en lenguaje C.

Problema 4.1: 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

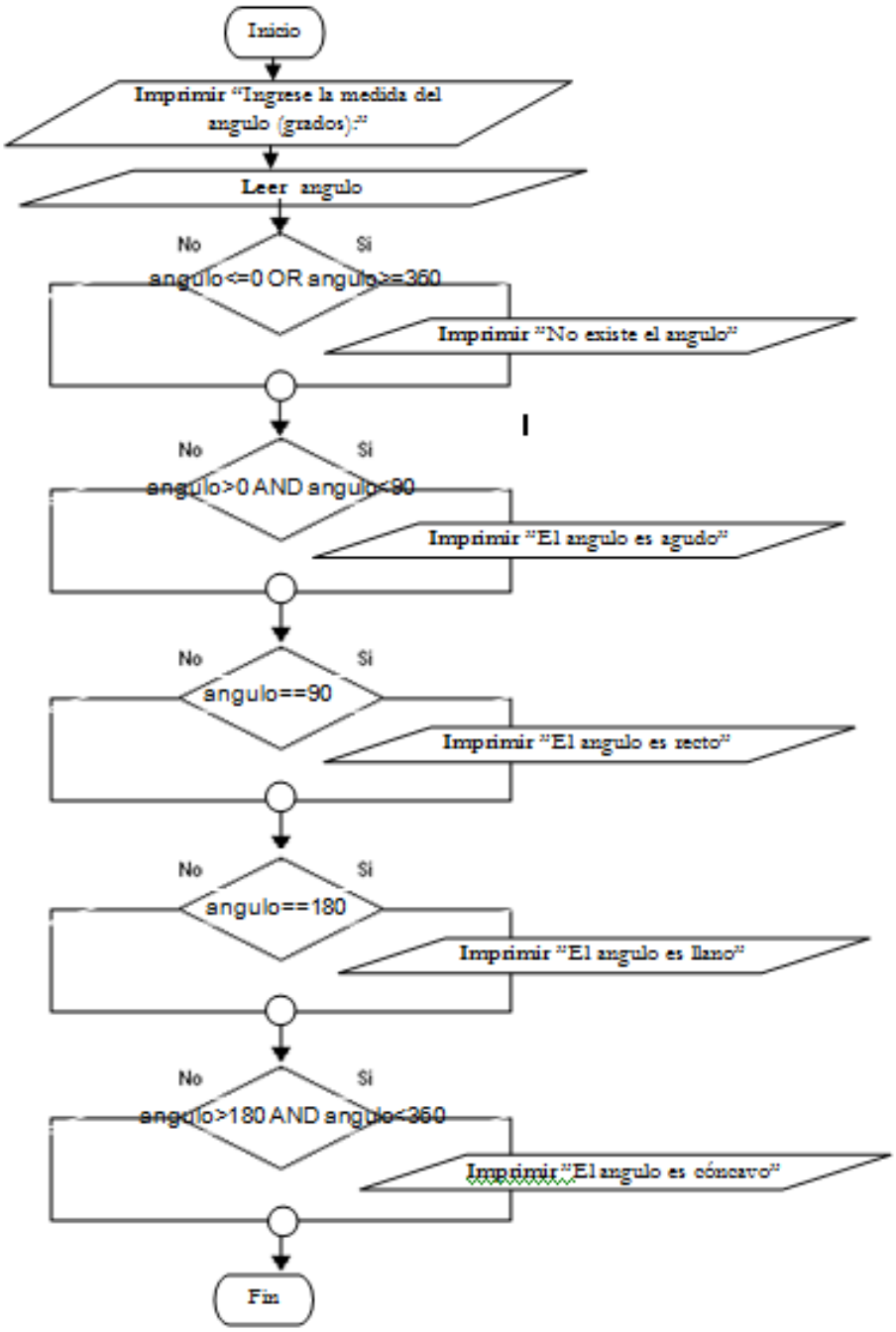
Tabla 4.2: Análisis del problema 4.1

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, tanto en pseudocódigo como en diagrama de flujo:

```
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"
 Fin_Si

 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 4.1.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.



Algoritmo 4.1.b: Tipo de ángulo – diagrama de flujo

Para reafirmar el funcionamiento de la estructura condicional simple y verificar que el algoritmo propuesto sea correcto, en la siguiente tabla se muestra su ejecución paso a paso, tomando comodatos de entrada 120 grados.

Instrucción	Dato de entrada	Operaciones (ALU)	Estado de las variables	Dato de salida
Inicio	-	-	angulo -	-
Imprimir "Ingrese la medida del ángulo (grados):".	-	-	angulo -	Ingrese la medida del ángulo (grados):
Leer angulo	120	-	angulo 120	-
Si (angulo<=0 AND angulo>=360) entonces...	-	(angulo <= 0 AND angulo >= 360) (120 <= 0 AND 120 >= 360) Falso AND Verdadero Falso	angulo 120	-
Si (angulo<90) entonces ...	-	(angulo >0 AND angulo < 90) (120 >0 AND 120<90) (1 AND 0) Falso	angulo 120	-
Si (angulo==90) entonces ...	-	(angulo == 90) (120 == 0) Falso	angulo 120	-
Si (angulo>90 AND angulo<180) entonces (La condición se evalúo como verdadera por lo tanto se ejecuta la instrucción del cuerpo)	-	(angulo >90 AND angulo < 180) (120>90 AND 120<180) (1 AND 1) Verdadero	angulo 120	
Imprimir "El ángulo es obtuso"	-	-	angulo 120	El ángulo es obtuso
Si (angulo==180) entonces ...	-	(angulo == 180) (120 == 180) Falso	angulo 120	-
Si (angulo >180 AND angulo<360) entonces...	-	(angulo > 180 AND angulo<360) (120>18 AND 120<360) Falso	angulo 120	-
Fin	-	-	angulo 120	-

Tabla 4.3: Prueba de escritorio del algoritmo 4.1

Al ejecutar paso a paso el algoritmo la única condición que satisface el estado de la memoria es la que sombreamos (angulo>90 AND angulo<180), por lo tanto, la única instrucción que se toma en cuenta es la del cuerpo del *Si* correspondiente.

El último paso es 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 {}.

```
/* 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 */
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");

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 4.1: tipoAngulo.c

A continuación se muestra la pantalla correspondiente a la ejecución del programa anterior introduciendo el mismo ángulo que se utilizó en la prueba de escritorio.

