ÍNDICE

Capítulo: Conceptos Básicos

Capítulo II: Estructuras lógicas selectivas

Capitulo III: Estructuras lógicas repetitivas

Capítulo IV: Vectores o Arreglos

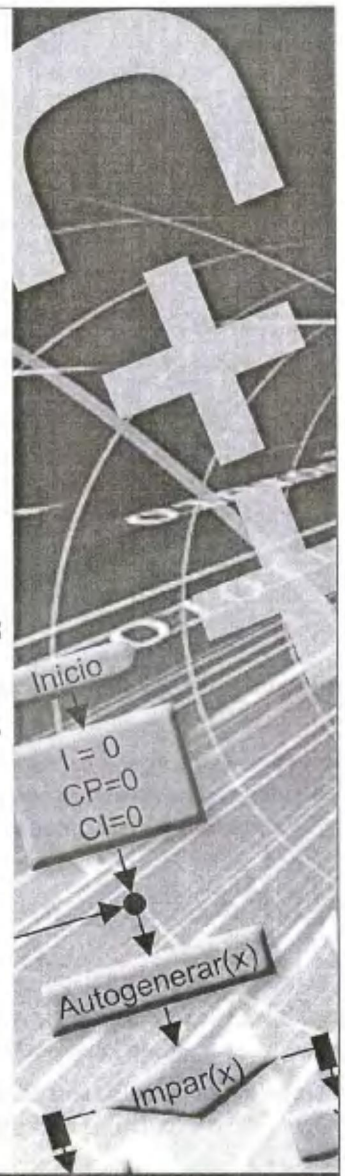
Capítulo V: Uso de funciones para desarrollar programas

Capítulo I

Conceptos Básicos

Este capítulo contiene:

- Conceptos básicos de algoritmia, peudocódigo y diagramas de flujo
- Elementos presentes en un algoritmo como variables, constantes, expresiones
- Tipo de operadores
- Declaración de variables y asignaciones
- Técnicas de desarrollo de algoritmos
- Ejemplos desarrollados



Conceptos básicos de algoritmia

Algoritmos

Los algoritmos constituyen un listado de instrucciones que indican el camino a seguir para dar solución a un problema.

Podríamos decir que un algoritmo es la suma de una parte lógica mas una parte de control, en donde la parte lógica especifica el conocimiento en la solución del problema y la parte de control es la estrategia para solucionar el problema.

Características de los algoritmos

- Un algoritmo no debe de ser ambiguo.
- Debe de tener una secuencia inicial
- Cada paso deberá tener una secuencia sucesiva y única es decir que deben indicar claramente el camino a seguir en la solución del problema.
- El algoritmo debe de ser siempre eficiente y dar una solución al problema o de lo contrario deberá dar un mensaje que diga "Sin solución"

Programa

Un programa define un "algoritmo", porque constituye el conjunto de

instrucciones que forman el algoritmo (ya codificados en un lenguaje de programación) y que se dan a una computadora para solucionar un problema específico.

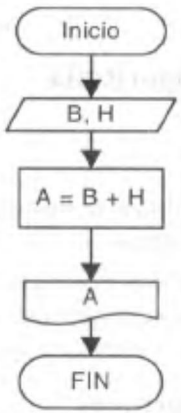
Heurística

Un algoritmo que da o produce una solución a un problema planteado aunque esta solución no sea la óptima es llamado "Heurístico".

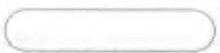
Diagrama de flujo

El diagrama de flujo es la representación gráfica de dicha secuencia de instrucciones que conforman el algoritmo.

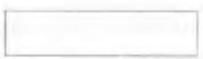
El siguiente es un ejemplo de un diagrama de flujo para sumar dos variables B y H, el resultado es almacenado en la variable A.



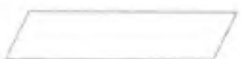
Los símbolos mas comunes y los cuales usaremos son:



Terminal: Se usa para indicar el inicio o fin de un diagrama de flujo



Proceso: Cualquier tipo de operación que pueda originar cambio de valor, operaciones aritméticas, de transformaciones, etc.

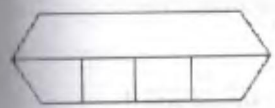


Entrada/Salida: Se usa para indicar el ingreso o salida de datos.

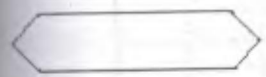
Salida: Se utiliza para mostrar datos, será el símbolo usado en este texto.

Conector: Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada.

Salida: Indica operaciones lógicas o de comparación entre datos y en función del resultado de la misma determina cual de los distintos caminos alternativos del programa se debe seguir.



En caso de: Usado para indicar varias acciones posibles según sea un dato de entrado al control.



Desde: Estructura repetitiva q indica un ciclo de N repeticiones de una o mas acciones.

Variables

Son los elementos que se utilizan para contener datos de distintos tipos: números, letras, cadenas de caracteres, valores lógicos, etc. El valor contenido en una variable puede cambiar a lo largo de la ejecución de un programa.

Constantes

Son elementos que contienen datos, el valor asignado a una constante es fijo y no se puede cambiar durante toda la ejecución de un programa.

Expresiones

Las expresiones son combinaciones de constantes, variables, simbolo de operación, paréntesis y nombres de funciones especiales.

Por ejemplo

$$a + (b + 3) / c$$

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones implicadas.

Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

Aritméticas

Relacionales

Lógicas

Operadores

Operadores Aritméticos

Los operadores aritméticos nos permiten, básicamente, hacer cualquier operación aritmética (suma, resta, multiplicación y división). En la siguiente tabla se muestran los operadores de los que se dispone.

Operador	Acción	Ejemplo	Resultado
-	Resta	x = 5 - 3	2
+	Suma	x = 2 + 3	5
*	Multiplicación	x = 2 * 3	6
/	División	x = 6 / 2	3
^	Potencia	x = 3 ^2	9
MOD	Módulo	x = 5 MOD 2	1
DIV	División entera	x= 5 DIV 2	2

El operador MOD nos devuelve el residuo de una división entera, mientras que el operador DIV permite realizar una división entre dos números enteros, allí radica la diferencia con el operador "/".

Operadores relacionales

Al igual que en matemáticas, estos operadores nos permitirán evaluar las relaciones (igualdad, mayor, menor, etc) entre un par de operandos (en principio, pensemos en números). Los operadores relacionales de los que disponemos son:

Operador	Acción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
=	Igual
!= o <>	Distinto

Operadores Lógicos

Los operadores lógicos producen un resultado **booleano**, y sus operandos son también valores lógicos o asimilables a ellos (los valores numéricos son asimilados a **cierto** o **falso** según su valor sea cero o distinto de cero).

Los operadores lógicos son tres; dos de ellos son binarios, el último (negación) es unario.

AND (Y)

Produce un resultado con valor de verdad true cuando ambos operandos tienen valor de verdad true; en cualquier otro caso el resultado tendrá un valor de verdad false.

Sintaxis: operando1 **AND** operando2

OR (O)

Produce un resultado con valor de falso cuando ambos operadores tienen valores falsos; en cualquier otro caso el resultado tendrá un valor de verdad.

Sintaxis: operando1 **OR** operando2

NOT (NO)

Invierte el valor de verdad de operando.

Sintaxis: **NOT** operando

Ejemplo:

Si operando tiene un valor de verdad, not operando produce un resultado con valor falso.

Prioridad de los Operadores

Los operadores deben ser evaluados según la siguiente prioridad

1.- ( )

2.- ^

3.- *, /, Mod, Not

4.- +, -, And

5.- >, <, >=, <=, <>, =, Or

ENTRADA / SALIDA de datos

Los dispositivos de entrada / salida permiten que el usuario interactúe con la

máquina. Por medio de los dispositivos de entrada el usuario ingresa los datos a procesar en el sistema y los dispositivos de salida presentan los resultados en un formato legible.

Las instrucciones de E/S dan acceso al programador a las funciones básicas de estos dispositivos, permitiéndole capturar datos de los dispositivos de entrada y asignarlos a variables para operar con ellos y mostrar resultados del proceso en los dispositivos de salida.

Instrucción LEER

Nos permite el ingreso de datos al computador, el dato leido debe ser almacenado en una variable, la sintaxis es la siguiente:

Leer variable

Instrucción ESCRIBIR

Esta instrucción permite reportar resultados de un proceso, también se usa para enviar un mensaje al operario. La sintaxis es la siguiente:

ESCRIBIR Variable

ESCRIBIR 'Texto'

ESCRIBIR Expresión

Veamos unos ejemplos, según sean las sintaxis anteriores respectivamente

ESCRIBIR Resultado

Esta instrucción devuelve el contenido de la variable Resultado

ESCRIBIR 'Prepárese para el ingreso de datos'

Esta instrucción muestra al usuario el contenido de los apóstrofes, nótese que para poder emitir un texto este debe ir encerrado en apóstrofes

ESCRIBIR 4*n

Esta instrucción primero calcula la expresión 4*n y luego muestra ese resultado.

Tipos de datos escalares

Son aquellos tipos de datos cuyos miembros están compuestos por un solo ítem (dato). Los tipos de datos escalares nativos son aquellos tipos de datos escalares que ya están implementados en el lenguaje junto a sus respectivas operaciones.

Entre estos tipos de datos tenemos Entero, Real, Caracter, Booleano; más adelante veremos otros tipos de datos como por ejemplo los tipos de datos agregados.

Asignaciones

Una asignación consiste en darle un determinado valor a una variable o constante, por ejemplo en la siguiente sentencia observamos que a la variable A, se le da el valor de 5.

A = 5

De manera similar podemos tener la siguiente asignación

A = 4 + (3 * Y^2)

vemos que una expresión a sido asignada a la variable A

Algunos autores usan el símbolo ← en vez de igual (=) para una asignación.

Declaración de variables

Mediante la declaración de variables describimos las características de las mismas. La sintaxis que usaremos es la siguiente:

Nombre_de_variable : Tipo

Entiéndase por tipo, al tipo de dato de la variable.

Técnicas de desarrollo de algoritmos

Existen dos principales técnicas de diseño de algoritmos de programación, el Top Down y el Bottom Up.

Top Down

También conocida como de arriba-abajo y consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al problema. Consiste en efectuar una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante entradas y salidas de información. Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.

La utilización de la técnica de diseño Top-Down tiene los siguientes objetivos básicos:

- Simplificación del problema y de los subprogramas de cada descomposición.
- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.

❖El programa final queda estructurado en forma de bloque o módulos lo que hace mas sencilla su lectura y mantenimiento.

Bottom Up

El diseño ascendente se refiere a la identificación de aquellos procesos que necesitan computarizarse conforme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Cuando la programación se realiza internamente y haciendo un enfoque ascendente, es difícil llegar a integrar los subsistemas al grado tal de que el desempeño global, sea fluido. Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha limite para la integración total del sistema. En esta fecha, ya se cuenta con muy poco tiempo, presupuesto o paciencia de los usuarios, como para corregir aquellas delicadas interfaces, que en un principio, se ignoran. Aunque cada subsistema parece ofrecer lo que se requiere, cuando se contempla al sistema como una entidad global, adolece de ciertas limitaciones por haber tomado un enfoque ascendente.

Uno de ellos es la duplicación de esfuerzos para acceder el software y mas aún al introducir los datos.

Otro es, que se introducen al sistema muchos datos carentes de valor.

Un tercero y tal vez el mas serio inconveniente del enfoque ascendente, es que los objetivos globales de la organización no fueron considerados y en consecuencia no se satisfacen.

Entonces...

La diferencia entre estas dos técnicas de programación se fundamenta en el resultado que presentan frente a un problema dado.

Imagine una empresa, la cual se compone de varios departamentos (contabilidad, mercadeo, ...), en cada uno de ellos se fueron presentando problemas a los cuales se les dio una solución basada en un enfoque ascendente (Bottom Up): creando programas que satisfacían sólo el problema que se presentaba.

Cuando la empresa decidió integrar un sistema global para suplir todas las necesidades de todos los departamentos se dio cuenta que cada una de las soluciones presentadas no era compatible la una con la otra, no representaba una globalidad, característica principal de los sistemas.

Como no hubo un previo análisis, diseño de una solución a nivel global en todos sus departamentos, centralización de información, que son características propias de un diseño Descendente (Top Down) y características fundamentales de los sistemas; la empresa no pudo satisfacer su necesidad a nivel global.

La creación de algoritmos es basado sobre la técnica descendente, la cual brinda el diseño ideal para la solución de un problema.

Pseudocódigo

Diremos que una notación es un pseudocódigo si mediante ella podemos describir el algoritmo, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas.

Todo pseudocódigo debe posibilitar la descripción de:

❖Instrucciones de entrada / salida.

❖Instrucciones de proceso.

❖Sentencias de control del flujo de ejecución.

❖Acciones compuestas, a refinar posteriormente.

Para entender mejor estos conceptos veamos algunos ejemplos.

Ejemplo 1.1

Elabore un algoritmo y su pseudocódigo para calcular e imprimir el área de un triángulo.

Solución

Declarar Variables

b,h : real

Entrada: (datos a introducir al computador)

Base = b

Altura = h

Operación: calcular área del triángulo (Base x Altura)/2

Salida: (Resultado que mostrará el computador)

Pseudocódigo:

```
1. Iniciar proceso
2. Declarar variables
```

```
        h : real
        b : real
        A : real
3. LEER b
4. LEER h
5. A = ( b * h ) / 2
6. ESCRIBIR A
7. Terminar el proceso
```

Este pseudocódigo es fácil pero daremos una pequeña explicación de ello.

1. Se inicia el proceso, todo programa o bloque de programa, debe tener un indicativo de inicio, ya sea inicio de bloque o inicio de programa.
2. Declaramos las variables a usar, nótese que las variables h y b son de tipo real, de eso se deduce que el resultado de una operación entre estas dos variables también sea del tipo real, por eso A es del tipo real.

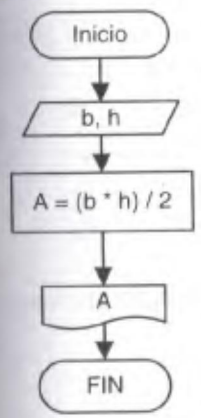
   Las siguiente línea, tiene el mismo resultado para la declaración de las variables b, h y A.

   b,h, A : Real
3. Leemos el valor de la variables b, por ejemplo digamos el valor introducido sea 5
4. Leemos el valor de la variable h, para este ejemplo ingresemos 7.
5. Calculamos la expresión (b*h)/2 y dicho resultado lo almacenamos en la variable A, entonces será: A = (5*7)/2, por lo tanto tendremos A = 17.5
6. Escribimos el resultado de la operación, en este caso será 7.
7. Cuando un proceso o programa se inicia, debemos indicar su fin, caso contrario, dicho proceso puede ejecutarse infinitamente, es decir al ser traducido a un lenguaje de programación

Una vez elaborado el algoritmo, el problema a programar está prácticamente resuelto, ya que lo único que faltaría sería la codificación correspondiente en el lenguaje de programación que se esté usando, técnicamente un algoritmo es codificable en cualquier lenguaje de programación.

Primero veamos cual es el diagrama de flujo de este ejercicio y luego su codificación en C++

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float h;
float b;
float A;
clrscr();
cin>>b;
cin>>h;
A = ( b * h ) / 2;
cout<<A;
getch();}
```

Las dos primeras líneas del programa en C++ hacen referencia a las librerías que deben usar para la ejecución de los comandos usados, tales como clrscr() y getch(), funciones que nos permiten limpiar la pantalla y detener la acción hasta presionar una tecla.

El comando cin nos permite leer un dato desde el teclado mientras que el comando cout nos muestra por pantalla el contenido de una variable o un comentario cualquiera.

La declaración de las variables os del tipo float q es lo mismo que decir del tipo real

Ejemplo 1.2

Construya un pseudocódigo, que dados los datos A, B, C y D que representan números enteros, escriba los mismos en orden inverso.

Solución

Declarar variables

A, B, C, D : Entero

Ingresar datos A, B, C, D

Imprimir datos invertidos

Nótese que por ahora este ejercicio solo es una simple lectura y escritura de datos. La declaración de datos lo hemos hecho en una sola línea debido a que todas las variables son del mismo tipo.

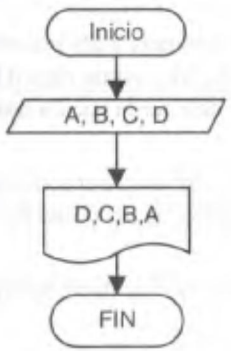
Suponiendo que se ingresan los valores: 7, 28, 150, 35, la impresión produce lo siguiente: 35, 150, 28, 7.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
      A, B, C, D : Entero
3. LEER A, B, C, D
4. ESCRIBIR D, C, B, A
5. Terminar el proceso
```

Nótese que en la línea 3 y 4, usamos una línea para leer y escribir el contenido de las variables respectivamente.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int A, B, C, D;
clrscr();
cin>>A>>B>>C>>D;
cout<<D<<"  "<<C<<"  "<<B<<"
"<<A;
getch();
}
```

En la escritura de datos, después de cada variable mostrada, se imprime dos espacios en blanco descritos entre comillas, con la finalidad de no mostrar en pantalla el valor de las variables juntas.

Ejemplo 1.3

Construya un pseudocódigo, que dado los datos enteros A y B, escriba el resultado de la siguiente expresión:

$$\frac{(A+B)^2}{3}$$

Solución:

Declarar variables

A, B : Entero

Ingresar A, B

Realizar un proceso, donde se ejecute la formula indicada ((A + B)2)/3.

Mostrar el resultado

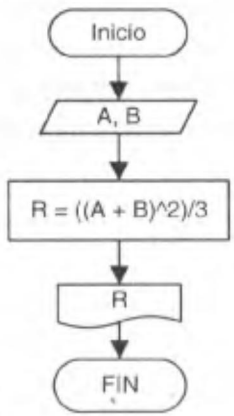
Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
      A, B : Entero
      R    : Real
3. LEER A, B
4. Calcular R = ((A + B)^2)/3
4. ESCRIBIR R
5. Terminar el proceso
```

R: Variable de tipo real. Almacena el resultado de la expresión. Observe que aun siendo A y B de tipo enteros, R es de tipo real, esto es debido a que no tenemos la seguridad de que la expresión ((A+B)^2/3 será entera, lo mas probable es que será de tipo real, previniendo eso declaramos R de tipo real.

En la codificación en C++ podemos notar que mostramos a través de cout un mensaje al usuario pidiéndole que ingrese una valor y luego lee dicho valor, esta es la forma en que se debe programar, de tal manera que nuestro programa es mas amigable con el usuario; los ejemplos siguientes tendrán esta forma de programación.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main(){
int A, B;
float R;
cout<<"Ingrese el valor de
      A:";cin>>A;
cout<<"Ingrese el valor de
      B:";cin>>B;
R=(pow((A+B),2))/3;
cout<<"El resultado es:
"<<R;}
```

La función pow(N,P) es usada en C++ para elevar un número o expresión numérica N a la potencia P

Ejemplo 1.4

Dada el código de matrícula y 5 calificaciones de un alumno obtenidas a lo largo del semestre; construya un pseudocódigo imprima la matrícula del alumno y el promedio de sus calificaciones.

Solución:

Declarar Variables

CODIGO : Entero largo

C1, C2, C3, C4, C5, PRO : Real

Ingresar el código de matrícula y sus calificaciones.

Calcular el promedio de las calificaciones.

Reportar la matrícula y el promedio obtenido.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
      CODIGO : Entero largo
      C1, C2, C3, C4, C5, PRO : Real
3. LEER Mat, C1, C2, C3, C4, C5
4. Calcular PRO = (C1 + C2 + C3 + C4 + C5)/5
4. ESCRIBIR MAT, PRO
5. Terminar el proceso
```

CODIGO es una variable de tipo entero, que representa la matrícula del alumno.

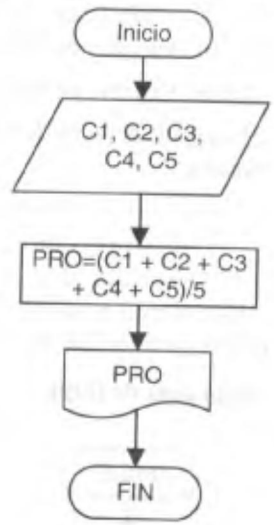
PRO: Variable de tipo real. Almacena el promedio de las calificaciones del alumno.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
```



```
#include<math.h>

void main()
{
int long CODIGO;
float C1, C2, C3, C4, C5, PRO;
clrscr();
cout<<"Ingrese código del alumno:";
cin>>CODIGO;
cout<<"Ingrese las 5 notas:";
cin>>C1>>C2>>C3>>C4>>C5;
PRO=(C1 + C2 + C3 + C4 + C5)/5;
cout<<"El alumno: "<<CODIGO;
cout<<" Tiene de promedio: "<<PRO;
getch();
}
```



Ejemplo 1.5

Escriba un pseudocódigo que permita calcular e imprimir el cuadrado y el cubo de un número entero positivo NUM.

Solución:

Declarar Variables

NUM : Entero

Ingresar el número.

Calcular el cuadrado y cubo del número ingresado.

Mostrar los resultados obtenidos.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
      NUM : Entero
      CUA, CUB : Real
3. LEER NUM
4. Calcular CUA = NUM ^ 2
5. Calcular CUB = NUM ^ 3
6. ESCRIBIR CUA, CUB
```

```
7. Terminar el proceso
```

CUA: Variable de tipo real. Almacena el cuadrado del número que se ingresa.

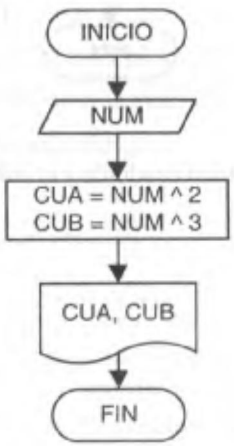
CUB: Variable de tipo real. Almacena el cubo del número que se ingresa.

El cálculo del cubo de NUM, también lo hubiéramos podido realizar de la siguiente manera:

```
Calcular CUB = NUM * NUM * NUM
Calcular CUB = (NUM ^ 2) * NUM
Calcular CUB = CUA * NUM
```

Este último es válido ya que anteriormente en la línea 4, CUA ya tiene el valor de la expresión NUM^2.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
int NUM;
float CUA, CUB;
cout<<"Ingrese número: ";cin>>NUM;
CUA = pow(NUM,2);
CUB = pow(NUM,3);
cout<<"El cuadrado es: "<<CUA;
cout<<"El cubo es: "<<CUB;
getch();
}
```

Otra vez usamos la función pow() para hallar tanto el cuadrado como el cubo del número. Es importante notar que la librería Math.h es usada para poder utilizar la función pow().

Ejemplo 1.6

Construya un pseudocódigo, que dados como datos la base y la altura de un rectángulo, calcule el perímetro y la superficie del mismo.

Consideraciones:

❖La superficie de un rectángulo se calcula aplicando la siguiente fórmula:

Superficie = base * altura

❖El perímetro se calcula como:

Perímetro = 2 * (base + altura)

Solución:

Declarar Variables B, A : Reales

Ingresar base y altura (B y A)

Calcular la superficie y perímetro del rectángulo.

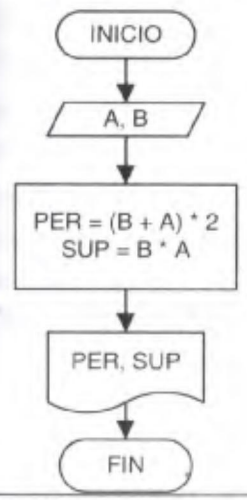
Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
      B, A     : Real
      PER, SUP : Real
3. LEER B, A
4. Calcular PER = (B + A) * 2
5. Calcular SUP = B * A
6. ESCRIBIR PER, SUP
7. Terminar el proceso
```

SUP: Variable de tipo real. Almacena la superficie del rectángulo

PER: Variable de tipo real. Almacena el perímetro del rectángulo.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main(){
float B, A, PER, SUP;
clrscr();
cout<<"Ingrese altura del
rectángulo: ";
cin>>A;
cout<<"Ingrese base del
rectángulo: ";
cin>>B;
PER = (B + A) * 2;
SUP = B * A;
```

```
cout<<"El perímetro es: "<<PER;
cout<<"La superficie es: "<<SUP;
getch();}
```

Ejemplo 1.7

Construya un pseudocódigo que dado el radio, calcule e imprima el área de una circunferencia.

Consideraciones:

La superficie de una circunferencia se calcula aplicando la siguiente fórmula:

$$Area = pi * radio^2$$

Solución

Declarar Variables

R : Real

Declarar constate PI = 3.1416

Ingresar radio

Calcular la superficie de la circunferencia.

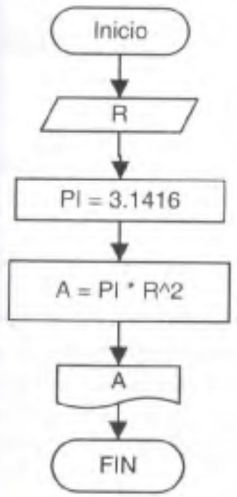
Mostrar resultados obtenidos.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
        R,A : Real
3. Declarar constante
        PI : Real
4. Asignar PI = 3.1416
5. LEER R
6. Calcular A = PI * R^2
7. ESCRIBIR A
8. Terminar el proceso
```

Nótese que estamos declarando una contante, para ello el valor de PI no podrá cambiar en la ejecución del programa.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
float R,A,PI;
PI=3.1416;
clrscr();
cout<<"Ingrese radio: ";
cin>>R;
A = PI * pow(R,2);
cout<<"El área es: "<<A;
getch();
}
```

Problema 1.8

Escriba un pseudocódigo, que dado el nombre de un dinosaurio, su peso y su longitud, expresados estos dos últimos en libras y pies, respectivamente; escriba el nombre del dinosaurio, su peso expresado en kilogramos y su longitud expresada en metros.

Consideraciones:

- Para convertir de libras a kilogramos, multiplica por 0.4535924
- Para convertir de pies a metros, multiplicar por 0.3048006

Solución:

Declarar Variables: NOM, PES, LON

Ingresar nombre, peso y longitud

Transformar cantidades a kilogramos y metros respectivamente.

Mostrar resultados

Donde: NOM es una variable de tipo cadena de caracteres, que expresa el nombre el dinosaurio.

PES una variable de tipo real, que expresa el peso del dinosaurio en libras.

LON es una variable de tipo real, que expresa la longitud del dinosaurio en pies.

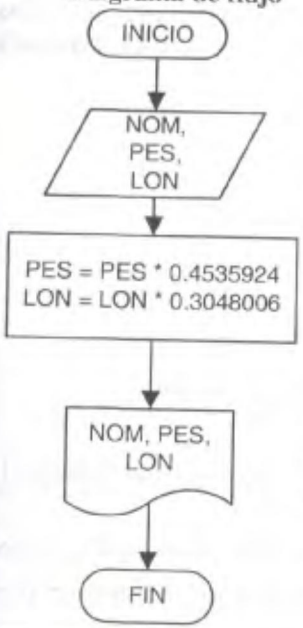
Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
        NOM : Cadena de caracteres
        PES, LON : Real
3. LEER NOM, PES, LON
4. Calcular PES = PES * 0.4535924
5. Calcular LON = LON * 0.3048006
6. ESCRIBIR NOM, PES, LON
7. Terminar el proceso
```

Nótese que no hemos declarado variables de almacenamiento para los resultados, por lo que el valor que se ingreso de la variable PES será reemplazado con el de la expresión PES * 0.4535924, lo mismo sucede con la variable LON, por ejemplo: sea el valor ingresado de PES = 500 luego de ejecutar la línea 4 el nuevo valor de PES será 500*0.4535924 ósea 226.7962, de manera análoga si LON = 250 pies es igual a 76.20 metros.