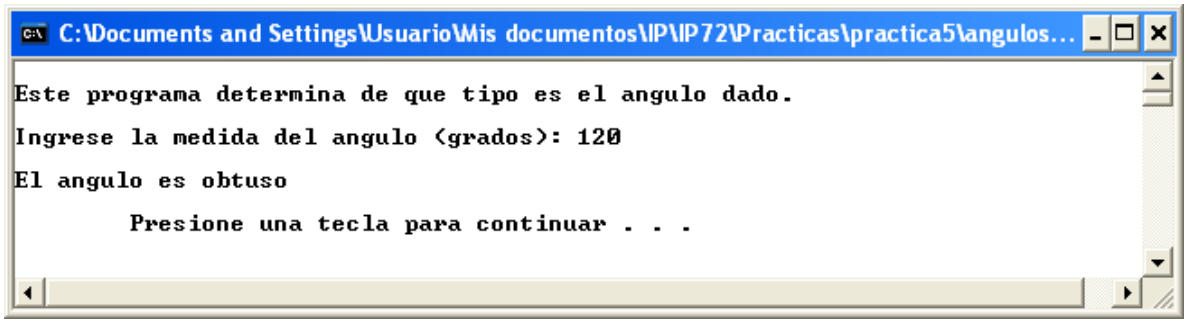


Figura 4.1: Ejecución del programa tipoAngulo.c

4.1.2. 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.

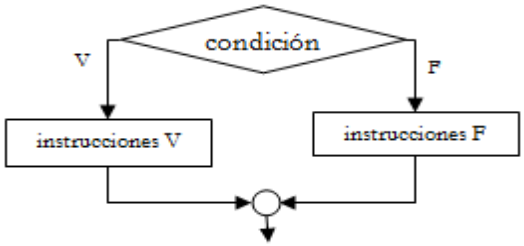
Pseudocódigo	Diagrama de Flujo	Lenguaje C
Si (<condición>) entonces <instruccionesV> sino <instruccionesF> Fin Si-Sino		**if(<**condición>**)** <instruccionesV> **else** <instruccionesF>

Tabla 4.4: 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 ésta estructura.

Problema 4.2: Se requiere un programa que simule el+ cálculo y muestre *las raíces reales para una ecuación de segundo grado de la forma:*

$$ax^2 + bx + c = 0.$$

Para resolver este problema se utilizará la fórmula general:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Por lo que los datos que requerimos son los coeficientes de la ecuación (a, b y c) y el resultado deseado serán las raíces. También se debe considerar que un polinomio tenga raíces reales se debe cumplir la condición:

$$b^2 \geq 4ac$$

De lo contrario el resultado de la raíz cuadrada sería un número imaginario. Para esto se propone una estructura selectiva doble para verificar que existan raíces reales.

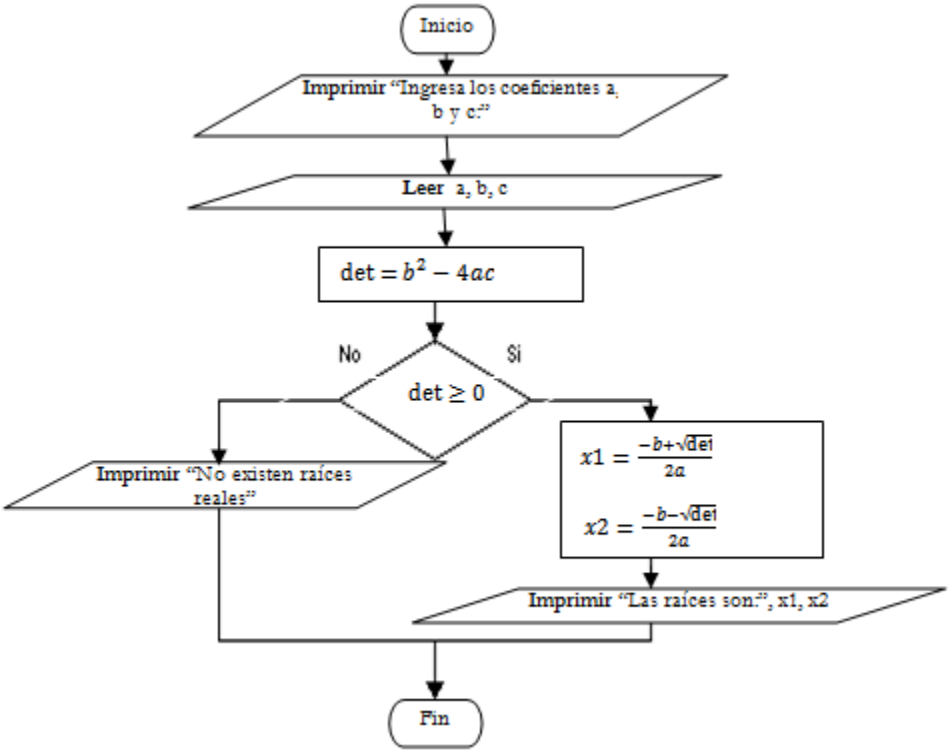
El algoritmo en pseudocódigo es el siguiente:

```
Inicio
   Imprimir “Ingresa los coeficientes a, b y c:”
   Leer a, b, c
det←b^2 − 4ac
   Si (det ≥ 0 ) entonces
                x1← (-b+√det)/2a
                x2 ← (-b-√det)/2a
                Imprimir “Las raíces son: ”, x1 , x2
   Si no
                Imprimir “No existen raíces reales”
Fin_Si-Sino
Fin
```

Algoritmo 4.2.a: Ecuación de segundo grado - pseudocódigo

Observa que en el algoritmo se utiliza la variable auxiliar det, la razón es porque el cálculo del determinante ($b^2 - 4ac$) se emplea en más de un lugar del algoritmo, por lo tanto, se recomienda guardar este resultado para no volverlo a calcular.

Ahora, la representación del mismo algoritmo en diagrama de flujo se muestra a continuación.



Algoritmo 4.2.b: Ecuación de segundo grado – diagrama de flujo

Para validar el algoritmo, veamos su ejecución paso a paso con el polinomio $x^2 + 5x + 4 = 0$.

Instrucción	Dato de entrada	Operaciones (ALU)	Estado de las variables	Dato de salida
Inicio	-	-	a b c det x1 x2 - - - - - -	-
Imprimir "Ingresa los coeficientes a,b y c".	-	-	a b c det x1 x2 - - - - - -	Ingresa los coeficientes a,b y c
Leer a, b, c	1,5,4	-	a b c det x1 x2 1 5 4 - - -	-
$det = b^2 - 4ac$	-	$det = 5^2 - 4 * 1 * 4 = 9$	a b c det x1 x2 1 5 4 9 - -	-
Si (det≥0) entonces	-	**(9 ≥ 0) verdadero**	a b c det x1 x2 1 5 4 9 - -	-
$x1 = \frac{-b + \sqrt{det}}{2a}$	-	$x1 = \frac{-5 + \sqrt{9}}{2 * 1} = -1$	a b c det x1 x2 1 5 4 9 -1 -	-
$x2 = \frac{-b - \sqrt{det}}{2a}$	-	$x1 = \frac{-5 - \sqrt{9}}{2 * 1} = -4$	a b c det x1 x2 1 5 4 9 -1 -4	-
Imprimir "Las raíces son:", x1, x2	-	-		Las raíces son: -1,-4
Fin	-	-	a b c det x1 x2 1 5 4 9 -1 -4	-