Nótese aquí el uso de una variable de tipo cadena de caracter y también el uso de cin para poder leer cualquier tipo de variable indistintamente.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main(){
char NOM[50];
float PES, LON;
clrscr();
cout<<"Ingrese nombre dinosaurio: ";
cin>>NOM;
cout<<"Ingrese su peso en libras: ";
cin>>PES;
cout<<"Ingrese longitud en pies: ";
cin>>LON;
PES = PES * 0.4535924;
LON = LON * 0.3048006;
cout<<"El Dinosaurio es : "<<NOM;
cout<<"su peso en KG es : "<<PES;
```



```
cout<<"Su longitud en metros es : "<<LON;
getch();}
```

Problema 1.9

En una Casa de Cambio necesitan construir un programa tal que dado como dato una cantidad expresada en dólares, convierta esa cantidad a nuevos soles.

Consideraciones:

❖Trabajaremos con el tipo de cambio establecido en : 1 dólar = 3.30 soles

Solución

Declarar Variables CANT : Real

Ingresar cantidad en soles

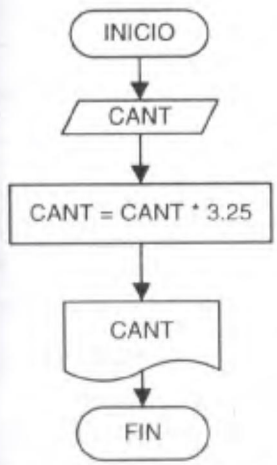
Realizar la conversión

Mostrar resultados

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
        CANT : Real
3. Calcular CANT = CANT * 3.25
4. ESCRIBIR CANT
5. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float CANT;
clrscr();
cout<<"Ingrese cantidad $: ";
cin>>CANT;
CANT = CANT * 3.25;
cout<<"Cambio en Soles : "<<CANT;
getch();
}
```

Problema 1.10

Construya un algoritmo, que dado el radio y la altura de un cilindro, calcule e imprima el área y su volumen.

Consideraciones:

❖El volumen de un cilindro lo calculamos aplicando la siguiente fórmula:

Volumen = pi * radio^2 * altura

❖La superficie del cilindro lo calculamos como:

Area = 2 * pi * radio * altura

Solución:

Declarar Variables

RADIO, ALTU: Real

Leer RADIO y ALTU

Realizar los cálculos del área y volumen

Escribir resultados

Pseudocódigo

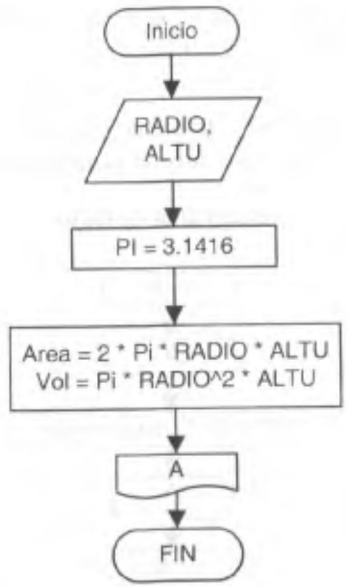
```
1. Iniciar proceso
2. Declarar variables
        RADIO, ALTU, Area, Vol : Real
3. Declarar constante
        PI = 3.1416
4. LEER RADIO,ALTU
5. Calcular Area = 2 * Pi * RADIO * ALTU
6. Calcular Vol = Pi * RADIO^2 * ALTU
7. ESCRIBIR Area, Vol
7. Terminar el proceso
```

Vol, almacena el volumen del cilindro.

Area, almacena el área del cilindro.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#define PI 3.1416
void main()
{
float RADIO, ALTU, Area, Vol;
clrscr();
cout<<"Ingrese Radio: ";
cin>>RADIO;
cout<<"Ingrese Altura: ";
cin>>ALTU;
Area = 2 * PI * RADIO * ALTU;
Vol = PI * pow(RADIO,2) * ALTU;
cout<<"El área es: "<<Area;
cout<<"El volumen es: "<<Vol;
getch();
}
```



Nótese en la codificación en C++ la línea #define PI 3.1416, en donde se declara una constante llamada PI con valor 3.1416

Problema 1.11

Una persona compró una estancia en un país sudamericano. La extensión de la estancia está especificada en acres. Construya un algoritmo, tal que dado como dato la extensión del campo en "acres", calcule e imprima la extensión del mismo en hectáreas.

Consideraciones:

❖1 acre es igual a 4 047 m^2 ❖1 hectárea tiene 10 000 m^2

Solución:

Declarar Variables

EXT: Real

Leer EXT

Realizar la conversión

Escribir EXT

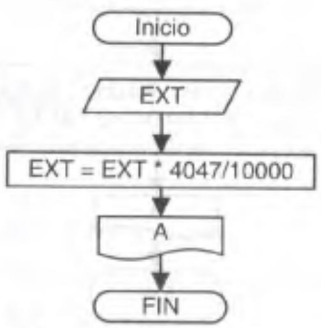
Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables  EXT : Real
```

```
3. Calcular EXT = EXT * 4047/10000
4. ESCRIBIR EXT
5. Terminar el proceso
```

EXT, almacena en primer lugar en dato introducido en acres, luego de la línea 3 esta misma variable almacena en dato en hectáreas.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float EXT;
clrscr();
cout<<"Acres: ";cin>>EXT;
EXT = EXT * 4047/10000;
cout<<"Hectareas: "<<EXT;
getch();}
```

Problema 1.12

En las olimpiadas de invierno el tiempo que realizan los participantes en la competencia de velocidad en pista, se mide en minutos, segundos y centésimas. La distancia que recorren, por otra parte, se expresa en metros.

Construya un pseudocódigo que calcule la velocidad de los participantes, en kilómetros por hora, de las diferentes competencias.

Consideraciones:

- El tiempo debemos expresarlo en segundos, por lo que para hacerlo aplicaremos la siguiente fórmula:

  TIEMSEG = Minutos*60 + Segundos + Centesimas/100

- Luego podemos calcular la velocidad, expresada en metros sobre segundos:

$$\text{VELOMS} = \frac{\text{Distancia (metros)}}{\text{TIEMSEG (Segundos)}}$$

- Para obtener la velocidad en kilómetros por hora, aplicamos la siguiente fórmula:

$$\text{VELOKH} = \text{VELOMS} * \frac{3600 \quad \text{(Kilometraje)}}{1000 \quad \text{(Hora)}} = \text{VELOMS} * 3.6 \text{ KM/H}$$

Solución:

Declarar Variables

MIN, SEG, CEN : Entero

DIST : Real

Leer MIN, SEG, GEN, DIST

Realizar los cálculos

Escribir resultados

En donde:

- MIN expresa el número de minutos que empleó el o la participante para realizar el recorrido.
- SEG expresa el número de segundos que utilizó el o la participante para realizar el recorrido.
- CEN expresa las centésimas que empleó el participante para realizar el recorrido.
- DIST es una variable de tipo real, que expresa la distancia del recorrido en metros.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
                MIN, SEG, CEN : Entero
                DIST, TIEMSEG, VELOKH : Real
3. Leer MIN, SEG, CEN, DIST
4. Calcular VELOKH = (DIST/(MIN*60+SEG+CEN/100))*3.6
5. ESCRIBIR 'Distancia:', DIST/1000, 'km'
6. ESCRIBIR 'Velocidad:', VELOKH,'(K/H)'
7. Terminar el proceso
```

TIEMSEG, expresa el tiempo de los participantes en segundos.

VELOKH, expresa la velocidad de los participantes en kilómetros por hora.

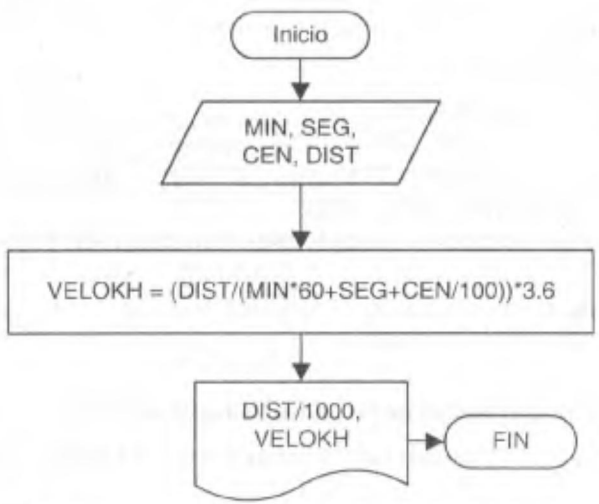
Analicemos la línea 6

```
5. ESCRIBIR 'Distancia:', DIST/1000, 'km'
```

Suponiendo que DIST sea igual a 2000 m, al momento de dar el resultado también transformamos los metros a kilómetros mediante DIST/1000, el resultado de la línea 6 sería: Distancia: 2 Km

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int MIN, SEG, CEN;
float DIST, TIEMSEG, VELOKH;
clrscr();
cout<<"Min: ";cin>>MIN;
cout<<"SEG: ";cin>>SEG;
cout<<"CEN: ";cin>>CEN;
cout<<"Distancia: ";cin>>DIST;
VELOKH = (DIST/(MIN*60+SEG+CEN/100))*3.6;
cout<<"Distancia: "<<DIST/1000<<" km";
cout<<"Velocidad: "<<VELOKH<<" (K/H)";
getch();
}
```



Problema 1.13

Construya un algoritmo, que dados los tres lados de un triángulo, pueda determinar su área. Esta la calculamos aplicando la siguiente fórmula:

$$\text{Area} = \sqrt{S * (S - L1) * (S - L2) * (S - L3)}$$

En donde S es: $S = (L1 + L2 + L3) / 2$

Consideraciones

- Recuerde que es lo mismo sacar la raíz cuadrada de un número que elevar dicho número a la potencia 0.5

Solución

Declarar Variables

L1, L2, L3,S,Area : Real

Leer L1, L2, L3

Realizar el cálculo del área

Escribir resultado

L1, L2, L3, son variables de tipo real, que representan los lados de un triángulo.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
        L1, L2, L3, S, Area : Real
3. Calcular S = (L1 + L2 + L3) / 2
4. Calcular Area = (S*(S-L1)*(S-L2)*(S-L3))^0.5
5. ESCRIBIR Area
7. Terminar el proceso
```

S se utiliza como una variable auxiliar para el cálculo del área.

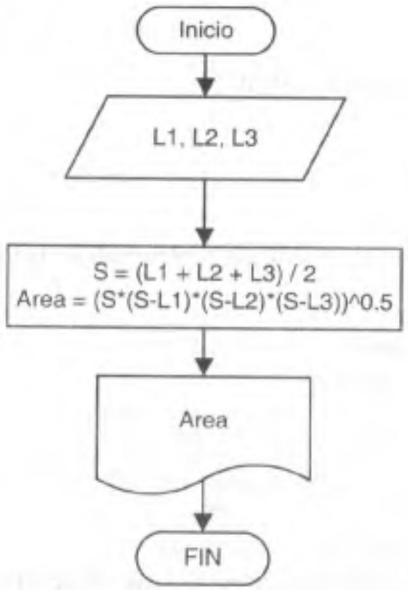
AREA almacena el área del triángulo.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
float L1, L2, L3, S, Area;
```

```
clrscr();
cout<<"Lado 1: ";cin>>L1;
cout<<"Lado 2: ";cin>>L2;
cout<<"Lado 3: ";cin>>L3;
S = (L1 + L2 + L3) / 2;
Area = pow(S*(S-L1)*(S-L2)*(S-L3),0.5);
cout<<"El área es: "<<Area;
getch();
}
```

Diagrama de flujo



El cálculo del área también hubiera podido ser con la siguiente fórmula

```
sqrt(S*(S-L1)*(S-L2)*(S-3))
```

Problema 1.14

Construya un algoritmo que sea capaz de intercambiar el valor de tres variables, de tal manera que sean las variables A, B, C, el valor de B se almacene en A, B obtenga el valor de C y C el valor de A.

Solución

Este ejercicio es muy intersante, analicemos, con valores reales, por ejemplo:

A= 2; B= 5; C=8

veamos como resultaría el intercambio de variables

1) Si A = B tendríamos A=5

2) Si B= C tendríamos B= 8

3) Si C=A tendríamos C= 2

Revisando esta solución concluimos que las líneas 1 y 2 con correctas pero la tercera no, esto es debido que al haberse ejecutado la línea 1 el valor de A es 5, y al llegar a la línea 3 el valor correcto de C seria 5 y no 2 como lo planteamos. El valor anterior de A ya no existe después de la ejecución de la línea 1, entonces ¿como hacemos para resolver esto?, pues es simple, debemos usar una variable auxiliar en donde podamos almacenar el primer valor de A, de esa manera no perderemos este dato, por lo tanto siguiendo con el ejemplo tendríamos lo siguiente

1) AUX = A tendríamos AUX=2 ,aquí almacenamos el valor de A

2) Si A = B tendríamos A=5

3) Si B= C tendríamos B= 8

4) Si C=AUX tendríamos C= 2

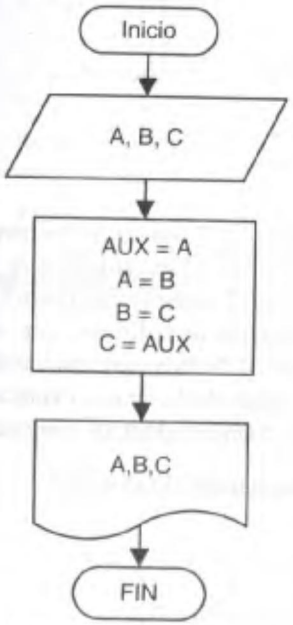
De esta manera los valores intercambiados de las variables serían A=5, B=8, C=2.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
        A, B, C, AUX : Entero
3. Calcular AUX = A
4. Calcular A=B
5. Calcular B=C
6. Calcular C=AUX
7. ESCRIBIR A, B, C
8. Terminar el proceso
```

Podemos ver claramente el uso de la variable AUX, la cual nos permite almacenar el valor de A, por lo tanto AUX debe ser del mismo tipo que las otras variables.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int A, B, C, AUX;
clrscr();
cout<<"Ingrese A: ";cin>>A;
cout<<"Ingrese B: ";cin>>B;
cout<<"Ingrese C: ";cin>>C;
AUX = A;
A = B;
B = C;
C = AUX;
cout<<"El valor de A es: "<<A;
cout<<"El valor de B es: "<<B;
cout<<"El valor de C es: "<<C;
getch();
}
```

Problema 1.15

Reforcemos en ejercicio anterior construyendo un algoritmo que sea capaz de intercambiar el valor de cinco variables, de tal manera que sean las variables A, B, C, D, E. El valor de C se almacene en A, D obtenga el valor de B, B el valor de E, E sea igual a A y C posea el valor de D.

Solución

Este ejercicio es similar al anterior, solo que con dos variables más, la forma de resolver es muy similar, la única variantes que debemos usar dos variables auxiliares, veamos el siguiente pseudocódigo.

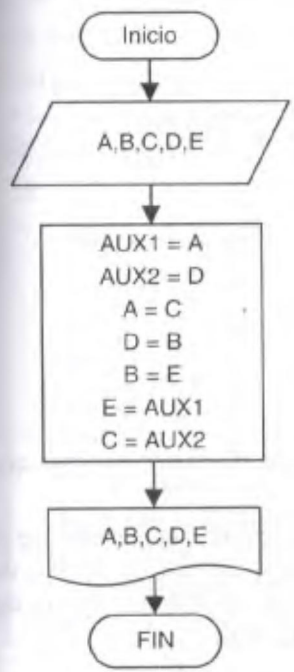
Pseudocódigo

```
1. Iniciar proceso
2. Declarar variables
        A, B, C, D, E, AUX1, AUX2 : Entero
3. Calcular AUX1 = A
4. Calcular AUX2 = D
5. Calcular A = C
```



```
6. Calcular D = B
7. Calcular B = E
8. Calcular E = AUX1
9. Calcular C = AUX2
10. ESCRIBIR A, B, C, D, E
11. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int A, B, C,D, E, AUX1, AUX2;
clrscr();
cout<<"Ingrese A: ";cin>>A;
cout<<"Ingrese B: ";cin>>B;
cout<<"Ingrese C: ";cin>>C;
cout<<"Ingrese D: ";cin>>D;
cout<<"Ingrese E: ";cin>>E;
AUX1 = A;
AUX2 = D;
A = C;
D = B;
B = E;
E = AUX1;
C = AUX2;
cout<<"El valor de A es: "<<A;
cout<<"El valor de B es: "<<B;
cout<<"El valor de C es: "<<C;
cout<<"El valor de D es: "<<D;
cout<<"El valor de E es: "<<E;
getch();}
```

Problema 1.16

Construya un pseudocódigo que ingresado un número entero de 3 cifras, se obtenga como resultado el número ingresado y el inverso de dicho número.

Solución

Este es un ejemplo interesante, podríamos resolver leyendo el número digito

por dígito y luego simplemente imprimirlo invertido, pero no es una buena solución. Para resolver este ejercicio necesitaremos de las funciones DIV y MOD. El algoritmo seria el siguiente:

Declarar Variables

Invertir el número

Escribir el número normal y el número invertido

Bueno es un algoritmo muy sencillo, veamos su implementación en Pseudocódigo.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        NUM, NUMINV, U, D, C, AUX : Entero
3. Calcular AUX = NUM
4. Calcular C = NUM DIV 100
5. Calcular NUM = NUM MOD 100
6. Calcular D = NUM DIV 10
7. Calcular U = NUM MOD 10
8. Calcular NUMINV = U*100 + D*10 + C
9. ESCRIBIR AUX, NUMINV
10. Terminar el proceso
```

Las variables U, D, C, están referidos a las unidades, decenas y centenas que tiene un número de tres cifras, respectivamente.

Veamos como funciona este pseudocódigo con un ejemplo, suponiendo que el número ingresado es NUM=734, por ahora no vamos a entrar en detalles de consistenciar si el valor leido es o no de tres cifras, por lo que asumiremos que así es. Entonces al ejecutar el algoritmo tendríamos lo siguiente:

El primer paso es guardar el valor del número ingresado

3. AUX = NUM ==> AUX = 734

Luego ejecutamos la línea 4 para poder extraer el primer valor, recordemos que el operador DIV realiza una división entera.

4. C = NUM DIV 100 = 734 DIV 100 = 7 ==> C = 7

Una vez extraido el primer valor es necesario eliminarlo del número para poder seguir con los otros, esto lo logramos con el uso del operador MOD, el cual nos devuelve el residuo de una división entera.

5. NUM = NUM MOD 100 = 734 MOD 100 = 34 ==> NUM = 34

El nuevo valor de NUM ahora es 34, de aqui ya es mas fácil usamos otra vez DIV y MOD para obtener lo números que nos faltan

6. D = NUM DIV 10 = 34 DIV 10 = 3 ==> D = 3

7. U = NUM MOD 10 = 34 DIV 10 = 4 ==> U = 4

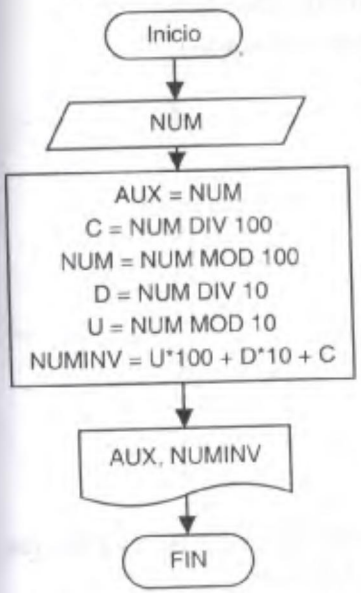
Ya tenemos el número descompuesto, ahora solo nos queda realizar una operación sencilla para unirlo, solo que el resultado ahora será el número original 734 invertido, ósea debemos obtener 437.

8. NUMINV = U*100 + D*10 + C = 4*100 + 3*10 + 7 ==> NUMINV = 437

El ejercicio nos pide, devolver el número original y el invertido, recordemos que el número original fue almacenado en la variable AUX, ya que NUM, después de del proceso, terminó con el valor de 34.

9. ESCRIBIR AUX, NUMINV

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int NUM, NUMINV, U, D, C, AUX;
clrscr();
cout<<"Ingrese número: ";
cin>>NUM;
AUX = NUM;
C = NUM/100;
NUM = NUM%100;
D = NUM/10;
U = NUM%10;
NUMINV = U*100 + D*10 + C;
cout<<"El numero es : "<<AUX;
cout<<"El invertido: "<<NUMINV;
getch();}
```

Nótese en C++ el uso del operador / en vez de Div, esto es posible solo si los números son de tipo entero, decido a eso podemos obtener la parte entera de la división de ambos números. Por otro lado el operador MOD es reemplazado por

% pudiendo obtener a través de este el residuo de la división.

Problema 1.17

Escriba un algoritmo que calcule el número mínimo de billetes 20, 10, 5, 1 dólares, que se necesita para cambiar un cheque. Considere que el valor del cheque es un número entero.

Solución

Este ejemplo es muy sencillo, el uso de los operadores DIV y MOD nos ayudará en la resolución del problema, el cual consiste en cambiar un cheque, en el número óptimo de billetes de cuatro denominaciones distintas (20, 10, 5, 1), este es un ejemplo típico en algoritmia y programación para principiantes, su algoritmo es el siguiente:

LEER Importe del cheque

Calcular número máximo de billetes en denominaciones de 20.

Calcular número máximo de billetes en denominaciones de 10.

Calcular número máximo de billetes en denominaciones de 5.

Calcular número máximo de billetes en denominaciones de 1.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          CANT, B20, B10, B5, B1 : Entero
3. Calcular B20 = CANT DIV 20
4. Calcular CANT = CANT MOD 20
5. Calcular B10 = CANT DIV 10
6. Calcular CANT = CANT MOD 10
7. Calcular B5 = CANT DIV 5
8. Calcular B1 = CANT MOD 5
9. ESCRIBIR B20, B10, B5, B1
10. Terminar el proceso
```

B20, B10, B5, B1, representan la cantidad de billetes de 20, 10, 5 y 1 dólares, respectivamente, en los cuales será cambiado el cheque.