Tabla 4.5: Prueba de escritorio del algoritmo 4.2

Para la codificación en lenguaje C se debe notar que el cuerpo del *Si* (if) tiene un bloque de tres instrucciones, por lo que deberán ir encerradas entre llaves {}, en cambio el cuerpo del *Sino* (else) sólo tiene una instrucción por lo que no son necesarias.

```
/* Programa: ecuacion.c
  Descripción: Solución de ecuaciones de segundo grado utilizando
*/

/* Bibliotecas */
#include<stdio.h>/* Biblioteca de entrada y salida estándar */
#include<stdlib.h>/* Biblioteca para las funciones del sistema */
#include<math.h>/* Biblioteca para utilizar funciones matemáticas:
pow para calcular la potencia
sqrt para calcular la raíz cuadrada*/
/* Función Principal*/
main ( )
 {

/*Declaración de variables */
double  a, b, c, x1, x2, det;

/* Mensaje de bienvenida */
printf("El siguiente programa calcula las raices de un polinomio de segundo grado\n");
printf("\n\t\t ax^2 + bx + c = 0");

/* Datos de entrada */
printf ("\nIntroduzca los coeficientes de a,b y c:");
scanf ("%lf,%lf,%lf",&a,&b,&c);
det = pow (b,2)-4*a*c;

/* Verificamos que la ecuación tenga raices reales */
if (det>= 0)
     {
       x1=(-b + sqrt(det))/2*a;
       x2=(-b - sqrt(det))/2*a;
       printf ("\n La raices son: %.2lf, %.2lf",x1, x2);
     }
else
       printf ("\nNo existen raices reales.");
       printf ("\n");
       system ("pause");
}
```

Programa 4.2: ecuacion.c

La ejecución del programa con los mismos datos de entrada de la prueba de escritorio es la siguiente.

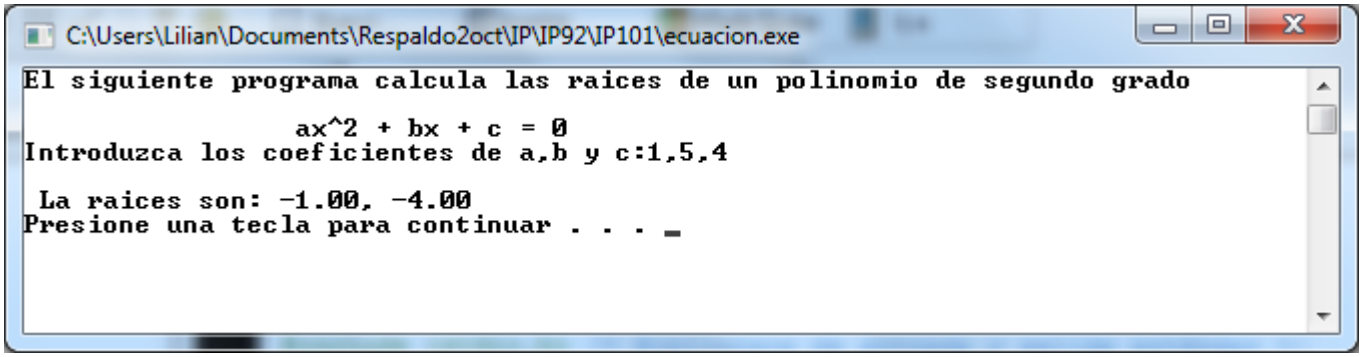


Figura 4.2: Ejecución del programa ecuacion.c

4.1.3. 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 al 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	

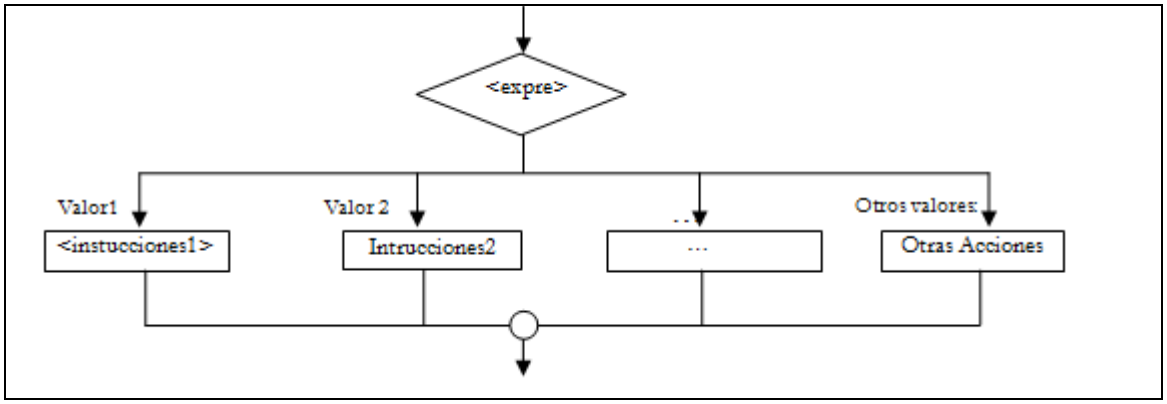


Tabla 4.6: Representaciones de la estructura condicional múltiple

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

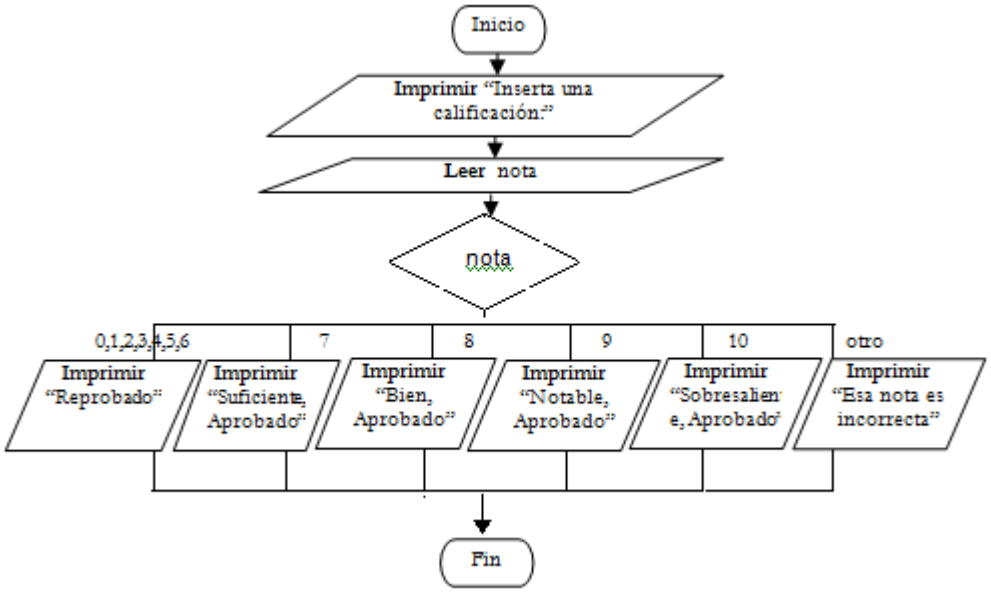
0-6: Reprobado
7: Suficiente, Aprobado
8: Bien, Aprobado
9: Notable, Aprobado
10: Sobresaliente, Aprobado

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 4.3.a: Conversión de calificación numérica a letra - pseudocódigo



Algoritmo 4.3.b: Conversión de calificación numérica a letra – diagrama de flujo

Observa que tanto en el diagrama de flujo como en el algoritmo, hay siete casos ($0 \leq nota \leq 6$) en los que la instrucción a ejecutar es la misma, por lo que se agrupan, más no se puede poner una condición que los contenga, como se haría en las estructuras vistas en las subsecciones anteriores, así que en el pseudocódigo se especifica cada valor antecedido por la palabra *caso* pero sólo se escribe una vez la instrucción y, de manera similar, se listan todos los valores. Para aclarar el funcionamiento de esta estructura y verificar si el algoritmo funciona adecuadamente, en la siguiente tabla se muestra una prueba de escritorio cuando la nota es igual a 8.

Instrucción	Dato de entrada	Operaciones	Estado de las variables	Dato de salida
Inicio	-	-	nota -	-
Imprimir "Inserta una calificación:"	-	-	nota -	Inserta una calificación:
Leer nota	8	-	nota 8	-
Casos (nota)	-	-		-
caso 0: caso1: caso2: caso3: caso4: caso5: caso6: Imprimir "Reprobado"	-	falso		-
caso 7: Imprimir "Suficiente, Aprobado"	-	falso		-
caso 8: Imprimir "Bien, Aprobado"	-	verdadero	nota 8	Bien, Aprobado
caso 9: Imprimir "Notable, Aprobado"	-	falso		-
caso 10: Imprimir "Sobresaliente, Aprobado"	-	falso		-
otros casos: Imprimir "Esa nota es incorrecta"	-	falso		-
Fin	-	-	nota 8	-

Tabla 4.7: Prueba de escritorio del algoritmo 4.3

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 éstas 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.

```
/* 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>

/*Función principal*/
main()
 {
  /*Declaración de variables*/
int nota;
  /*Mensaje de bienvenida */
```

```
printf("\nEl siguiente programa lee una calificacion con número,  \ndetermina 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 4.3: calificacion.c

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

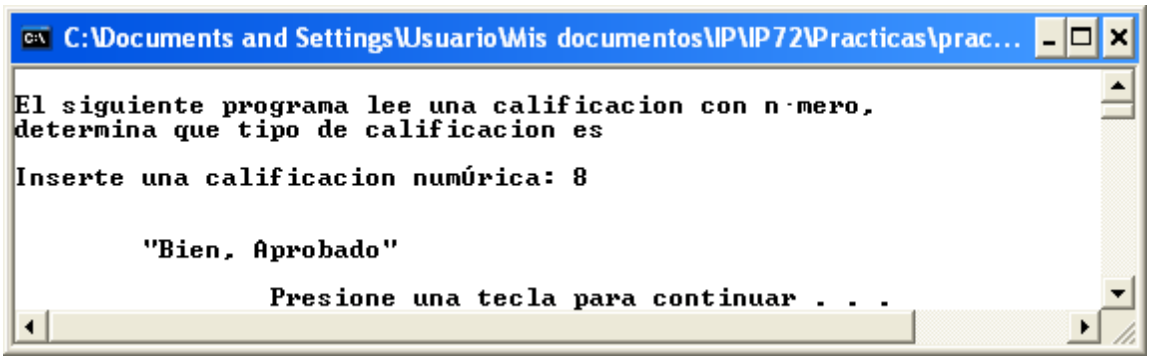


Figura 4.3: Ejecución del programa calificacion.c

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.

4.2. 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 mayorías 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 repiten 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 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--

En contraste, un *interruptor* o *bandera* es una variable que puede tomar dos posibles valores a lo largo de la ejecución del programa, éstos son: 1 (encendido/abierto) y 0 (apagado/cerrado), de ahí su nombre. Y se utilizan principalmente para registrar la ocurrencia o no de un suceso.

Por último, un *acumulador* es una variable cuyo objetivo es acumular cantidades sucesivas obtenidas al realizar la misma operación. El uso más habitual de un acumulador es obtener sumas y productos. Al igual que con los contadores, para poder utilizar un acumulador hay que realizar sobre ellos dos operaciones básicas: inicialización e incremento.

En el caso de obtener sumas el acumulador se inicializa en cero y en el caso de los productos en uno, para no afectar el resultado.

SumaTotal=0;
ProductoFinal=1;

Una vez obtenido y almacenado en una variable la cantidad a acumular la añadimos a la variable acumulador:

Variable Acumulador   Cantidad a Acumular

SumaTotal = SumaTotal + cantidad;
ProductoFinal = ProductoFinal * cantidad;

En resumen, los contadores permiten llevar la cuenta del número de iteraciones que se realizan en un ciclo (definido o indefinido), y en el caso específico de ciclos definidos son el medio por el cual se controla el fin del ciclo. Por otro lado los acumuladores nos permiten guardar resultados parciales de operaciones que se realizan como parte del cuerpo de un ciclo (puede ser definido o indefinido). En este punto, 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 equivalente, es decir, cualquiera de ellas se puede expresar en términos de las otras. En los siguientes subtemas estudiarás a profundidad cada una de ellas y verás su equivalencia.

4.2.1. 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).

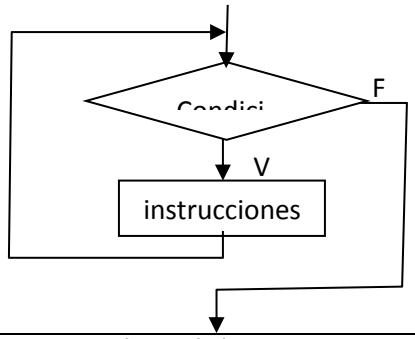
Pseudocódigo	Diagrama de Flujo	Lenguaje C
Mientras <condición>**hacer** <instrucciones> **Fin mientras**		**while** *(<condición>)* *<instrucciones>;*

Tabla 4.8: 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 4.4: Se requiere un programa que calcule el promedio de temperaturas que registra una ciudad, las temperaturas se introducirán en grados Farenheit °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 Farenheit, 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 ($tempF_1 + tempF_2 + \ldots, + tempF_n$) y dividirlas entre el número total de temperaturas que se hayan leído, digamos n. Lo anterior se expresa con la siguiente fórmula.