Veamos como se ejecuta este pseudocódigo, suponiendo que el valor del cheque es 4152, tenemos lo siguiente:

3. B20 = CANT DIV 20 = 4152 DIV 20 ==> B20 = 207

4. CANT = CANT MOD 20 = 4152 DIV 20 ==> CANT = 12
5. B10 = CANT DIV 10 = 12 DIV 10 ==> B10 = 1
6. CANT = CANT MOD 10 = 12 MOD 10 ==> 2
7. B5 = CANT DIV 5 = 2 DIV 5 ==> B5 = 0
8. B1 = CANT MOD 5 = 2 MOD 5 ==> B1 = 2

Por lo tanto el cheque de 4152 dólares se cambiaría de la siguiente manera:

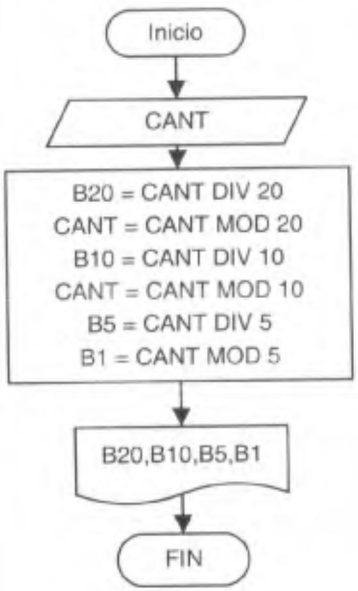
207 billetes de 20 dólares

1 billete de 10 dólares

0 billetes de 5 dólares

2 billetes de 1 dolar

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int CANT, B20, B10, B5, B1;
clrscr();
cout<<"Ingrese cantidad: ";
cin>>CANT;
B20 = CANT/20;
CANT = CANT%20;
B10 = CANT/10;
CANT = CANT%10;
B5 = CANT/5;
B1 = CANT%5;
cout<<"Billetes de 20: "<<B20;
cout<<"Billetes de 10: "<<B10;
cout<<"Billetes de 5: "<<B5;
cout<<"Billetes de 1: "<<B1;
getch();}
```

Problema 1.18

Construya un pseudocódigo, que dado el radio, la generatriz y la altura de un cono; calcule e imprima el área de la base, el área lateral, el área total y su

volumen.

Consideraciones:

❖Un cono tiene la siguiente forma:

❖El área de la base se calcula con base en la siguiente fórmula:

$$AB = Pi * RADIO^2$$

❖El área lateral se calcula:

$$AL = Pi * RADIO * GENE$$

❖El área total se calcula como:

$$AT = AB + AL$$

❖El volumen se calcula de esta forma:

$$VOL = \frac{1}{3} * AB * ALTU$$

Solución

Declarar Variables

Declarar constante

Leer RADIO, ALTU, GENE

Calcular AB = Pi * (RADIO^2),

AL = Pi * RADIO * GENE

AT = AB + AL

VOL = 1/3 * AB * ALTU

Escribir resultados

AB: Variable de tipo real. Almacena el área de la base.

AL: Variable de tipo real. Almacena el área lateral del cono.

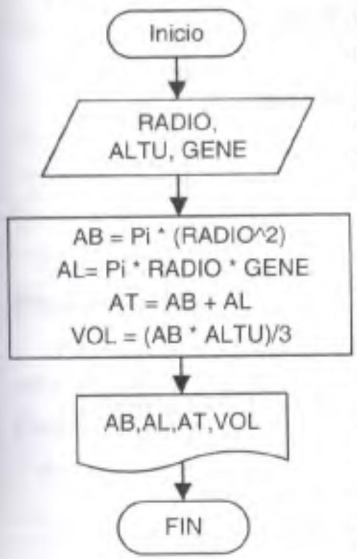
AT: Variable de tipo real. Almacena el área total del cono.

VOL: Variable de tipo real. Almacena el volumen.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        RADIO, ALTU, GENE, : Real
        AB, AL, AT, VOL    : Real
3. Declarar constante Pi = 3.141592 : Real
4. Leer RADIO, ALTU, GENE
5. Calcular AB = Pi * (RADIO^2)
           AL = Pi * RADIO * GENE
           AT = AB + AL
           VOL = (AB * ALTU)/3
6. ESCRIBIR 'Area de la base:', AB,
           'Area lateral:', AL,
           'Area total:', AT,
           'Volumen:', VOL
7. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#define PI 3.141592
void main()
{
float RADIO, ALTU, GENE, AB, AL,
AT, VOL;
clrscr();
cout<<"Ingrese radio: ";
cin>>RADIO;
cout<<"Ingrese Generatriz: ";
cin>>GENE;
cout<<"Ingrese Altura: ";
cin>>ALTU;
AB = PI * pow(RADIO,2);
AL = PI * RADIO * GENE;
AT = AB + AL;
```

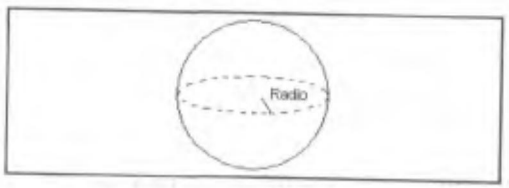
```
VOL =  (AB * ALTU)/3;
cout<<"Area de la base: "<<AB;
cout<<"Area lateral: "<<AL;
cout<<"Area total: "<<AT;
cout<<"Volumen: "<<VOL;
getch();
}
```

Problema 1.19

Construya un pseudocódigo, que dado el radio de una esfera, calcule e imprima el área y su volumen.

Consideraciones:

❖Una esfera tiene la siguiente forma:



❖El área de una esfera la calculamos de esta forma:

$$AREA = 4 * Pi * RADIO^2$$

❖El volumen de una esfera lo calculamos de esta forma:

$$VOL = \frac{4}{3} * Pi * RADIO^3$$

Solución

Declarar Variables RADIO : Real

Leer RADIO

Hacer AREA = 4 * Pi * (RADIO^2) y VOL = 4/3 * Pi * (RADIO^3)

Escribir Resultados

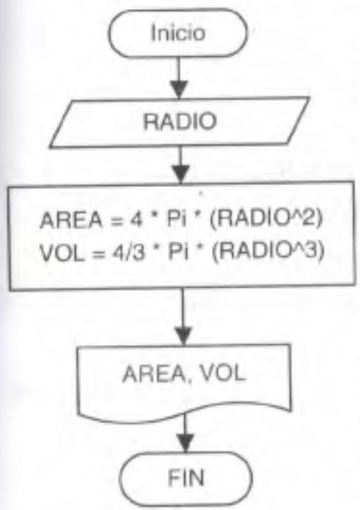
AREA: Variable de tipo real. Almacena el área de la esfera.

VOL: Variable de tipo real. Almacena el volumen de la esfera.

Pseudocódigo


```
1. Iniciar proceso
2. Declarar Variables
        RADIO, AREA, VOL : Real
3. Declarar constante Pi = 3.141592 : Real
4. Leer RADIO
5. Calcular AREA = 4 * Pi * (RADIO^2)
6. Calcular VOL = 4/3 * Pi * (RADIO^3)
7. ESCRIBIR 'Area de la esfera', AREA,
           'Volumen de la esfera:', VOL
8. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

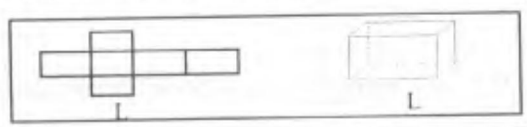
```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#define PI 3.141592
void main()
{
float RADIO, AREA, VOL;
clrscr();
cout<<"Ingrese radio: ";
cin>>RADIO;
AREA = 4 * PI * pow(RADIO,2);
VOL = 4/3*PI * pow(RADIO,3);
cout<<"Area de esfera: "<<AREA;
cout<<"Volumen: "<<VOL;
getch();
}
```

Problema 1.20

Construya un pseudocódigo, que dado como dato el lado de un hexaedro o cubo; calcule el área de la base, el área lateral, el área total y el volumen.

Consideraciones:

❖Un hexaedro o cubo tiene la siguiente forma:



❖Para calcular el área de la base aplicamos la siguiente fórmula:

$$AB = L^2$$

❖Para calcular el área lateral hacemos:

$$AL = 4 * L^2$$

❖Para calcular el área total hacemos

$$AT = 6 * L^2$$

❖Para calcular el volumen hacemos:

$$V = L^3$$

Solución

Declarar Variables

Leer L

Hacer AB = L^2, AL = 4 * (L^2), AT = 6 * (L^2) y V = L^3

Escribir Resultados

AB : Variable de tipo real. Almacena el área de la base del hexaedro.

AL : Variable de tipo real. Almacena el área lateral.

AT : Variable de tipo real. Almacena el área total.

V : Variable de tipo real. Almacena el volumen del hexaedro.

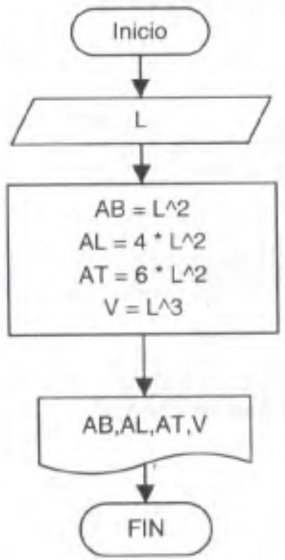
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        L, AB, AL, AT, V : Real
3. Leer L
4. Calcular AB = L^2
            AL = 4 * L^2
            AT = 6 * L^2
            V = L^3
5. ESCRIBIR 'Area de la base:', AB,
            'Area lateral:', AL,
            'Area total:', AT,
            'Volumen:', V
6. Terminar el proceso
```

Nótese que en la linea 4, agrupamos las acciones de cálculo, este forma para el

pseudocódigo y como más se usa, de ahora en adelante agruparemos las acciones de cálculos, reportes de datos, entre otros.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
float L, AB, AL, AT, V;
clrscr();
cout<<"Ingrese Lado:
";cin>>L;
AB= pow(L,2);
AL= 4 * pow(L,2);
AT= 6 * pow(L,2);
V = pow(L,3);
cout<<"Area base: "<<AB;
cout<<"Area lateral: "<<AL;
cout<<"Area total: "<<AT;
cout<<"Volumen: "<<V;
getch();
}
```

Problema 1.21

Construya un pseudocódigo que calcule la distancia entre dos puntos dados P1 y P2.

Consideraciones:

❖Para calcular la distancia "D" entre dos puntos dados P1 y P2 aplicamos la siguiente fórmula:

$$D = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

Solución

Declarar Variable

Leer coordenadas

Realizar el cálculo de la distancia

Mostrar resultados

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables X1, Y1, X2, Y2, Dis : Real
3. Leer X1, Y1, X2, Y2
3. Calcular D =((X1 - X2)^2 + (Y1 - Y2)^2)^0.5
4. ESCRIBIR D
5. Terminar el proceso
```

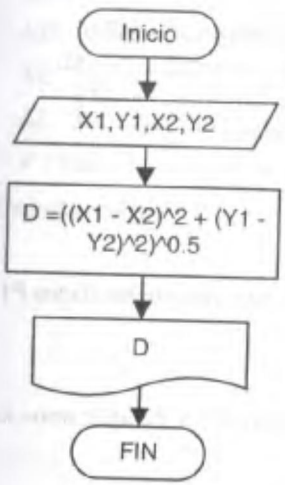
Donde:

X1 y Y1 expresan las coordenadas del punto P1 en el eje de las XY.

X2 y Y2 expresan las coordenadas del punto P2 en el eje de las XY.

D almacena la distancia entre dos puntos P1 y P2

Diagrama de flujo



Codificación en C++

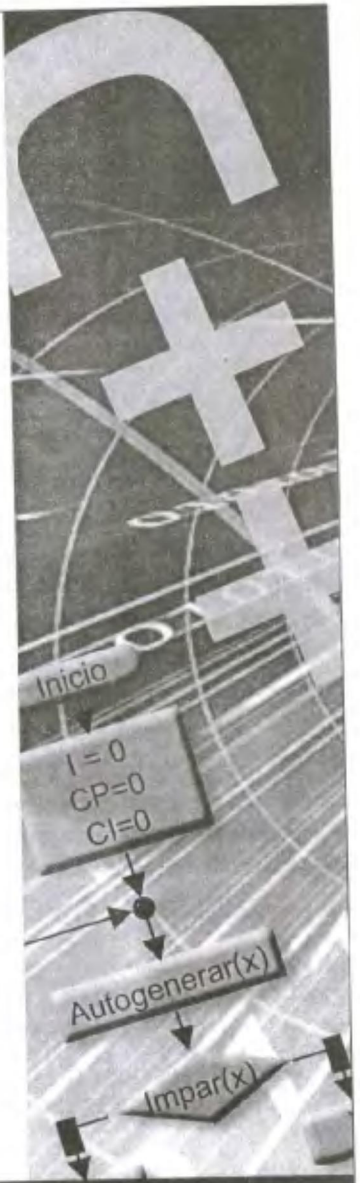
```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
float X1, Y1, X2, Y2, Dis;
clrscr();
cout<<"Ingrese primer punto (X1,
      Y1): ";cin>>X1>>Y1;
cout<<"Ingrese segundo punto (X2,
      Y2): ";cin>>X2>>Y2;
Dis=sqrt(pow((X1-X2),2) + pow((Y1-
Y2),2));
cout<<"Distancia: "<<Dis;
getch();
}
```


Capítulo II

Estructuras Lógicas Selectivas

Este capítulo contiene:

- Estructura Si...Entonces (Selección simple)
- Estructura Si...Entonces...Si no (Alternativa doble)
- Anidamiento de Estructuras condicionales
- La estructura de selección múltiple
- Ejemplos desarrollados



Algoritmos y Diagramas de Flujo aplicados en C++

e-book
www.FreeLibros.com

Introducción

Las estructuras lógicas selectivas se encuentran en la solución algoritmica de casi todo tipo de problemas. La utilizamos cuando en el desarrollo de la solución de un problema debemos ***tomar una decisión***, para establecer un proceso o señalar un camino alternativo a seguir.

Esta toma de decisión se basa en la evaluación de una o más condiciones que nos señalarán como alternativa o consecuencia, la rama a seguir.

Hay situaciones en las que la toma de decisiones se realiza en cascada. Es decir se toma una decisión, se marca la rama correspondiente a seguir, se vuelve a tomar otra decisión y así sucesivamente. Por lo que para alcanzar la solución de este problema o subproblema debemos aplicar prácticamente un árbol de decisión.

Las estructuras algoritmicas selectivas que se utilizan para la toma de decisiones lógicas las podemos clasificar de la siguiente forma:

1. SI...ENTONCES (Estructura selectiva simple)
2. SI...ENTONCES...SINO (Estructura selectiva doble)

Cabe señalar que cuando a la estructura selectiva las aplicamos en cascada, podemos utilizar una combinación de las estructuras señaladas anteriormente en la clasificación.

Estructura Si...Entonces (Selección simple)

La estructura selectiva ***Si...Entonces*** permite que el flujo del diagrama siga por

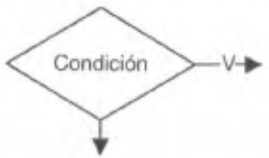
un camino específico si se cumple una condición o conjunto de condiciones. Si al evaluar la condición (o condiciones) el resultado es verdadero, entonces se ejecuta(n) cierta(s) operación(es). Luego se continúa con la secuencia normal del diagrama

La sintaxis para este tipo de estructura es la siguiente:

```
Si condición entonces
      Hacer operación
Fin_Si(Fin del condicional)
```

La *condición* es una expresión lógica. Y *operación* puede ser una operación simple o un grupo de operaciones.

El diagrama de flujo de la condicional Si simple es:



Si la condición es falsa no hace ninguna acción. Veamos unos ejemplos de una sentencia selectiva simple.

Ejemplo 2.1

Construya un pseudocódigo tal, que dado como dato la calificación de un alumno en un examen, escriba "Aprobado" en caso de que esa calificación fuese mayor que 10.5.

Solución

LEER nota

Comprobar si nota es mayor que 10.5

Escribir resultado en caso esté aprobado

Algoritmo

```
1. Iniciar proceso
2. Declarar Variables
          Nota : Real
3. LEER Nota
4. Si Nota > 10.5 Entonces
```

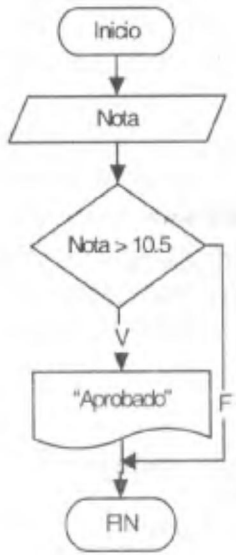


```
      4.1. Escribir "Aprobado"
5. Fin_Si
6. Terminar el proceso
```

En la línea 4 realizamos la comprobación de la condicional para este ejercicio, solo en el caso de ser verdadero, se ejecuta la línea 4.1, mostrando el mensaje "Aprobado", caso contrario se ejecuta la línea 5, por lo tanto el pseudocódigo no muestra ningún resultado, de ser falsa la condicional.

Nótese que para cada sentencia Si...Entonces, se debe colocar un fin de sentencia mediante Fin_Si

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float Nota;
clrscr();
cout<<"Ingrese nota: ";
cin>>Nota;
if (Nota > 10.5)
{
   cout<<"Aprobado";
}
getch();
}
```

Note q en la codificación en C++ la condicional Si es reemplazada por If luego viene la condición encerrada entre paréntesis; luego de eso vienen las acciones a realizar delimitadas por llaves las cuales indican el inicio y el fin del bloque de la condicional If. Para nuestro ejemplo solo se emite un mensaje de "Aprobado"

Por otro lado vemos claramente en el diagrama de flujo si la condición es falsa la secuencia del diagrama nos lleva a un punto en donde el flujo termina, para este caso en particular.

Ejemplo 2.2

Dado como dato el sueldo de un trabajador, apliquele un aumento del 17% si su

sueldo es inferior a $ 1000. Imprima en este caso, el nuevo sueldo del trabajador.

Solución

Leer Sueldo

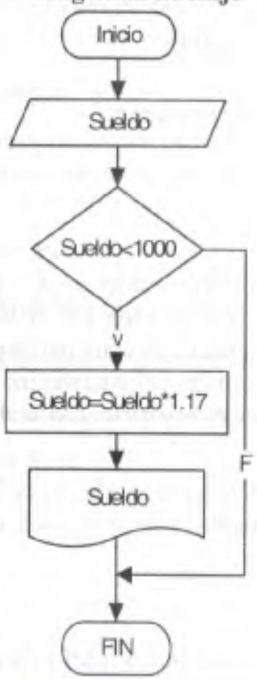
Comprobar si sueldo cumple con la condición

Si cumple la condición, realizar el aumento y reportar nuevo sueldo

Algoritmo

```
1. Iniciar proceso
2. Declarar Variables
        Sueldo : Real
3. LEER Sueldo
4. Si Sueldo < 1000 Entonces
     4.1. Sueldo = Sueldo *1.17
     4.2. Escribir Sueldo
5. Fin_Si
6. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float Sueldo;
clrscr();
cout<<"Ingrese sueldo:
";cin>>Sueldo;
if (Sueldo < 1000)
{
    Sueldo = Sueldo *1.17;
    cout<<"Nuevo sueldo:
"<<Sueldo;
}
getch();
}
```


Estructura Si...Entonces...Si no (Alternativa doble)

Como se puede ver la selección simple es limitada aunque muchas veces necesaria, por otro lado la alternativa doble nos permite tomar decisiones en ambos sentidos, es decir cuando la sentencia de comparación sea verdadero o cuando sea falso, en otras palabras cuando la respuesta de la comparación sea verdadera se ejecutarán una o más acciones, así mismo si la respuesta es falsa se ejecutarán acciones diferentes. Veamos el pseudocódigo el cual explica mejor el concepto de alternativa doble.

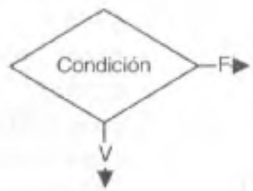
```
Si Condición Entonces
      Acciones1
 sino
      Acciones2
Fin_Si {Fin del condicional}
```

Donde:

Condición expresa la condición o conjunto de condiciones a evaluarse.

Acciones1 expresa la operación o conjunto de operaciones que se van a realizar si la condición resulta verdadera.

Acciones2 expresa la operación o conjunto de operaciones que se van a realizar si la condición resulta falsa.



Ejemplo 2.3

Construya un algoritmo, que dado como dato la calificación de un alumno en un examen, escriba "Aprobado" si su calificación es mayor que 10.5 y "Reprobado" en caso contrario.

Solución

Nótese que este ejercicio, es más completo que el Ejemplo 2.1. El algoritmo es el siguiente:

Leer Nota

Comprobar si la Nota es mayor a 10.5, reportar "Aprobado"

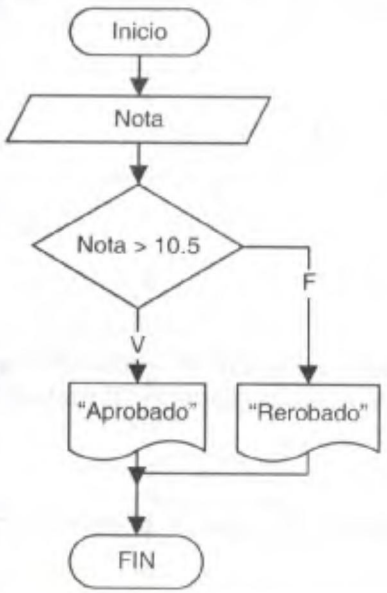
Si no, Reportar "Reprobado"

Algoritmo

```
1. Iniciar proceso
2. Declarar Variables
          Nota : Real
3. LEER Nota
4. Si Nota > 10.5 Entonces
     4.1. Escribir 'Aprobado'
5. Si no
     5.1. Escribir 'Reprobado'
6. Fin_Si
7. Terminar el proceso
```

En este ejemplo se asume que la nota es válida, entiéndase nota válida si está entre 0 y 20, incluso.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float Nota;
clrscr();
cout<<"Ingrese nota: ";
cin>>Nota;
if (Nota > 10.5)
{
    cout<<"Aprobado";
}
else
{
    cout<<"Reprobado";
}
getch();
}
```


En la codificación en C++ ahora tenemos la palabra "else" el reemplaza al "Si no" del algoritmo, por lo tanto se ejecutará else solo si la condición no se cumple.

Ejemplo 2.4

Construya un algoritmo, que dado como dato el sueldo de un trabajador, le aplique un aumento del 17% si su sueldo es inferior a S/. 1000 y 12% en caso contrario. Imprima el nuevo sueldo del trabajador.

Solución

Leer Sueldo

Comprobar si sueldo cumple con la condición

Si cumple la condición, incrementar el 17%

Si no, incrementar el 12 %

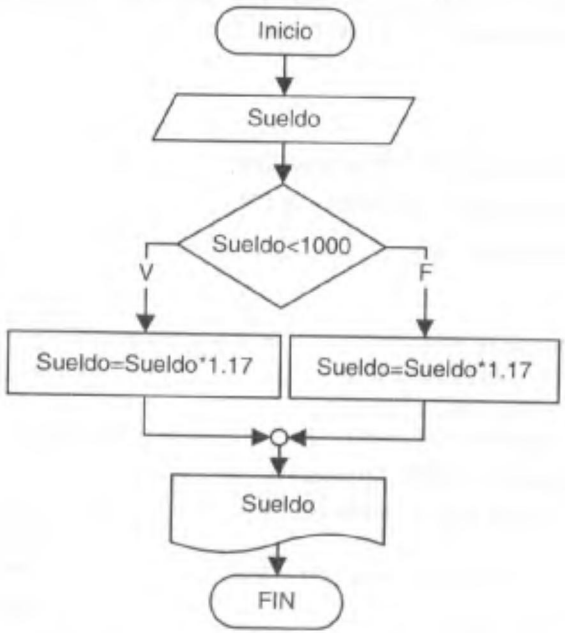
Algoritmo

```
1. Iniciar proceso
2. Declarar Variables
        Sueldo : Real
3. LEER Sueldo
4. Si Sueldo < 1000 Entonces
     4.1. Sueldo = Sueldo *1.17
5. Si no
     5.2. Sueldo = Sueldo *1.12
6. Fin_Si
7. Escribir Sueldo
8. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
float Sueldo;
clrscr();
cout<<"Ingrese sueldo: ";cin>>Sueldo;
if (Sueldo < 1000)
{   Sueldo = Sueldo *1.17;
} else
```

```
{   Sueldo = Sueldo *1.12;
}
cout<<"Nuevo sueldo: "<<Sueldo;
getch();
}
```



Nótese en el diagrama de flujo el uso de pequeño circulo en donde se encuentran los flujos de la condicional, este es un conector de tal manera que indica que sea verdad o falso la condición, luego de ejecutar las acciones correspondientes el flujo continua hasta el conector y luego su camino hacia Fin

Anidamiento de Estructuras condicionales

A menudo tendrá la necesidad de anidar una o más estructuras condicionales, ya sean simples o dobles, o combinaciones de ellas.

Se dice que las estructuras están anidadas, cuando hay unas dentro de ellas, esto lo veremos muy a menudo. Por ejemplo veamos el siguiente pseudocódigo

```
Si Condición1 Entonces
```



```
        Si Condición2 Entonces
            Acciones1
        Fin_si
    Fin_Si
```

Aquí observamos que primero se debe cumplir la Condición1, para luego evaluar la Condición2, solo al cumplirse esta última, se procederá a la ejecución de Acciones1.

```
Si Condición1 Entonces
    Acciones1
    Si Condición2 Entonces
        Acciones2
    Si no
        Acciones3
    Fin_si
Fin_Si
```

En este ejemplo, una vez cumplida la Condición1, realizamos la Acción1 y luego evaluamos la Condición2, dependiendo el resultado de la Condición2, se procederá a realizar Acciones2 o Acciones3.

Podemos tener muchas combinaciones de anidamiento, pero en el fondo todos se manejan de la misma manera. Veamos con unos ejemplos.

Ejemplo 2.5

Implemente la validación de la nota ingresada en el Ejemplo 2.3.

Solución

En el Ejemplo 2.3, mencionamos que se suponía una nota válida, en la práctica veremos que nada se supone, todo ingreso de dato debe estar correctamente validado. Para esto, las estructuras condicionales nos ayudan en esta tarea.

Leer nota

Si es nota válida

Comprobar si la Nota es mayor a 10.5, reportar "Aprobado"

Si no, Reportar "Reprobado"

En caso de no ser una nota válida, mandarle un mensaje al usuario.