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

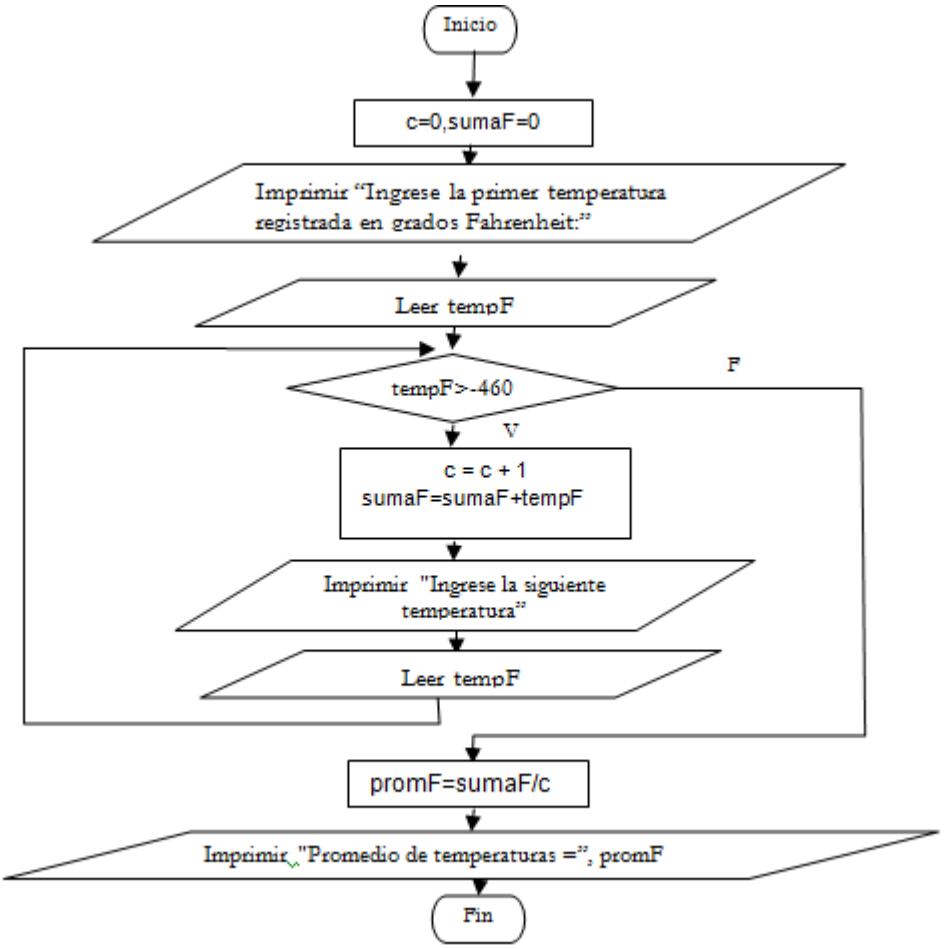
Así que en este caso se usará un ciclo que vaya leyendo una a una las temperaturas (almacenándolas en la variable $tempF$) y acumulando la suma en la variable $sumaF$, estas acciones se repetirán hasta que el usuario introduzca un número menor a -460. De esta manera, la condición de término es: $sumaF \geq -460$; 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 4.4.a: Promedio temperaturas - pseudocódigo

La representación del algoritmo en diagrama de flujo que se muestra en la siguiente figura:



Algoritmo 4.4.b: Promedio temperaturas – diagramas de flujo

En la siguiente tabla se muestra una prueba de escritorio para las temperaturas 75, 78, 79 y para concluir el ciclo -2000.

Instrucción		Datos de entrada	Operaciones (ALU)	Estado de las variables	Salida a monitor
Inicio		-	-	-	-
c=0,sumaF=0		-	-	c: 0 | sumaF: 0	-
Imprimir "Ingrese la primer temperatura registrada en grados Fahrenheit:"		-	-	c: 0 | sumaF: 0	Ingrese la primer temperatura registrada en grados Fahrenheit:
Leer tempF		75	-	c: 0 | sumaF: 0 | tempF: 75	-
Mientras (tempF>-460)		-	**Mientras** (75>-460) **verdadero**	c: 0 | sumaF: 0 | tempF: 75	-
iteración	Instrucción				
1	c = c + 1	-	-	c: 1 | sumaF: 0 | tempF: 75	-
	sumaF=sumaF+tempF	-	suma=0+75	c: 1 | sumaF: 75 | tempF: 75	-
	Imprimir "Ingrese la siguiente temperatura..."	-	-	c: 1 | sumaF: 75 | tempF: 75	Ingrese la siguiente temperatura...
	Leer tempF	78	-	c: 1 | sumaF: 75 | tempF: 78	-
Mientras (tempF>-460)		-	**Mientras**(78>-460) **verdadero**	c: 1 | sumaF: 75 | tempF: 78	-
iteración	Instrucción				
2	c = c + 1	-	c=1+1	c: 2 | sumaF: 75 | tempF: 78	-
	sumaF=sumaF+tempF	-	sumaF=75+78	c: 2 | sumaF: 153 | tempF: 78	-
	Imprimir "Ingrese la siguiente temperatura..."	-	-	c: 2 | sumaF: 153 | tempF: 78	Ingrese la siguiente temperatura...
	Leer tempF	79	-	c: 2 | sumaF: 153 | tempF: 79	-

Mientras (tempF>-460)		-	**Mientras** (79>-460) **v e r d a d e r o**	c: 2 / sumaF: 153 / tempF: 79	-
iteración	Instrucción				
3	c = c + 1	-	c=2+1	c: 3 / sumaF: 153 / tempF: 79	-
	sumaF=sumaF+tempF	-	sumaF=153+79	c: 3 / sumaF: 232 / tempF: 79	-
	Imprimir "Ingrese la siguiente temperatura"	-	-	c: 3 / sumaF: 232 / tempF: 79	Ingrese la siguiente temperatura
	Leer tempF	77	-	c: 3 / sumaF: 232 / tempF: -2000	-
Mientras (tempF>-460)		-	**Mientras** (-20000>-460) **f a l s o**	c: 3 / sumaF: 232 / tempF: -20000	-
F I N D E L C I C L O					
promF.=sumaF/c		-	promF= 380 / 5	c: 3 / sumaF: 232 / tempF: -20000 / promF: 77.33	-
Imprimir "Promedio =" promF		-	-	c: 3 / sumaF: 232 / tempF: -20000 / promF: 77.33	Promedio = 77.33
F I N		-	-	c: 3 / sumaF: 232 / tempF: -20000 / promF: 77.33	-

Tabla 4.9: Prueba de escritorio del algoritmo 4.4

En la tabla 4.9 se puede observar como el mismo conjunto de instrucciones se repite tres veces (3 iteraciones), en cada una se valida la condición y el ciclo termina sólo cuando ésta no se satisface por el estado de las variables implicadas en la expresión booleana. Una vez que se ha ilustrado el funcionamiento del ciclo Mientras y verificado que si funciona, el siguiente paso es la codificación, para la cual se determinó utilizar una variable para representar el valor centinela que controla el ciclo.

```
/* Programa: promTemp.c
   Descripción: Calcula el promedio de las temperaturas que el usuario ingresa.
*/
#include<stdio.h>
#include<stdlib.h>
#define centinela -460

/* Función principal */
main () {
/*Declaración de acumuladores y contadores*/
float tempF,promF, sumaF=0;
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 4.4: promTemp.c

Por último, en la siguiente figura se muestra la ejecución del programa con los mismos datos que se utilizaron en la prueba de escritorio.

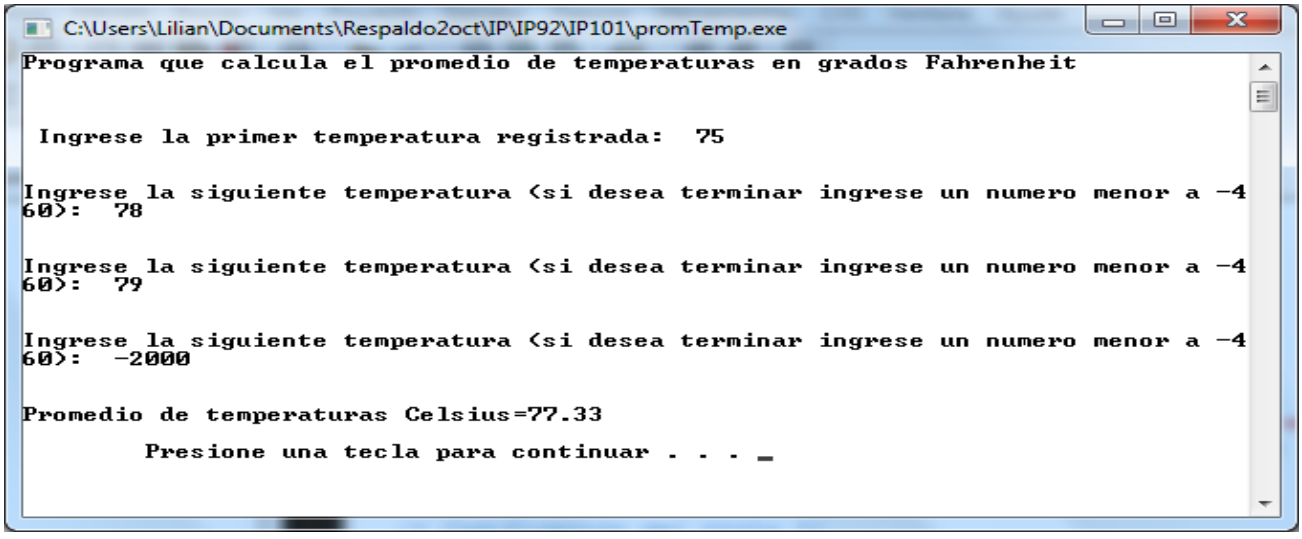


Figura 4.4: Ejecución del programa promTemp.c

4.2.2. 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.

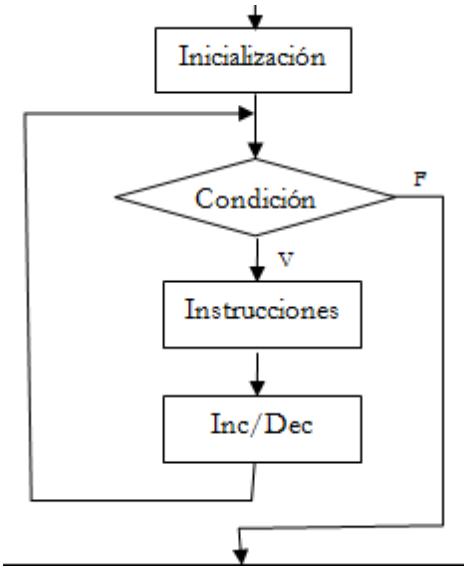
Pseudocódigo	Diagrama de Flujo
Desde <inicialización>**Mientras**<condición>, <incr/decr> <Instrucciones> **Fin desde**	
Lenguaje C	
for (<inicialización>;<condición>; <inc/dec>**)** <instrucciones>	

Tabla 4.8: 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.

Es importante señalar que si te centras en la representación en diagrama de flujo de este ciclo podrás darte cuenta que se parece al diagrama del ciclo while, salvo por la inicialización y el incremento (o decremento); de hecho si partes del diagrama de flujo para la codificación puedes utilizar un while. De igual manera, un ciclo while se puede representar

con un ciclo for cuya <inicialización> e <incremento/decremento> son vacíos, sólo se define la condición. Con lo anterior se muestra la equivalencia de las dos estructuras.

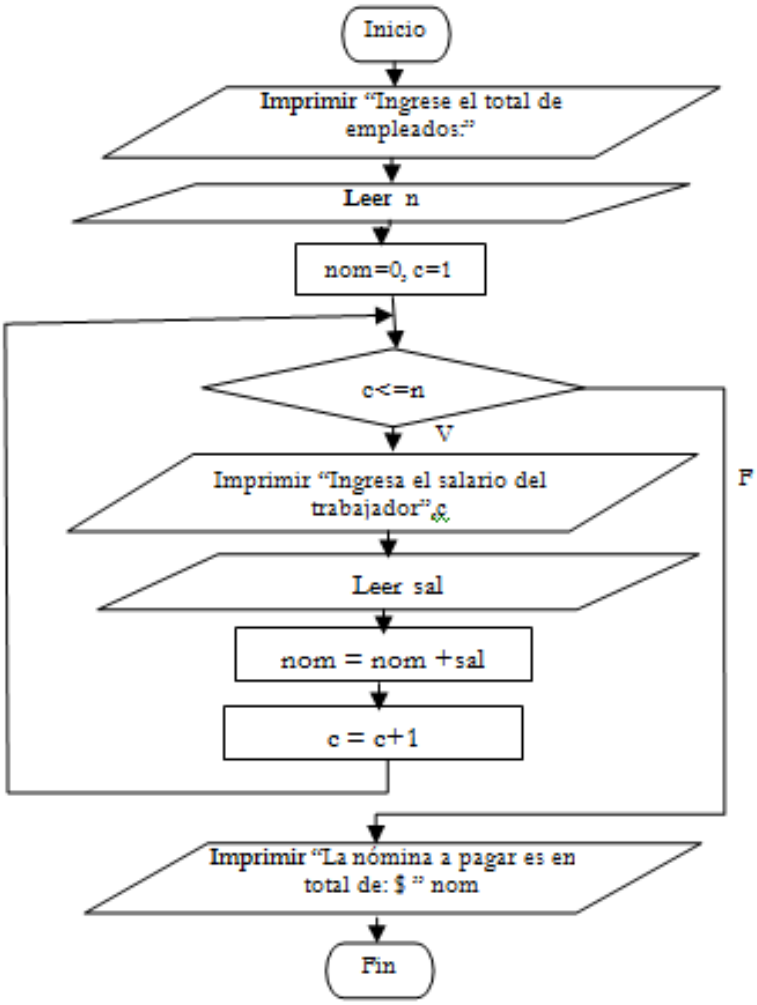
Para ejemplificar las representaciones, codificación y funcionamiento de esta estructura se presenta el siguiente problema desarrollado.

Problema 4.5: 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 4.5.a: Nómina - pseudocódigo



Algoritmo 4.5.b: Nómina – diagrama de flujo

En la siguiente tabla se muestra la ejecución paso a paso del ciclo suponiendo que se quiere calcular la nómina de tres empleados y ya se ha leído el dato y registrado en la variable n, la cual no cambia su valor a lo largo de la ejecución del ciclo.