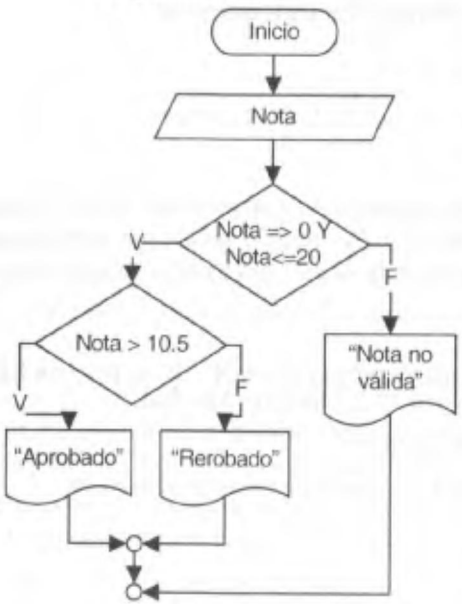
Pseudocódigo

```
1. Iniciar proceso
```

```
2. Declarar Variables
        Nota : Real
3. LEER Nota
4. Si Nota>=0 Y Nota<=20 Entonces
    4.1. Si Nota > 10.5 Entonces
        4.1.1. Escribir "Aprobado"
    4.2. Si no
        5.2.1. Escribir "Reprobado"
    4.3. Fin_Si
5. Si no
    5.1. Escribir "Nota no es válida"
6.Fin_Si
7. Terminar el proceso
```

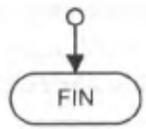
Una simple línea de código, puede hacer mucha diferencia. La línea 4 muestra el uso del operador Y, esto obliga a que tanto la condición Nota>=0 y la condición Nota<=20 sean verdaderas, para que la expresión "Nota>=0 Y Nota<=20" sea verdadera, solo así podemos decir que es una nota válida, en caso contrario, se reportará una nota no válida.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float Nota;
clrscr();
cout<<"Ingrese nota: ";
cin>>Nota;
if (Nota>=0 & Nota<=20)
{  if (Nota > 10.5)
   { cout<<"Aprobado";
   }else{
     cout<<"Reprobado";
   }
```


```
   }else{
     cout<<"Nota no válida";}
getch();}
```

Nótese que por motivo de espacio hemos tenido que usar el conector en el diagrama de flujo, esto se hace cuando los diagramas con muy grandes de esa forma sabemos por donde va el flujo del diagrama.

Ejemplo 2.6

Se requiere la implementación de un pseudocódigo que dado un número entero positivo mayor que cero de como resultado si dicho número es par o impar. El ejercicio requiere la validación del dato de entrada.

Solución

El problema especifica la validación del dato de entrada, eso quiere decir que si el número ingresado no es mayor a cero, se dará fin a la ejecución del programa o se reportará como número no válido (depende del programador). El algoritmo sería el siguiente;

Leer Dato

Comprobar dato de entrada, si es número entero positivo

Si dato es válido, realizar la comprobación para ver si es par

Si es Par, reportar par

Si no, reportar impar

Si el dato no es válido, fin del algoritmo

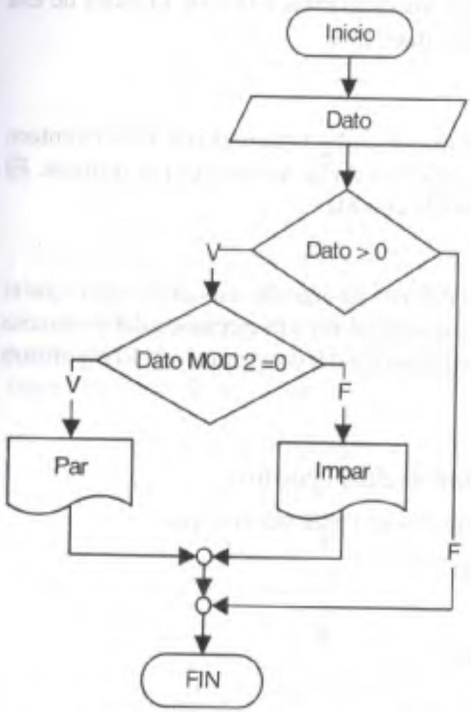
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        Dato : Entero
3. LEER Dato
4. Si Dato > 0 Entonces
    4.1. Si (Dato MOD 2)=0 Entonces
        4.1.1. Escribir 'Número ingresado es Par'
    4.2. Si no
        4.2.1. Escribir 'Número ingresado es Impar'
    4.3. Fin_Si
```

```
5. Fin_Si
6. Terminar el proceso
```

Un número es par cuando al dividirlo entre dos, es una división exacta, es decir no deja residuo, de allí que usamos el operador MOD.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int Dato;
clrscr();
cout<<"Ingrese
número:";
cin>>Dato;
if (Dato>0)
{
 if ((Dato % 2) == 0)
 {   cout<<"Par";
 }else{
     cout<<"Impar";
 }
}
getch();
}
```

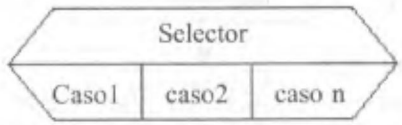
Note que en la comparación del operador Mod en la codificación C++ (%) usamos == debido a que esta es la sintaxis de comparación de igualdad en C++

La estructura de selección múltiple

La estructura de selección múltiple permite que el flujo del programa se bifurque por varias ramas en el punto de la toma de decisión(es), esto en función del valor que tome el selector. Así, si el selector toma el valor 1 se ejecutará la acción 1, si toma el valor 2 se ejecutará la acción 2, si toma el valor N se realizará la acción N, y si toma un valor distinto de los valores comprendidos entre 1 y N, se continuará con el flujo normal del diagrama realizándose la acción N+1.

A continuación presentamos el pseudocódigo que ilustra esta estructura selectiva.

```
En caso de Selector
      Caso Valor1: Hacer Acción1
      Caso Valor2: Hacer Acción2
      .
      .
      .
      Caso ValorN: Hacer AcciónN
      En otro caso : AccionN+1
Fin_Caso {Fin del condicional}
```



Donde:

Selector es la variable o expresión, a evaluarse, según la cual se tomará una de las "múltiples" decisiones o alternativas.

Acción1 expresa la operación o conjunto de operaciones que se van a realizar si el selector toma el valor1.

Acción2 expresa la operación o conjunto de operaciones que se van a realizar si el selector toma el valor 2.

AcciónN expresa la operación o conjunto de operaciones que se van a realizar si el selector toma el valor N.

La estructura selectiva ***En caso de*** es muy flexible, lo que permite aplicarla de diferentes formas. Obsérvense los siguientes aplicaciones.

Ejemplo 2.7

Construya un pseudocódigo, que dado como datos dos variables de tipo entero (NUM, V), obtenga el resultado de la siguiente función:

100*V	Si NUM = 1
100^V	Si NUM = 2
100/V	Si NUM = 3
0	Para cualquier otro valor de NUM

Solución

El algoritmo para este caso es simple:

Declarar variables

Leer NUM, V

Según sea el caso de NUM, realizar la acción correspondiente

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          NUM, V : Entero
          VAL    : Real
3. Leer NUM, V
4. En caso de NUM
      4.1. Caso 1 : Calcular VAL = 100*V
      4.2. Caso 2 : Calcular VAL = 100^V
      4.3. Caso 3 : Calcular VAL = 100/V
      4.4. En otro caso: Calcular VAL = 0
5. Fin_Caso
6. Escribir VAL
7. Terminar el proceso
```

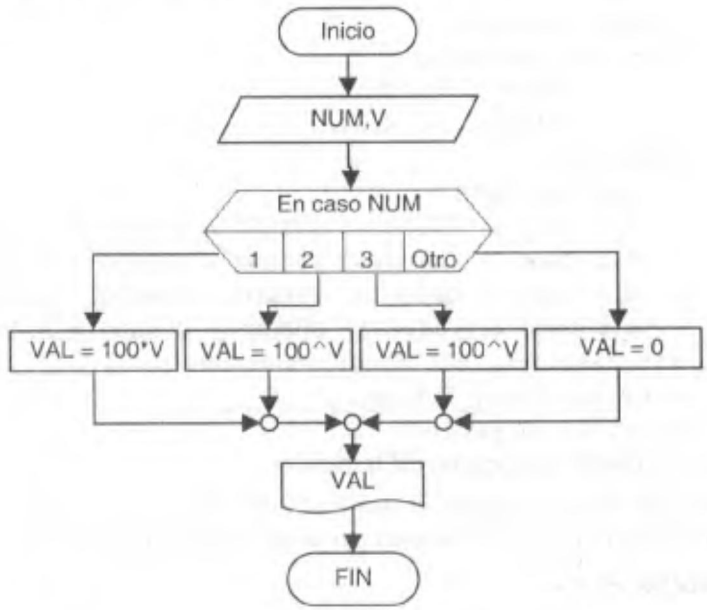
Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main(){
int NUM, V;
float VAL;
clrscr();
cout<<"Ingrese NUM: ";cin>>NUM;
cout<<"Ingrese V: ";cin>>V;
switch(NUM)
{case 1 : VAL = 100*V;break;
 case 2 : VAL = pow(100,V);break;
 case 3 : VAL = 100/V;break;
 default: VAL = 0;
}
```



```
cout<<"Resultado: "<<VAL;
getch();}
```

Switch es lo mismo que usar "En caso de" en el pseudocódigo. El uso de break en la codificación C++ nos permite que una vez evaluada una condición se termine la secuencia del flujo Switch en caso contrario puede seguir evaluando otras condiciones y producir un resultado erróneo.



Ejemplo 2.8

Construya un algoritmo, que dado como datos la categoría y el sueldo de un trabajador, calcule el aumento correspondiente teniendo en cuenta la siguiente tabla. Imprima la categoría del trabajador y su nuevo sueldo.

Categoría	Aumento
1	15%
2	10%
3	8%
4	5%

Solución

Declarar Variables

Leer Datos de entrada

Según sea el caso, realizar el aumento correspondiente.

Reportar el aumento y la categoría.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        CATE      : Entero
        SUELDO    : Real
3. Leer CATE
4. En caso de CATE
     4.1 Caso 1: Calcular SUELDO = SUELDO * 1.15
     4.2 Caso 2: Calcular SUELDO = SUELDO * 1.10
     4.3 Caso 3: Calcular SUELDO = SUELDO * 1.08
     4.3 Caso 4: Calcular SUELDO = SUELDO * 1.06
5. Fin_Caso {Fin del condicional del paso 3}
6. Escribir CATE, SUELDO
7. Terminar el proceso
```

CATE, representa la categoria del trabajador.

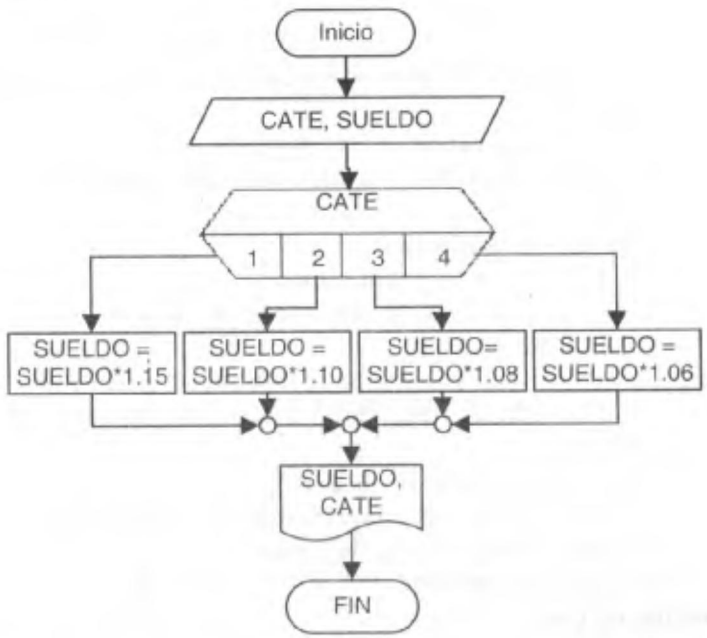
Nótese que no es necesario el uso de la opción "En otro caso", algunos programadores usan esta opción para enviar un mensaje de error al usuario.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int CATE;
float SUELDO;
clrscr();
cout<<"Ingrese Sueldo: ";cin>>SUELDO;
cout<<"Ingrese Categoría: ";cin>>CATE;
switch(CATE)
{
 case 1: SUELDO = SUELDO * 1.15;break;
```



```
 case 2: SUELDO = SUELDO * 1.10;break;
 case 3: SUELDO = SUELDO * 1.08;break;
 case 4: SUELDO = SUELDO * 1.06;break;
}
cout<<"Nuevo sueldo: "<<SUELDO;
cout<<"Categoría: "<<CATE;
getch();}
```



A continuación presentamos una serie de problemas resueltos, diseñados expresamente como elementos de ayuda para el análisis y la retención de los conceptos. Además se utilizan en muchos de ellos, tablas que permiten hacer el seguimiento del algoritmo para diferentes corridas.

Ejemplo 2.9

Dados los datos A, B y C, que representan números enteros diferentes, construya un pseudocódigo para escribir estos números en forma descendente.

Pseudocódigo

```
1. Iniciar proceso
```

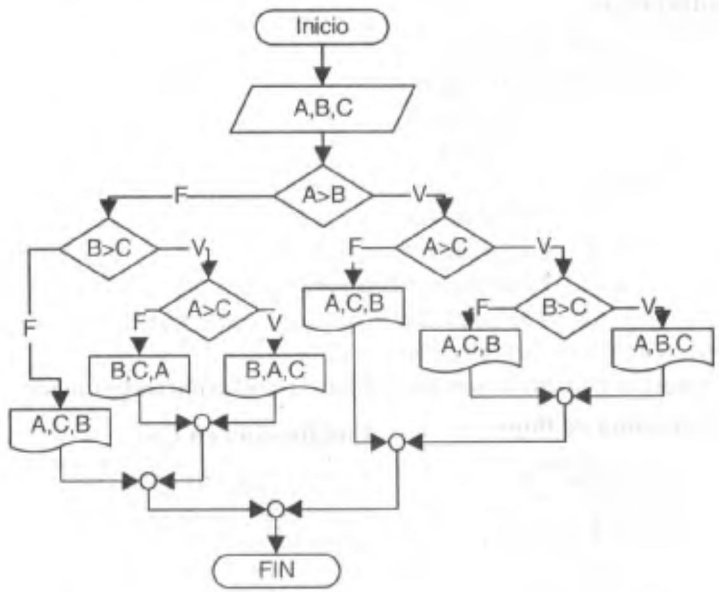
```
2. Declarar Variables
        A, B, C : Entero
3. Leer A, B. C
4. Sí A > B Entonces
 4.1. Si A > C Entonces
      4.1.1. Si B > C Entonces
               4.1.1.1. Escribir A, B, C
      4.1.2. Si no
               2.1.2.1. Escribir A, C, B
      4.1.3. Fin_Si{Fin del condicional del paso 2.1.1}
 4.2. Si no
      4.2.1. Escribir C, A, B
 4.3. Fin_Si {Fin del condicional del paso 2.1}
5. Si no
 5.1. Si B > C Entonces
      5.1.1. Si A > C Entonces
               5.1.1.1. Escribir B, A y C
      5.1.2 Si no
               5.1.2.1. Escribir B, C y A
      5.1.3. Fin_Si {Fin del condicional del paso 3.1.1}
 5.2. Si no
      5.2.1 Escribir C, B y A
 5.3. Fin_Si {Fin del condicional del paso 3.1}
6. {Fin del condicional del paso 3}
7. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int A,B,C;
clrscr();
cout<<"Ingrese A: ";cin>>A;
cout<<"Ingrese B: ";cin>>B;
cout<<"Ingrese C: ";cin>>C;
if (A > B)
{    if (A > C)
     {   if (B > C)
```



```
               cout<<A<<" "<<B<<" "<<C;
          else
               cout<<A<<" "<<C<<" "<<B;

     }else
          cout<<C<<" "<<A<<" "<<B;
}else
{    if (B > C)
          if (A > C)
               cout<<B<<" "<<A<<" "<<C;
          else
               cout<<B<<" "<<C<<" "<<A;
     else
          cout<<C<<" "<<B<<" "<<A;
}
getch();
}
```



Ejemplo 2.10

El número de sonidos emitidos por un grillo en un minuto, es una función de la temperatura. Como resultado de esto, es posible determinar el nivel de la temperatura haciendo uso de un grillo como termómetro.

La fórmula para la función es:

$$T = N/4 + 40$$

Donde: T representa la temperatura en grados Fahrenheit y N el número de sonidos emitidos por minuto.

Construya un pseudocódigo que le permita calcular la temperatura, teniendo en cuenta el número de sonidos emitidos por el grillo.

Solución

Declarar Variables

Comprobar si el número de sonidos es válido

Calcular la temperatura

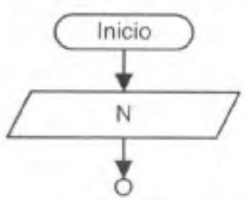
Reportar temperatura calculada

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          N : Entero
          T : Real
3. Leer N
4. Si N > 0 Entonces
      4.1. Calcular T = N/4 + 40
      4.2. Escribir "Temperatura", T
5. Fin_Si {Fin del condicional del paso 4}
6. Terminar el proceso
```

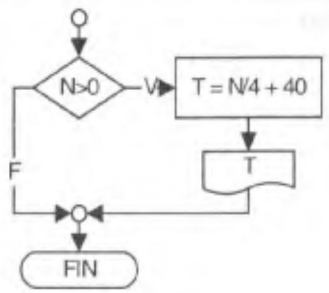
N expresa el número de sonidos emitidos por el grillo en un minuto

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int N;
```


```
float T;
clrscr();
cout<<"Ingrese N:
";cin>>N;
if (N > 0)
{     T = N/4 + 40;
      cout<<"Temperatura:
"<<T;}
getch();}
```

Ejemplo 2.11

Construya un pseudocódigo, que dado como datos los valores enteros P y Q, determine si los mismos satisfacen la siguiente expresión:

$$P^3 + Q^4 - 2*P^2 < 680$$

En caso afirmativo debe imprimir los valores P y Q.

Donde P y Q son variables de tipo entero que expresan los datos que se ingresan.

Solución

Declarar variables

Comprobar si P y Q satisfacen la expresión

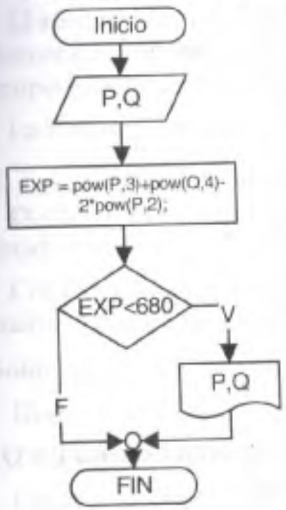
Si es satisfactorio mostrar P y Q

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          P, Q : Enteros
          EXP  : Real
3. Leer P, Q
4. Calcular EXP = P^ 3 + Q^ 4 - 2 * P^ 2
5. Si EXP < 680 Entonces
      5.1. Escribir P,Q
6. Fin_Si {Fin del condicional del paso 5}
7. Terminar el proceso
```

EXP almacena el resultado del cálculo de la expresión.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
int P,Q;
float EXP;
clrscr();
cout<<"Ingrese P: ";
cin>>P;
cout<<"Ingrese Q: ";
cin>>Q;
EXP = pow(P,3)+pow(Q,4)-2*pow(P,2);
if (EXP < 680)
{    cout<<P<<"   "<<Q;
}
getch();
}
```

Ejemplo 2.12

Escribir un programa que lea dos números enteros por teclado y determine cuál es el mayor y cual es el menor. También deberá considerar el caso en el que los dos números sean iguales.

Solución

Declarar Variables

Leer números

Comprobar cual es el mayor, o la igualdad

Reportar resultados

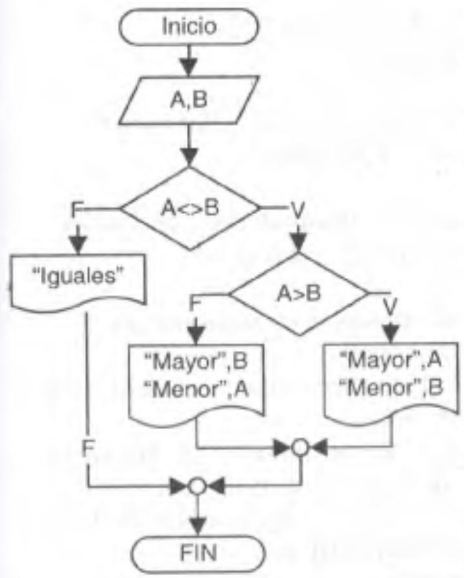
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                A, B : Entero
3. Leer A, B
4. Si A <> B Entonces
     4.1. Si A > B Entonces
          4.1.1. Escribir 'El mayor es', A,
                         'El menor es', B
```



```
     4.2. Si no
          4.2.1. Escribir 'El mayor es', B,
                          'El menor es', A
     4.3. Fin_Si
5. Si no
     5.1. Escribir 'Los números son iguales'
6. Fin_Si {Fin del condicional del paso 4}
7. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int A,B;
clrscr();
cout<<"Ingrese A:";cin>>A;
cout<<"Ingrese B:";cin>>B;
if (A != B)
 if (A > B)
 { cout<<"El mayor es"<<A;
   cout<<"El menor es"<<B;
 }else{
   cout<<"El mayor es"<<B;
   cout<<"El menor es"<<A;
 }
else
 cout<<"Números son
       iguales";
getch();}
```

Solo hay una cosa que resaltar en la codificación en C++, es el uso del operador != el cual implica diferencia, este es el que reemplaza al operador <> del pseudocódigo.

Ejemplo 2.13

Escribir un programa que lea una nota de un examen por teclado y devuelva la calificación que tiene. La calificación podrá ser: Suspenso (0-4.99), Aprobado (5-6.99), Notable (7-8.99), Sobresaliente (9-9.99) o Matricula de Honor (10).

Solución

Declarar Variables

Leer nota

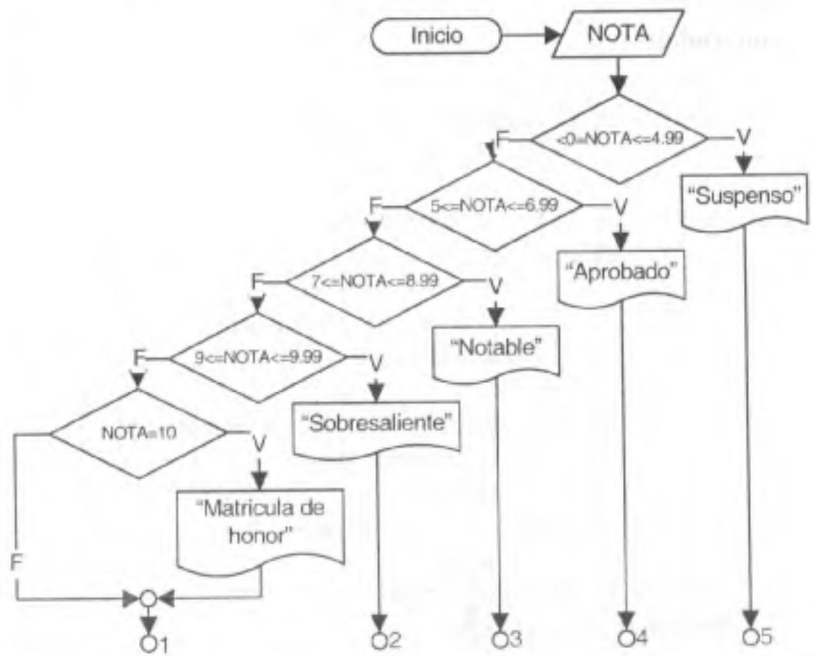
Calcular calificación y reportar resultados

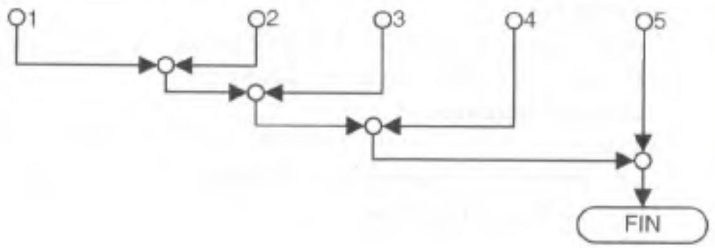
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                Nota : Real
3. Leer Nota
4. Si Nota >= 0 Y Nota <= 4.99 Entonces
     4.1. Escribir 'Suspenso'
5. Si no
     5.1. Si Nota >= 5 Y Nota <= 6.99  Entonces
          5.1.1. Escribir ' Aprobado'
     5.2. Si no
          5.2.1. Si Nota>=7 Y Nota<=8.99  Entonces
               5.2.1.1. Escribir 'Notable'
          5.2.2. Si no
               5.2.2.1. Si Nota>=9 Y Nota<=9.99
                        Entonces
                    5.2.2.1.1. Escribir 'Sobresaliente'
               5.2.2.2. Si no
                    5.2.2.2.1. Si Nota = 10 Entonces
                         5.2.2.2.1.1 Escribir
                                  'Matricula de honor'
                    5.2.2.2.2. Fin_Si
               5.2.2.3. Fin_Si
          5.2.3. Fin_Si
     5.3. Fin_Si
6. Fin_Si
7. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
float Nota;
clrscr();
cout<<"Ingrese Nota: ";cin>>Nota;
if (Nota >= 0 & Nota <= 4.99)
 cout<<"Suspenso";
else
 if (Nota >= 5 & Nota <= 6.99)
     cout<<"Aprobado";
 else
     if (Nota>=7 & Nota<=8.99)
          cout<<"Notable";
     else
          if (Nota>=9 & Nota<=9.99)
               cout<<"Sobresaliente";
          else
               if (Nota == 10)
                    cout<<"Matrícula de honor";
getch();}
```



Ejemplo 2.14

Escribir un programa que lea tres números enteros por teclado y muestre por pantalla el mayor de los tres.

Solución

Declarar Variables

Leer números

Calcular el mayor

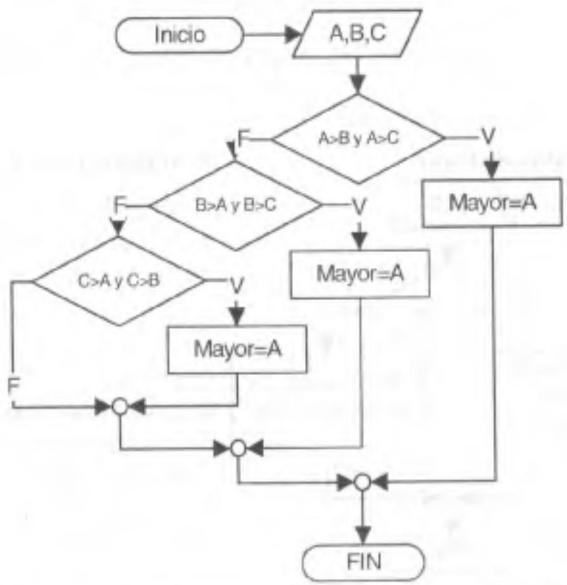
Reportar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                A, B, C, Mayor : Entero
3. Leer A, B, C
4. Si A > B Y A > C Entonces
      4.1. Hacer Mayor = A
5. Si no
      5.1. Si B > A Y B > C Entonces
           5.1.1 Hacer Mayor = B
      5.2. Si no
           5.2.1. Si C > A Y C > B Entonces
                5.1.1 Hacer Mayor = C
           5.2.2. Fin_Si
      5.3. Fin_Si
6. Fin_Si
7. Escribir 'El número mayor es: ', Mayor
8.Terminar el proceso
```


Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int A,B,C,Mayor;
clrscr();
cout<<"Ingrese A: ";cin>>A;
cout<<"Ingrese B: ";cin>>B;
cout<<"Ingrese C: ";cin>>C;
if (A>B & A>C)
 Mayor = A;
else
 if (B>A & B>C)
      Mayor = B;
 else
      if (C>A & C>B)
           Mayor = C;
cout<<"El número mayor es: "<<Mayor;
getch();}
```



Ejemplo 2.15

Escribir un programa que lea tres números enteros por teclado y emita un mensaje indicando si están o no ordenados crecientemente.

Solución

La solución a este problema es sencillo, ya que siendo las variables A, B y C, estas están en forma creciente si A<B<C. Teniendo en cuenta esto, el algoritmo sería el siguiente.

Declarar Variables

Leer números

Comparar las variables

Reportar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                A, B, C : Entero
3. Leer A, B, C
4. Si A < B Y B < C Entonces
     4.1. Escribir 'Están Ordenados en forma creciente'
5. Si no
     5.1. Escribir 'No están ordenado en forma creciente'
6. Fin_Si
7.Terminar el proceso
```

Diagrama de flujo

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{

clrscr();
cout<<"Ingrese
A:";cin>>A;
cout<<"Ingrese
B:";cin>>B;
```



```
cout<<"Ingrese C:";cin>>C;
if (A<B & B<C)
 cout<<"Ordenado en forma creciente";
else
 cout<<"No ordenado en forma creciente";
getch();
}
```

Ejemplo 2.16

Escribir un programa que permita introducir por teclado tres letras y responda si existen al menos dos letras iguales.

Solución

Las comparaciones que hemos estado realizando han sido en base a números, ahora tenemos la oportunidad de hacer comparación de caracteres, en el fondo tienen , en algoritmo, el mismo tratamiento, todos los operadores de comparación pueden ser aplicados a los caracteres.

Declarar Variables

Leer caracteres

Comparar si existe un par de caracteres iguales

Mostrar resultado en el caso de haber un par de caracteres iguales.

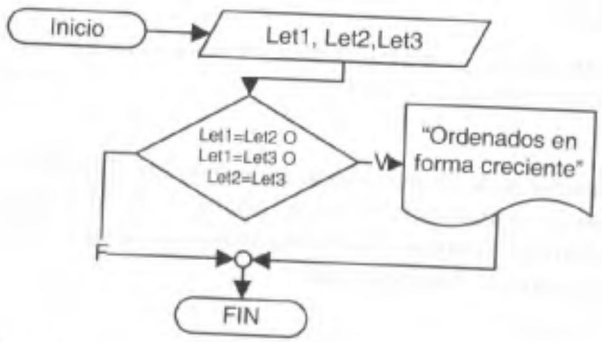
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                Let1, Let2, Let3 : Caracter
3. Leer Let1, Let2, Let3
4. Si Let1 = Let2 O Let1 = Let3 O Let2 = Let3 Entonces
     4.1. Escrbir 'Existe un par de caracteres iguales'
5. Fin_Si
6. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
char Let1, Let2, Let3;
```

```
clrscr();
cout<<"Ingrese letra: ";cin>>Let1;
cout<<"Ingrese letra: ";cin>>Let2;
cout<<"Ingrese letra: ";cin>>Let3;
if (Let1 == Let2 || Let1 == Let3 || Let2 == Let3)
 cout<<"Existe un par de caracteres iguales";
getch();}
```



Ejemplo 2.17

Realizar un programa que pida dos números enteros por teclado y muestre por pantalla el siguiente menú:

MENÚ

1. Sumar
2. Restar
3. Multiplicar
4. Dividir

Elija una opción:

El usuario deberá elegir una opción y el programa deberá mostrar el resultado por pantalla y finalizar.

Solución

El ejemplo nos pide ingresar dos números enteros, luego de eso debemos ingresar 1, 2, 3 o 4 según sea la operación que deseemos realizar con los números

ingresados. Además, se nos debe presentar un menú a través del cual introduciremos la opción correspondiente a la operación. El algoritmo es el siguiente:

Declarar Variables

Leer números

Mostrar menú

Leer opción

Según sea la opción ingresada, realizar operación

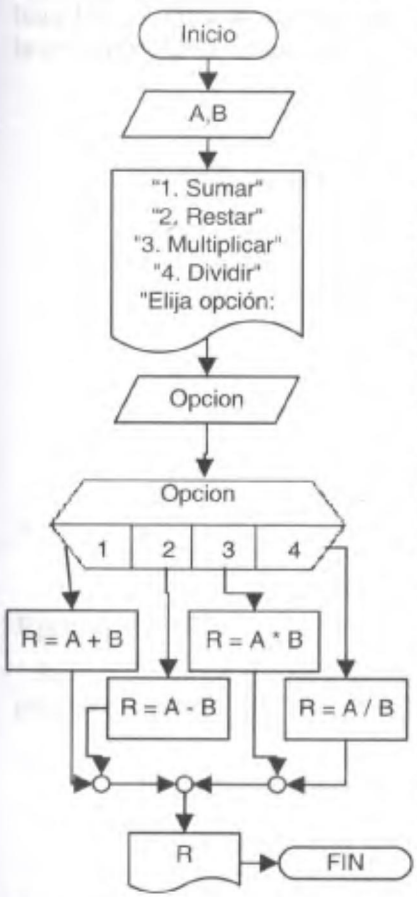
Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                A, B, Opcion : Entero
                R : Real
3. Leer A,B
4. Escribir "   MENU    "
            "1. Sumar"
            "2. Restar"
            "3. Multiplicar"
            "4. Dividir"
            "Elija opción:"
5. Leer Opcion
3. Leer A, B.
6. En caso de Opcion
      6.1. Caso 1 : R = A + B
      6.2. Caso 2 : R = A - B
      6.3. Caso 3 : R = A * B
      6.4. Caso 4 : R = A / B
7. Fin_Caso
8. Escribir 'el resultado es: ', R
9. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
```




```
void main()
{
int A, B, Opcion;
float R;
clrscr();
cout<<"Primer número";
cin>>A;
cout<<endl<<"Segundo número";
cin>>B;
cout<<"MENU"<<endl;
cout<<"1. Sumar"<<endl;
cout<<"2. Restar"<<endl;
cout<<"3. Multiplicar"<<endl;
cout<<"4. Dividir"<<endl;
cout<<"Elija
opción:";cin>>Opcion;
switch(Opcion)
{case 1 : R = A + B;break;
 case 2 : R = A - B;break;
 case 3 : R = A * B;break;
 case 4 : R = A / B;
}
cout<<endl<<"el resultado
es:"<<R;
getch();
   }
```

Nótese el uso de <<endl en el código en C++, esto permite insertar un fin de línea, es decir que cuando se muestre el contenido de cout el cursor se mostrará en la línea siguiente.

Ejemplo 2.18

Dado como dato el tiempo de servicio de un trabajador, considere un aumento del 15% si la categoría del trabajador es A, un 12% en caso la categoría sea B, Si la categoría es C, un aumento del 10% y para la categoría D se aumentará $15. Imprima el sueldo con el aumento incorporado, la categoría y el tiempo de servicio del trabajador.

La categoría esta dada por la siguiente tabla

Categoría	Años
A	20-30
B	De 15 a 20
C	De 10 a 15
D	de 0 a 10

Supóngase un sueldo general S, para todas las categorías

Solución

Nótese que para poder hallar el aumento correspondiente primero debemos de hallar la categoría. El algoritmo para este ejercicio es el siguiente:

Declarar Variables

Leer años de servicio y el sueldo

Obtener categoría de trabajador

Según sea la categoría, realizar el aumento

Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                TS, S : Real
                Cate : Caracter
3. Leer TS, S
4. Si TS >= 0 Y TS <10 Entonces
     4.1. Cate = 'D'
5. Si no
     5.1. Si TS >= 10 Y TS <15 Entonces
          5.1.1. Cate = 'C'
     5.2. Si no
          5.2.1. Si TS >= 15 Y TS <20 Entonces
               5.2.1.1. Cate = 'B'
          5.2.2. Si no
               5.2.2.1. Si TS>=20 Y TS<=30 Entonces
                    5.2.2.1.1. Cate = 'A'
               5.2.2.2. Fin_Si
          5.2.3. Fin_Si
     5.3. Fin_Si
```

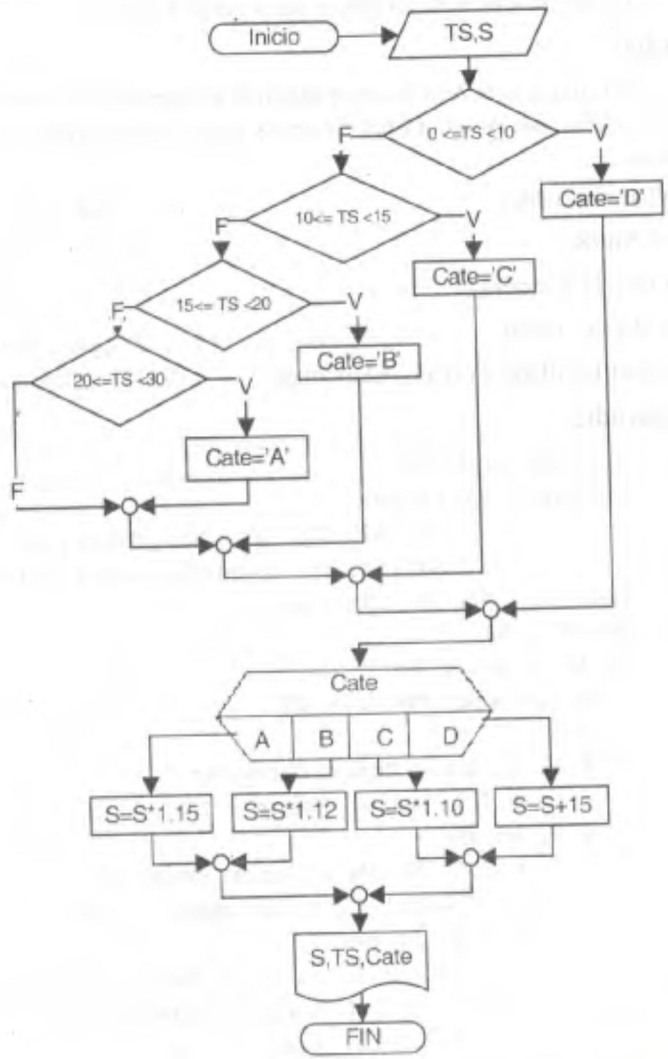
```
6. Fin_Si
7. En caso de Cate
      7.1. Caso 'A' : S = S * 1.15
      7.2. Caso 'B' : S = S * 1.12
      7.3. Caso 'C' : S = S * 1.10
      7.4. Caso 'D' : S = S + 15
8. Fin_Caso
9. Escribir 'El nuevo sueldo es: ', S
            'La categoría es: ', Cate
            'Tiempo de servicio es: ', TS
10. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float TS,S;
char Cate;
clrscr();
cout<<"Tiempo de servicio: ";cin>>TS;
cout<<"Sueldo: ";cin>>S;
if (TS >= 0 && TS <10)
 Cate = 'D';
else
 if (TS >= 10 && TS <15)
      Cate = 'C';
 else
      if (TS >= 15 && TS <20)
           Cate = 'B';
      else
           if (TS>=20 && TS<=30)
                Cate = 'A';
switch(Cate)
{     case 'A' : S = S * 1.15;break;
      case 'B' : S = S * 1.12;break;
      case 'C' : S = S * 1.10;break;
      case 'D' : S = S + 15;
```



```
}
cout<<endl<<"El nuevo sueldo es: "<<S;
cout<<endl<<"Categoría: "<<Cate;
cout<<endl<<"Tiempo de servicio: "<<TS;
getch();}
```



Ejemplo 2.19

Construya un pseudocódigo tal, que dados como datos la matrícula y 5 calificaciones de un alumno; imprima la matrícula, el promedio y la palabra "Aprobado" si el alumno tiene un promedio mayor o igual que 10.5, y la palabra "No aprobado" en caso contrario.

El promedio se calculará en base a las 4 notas mayores.

Solución

Lo primero que debemos hacer es eliminar la menor nota, esta tarea ya nos es familiar, dado que en anteriores ejemplos hemos desarrollado procedimientos similares.

Declarar variables

Leer Notas

Eliminar la menor nota

Calcular promedio

Reportar resultado y estado del alumno.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                N1, N2, N3, N4, N5, Menor, P : Real
                Matricula: cadena de caracteres
3. Leer N1, N2, N3, N4, N5
4. Menor = N1
5. Si N2 < Menor Entonces
      5.1. Hacer Menor = N2
6. Si no
      6.1. Si N3 < Menor Entonces
           6.1.1. Hacer Menor = N3
      6.2. Si no
           6.2.1. Si N4 < Menor Entonces
                6.2.1.1. Hacer Menor = N4
           6.2.2. Si no
                6.2.2.1. Si N5 < Menor Entonces
                     6.2.2.1.1. Hacer Menor = N5
                6.2.2.2. Fin_Si
```

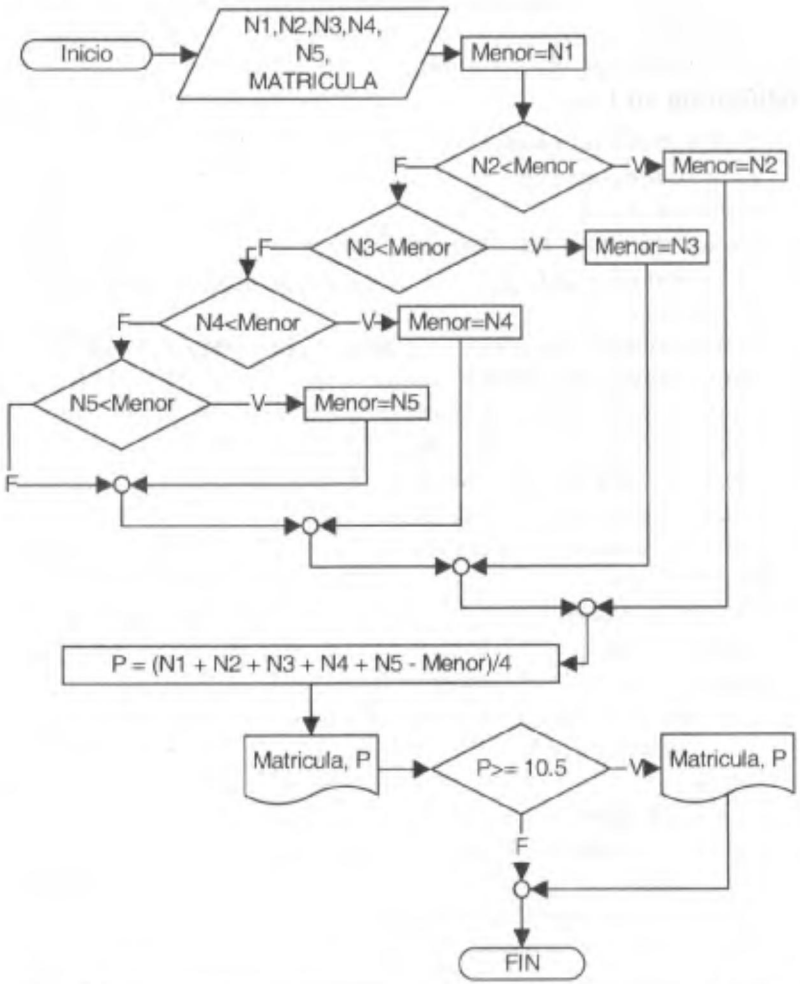


```
           6.2.3. Fin_Si
      6.3. Fin_Si
7. Fin_Si
8. Calcular P = (N1 + N2 + N3 + N4 + N5 - Menor)/4
9. Escribir Matricula, P
10. Si P >= 10.5 Entonces
      10.1. Escribir 'Aprobado'
11. Si no
      11.1. Escribir 'No aprobado'
12. Fin_Si
13. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
float N1, N2, N3, N4, N5, Menor,P;
char Matricula[10];
clrscr();
cout<<endl<<"Ingrese código: ";cin>>Matricula;
cout<<endl<<"Ingrese notas: ";
cout<<"Nota 1: ";cin>>N1;
cout<<"Nota 2: ";cin>>N2;
cout<<"Nota 3: ";cin>>N3;
cout<<"Nota 4: ";cin>>N4;
cout<<"Nota 5: ";cin>>N5;
Menor = N1;
if (N2 < Menor)
 Menor = N2;
else
 if (N3 < Menor)
    Menor = N3;
 else
      if (N4 < Menor)
         Menor = N4;
      else
           if (N5 < Menor)
              Menor = N5;
```

```
P = (N1 + N2 + N3 + N4 + N5 - Menor)/4;
cout<<endl<<"Codigo: "<<Matricula;
cout<<endl<<"Promedio: "<<P;
if (P >= 10.5)
 cout<<endl<<"Aprobado";
else
 cout<<endl<<"No aprobado";
getch();}
```



Hemos usado un pequeño truco para calcular el promedio de las cuatro notas mayores, primero hemos encontrado la menor nota, luego al calcular el promedio sumamos las cinco notas ingresadas y le restamos la nota menor antes calculada.

Ejemplo 2.20

Construya un algoritmo que permita saber si un número entero de 4 cifras, es capicúa.

Solución

Un número capicúa es cuando al ser leído de izquierda a derecha y de derecha a izquierda, nos debe dar el mismo número. Por ahora usaremos el método de invertir un número para poder ver si al invertirlo es igual al original.

Declarar variables

Leer número

Invertir número

Comparar número invertido con el original

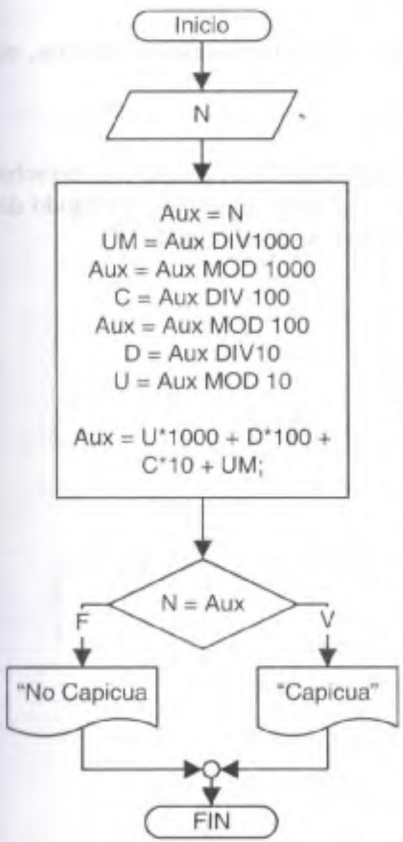
Reportar resultados

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                N , Aux, : Entero
                U, D, C, UM : Entero
3. Leer N
4. Hacer Aux = N
5. Calcular     UM = Aux Div 1000
                Aux = Aux Mod 1000
                C = Aux Div 100
                Aux = Aux Mod 100
                D = Aux Div 10
                U = Aux Mod 10
6. Calcular Aux = U*1000 + D*100 + C*10 + UM
7 Si N = Aux Entonces
     7.1. Escribir 'Número es capicúa'
8. Si no
     8.1. Escribir 'Número no es capicúa'
9. 4. Fin_Si
```

```
10. Terminar el proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int N , Aux,U, D, C, UM;
clrscr();
cout<<endl<<"Ingrese número:
";
cin>>N;
Aux = N;
UM = Aux/1000;
Aux = Aux%1000;
C = Aux/100;
Aux = Aux%100;
D = Aux/10;
U = Aux%10;
Aux = U*1000 + D*100 + C*10 +
UM;
if (N == Aux)
 cout<<endl<<"Capicúa";
else
 cout<<endl<<"No capicúa";
getch();
}
```

U, D, C, UM son variables usadas para la descomposición del número

Nótese que la variable auxiliar (Aux) primero la usamos para realizar la descomposición del número y luego nos sirve para almacenar el número invertido, esta metodología es muy usada, consta en usar una misma variable para diferentes usos, de esa manera no debemos de declarar una variable especifica para cada acción a realizar. Solo se deben de declarar las variables necesarias.

Ejemplo 2.21

Diseñe un algoritmo que ingresada una fecha en formato DD/MM/AAAA, reporte la fecha como: "Es DD del mes MM del año AAAA". Debe suponerse que la fecha ingresada es válida.

Solución

Se nos pide que ingresado la fecha 15/06/2005, nos de como resultado: Es 15 de Junio del año 2005. El único inconveniente es obtener el mes en letras, la cual es una tarea sencilla. El algoritmo es el siguiente:

Declarar variables

Leer fecha

Convertir mes a letras

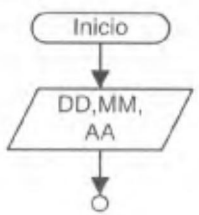
Reportar fecha

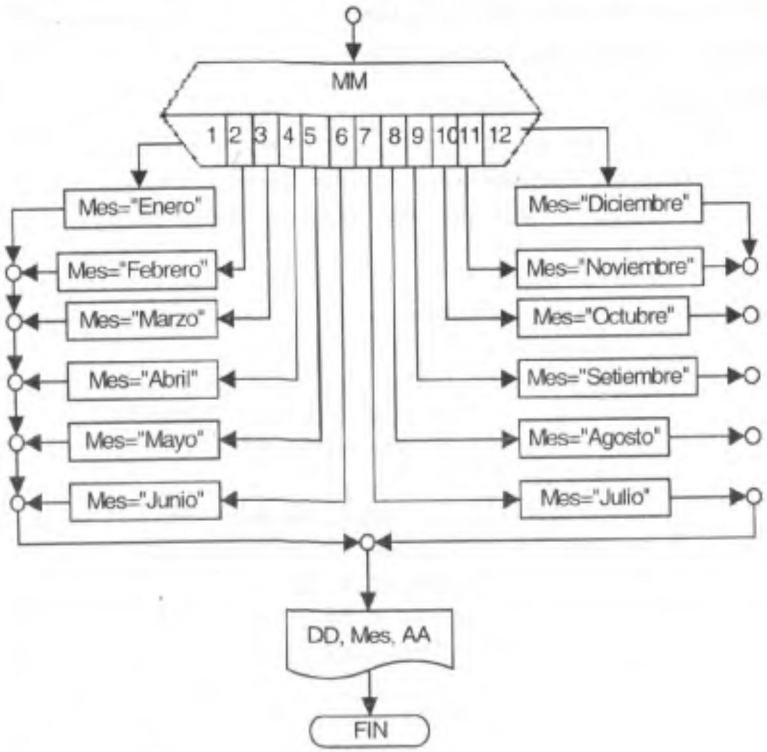
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                DD, MM, AA : Entero
                Mes : Cadena de caracteres
3. Leer DD, MM, AA
4. En caso de MM
      4.1. Caso 1 : Mes = 'Enero'
      4.2. Caso 2 : Mes = 'Febrero'
      4.3. Caso 3 : Mes = 'Marzo'
      4.4. Caso 4 : Mes = 'Abril'
      4.5. Caso 5 : Mes = 'Mayo'
      4.6. Caso 6 : Mes = 'Junio'
      4.7. Caso 7 : Mes = 'Julio'
      4.8. Caso 8 : Mes = 'Agosto'
      4.9. Caso 9 : Mes = 'Setiembre'
      4.10. Caso 10 : Mes = 'Octubre'
      4.11. Caso 11 : Mes = 'Noviembre'
      4.12. Caso 12 : Mes = 'Diciembre'
5. Fin_Caso
6. Escribir 'Es ', DD, 'de ', Mes, 'del año ', AA
7. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
int DD, MM, AA;
char Mes[20];
clrscr();

cout<<endl<<"Ingrese dia: ";cin>>DD;
cout<<endl<<"Ingrese mes: ";cin>>MM;
cout<<endl<<"Ingrese año: ";cin>>AA;
switch(MM)
{case 1 : strcmp(Mes,"Enero");break;
 case 2 : strcmp(Mes,"Febrero");break;
 case 3 : strcmp(Mes,"Marzo");break;
 case 4 : strcmp(Mes,"Abril");break;
 case 5 : strcmp(Mes,"Mayo");break;
 case 6 : strcmp(Mes,"Junio");break;
 case 7 : strcmp(Mes,"Julio");break;
 case 8 : strcmp(Mes,"Agosto");break;
 case 9 : strcmp(Mes,"Setiembre");break;
 case 10 : strcmp(Mes,"Octubre");break;
 case 11 : strcmp(Mes,"Noviembre");break;
 case 12 : strcmp(Mes,"Diciembre");break;
}
cout<<"Es "<<DD<<" de "<<Mes<<" del año "<<AA;
getch();
}
```





Ejemplo 2.22

Diseñe un algoritmo que permita analizar la validez de una fecha, considere que el año válido sea mayor a 1800.

Solución

Para comprobar la validez de una fecha, es necesario analizar , los componentes individualmente, es decir, debemos ver que el día sea válido, asimismo el mes y el año. Es recomendable que analice primero el año, luego el mes y finalmente el día, recuerde que sabiendo cual es el mes, sabremos también la cantidad de días que posee dicho mes. El algoritmo es el siguiente:

Declarar variables

Leer fecha

Obtener número máximo de días que contienen los meses del año.

Comprobar si año, mes y día son válidos

Reportar si la fecha ingresada es válida o no

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                DD, MM, AA, NDias : Entero
3. Leer DD, MM, AA
4. En caso de MM
     4.1. Caso 1 : NDias = 31
     4.2. Caso 2 :
          4.2.1. Si (AA Mod 4 =0 Y AA MOD 400 = 0) O
                    (AA Mod 400 =0) Entonces
               4.2.1.1. Hacer NDias = 29.
          4.2.2. Si no
               4.2.2.1. Hacer NDias = 28.
     4.3. Caso 3 : NDias = 31
     4.4. Caso 4 : NDias = 30
     4.5. Caso 5 : NDias = 31
     4.6. Caso 6 : NDias = 30
     4.7. Caso 7 : NDias = 31
     4.8. Caso 8 : NDias = 31
     4.9. Caso 9 : NDias = 30
     4.10. Caso 10 : NDias = 31
     4.11. Caso 11 : NDias = 30
     4.12. Caso 12 : NDias = 31
5. Fin_Caso
6. Si AA>1800 Y (MM>0 Y MM<=12) Y (DD>0 Y DD<=NDias)
     6.1. Escribir 'Fecha válida'
7. Si no
     7.1.Escribir 'Fecha no válida'
8. Fin_Si
9. Terminar el proceso
```

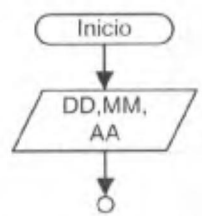
La variable NDias almacena el número de días que tiene el mes.