Instrucción		Datos de entrada	Operaciones (ALU)	Estado de las variables	Salida a monitor
Desde nom=0,c=1; **Mientras** (c<=n);		-	(1<=3) **verdadero**	nom: 0, c: 1	-
iteración	Instrucción				
1	**Imprimir** "Ingresa el salario del trabajador",c	-	-	nom: 0, c: 1	Ingresa el salario del trabajador 1
	Leer sal	10	-	nom: 0, c: 1, sal: 10	-
	nom=nom+sal	-	nom=0+10	nom: 10, c: 1, sal: 10	-
	c=c+1	-	c=1+1	nom: 10, c: 2, sal: 10	-
Desde nom=0,c=1; **Mientras** (c<=n);		-	(2<=3) **verdadero**	nom: 10, c: 2, sal: 10	-
iteración	Instrucción				
2	**Imprimir** "Ingresa el salario del trabajador",c	-	-	nom: 10, c: 2, sal: 10	Ingresa el salario del trabajador 2
	Leer sal	15	-	nom: 10, c: 2, sal: 15	-
	nom=nom+sal	-	nom=10+15	nom: 25, c: 2, sal: 15	-
	c=c+1	-	c=2+1	nom: 25, c: 3, sal: 15	-
Desde nom=0;c=1; **Mientras** (c<=n);		-	(3<=3) **verdadero**	nom: 25, c: 3, sal: 15	-
iteración	Instrucción				
3	**Imprimir** "Ingresa el salario del trabajador",c	-	-	nom: 25, c: 3, sal: 15	Ingresa el salario del trabajador 3
	Leer sal	20	-	nom: 25, c: 3, sal: 20	-
	nom=nom+sal	-	nom=25+20	nom: 45, c: 3, sal: 20	-
	c=c+1	-	c=3+1	nom: 45, c: 4, sal: 20	-
Desde c=1; **Mientras** (c<=n);		-	(4<=3) **Falso**	nom: 45, c: 4, sal: 20	-

Tabla 4.9: Prueba de escritorio del algoritmo 4.4

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>
```

```
/*Función Principal*/
main ()
{
/* Declaración de las variables */
int n,c;
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 4.5: nomina.c

En la siguiente figura se muestra la ejecución del programa con los mismos datos de entrada que se utilizaron en la prueba de escritorio.

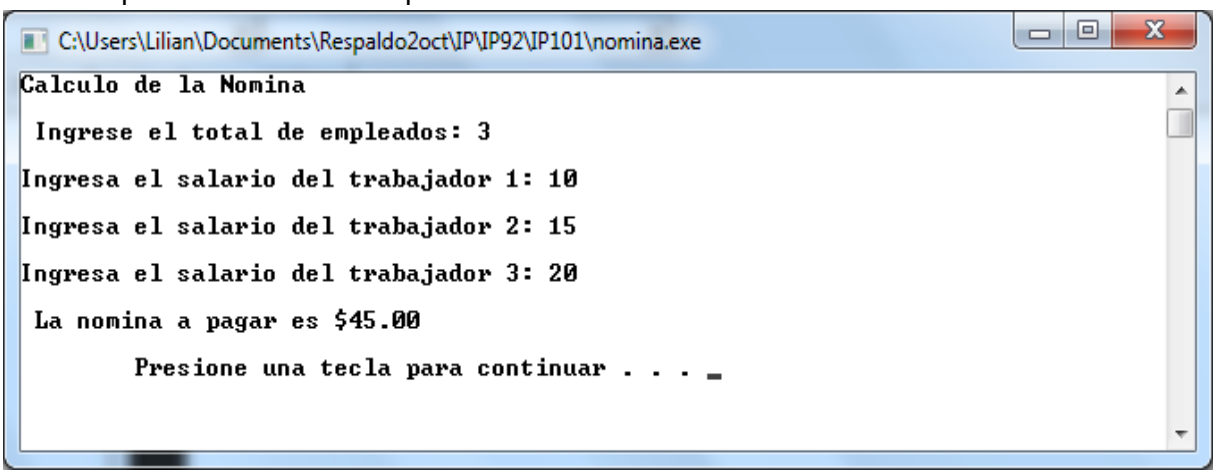


Figura 4.6: Ejecución del programa nomina.c

4.2.3 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.

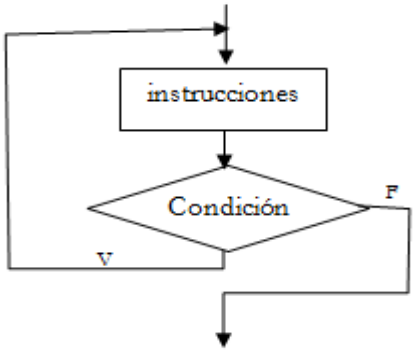
Pseudocódigo	Diagrama de Flujo	Lenguaje C
Hacer <instrucciones> **Mientras** <condición>**Fin**		**do** <instrucciones>; **while**(<condición>);

Tabla 4.9: 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 4.6: 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”

$$fuerza = masa * aceleración$$

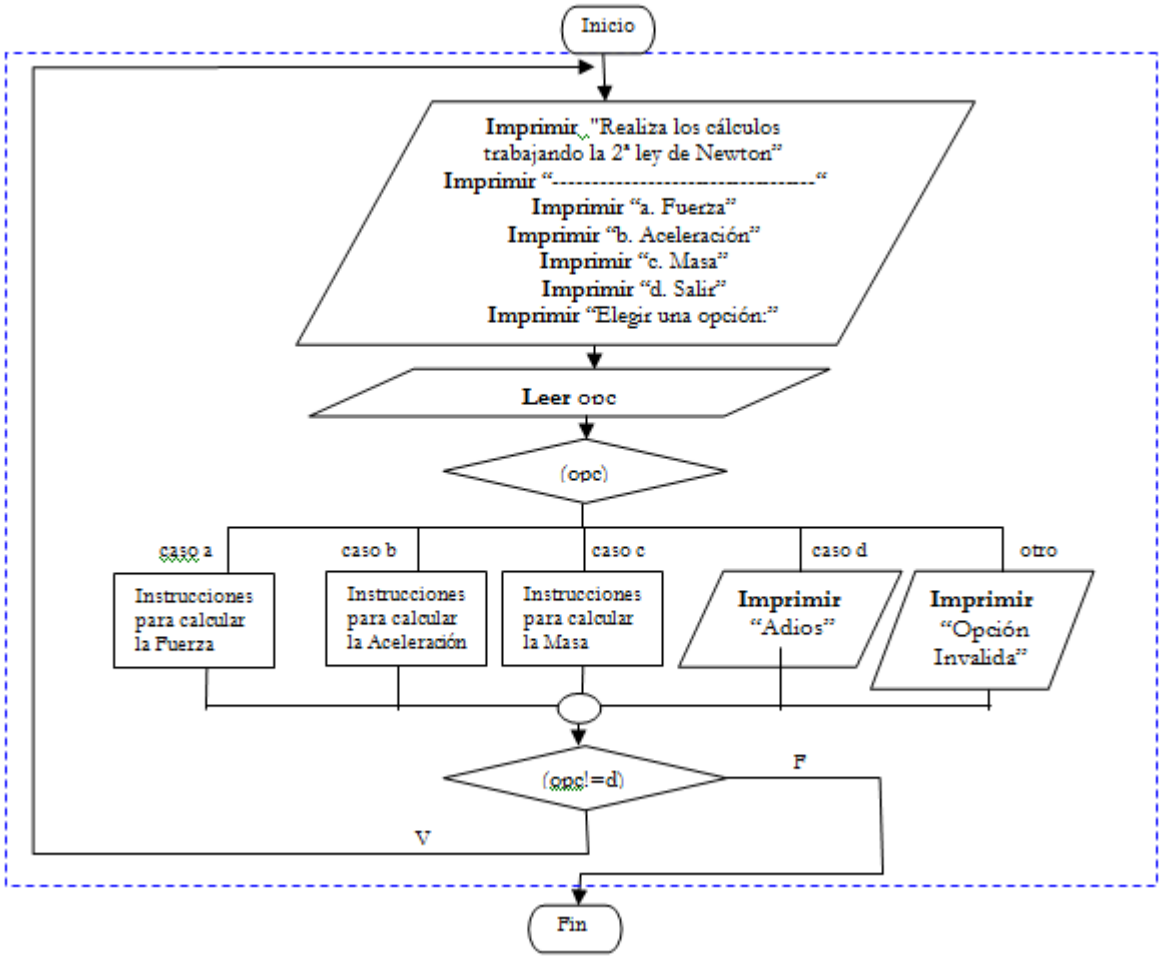
En este caso, para resolver la parte del menú se utilizará un switch-case, en el que cada opción del menú corresponda a un caso, así las instrucciones que lo forman deben ser: la lectura de los datos correspondientes y la operación apropiada (que se define despejando