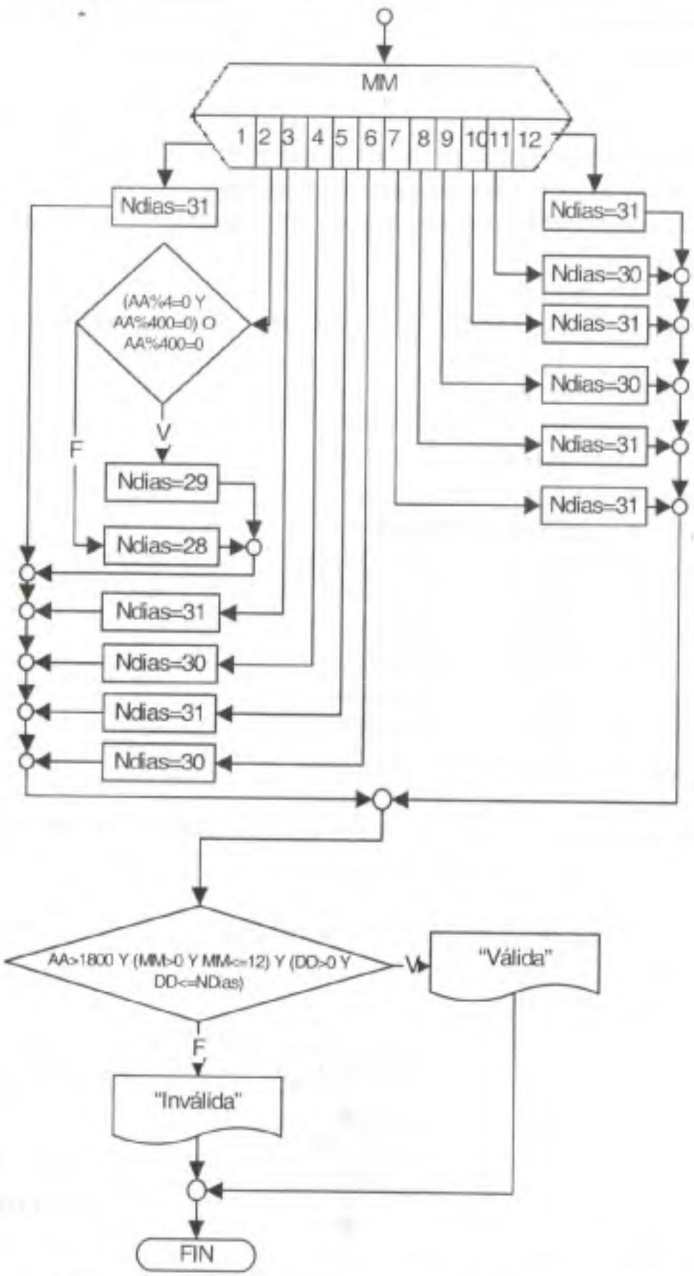
Codificación en C++

```
#include<iostream.h>
```



```
#include<conio.h>
void main(){
int DD, MM, AA, NDias;
clrscr();
cout<<endl<<"Ingrese dia: ";cin>>DD;
cout<<endl<<"Ingrese mes: ";cin>>MM;
cout<<endl<<"Ingrese año: ";cin>>AA;
switch(MM)
{case 1 : NDias = 31;
 case 2 : if ((AA%4==0 && AA%400==0) || (AA%400==0))
               NDias = 29;
          else
               NDias = 28;
          break;
 case 3 : NDias = 31;break;
 case 4 : NDias = 30;break;
 case 5 : NDias = 31;break;
 case 6 : NDias = 30;break;
 case 7 : NDias = 31;break;
 case 8 : NDias = 31;break;
 case 9 : NDias = 30;break;
 case 10 : NDias = 31;break;
 case 11 : NDias = 30;break;
 case 12 : NDias = 31;break;}
if(AA>1800 && (MM>0 && MM<=12) && (DD>0 && DD<=NDias))
     cout<<endl<<"Fecha válida";
else
     cout<<endl<<"Fecha no válida";
getch();}
```





Ejemplo 2.23

Diseñe un algoritmo que permita calcular el dia siguiente de una fecha dada. Asumir que la fecha ingresada es válida.

Solución

Al calcular el día siguiente de una fecha, debemos de ver que sucede cuando el dia ingresado es el ultimo dia del mes, por lo tanto el siguiente día deberá ser el primer dia del mes siguiente, y que pasa si la fecha ingresada es el último dia del ultimo mes del años, el dia siguiente seria el primer dia del primer mes del año siguiente. Por lo tanto el algoritmo será el siguiente:

Declarar variables

Leer fecha

Obtener número máximo de dias que contienen los meses del año.

Reportar fecha

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                DD, MM, AA : Entero
3. Leer DD, MM, AA
4. En caso de MM
    4.1. Caso 1 : NDias = 31
    4.2. Caso 2 :
        4.2.1. Si (AA Mod 4 =0 Y AA MOD 400 = 0) O
                  (AA Mod 400 =0) Entonces
            4.2.1.1. Hacer NDias = 29.
        4.2.2. Si no
            4.2.2.1. Hacer NDias = 28.
    4.3. Caso 3 : NDias = 31
    4.4. Caso 4 : NDias = 30
    4.5. Caso 5 : NDias = 31
    4.6. Caso 6 : NDias = 30
    4.7. Caso 7 : NDias = 31
    4.8. Caso 8 : NDias = 31
    4.9. Caso 9 : NDias = 30
    4.10. Caso 10 : NDias = 31
    4.11. Caso 11 : NDias = 30
```

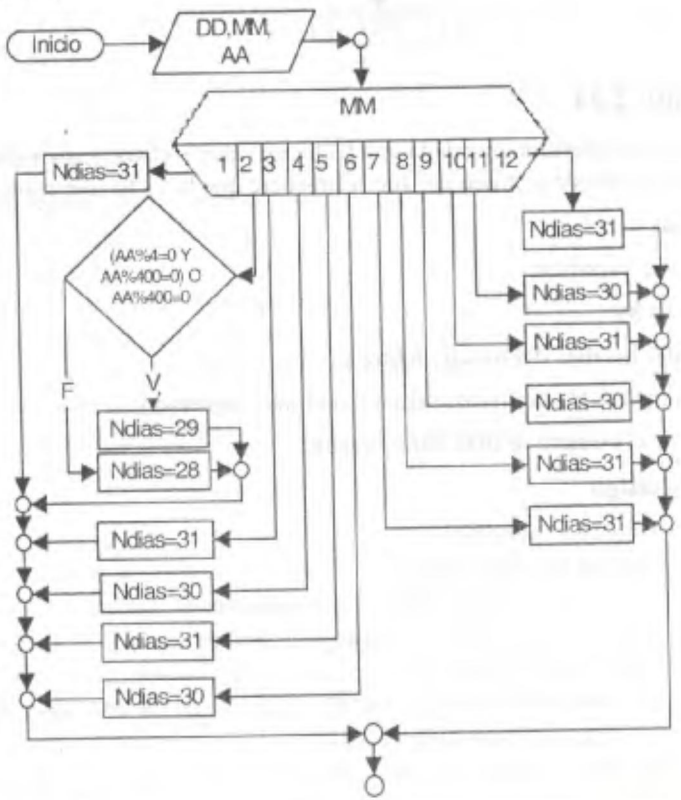
```
        4.12. Caso 12 : NDias = 31
    5. Fin_Caso
    6. DD = DD + 1
    7. Si DD > NDias Entonces
        7.1. Calcular DD = 1
                      MM = MM + 1
        7.2. Si MM > 12 Entonces
            7.2.1. Calcular MM = 1
                           AA = AA + 1
        7.3. Fin_Si
    8. Fin_Si
    9. Escribir 'Día siguiente es: ', DD, MM, AA
    10. Terminar el proceso
```

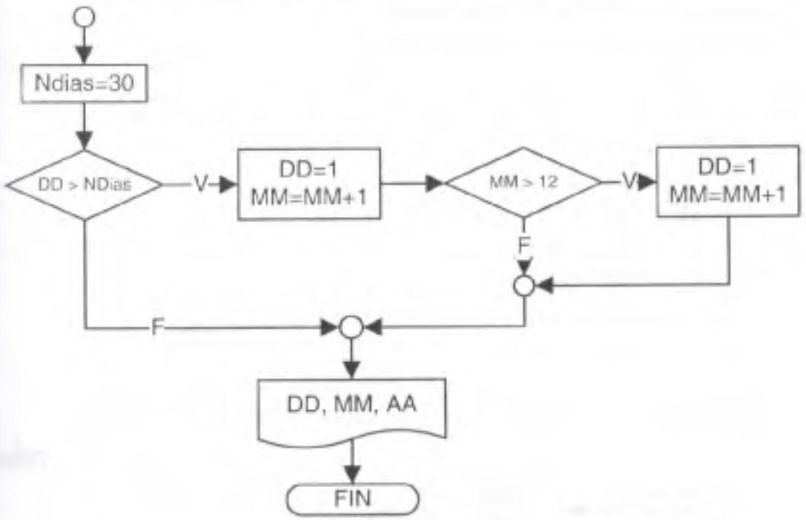
Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int DD, MM, AA, NDias;
clrscr();
cout<<endl<<"Ingrese día: ";cin>>DD;
cout<<endl<<"Ingrese mes: ";cin>>MM;
cout<<endl<<"Ingrese año: ";cin>>AA;
switch(MM)
{case 1 : NDias = 31;
 case 2 : if ((AA%4==0 && AA%400==0) || (AA%400==0))
              NDias = 29;
          else
              NDias = 28;
          break;
 case 3 : NDias = 31;break;
 case 4 : NDias = 30;break;
 case 5 : NDias = 31;break;
 case 6 : NDias = 30;break;
 case 7 : NDias = 31;break;
 case 8 : NDias = 31;break;
 case 9 : NDias = 30;break;
 case 10 : NDias = 31;break;
```



```
 case 11 : NDias = 30;break;
 case 12 : NDias = 31;break;
}
DD = DD + 1;
if (DD > NDias)
{    DD = 1;
     MM = MM + 1;
     if (MM > 12)
     {    MM = 1;
          AA = AA + 1;
     }
}
cout<<"Dia siguientes es:"<<DD<<"/"<<MM<<"/"<<AA;
getch();}
```





Ejemplo 2.24

Diseñe un algoritmo que dada una fecha, se calcule el número de días que han transcurrido desde el inicio del año. Considere que la fecha ingresada es válida.

Solución

Declarar variables

Leer fecha

Calcular los dias del mes de febrero

Calcular dias transcurridos según sea el mes ingresado

Escribir el número de días transcurridos

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                DD, MM, AA   : Entero
                Aux, Suma    : Entero
3. Leer DD, MM,AA
4. Si (AA Mod 4 =0 Y AA MOD 400 = 0) O (AA Mod 400 =0)
     4.1. Hacer Aux = 29.
5. Si no
```



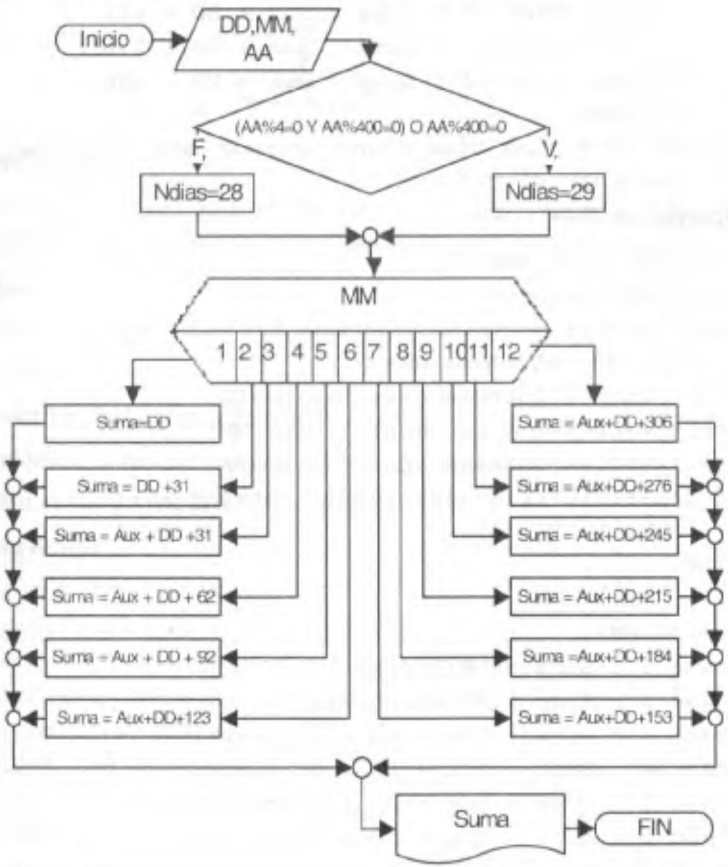
```
     5.1. Hacer Aux = 28.
6. Fin_Si
7. En caso de MM
     7.1. Caso 1 : Suma = DD
     7.2. Caso 2 : Suma = DD +31
     7.3. Caso 3 : Suma = Aux + DD +31
     7.4. Caso 4 : Suma = Aux + DD + 62
     7.5. Caso 5 : Suma = Aux + DD + 92
     7.6. Caso 6 : Suma = Aux + DD + 123
     7.7. Caso 7 : Suma = Aux + DD + 153
     7.8. Caso 8 : Suma = Aux + DD + 184
     7.9. Caso 9 : Suma = Aux + DD + 215
     7.10. Caso 10 : Suma = Aux + DD + 245
     7.11. Caso 11 : Suma = Aux + DD + 276
     7.12. Caso 12 : Suma = Aux + DD + 306
5. Fin_Caso
9. Escribir "Los días transcurridos son: ", Suma
10. Terminar el proceso
```

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int DD, MM, AA, Suma,Aux;
cout<<endl<<"Ingrese día: ";cin>>DD;
cout<<endl<<"Ingrese mes: ";cin>>MM;
cout<<endl<<"Ingrese año: ";cin>>AA;
if ((AA%4==0 && AA%400==0) || (AA%400==0))
     Aux = 29;
else
     Aux = 28;
switch(MM)
{case 1 : Suma = DD;break;
 case 2 : Suma = DD +31;break;
 case 3 : Suma = Aux + DD +31;break;
 case 4 : Suma = Aux + DD + 62;break;
 case 5 : Suma = Aux + DD + 92;break;
 case 6 : Suma = Aux + DD + 123;break;
```

```
    case 7 : Suma = Aux + DD + 153;break;
    case 8 : Suma = Aux + DD + 184;break;
    case 9 : Suma = Aux + DD + 215;break;
    case 10 : Suma = Aux + DD + 245;break;
    case 11 : Suma = Aux + DD + 276;break;
    case 12 : Suma = Aux + DD + 306;break;
   }
   cout<<endl<<"Dias transcurridos: "<<Suma;
   getch();}
```

Nótese que los días del mes de febrero no lo calculamos en la sentencia "En caso de", dado que si lo hiciéramos así, tendríamos que hacer la comparación desde el mes de marzo hasta diciembre y sería redundante.



Capítulo III

Estructuras Lógicas Repetitivas

Este capítulo contiene:

- Estructura Hacer Mientras
- Estructura Mientras
- Estructura Desde
- Ejemplos desarrollados

Introducción

Es muy común encontrar en los algoritmos operaciones que se deben ejecutar un número repetido de veces. Si bien las instrucciones son las mismas, los datos sobre los que se opera varían. El conjunto de instrucciones que se ejecuta repetidamente se llama ciclo.

Todo ciclo debe terminar de ejecutar luego de un número finito de veces, por lo que es necesario en cada iteración del mismo, evaluar las condiciones necesarias para decidir si debe seguir ejecutándose o debe detenerse. En todo ciclo, siempre debe existir una condición de parada o fin de ciclo.

En algunos algoritmos podemos establecer a priori que el ciclo se repetirá un número definido de veces. Es decir, el número de repeticiones no dependerá de las proposiciones dentro del ciclo. Llamaremos ***Desde*** a la escritura algorítmica repetitiva que se ejecuta un número definido de veces.

Por otra parte, en algunos algoritmos no podemos establecer a priori el número de veces que ha de ejecutar el ciclo, sino que este número dependerá de las proposiciones dentro del mismo. Llamaremos ***mientras*** a la estructura algorítmica repetitiva que se ejecuta mientras la condición evaluada resulta verdadera. También tenemos la estructura *Hacer Mientras*.

En la práctica tanto la estructura *Hacer Mientras* como la estructura *Mientras* se pueden usar indistintamente para obtener un resultado.

Estructura *Hacer Mientras*

La estructura *Hacer Mientras* como lo señalamos anteriormente, es la

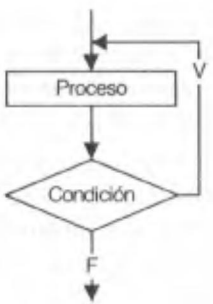
estructura algoritmica adecuada para utilizar en un ciclo que se ejecutará un número definido de veces. Por ejemplo cuando necesitamos calcular la nómina total de la empresa, tenemos que sumar los sueldos de los N empleados de la misma. Cuando necesitamos obtener el promedio de calificaciones de un curso, debemos sumar las N calificaciones de los alumnos y dividir esa suma entre N. Es decir, sabemos de antemano cuántas veces tenemos que repetir una determinada operación, acción o tarea. El número de repeticiones no depende de las proposiciones dentro del ciclo. El número de veces se obtiene del planteamiento del problema o de una lectura que indica que el número de iteracciones se debe realizar para N ocurrencias.

La sintaxis para esta estructura es la siguiente:

```
Hacer
      {Proceso}
Mientras (Condición)
{Fin del ciclo}
```

Proceso, es cualquier operación o conjunto de operaciones que deban ejecutarse mientras se repita el ciclo. El ciclo se repetirá siempre que la Condición sea verdadera, es decir se cumplirá hasta que la Condición sea falsa.

Esta estructura garantiza por lo menos una iteración del bloque de proceso, debido a que primero realiza el proceso y al final evalúa la condición.



Muchas veces usaremos esta estructura con el uso de un contador, y la condición estará ligada a dicho contador, obteniendo así la siguiente sintaxis:

```
V = Vi
Hacer
```



```
      {proceso}
      Calcular V = V + INC
Mientras que (Condición)
{Fin del ciclo}
```

Donde *V* : es una variable de control
Vi : es el valor inicial.
INC : es el incremento

V (contador del ciclo, generalmente representado por las letras I, J, K, V) toma un valor inicial (Vi) y se compara con el valor esperado en la condición. El ciclo se ejecuta hasta que la condición sea verdadera. El valor de V se incrementa en cada iteración.

El incremento de INC por lo general es en un unidad, pero eso depende mucho de la naturaleza del problemas, veremos casos en que el incremento será de dos unidades o de tres unidades, etc , en general podremos tener un incremento en X unidades, donde X es un número entero.

No es una regla que los valores Vi, V, VF e INC sean enteros, veremos casos en que será necesario un aumento de 0.02 por ejemplo, en ese caso dichas variables tendrán que ser reales.

En resumen las variables de control son enteras o son reales o cualquier otro tipo, pero no pueden ser una mezcla de ambas.

Por otro lado no siempre tendremos incrementos, si no decrementos, en este caso el algoritmo sufre un pequeño cambio, como mostramos a continuación

```
V=Vi
Hacer
      {proceso}
      Calcular V = V - INC
Mientras (Condicion)
{Fin del ciclo}
```

Veamos a continuación el siguiente ejemplo:

Ejemplo 3.1

Construya un algoritmo, que dados como datos los sueldos de los 10 trabajadores de una empresa, obtenga el total de nómina de la misma. Considere además que no puede utilizar estructuras algoritmicas repetitivas en la solución del problema.

Solución

La restricción de no usar estructuras repetitivas, es para ver como se solucionaría el ejemplo con lo que hemos aprendido hasta el momento, luego haremos una comparación, para ver cual es mas práctica. El algoritmo es el siguiente:

Algoritmo

Declarar variables

Leer Sueldos

Calcular la nómina

Mostrar nómina

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          SUE1, SUE2, SUE3, SUE4, SUE5  : Real
          SUE6, SUE7, SUE8, SUE9, SUE10 : Real
          Nomina : Real
3. Leer SUE1, SUE2, SUE3, SUE4, SUE5, SUE6, SUE7,
      SUE8, SUE9, SUE10
4. Calcular Nomina = SUE1 + SUE2 + SUE3 + SUE4 + SUE5
                   + SUE6 + SUE7 + SUE8 + SUE9 + SUE10
5. Escribir 'El total de la nómina es', Nomina
6. Terminar el proceso
```

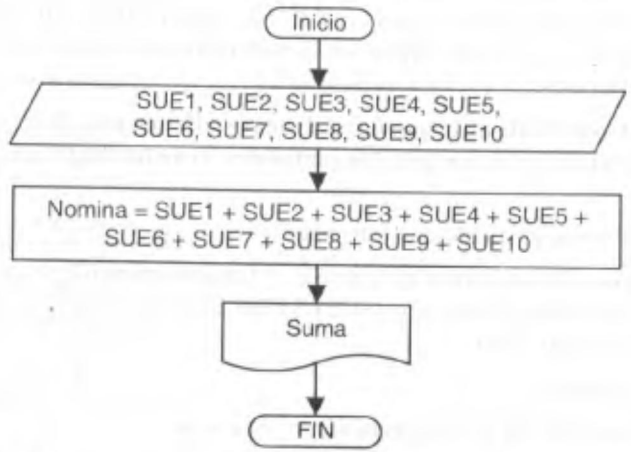
Como podemos ver, tuvimos que declarar 10 variables para poder calcular la nómina de 100 trabajadores, un mal uso de recursos a la hora de programar, pero aun si fuera el caso de 100 o 1000 trabajadores, este forma de desarrollar el ejemplo resulta inútil. En el ejercicio siguiente observaremos el uso de la estructura *Hacer Mientras*.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
float SUE1, SUE2, SUE3, SUE4, SUE5, SUE6, SUE7, SUE8,
      SUE9, SUE10, Nomina;
clrscr();
```



```
cout<<"Ingrese datos: ";
cout<<endl<<"Sueldo: ";
cin>>SUE1>>SUE2>>SUE3>>SUE4>>SUE5>>SUE6>>SUE7>>SUE8>>SUE9>>SUE10;
Nomina = SUE1 + SUE2 + SUE3 + SUE4 + SUE5 + SUE6 +
         SUE7 + SUE8 + SUE9 + SUE10;
cout<<"El total de la nómina es: "<<Nomina;
getch();}
```



Ejemplo 3.2

A continuación presentamos la solución del problema anterior, utilizando una estructura algoritmica *Hacer Mientras*.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          I          : Entero
          Nomina,SUE : Real
3 Hacer I = 0
     NOMINA = 0
4. Hacer
   4.1. Leer SUE
   4.2. Calcular NOMINA = NOMINA + SUE
   4.3. I = I + 1
5. Mientras I < 10  {Fin del ciclo del paso 2}
```

```
6. Escribir NOMINA
7. Terminar el proceso
```

Modificando la condición (I = 10), podemos utilizar esta solución para N empleados, incluso podemos leer el valor de N y formalizar la condición como (I = N).

I representa la variable de control del ciclo. Contabiliza el número de veces que ha de repetirse una determinada acción. El contador toma un valor inicial (generalmente 0 ó 1) y se incrementa en la mayoría de los casos en una unidad en cada vuelta del ciclo.

NOMINA representa un *acumulador*. Este se utiliza cuando debemos obtener el total acumulado de un conjunto de cantidades. Generalmente se inicializa en cero.

SUE Representa el sueldo del trabajador.

Realicemos un ejemplo para ver como es el funcionamiento de esta estructura, utilicemos los siguientes datos para (SUE): $1500, $890, $700, $950, $2300, $1650, $1800, $1400, $760, $900.

3. I=0 y Nomina=0

Primera iteración de la estructura *Hacer Mientras*

4. Hacer

4.1. Leer SUE ==> SUE = 1500

4.2. NOMINA = NOMINA + 1500 ==> 0 +1500 ==> 1500

4.3. I = I+1 ==> 0+1 ==> 1

5. Mientras I < 10 ==> 1< 10 (Verdad, se repite bucle)

Segunda iteración de la estructura *Hacer Mientras*, los valores anteriores de las variables son I=1, NOMINA=1500

4. Hacer

4.1. Leer SUE ==> SUE = 890

4.2. NOMINA = NOMINA + 1500 ==> 1500 + 890 ==> 2390

4.3. I = I+1 ==> 1+1 ==> 2

5. Mientras I < 10 ==> 2 < 10 (Verdad, se repite bucle)

Tercera iteración de la estructura *Hacer Mientras*, los valores anteriores de las variables son I=2, NOMINA=2390

4. Hacer

4.1. Leer SUE ==> SUE = 700

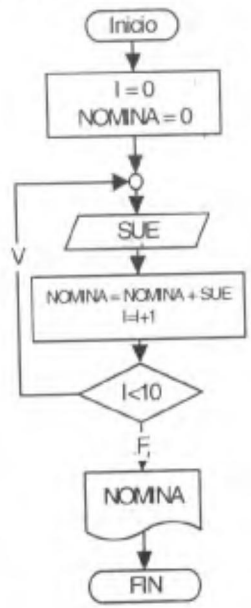
4.2. NOMINA = NOMINA + 1500 ==> 2390 + 700 ==>3090

4.3. I = I+1 ==> 2+1 ==> 3

5. Mientras I < 10 ==> 3 < 10 (Verdad, se repite bucle)

El ciclo se repite hasta que I tenga el valor de 10, entonces la condición I<10 será falsa y terminará el ciclo.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int I;
float NOMINA,SUE;
clrscr();
I = 0;
NOMINA = 0;
do
{     cout<<endl<<"Ingrese sueldo
                "<<I+1<<" :";
      cin>>SUE;
 NOMINA = NOMINA + SUE;
 I = I + 1;
}while(I<10);
cout<<"El total de la nómina es:
     "<<NOMINA;
getch();}
```

Observemos a continuación otro ejemplo:

Ejemplo 3.3

Escriba un pseudocódigo, que dados como datos N números enteros, obtenga el número de ceros que hay entre estos números.

Algoritmo

Declarar variables

Leer cantidad de números a leer

Hacer

Comprobar si número leido es cero, de ser así agregar una unidad al contador de ceros

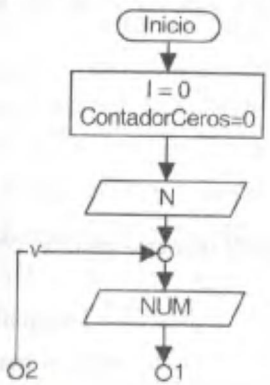
Mientras que no se cumplan el total de número leidos.

Mostrar cantidad de ceros leidos

Pseudocódigo

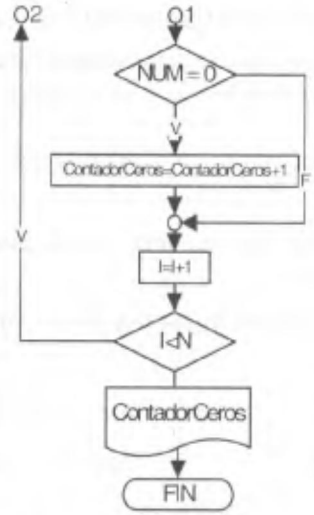
```
1. Iniciar proceso
2. Declarar Variables
                I, ContadorCeros, N, NUM : Entero
3. Hacer I = 0
       ContadorCeros = 0
4. Leer N
5. Repetir
     5.1. Leer NUM
     5.2. Si NUM = 0 Entonces
          5.2.1. Hacer ContadorCeros = ContadorCeros + 1
     5.3. Fin_Si
     5.4. Calcular I = I + 1
6. Hasta que I = N
7. Escribir ContadorCeros
8. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int I, ContadorCeros, N, NUM;
clrscr();
I = 0;
ContadorCeros = 0;
cout<<endl<<"cantidad números:";
cin>>N;
do{
```


```
      cout<<endl<<"Número
           "<<I+1<<" :";
      cin>>NUM;
      if (NUM == 0)
           ContadorCeros++;
      I = I + 1;
}while (I<N);
cout<<"Cant. ceros es: "
           <<ContadorCeros;
getch();
}
```

I representa al contador del ciclo.

N indica la cantidad de números que serán leidos

NUM es el número leido, el cual comprobaremos si es o no, cero.

ContadorCeros, es un contador. Cuenta el número de ceros ingresados.

Nótese en la codificación en C++ el uso del operador ++ en:

ContadorCeros++;

que es lo mismo que decir

ContadorCeros = ContadorCeros + 1;

Dicho operador aumenta en una unidad al contenido de la variable.

Estructura Mientras

La estructura algoritmica *mientras* es la estructura adecuada para utilizar en un ciclo cuando no sabemos el número de veces que éste se ha de repetir. Dicho número depende de las proposiciones dentro del ciclo. Ejemplos en la vida cotidiana encontramos muchos. Por ejemplo, supongamos que tenemos que obtener el total de una serie de gastos, pero no sabemos exactamente cuántos son; o cuando tenemos que sacar el promedio de calificaciones de un examen, pero no sabemos precisamente cuántos alumnos lo aplicaron. Tenemos que sumar las calificaciones e ir contando el número de alumnos, ésto con el fin de poder obtener posteriormente el promedio. El ciclo se repite mientras tengamos calificaciones de alumnos.

En la estructura *mientras* se distinguen dos partes:

- *Ciclo:* Conjunto de instrucciones que se ejecutarán repetidamente.
- *Condición de terminación:* La evaluación de esta condición permite decidir cuando finalizará la ejecución del ciclo. La condición se evalúa al inicio del mismo.

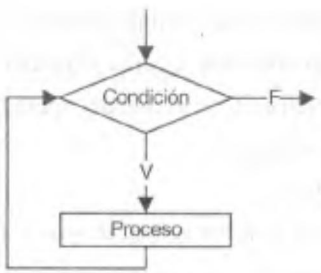
Esta estructura se ejecuta mientras que la condición es verdadera, en caso contrario terminará el ciclo.

Debe existir también un anunciado dentro del ciclo que afecte la condición, para evitar que el ciclo se ejecute indefinidamente.

En lenguaje algorítmico de la estructura *mientras* la expresamos de esta forma:

Algoritmo

```
Mientras Condición
      {PROCESO}
Fin_Mientras
```



Veamos a continuación el siguiente ejemplo.

Ejemplo 3.4

Supongamos que debemos obtener la suma de los gastos que hicimos en nuestro último viaje, pero no sabemos exactamente cuántos fueron.

Solución

Como resolver un ejercicio donde no se saben cuantos datos serán leídos, es simple, debemos de ingresar un valor que indique el término de los valores ingresado, puede usarse un artificio en este caso, por ejemplo podemos ingresar el valor -1 como la cantidad que indique el fin, podemos usar -1 ya que no es ningún valor válido que indique un gasto echo en el viaje.

Datos : Gasto1, Gasto2, ..., -1

Donde : Gasto(i) es una variable de tipo real, que representa el gasto número

Algoritmo

Declarar variables

Inicializar acumulador de gastos

Leer Primer gasto

Mientras gasto sea válido

Incrementar acumulador con el valor de gasto

Leer nuevo gasto

Reportar total de gastos

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                SUMAGAS, GASTO : Real
3. Hacer SUMAGAS = 0
4. Leer GASTO
5. Mientras GASTO <> -1
     5.1. Calcular SUMAGAS = SUMAGAS + GASTO
     5.2. Leer GASTO
6. Fin_Mientras{Fin de ciclo del paso 5}
7. Escribir SUMAGAS
8. Terminar proceso
```

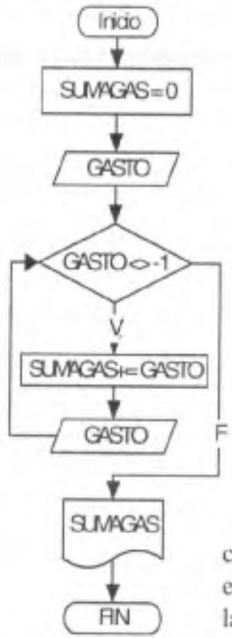
SUMAGAS es un acumulador. Acumula los gastos efectuados.

GASTO su valor en la primera lectura debe ser verdadero, es decir distinto de -1. Su valor se modifica en cada vuelta del ciclo. Cuando gasto tome el valor de -1, entonces el ciclo se detendrá.

Nótese que la condición de la estructura Mientras es que Gasto debe ser diferente de -1, mientras que la condición sea verdadera, se irá acumulando el valor de Gasto ingresado, y se pedirá ingresar un dato nuevo para Gasto.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
```




```
void main()
{
float SUMAGAS, GASTO;
clrscr();
SUMAGAS = 0;
cout<<endl<<"Ingrese Gasto: ";
cin>>GASTO;
while(GASTO != -1){
 SUMAGAS+= GASTO;
 cout<<endl<<"Ingrese Gasto: ";
 cin>>GASTO;
}
cout<<"Total gastos: "<<SUMAGAS;
getch();
}
```

Hay que resaltar el uso del operador != en la codificación C++, este operador significa diferencia entre dos componentes, por otro lado tenemos también la siguiente línea:

```
SUMAGAS+= GASTO;
```

la cual es la forma abreviada de:

```
SUMAGAS = SUMAGAS + GASTO;
```

Ejemplo 3.5

Escriba un pseudocódigo que dado un grupo de números naturales positivos, calcule e imprima el cubo de estos números.

Solución

En este ejercicio podemos usar un artificio similar al del anterior ejemplo, ya que solo nos piden números enteros positivos, por lo tanto cualquier número negativo nos podría servir como medio de comparación, entonces solo nos bastará la comprobación que el número es positivo en caso contrario se terminará el ciclo.

Algoritmo

Declarar variables

Leer Primer número

Mientras número sea válido

Hallar el cubo

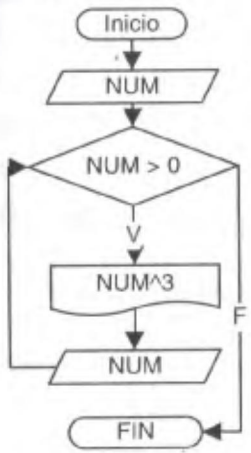
Mostrar cubo

Leer nuevo número

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                NUM : Entero
3. Leer NUM
4. Mientras (NUM > 0)
     4.1. Escribir NUM^3
     4.2. Leer NUM
5. Fin_Mientras {Fin del ciclo del paso 4}
6. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main(){
int NUM;
clrscr();
cout<<endl<<"Ingrese Número: ";
cin>>NUM;
while(NUM > 0){
 cout<<endl<<"El cubo es :
                "<<(pow(NUM,3));
 cout<<endl<<"Ingrese Número: ";
 cin>>NUM;
}}
```

En el código no hemos utilizado algún tipo de cálculo previo del cubo del número ingresado, este cálculo lo realizamos al momento de mostrar el resultado, de este modo nos ahorramos algunas líneas de código.

A continuación presentamos una serie de problemas diseñados expresamente como elementos de ayuda para el análisis y la retención de los conceptos. Además se utiliza en muchos de ellos.

Ejemplo 3.6

En un supermercado una ama de casa pone en su carrito los artículos que va tomando de los estantes. La señora quiere asegurarse de que el cajero le cobre bien lo que ella ha comprado, por lo que cada vez que toma un artículo anota su precio junto con la cantidad de artículos iguales que ha tomado y determina cuanto dinero gastara en ese articulo; a esto le suma lo que ira gastando en los demás artículos, hasta que decide que ya tomo todo lo que necesitaba. Ayúdale a esta señora a obtener el total de sus compras.

Algoritmo

Leer la cantidad de artículos y su precio

Calcular el total a pagar por dichos artículos y sumar el resultado al total general

Comprobar si el precio y la cantidad son válidos o no

Mostrar el total de la compra

Pseudocódigo

Declarar e inicializar variables

Hacer

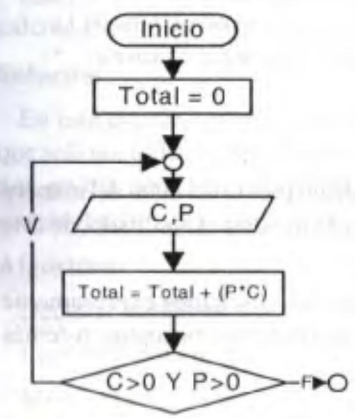
Leer P y C

Total = Total + (P*C)

Mientras (C>0 Y P>0)

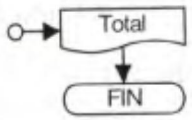
Escribir Total

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
float Total, P, C;
clrscr();
Total=0;
do{
   cout<<endl<<"Precio: ";cin>>P;
   cout<<endl<<"Cantidad: ";cin>>C;
   Total+=(P*C);
```


```
}while(C>0 && P>0);
cout<<endl<<"El Total es : "<<Total;
getch();}
```

Ejemplo 3.7

Determinar la cantidad semanal de dinero que recibirá cada uno de los N obreros de una empresa. Se sabe que cuando las horas que trabajo un obrero exceden de 40, el resto se convierte en horas extras que se pagan al doble de una hora normal, cuando no exceden de 8; cuando las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se paga por una hora normal y el resto al triple. Considere que todos los obreros ganan el mismo sueldo ($200)

Algoritmo

Declarar variables y constantes

Hacer

Leer número de horas trabajadas

Dependiendo del número total de horas trabajadas calcular el sueldo final

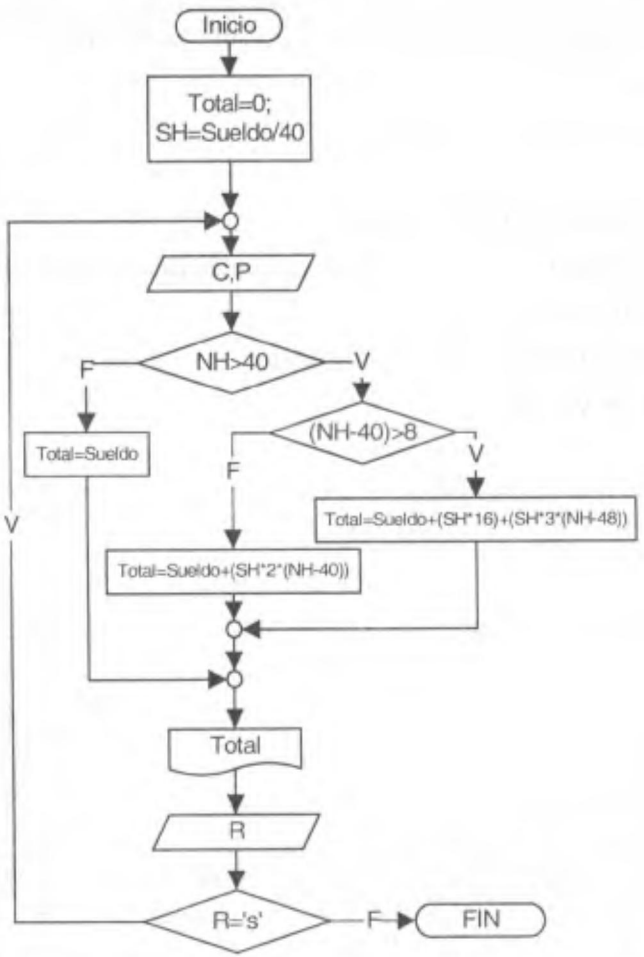
Mostrar sueldo

Mientras variable de condición sea falsa

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#define Sueldo 200
void main()
{
float Total, SH;
int NH;
char R;
clrscr();
Total=0;
SH=Sueldo/40;
do{
 cout<<endl<<"Cálculo de sueldos";
 cout<<endl<<"Nº de Horas: ";cin>>NH;
```

```
if(NH>40)
     if((NH-40)>8)
          Total=Sueldo+(SH*16)+(SH*3*(NH-48));
     else
          Total=Sueldo+(SH*2*(NH-40));
else
     Total=Sueldo;
cout<<endl<<"Total a pagar: "<<Total;
cout<<endl<<"Desea continuar? (s/n): ";cin>>R;
}while(R=='s');}
```



Ejemplo 3.8

Calcular la suma siguiente:

100 + 98 + 96 + 94 + . . . + 0 en este orden

Algoritmo

Declarar variables y constantes

Inicializar variables

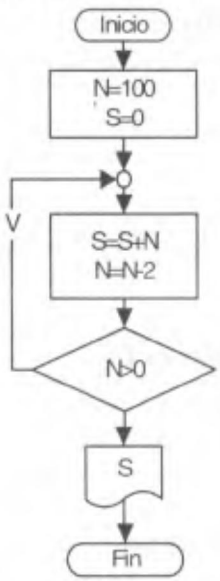
Hacer

Calcular suma y decrementar el número en dos unidades

Mientras número sea mayor que cero.

Mostrar suma

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float S;
int N;
clrscr();
N=100;
S=0;
do{
 S+=N;
 N-=2;
}while(N>0);
cout<<"La suma es: "<<S;
getch();
}
```

Ejemplo 3.9

Construya un pseudocódigo que calcule e imprima la suma de los N primeros números naturales.

Solución

En este caso deberemos usar otra forma para la condición, usaremos un

cantador y lo compararemos con N, si el contador es mayor que el valor de N significa que ya sean leido todos los números menores que N, inclusive este.

Algoritmo

Declarar variables

Leer Primer número

Mientras número sea válido

Acumular número

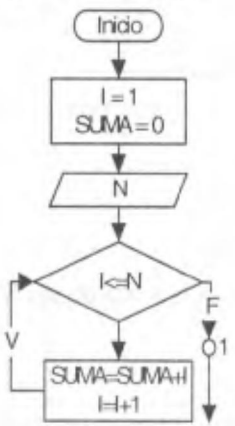
Leer nuevo número

Mostrar suma de lo N primero números

Pseudocódigo

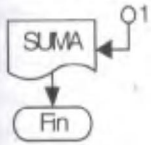
```
1. Iniciar proceso
2. Declarar Variables
                I, N, SUMA : Enteros
3. Hacer I = 1 y Suma = 0
4. Leer N
5. Mientras I <= N
     5.1. Calcular SUMA = SUMA + I
                  I = I + 1
4. Fin_Mientras {Fin del ciclo del paso 5}
5. Escribir SUMA
6. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int I, N, SUMA;
clrscr();
I = 1;
SUMA = 0;
cout<<"Ingrese número : ";
cin>>N;
while(I<=N)
{ SUMA+=I ;
```


```
I++;}
cout<<"La suma es: "<<SUMA;
getch();
}
```

N, representa el número de números naturales que se van a ingresar.

I, representa al contador del ciclo.

SUMA, acumulador, acumula la suma de los N números naturales.

Ejemplo 3.10

Se tienen las calificaciones de un grupo de alumnos que presentaron un examen. El profesor desea obtener el promedio de estas calificaciones. Escriba un pseudocódigo para resolver lo planteado.

Solución

En este ejercicio podemos usar el mismo artificio que en el ejemplo 3.4, esto es posible ya que las notas son números positivos.

Algoritmo

Declarar variables

Leer nota

Mientras Nota sea válida

Sumar notas

Leer nuevo número

Calcular promedio

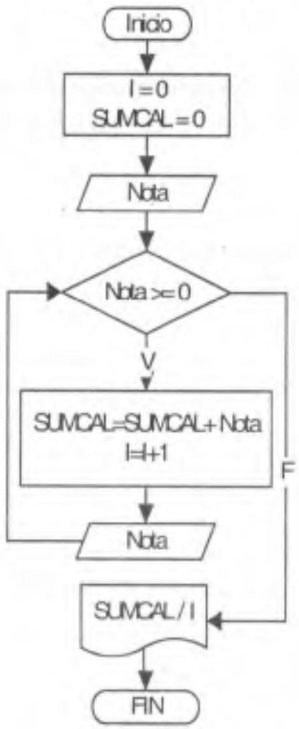
Reportar Promedio

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          I   : Enteros
          Nota,SUMCAL : Real
3. Hacer I = 0
         SUMCAL = 0
4. Leer Nota
5. Mientras (Nota >= 0)
```

```
        5.1. Calcular SUMCAL = SUMCAL + Nota
                       I = I + 1
        5.2. Leer Nota
    6. Fin_Mientas
    7. Escribir SUMCAL/I
    8. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int I;
float Nota,SUMCAL;
clrscr();
I = 0;
SUMCAL = 0;
cout<<"Ingrese nota: ";
cin>>Nota;
while(Nota >= 0)
{
 SUMCAL+= Nota;
 I++;
 cout<<"Ingrese nota: ";
cin>>Nota;
}
cout<<"Promedio:"<<(SUMCAL/I);
getch();
}
```

Nótese que es necesario llevar la contabilidad del número de notas ingresadas, es por ello que usamos el contador I.

SUMCAL acumula las calificaciones.

Nota expresa la calificación del alumno i

PRO almacena el promedio de las calificaciones.

Ejemplo 3.11

Escriba un pseudocódigo, que dados como datos 270 números enteros, obtenga la suma de los números pares y el promedio de los números impares. Además indique cuantos ceros se ingresaron.

Algoritmo

Declarar variables

Inicializar contadores y acumuladores

Mientras contador sea menor a 270

Leer número

Si el número es par, sumar números pares

Si el número es impar, sumar impares y contabilizar los número impares

Si el número es cero, contabilizar cuanto ceros se ingresan.

Calcular promedio de números impares

Reportar Promedio de números de impares, suma números pares y la cantidad de ceros que fueron ingresados.

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                SUMPAR, SUMIMPAR, PROIMPAR : Real
                CONIMPAR, CONCEROS, I, NUM : Entero
3. Hacer I = 0
          SUMPAR = 0
          SUMIMP = 0
          CONCEROS = 0
          CONIMPAR = 0
4. Mientras I < 270
      4.1. Leer NUM
      4.2. Si (NUM <> 0) Entonces
           4.2.1. Si (-1^NUM) > 0 Entonces
                4.2.1.1. Calcular SUMPAR = SUMPAR + NUM
           4.2.2. Si no
                4.2.2.1. Calcular CONIMPAR = CONIMPAR + 1
                                  SUMIMPAR = SUMIMPAR+NUM
```

```
            4.2.3. Fin_Si
        4.3. Si no
            4.3.1. Calcular CONCEROS = CONCEROS + 1
        4.4. Fin_Si
        4.5. Calcular I = I + 1
    5. Fin_Mientras
    6. Calcular PROIMPAR = SUMIMPAR / CONIMPAR
    7. Escribir PROIMPAR, SUMPAR, CONCEROS
    8. Terminar proceso
```

Podemos sumar un número entero con un real si la respuesta es almacenada en una variable real, pero sería erróneo si dicha suma se almacena en una variable entera.

SUMPAR y **SUMIMPAR**, son acumuladores. Acumulan los números pares e impares, respectivamente.

CONIMPAR, es un contador. Cuenta la cantidad de números pares.

PROIMPAR, Almacena el promedio de los números pares.

La condición:

```
Mientras I < 270
```

llega a exactamente a 270 si I empieza en cero, en el caso que I empiece en 1, la condición debería ser

```
Mientras I <= 270
```

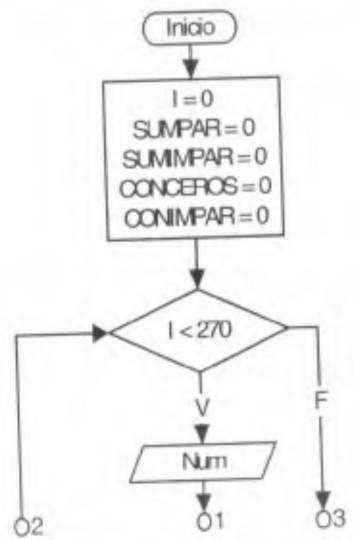
Por otro lado tenemos la expresión (-1^NUM), si elevamos -1 a un número par este siempre será positivo , por el contrario si el número es impar el resultado es negativo, esta es una manera práctica de saber si un número es par o impar.