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
Mientras (opc!=4) Fin
 Fin
```

Algoritmo 4.6.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 4.6.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 a continuación.

```
/* Programa: newton.c
   Descripción: Muestra un menú para calcular la aceleración,          fuerza o masa,
conforme a la segunda ley de Newton */
/*directivas de preprocesador*/
#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 Selección Múltiple*/
switch (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 4.5: newton.c

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

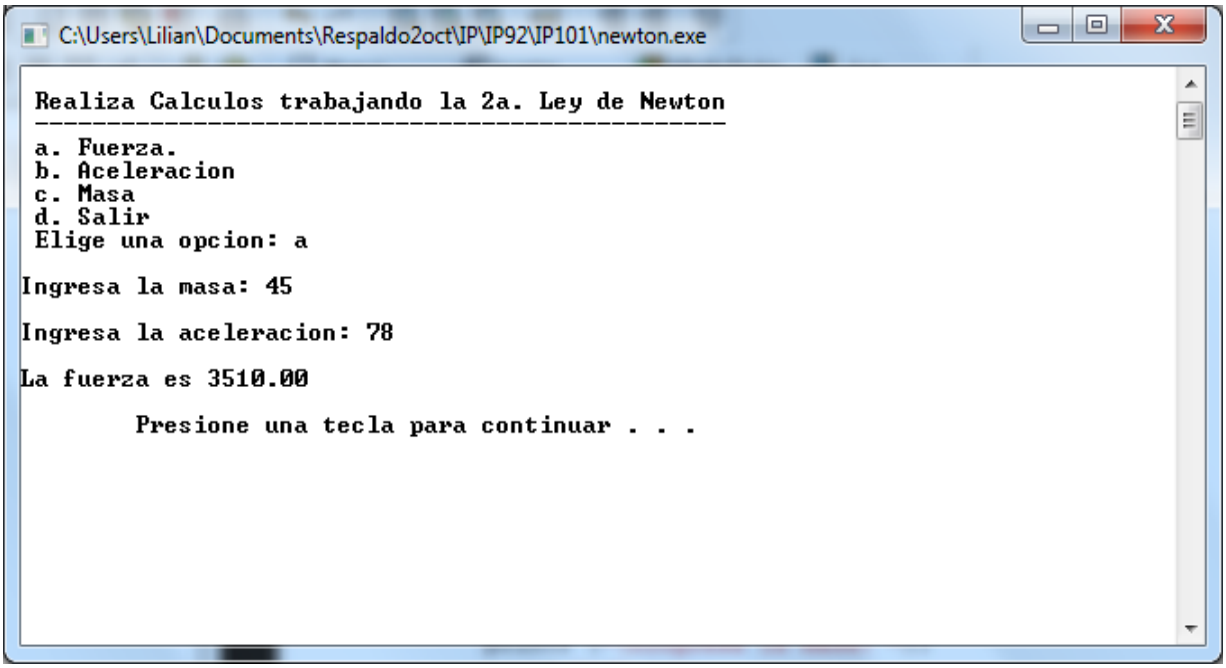


Figura 4.6: Ejecución del programa newton.c

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 anidamientos 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.

4.3. 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 a 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 4.6 y programa 4.6), 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 4.1, 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 4.7: 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 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 ("pause");
}
```

Programa 4.7: tipoAngulo2.c

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

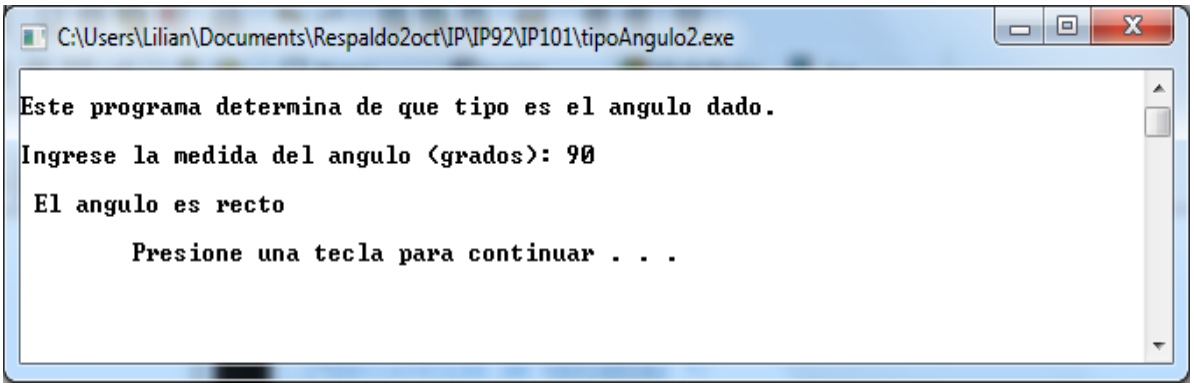


Figura 4.7: Ejecución del programa tipoAngulo2.c

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.

Cierre de la unidad

Con esto damos 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. Recuerda que debes practicar mucho para dominar cada uno de los componentes del lenguaje C. No dejes de repasar los temas y de realizar tus propios programas. ¡Adelante!

Fuentes de consulta

- Joyanes, L., & Zohanero, I. (2005). *Programación en C. Metodología, algoritmos y estructuras de datos.* aspaño: 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.
- Pérez, H. (1992). *Física General* (Primera Edición ed.). México: Publicaciones Cultura.