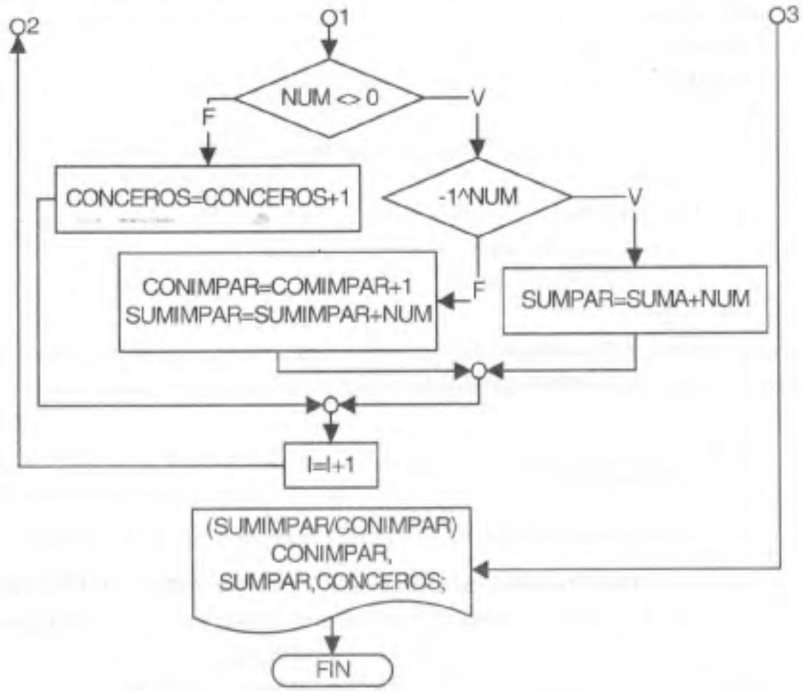
Codificación en C++

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main(){
float SUMPAR, SUMIMPAR;
int CONIMPAR, CONCEROS, I, NUM;
clrscr();
I = 0;
SUMPAR = 0;
```



```
SUMIMPAR = 0;
CONCEROS = 0;
CONIMPAR = 0;
while (I < 270)
{    cout<<endl<<"Ingrese número "<<I+1<<": ";
 cin>>NUM;
 if (NUM != 0)
      if (pow(-1,NUM) > 0)
           SUMPAR+=NUM;
      else{
           CONIMPAR++;
           SUMIMPAR+=NUM;
      }
 else
      CONCEROS++;
 I++;
}
 cout<<endl<<"Promedio impares: "<<(SUMIMPAR/CONIMPAR);
 cout<<endl<<"cantidad de números impares: "<<CONIMPAR;
 cout<<endl<<"suma de pares: "<<SUMPAR;
 cout<<endl<<"cantidad de ceros: "<<CONCEROS;
 getch();}
```





Ejemplo 3.12

Hacer un pseudocódigo para obtener la tabla de multiplicar de un número entero K, comenzando desde 1.

Algoritmo

Declarar variables

Inicializar contador

Leer el valor de K

Mientras contador sea menor a K

Leer número

Calcular tabla de multiplicar

Imprimir tabla

Pseudocódigo

```
1. Iniciar proceso
```



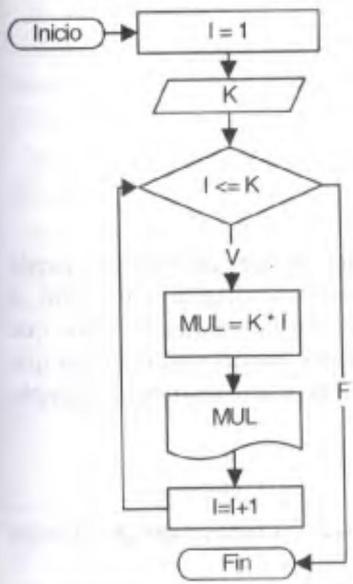
```
2. Declarar Variables
                K, I : Entero
                MUL  : Real
3. Hacer I = 1
4. Leer K
5. Mientras I <= K
      5.1. Calcular MUL = K * I
      5.2. Escribir K, "*", I, "=", MUL
      5.3. Hacer I = I + 1
6. Fin_Mientras
7. Terminar proceso
```

K expresa el número entero del cual queremos obtener la tabla de multiplicar.

MUL Almacena el resultado de la multiplicación, del tipo real debido a que dependiendo del valor de K, MUL puede ser muy grande, rebasando los límites de un número entero.

En este caso el contador I debió ser inicializado con 1 y no con 0, como lo hemos estado viendo en ejemplos anteriores, esto debido a las condiciones propias del problema .

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int K, I;
float MUL;
clrscr();
I = 1;
cout<<endl<<"ingrese número: ";
cin>>K;
while (I <= K)
{MUL = K * I;
 cout<<K<<"x"<<I<<"="<<MUL;
 I++;
}
getch();
}
```

Estructura Desde

Cuando se sabe el número de iteraciones que tendrán que ejecutarse cierta cantidad de acciones, es recomendable usar la estructura Desde, esta estructura no solo ejecuta un número de acciones, además cuenta internamente el número de iteraciones, esto elimina la necesidad de un contador, lo que no sucede con las estructuras Mientras y Hacer Mientras.

A continuación veremos el pseudocódigo de flujo con su respectivo algoritmo para la estructura:

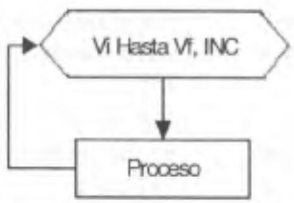
```
Desde I = Vi Hasta Vf INC
 Hacer
      Acciones
Fin_Desde
```

I: Variable de tipo entero, representa la variable de control del ciclo.

Vi: Variable de tipo entero, representa el inicio del ciclo.

Vf: Variable de tipo entero, representa el final del ciclo.

INC: Expresión de tipo entero. Representa el incremento/decremento de la variable de control, cuando se omite esta expresión se supone que el incremento es en una unidad.



Como podemos observar esta estructura es fácil de usar, además nos puede ahorrar muchas líneas de código e iteraciones en el diagrama de flujo, a comparación de las otras estructuras repetitivas. En la práctica veremos que muchos ejercicios podrán ser resueltos con las tres estructuras, mientras que para resolver otros, tendremos que optar por una de ellas. Veamos un ejemplo sencillo para entender mejor la estructura Desde.

Ejemplo 3.13

Haga un pseudocódigo para obtener la suma de los 20 primeros números naturales enteros positivos.

Algoritmo

Declarar variables

Inicializar acumulador

Desde I=1 hasta 20 hacer

Acumular el valor de I

Reporta sumatoria

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
                I : Entero
                S : Real
3.Hacer S = 0
4. Desde I = 1 Hasta 20 INC I = I + 1 Hacer
     4.1. S = S + I
5. Fin_Desde {Fin del ciclo del paso 4}
6. Imprimir S
7. Terminar proceso
```

I: Variable de tipo entero. Representa la variable de control del ciclo.

S: Variable de tipo real, en ella se almacena la suma.

Nótese en la línea 4 que el contador está siendo inicializado en 1 y finaliza cuando llegue a 20, el contador irá avanzando en una unidad según I=I+1. Por otro lado en este ejemplo no se necesita ningún dato de entrada.

Hagamos un seguimiento de este ejemplo para ver como actúa la estructura Desde.

3. S = 0

4. Desde I = 1 Hasta 20 INC I = I + 1 Hacer

Primera iteración, I = 1

4.1. S = S + I ==> S = 0 + 1 ==> S = 1

Segunda iteración, I = 2

4.1. S = S + I ==> S = 1 + 2 ==> S = 3

Tercera iteración, I = 3

4.1. S = S + I ==> S = 3 + 3 ==> S = 6

Cuarta iteración, I = 4

4.1. S = S + I ==> S = 6 + 4 ==> S = 10

Quinta iteración, I = 5

4.1. S = S + I ==> S = 10 + 5 ==> S = 15

Sexta iteración, I = 6

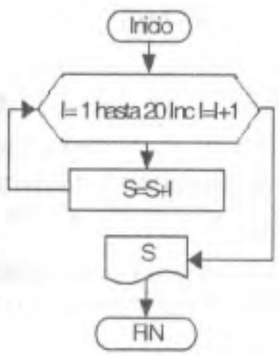
4.1. S = S + I ==> S = 15 + 6 ==> S = 21

Las iteraciones terminan cuando I=20

4.1. S = S + I ==> S = 190 + 20 ==> S = 210

5. Fin_Desde

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
float S;
clrscr();
S = 0;
for (int I= 1;I<=20;I++)
 S+=I;
cout<<endl<<"La suma es:
"<<S;
getch();
}
```

Veamos la sintaxis de la estructura for (Desde) en C++

```
for (int I=1;I<=20;I++)
```

Primero tenemos el valor de inicio I=1, seguido por la expresión que al ser falsa indicará el término del ciclo, por lo tanto tenemos que si I<=20 es falso termina el bucle for, pero ¿porque no poner hasta I==20? la respuesta es simple, C++ define que la estructura for debe ser usada con esa sintaxis, por lo tanto el sin del bucle siempre estará dado por una expresión de comparación que para este ejemplo es I<=20. Por otro lado C++ permite la declaración de la variable a usar dentro del la sintaxis de la estructura for.

Intente realizar este sencillo ejemplo con las estructuras Mientras y Hacer Mientras así verá la diferencia. Claro está que dependiendo de la naturaleza del

problema, usando esta estructura repetitiva se podría ahorrar varias líneas de código en un programa. Veamos un ejemplo más.

Ejemplo 3.14

Realice un pseudocódigo para identificar si un número ingresado es primo o no

Solución

Un número es primo cuando sólo puede ser divido entre la unidad y el mismo número, teniendo en cuenta esto podemos decir que si dividimos el número entre valores consecutivos empezando en la unidad y terminando en el número en mención, sólo debe de haber dos valores que dividan al número, si hay más de dos entonces el número no es primo.

Algoritmo

Declarar variables

Inicializar contador

Leer número

Desde I=1 hasta Número hacer

Comprobar si la división entre número e I tiene un residuo cero

Si es cierto entonces incrementar contador

Si el contador es menor que dos, entonces el número es primo

Pseudocódigo

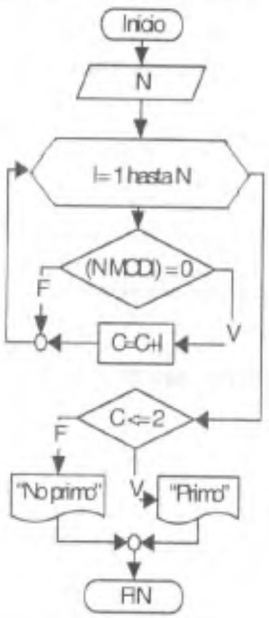
```
1. Iniciar proceso
2. Declarar Variables
                N, C, I : Enteros
3. Leer N
4. Hacer C = 0
5. Desde I = 1 Hasta N Hacer
     5.1. Si (N Mod I) = 0 Entonces
          5.1.1. Calcular C = C + 1
     5.2. Fin_Si
6. Fin_Desde{Fin del ciclo del paso 3}
7. Si c <= 2 Entonces
     7.1. Escribir 'Es primo'
8. Si no
     8.1. Escribir 'No es primo'
```

```
9. Fin_Si
10. Terminar proceso
```

C: Variable de tipo entero. Usado como contador, para determinar cuantas veces el número N es dividido exactamente.

Nótese que en la estructura Desde no hemos colocado el incremento, esto es cuando el incremento es en una unidad, podemos omitirla, quedando sobre entendido.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int N, C;
clrscr();
cout<<endl<<"Ingrese número:";
cin>>N;
C = 0;
for (int I= 1;I<=N;I++)
 if ((N%I) == 0)
      C++;
if (C <= 2)
 cout<<endl<<"Es primo";
else
 cout<<endl<<"No es primo";
getch();
}
```

Ejemplo 3.15

Leer el sueldo de 10 trabajadores, calcular el promedio de los sueldos, además reportar a cuanto asciende el sueldo más alto.

Solución

En este caso, debemos de aprovechar la estructura Desde para acumular los sueldos y a la vez hallar el sueldo más alto

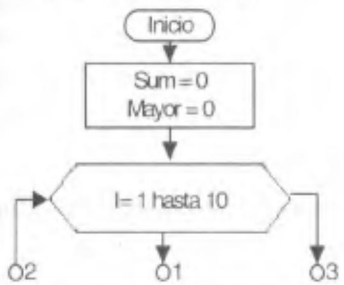
Algoritmo

Declarar variables
Inicializar Acumulador
Desde I=1 hasta 10 hacer
  Leer datos
  Acumular sueldos
  Comprobar sueldo mayor
Reportar resultados

Pseudocódigo

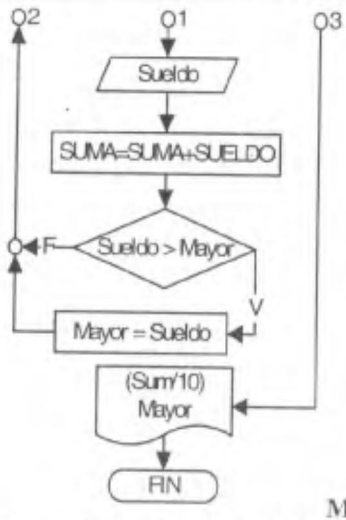
```
1. Iniciar proceso
2. Declarar Variables
          Nombre : Cadena de caracteres
          Sum, Mayor, Sueldo: Real
3. Hacer Sum = 0
        Mayor = 0
5. Desde I = 1 Hasta 10 Hacer
     5.1. Leer Nombre, Edad, Sueldo
     5.2. Calcular Sum = Sum + Sueldo
     5.3. Si Sueldo > Mayor Entonces
          5.3.1. Hacer Mayor = Sueldo
     5.4. Fin_Si
6. Fin_Desde
7. Imprimir 'El promedio es: ', (Sum / 10)
            'El mayor de los sueldos es: ', Mayor
8. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
float Sum, Mayor,Sueldo;
clrscr();
Sum = 0;
Mayor = 0;
```




```
for (int I= 1;I<=10;I++)
{cout<<endl<<"Ingrese
                  sueldo:";
 cin>>Sueldo;
 Sum += Sueldo;
 if (Sueldo > Mayor)
     Mayor = Sueldo;
}
cout<<endl<<"El promedio :
           "<<(Sum/10);
cout<<endl<<"Sueldo mayor:
           "<<Mayor;
getch();
}
```

Mayor: Variable de tipo entero. Aquí se almacena el mayor de los sueldos.

Sum : Variable de tipo real en donde se almacena la suma de los sueldos.

Ejemplo 3.16

Dado 4 notas de un alumno, eliminar la menor y calcular el promedio de las tres notas restantes. Realice el algoritmo para aplicarlo a N alumnos.

Solución

Anteriormente vimos un ejemplo de como hallar la menor nota y excluirla del cálculo del promedio, aqui usaremos el mismo algoritmo, con una pequeña modificación.

Un alumno tendrá cuatro notas, eso nos indica el uso de una estructura Desde de 1 a 4, pero por otro lado son N alumnos, eso quiere decir que necesitaremos otra estructura Desde, de 1 hasta N, ambas estructuras deben ir anidadas.

Algoritmo

Declarar variables

Leer cantidad de alumnos

Desde I=1 hasta Número de alumnos hacer

Colocar como nota menor 21 e inicializar el acumulador de notas a cero

Desde J = 1 hasta numero de notas hacer

Si la nota leida es menor que la nota menor, actualizar nota menor

Calcular promedio

Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          Nota, S, Menor : Real
          N, I, J : Entero
3. Leer N
4. Desde I = 1 Hasta N Hacer
   4.1. Hacer Menor = 21
              S = 0
   4.2. Desde J = 1 Hasta 4
      4.2.1. Leer Nota
      4.2.2. Hacer S = S + Nota
      4.2.3. Si Nota < Menor Entonces
         4.2.3.1. Hacer Menor = Nota
      4.2.4. Fin_Si
   4.3. Desde
   4.4 Imprimir "El promedio es", (S - Menor)/3
5. Fin_Desde
6. Terminar proceso
```

I, J: Variable de tipo entero. Representa la variable de control del ciclo.

Menor: Variable de tipo real. Almacena la menor de las notas

Se coloca como la nota menor inicial 21, este artificio puede usarse, ya que las notas son en el sistema vigesimal, y el máximo es 20; entonces suponiendo que todas las notas de un alumno sean 20, se cumple que 20<21, entonces se actualizará la nota menor a 20.

Veamos como se ejecuta este pseudocódigo.

3. N = 3, suponiendo que son tres alumnos

4. Desde I = 1 Hasta 3 Hacer

Para I = 1 tenemos

4.1. Hacer Menor = 21

S = 0

Obtendremos el promedio del alumno I=1

4.2. Desde J = 1 Hasta 4 Hacer

Para J = 1 tenemos S=0 y Menor = 21

4.2.1. Leer Nota = 16

4.2.2. S = S + Nota ==> 0 + 16 ==> S = 16

4.2.3. Si 16 < 21 Entonces

4.2.3.1. Menor = 16

Para J = 2 tenemos, S=16 y Menor = 16

4.2.1. Leer Nota = 18

4.2.2. S = S + Nota ==> 16 + 18 ==> S = 34

4.2.3. Si 18 < 16 (Falso, Menor se queda en 16

Para J = 3 tenemos, S=34 y Menor = 16

4.2.1. Leer Nota = 10

4.2.2. S = S + Nota ==> 34 + 10 ==> S = 44

4.2.3. Si 10 < 16 Entonces Menor = 10

4.2.3.1. Menor = 16

Para J = 4 tenemos S=44 y Menor 10

4.2.1. Leer Nota = 15

4.2.2. S = S + Nota ==> 44 + 15 ==> S = 59

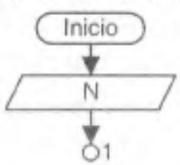
4.2.3. Si 10 < 16 (Falso)

4.3. Fin_Desde

4.4. Imprimir "El promedio es:", (S - Menor)/3 ==> (59-10)/3 ==> 16.33

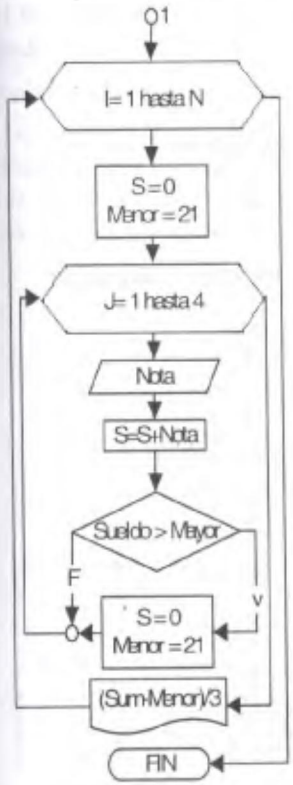
5. Fin_Desde

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
```


```
float Nota, S, Menor;
int N;
clrscr();
cout<<endl<<'Ingrese Cantidad
                de alumnos:";
cin>>N;
for (int I = 1;I<=N;I++)
{     Menor = 21;
      S = 0;
      for (int J=1;J<=4;J++)
      {    cout<<endl<<"Ingrese
                          Nota:";
           cin>>Nota;
           S+= Nota;
           if(Nota < Menor)
                Menor = Nota;
      }
      cout<<endl<<"El promedio
es: "<<((S-Menor)/3);
}
getch();
}
```

Luego de terminar el bucle **desde**, con la variable de control J, se devuelve el control al Desde principal con la variable I, entonces I = 2; nuevamente tendremos Menor = 21 y S = 0, ya que se trata de un nuevo alumno y por lo tanto se trata de un nuevo promedio.

Ejemplo 3.17

Calcular el menor y el mayor de una lista de N números enteros

Solución

Resultaría imposible usar el artificio anterior para hallar el menor número si no se sabe entre que límites se encuentra, en este caso podemos hacer que el primer número leído sea el menor y el mayor a la vez, este sería el mejor artificio para encontrar el menor y el mayor de una lista.

Algoritmo

Declarar variables

Leer primer número

Hacer que primer número sea el mayor y el menor a la vez

Desde I=2 hasta N hacer

Leer número

Comprobar si el leído es menor o el mayor de los leidos y actualizar

Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        Numero, I, J,N : Entero
3. Leer N
4. Leer Numero
5. Hacer Menor = Numero
       Mayor = Numero
6. Desde I = 2 Hasta N Hacer
     6.1. Leer Numero
     6.2. Si Numero < Menor Entonces
          6.1.1. Hacer Menor = Numero
     6.3. Si no
          6.3.1. Si Numero > Mayor Entonces
               6.3.1.1. Hacer Mayor = Numero
          6.3.2. Fin_Si
     6.4. Fin_Si
7. Fin_Desde
8. Escribir 'El número mayor es: ', Mayor
          'El número menor es: ', Menor
9. Terminar proceso
```

N: Numero de tipo entero. Representa la cantidad de números.

Numero: Número de tipo entero. Representa el número a ingresar.

Nótese que el bucle Desde empieza en 2, a diferencia de los anteriores vistos, esto se debe a que en la línea 4 ya leemos el primer valor de la lista, por lo tanto

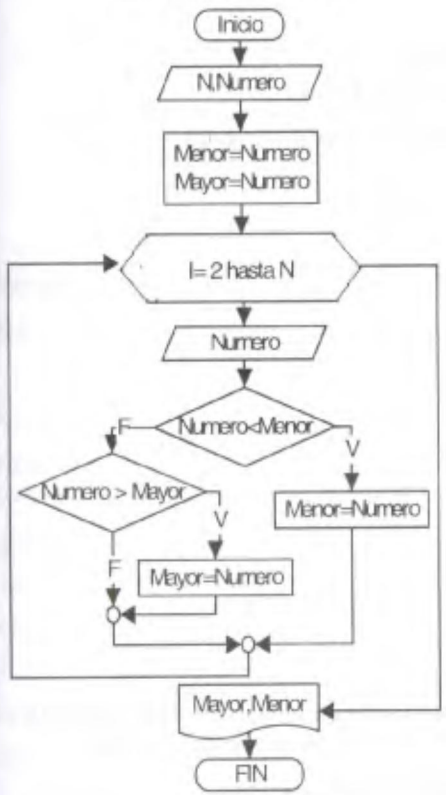
en el bucle Desde debemos empezar a leer los números de la lista a partir del segundo.

Se obtiene el mismo resultado reemplazando la línea 6 por la siguiente

```
6. Desde I = 1 Hasta N-1 Hacer
```

recuerde que la estructura Desde sólo la usamos para contabilizar los números que se van ingresando, por este motivo es indiferente, en este caso, usar cualquier forma, de las ya presentadas, para la estructura Desde.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int Numero, N,
      Menor,Mayor;
clrscr();
cout<<"Ingrese
      Cantidad de
      Números:";
cin>>N;
cout<<"Ingrese
      Número:";
cin>>Numero;
Menor = Numero;
Mayor = Numero;
for(int
      I=2;I<=N;I++){
 cout<<"Ingrese
            Número:";
 cin>>Numero;
 if (Numero<Menor)
      Menor = Numero;
 else
    if(Numero>Mayor)
       Mayor=Numero;
}
cout<<"El mayor es: "<<Mayor;
cout<<"El menor es: "<<Menor;
getch();}
```

Ejemplo 3.18

Hallar cuantos términos hay en la progresión aritmética mostrada a continuación, También halle la suma de los términos.

5, 8, 11, 14, 17, 20, 23,.....n

Algoritmo

Declarar variables

Inicializa acumuladores y contadores

Leer último término

Desde I=5 hasta N INC I=I+3 hacer

Acumular el valor de la progresión dado por I

Aumentar contador de la cantidad de número sumados.

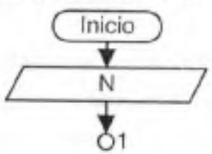
Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          Suma, N, I, C : Entero
3. Hacer C = 0
     Suma = 0
4. Leer N
5. Desde I = 5 Hasta N INC I = I + 3
     5.1. Suma = Suma + I
     5.2. C = C + 1
6. Fin_Desde
7. Escribir 'El número de términos es : ', N
          'La suma de los términos es: ', Suma
8. Terminar proceso
```

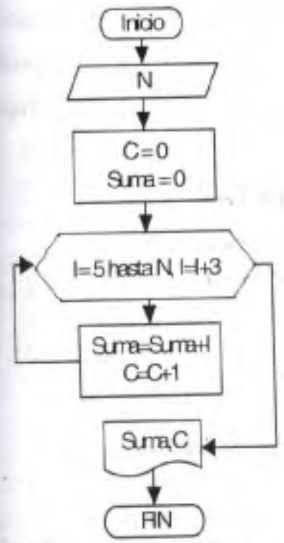
Véase el incremento de la estructura Desde, el cual es en 3 unidades, por cada corrida, esto es debido a que la razón de la progresión es de 3 unidades.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
```


```
int Suma, N, C;
clrscr();
C = 0;
Suma = 0;
cout<<endl<<"Ingrese Número:";
cin>>N;
for (int I=5;I<=N;I+=3)
{     Suma+= I;
      C++;
}
cout<<endl<<"El número de
          términos es : "<<C;
cout<<endl<<"La suma de los
          términos es: "<<Suma;
getch();
}
```

Ejemplo 3.19

Halle el número de término para la progresión siguiente:

1, 2, 4, 8, 16, 32, ..., 10000

Solución

Nótese que la progresión aritmética tiene una razón de I * 2, siendo I un término de la progresión, entonces tenemos:

Algoritmo

Declarar variables

Iniciar contador

Desde I=1 hasta 10000 INC I=I*2 hacer

Aumentar contador de la cantidad de número de la progresión.

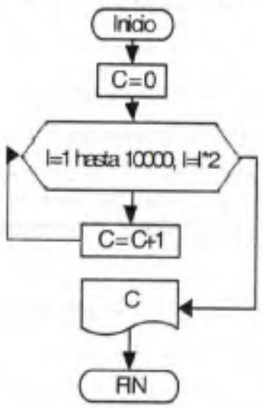
Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          I, C : Entero
3. C = 0
```

```
4. Desde I = 1 Hasta 10000 INC I = I * 2
     4.1. C = C + 1
5. Fin_Desde
6. Escribir 'El número de términos es : ', C
7. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int C;
clrscr();
C = 0;
for (int I=1;I<=10000;I*=2)
 C++;
cout<<endl<<"El número de
                términos es : "<<C;
getch();
}
```

Ejemplo 3.20

Escriba un algoritmo que nos permita hallar el factorial de un número

Solución

Dado un número N el factorial de dicho número es

$$N*(N-1)*(N-2)-...*1$$

Por ejemplo si el número es 4, su factorial es

$$4*3*2*1$$

Por otro lado hay un factorial especial que tenemos que considerar, es el factorial de cero, cuyo resultado es uno, entonces:

$$!0 = 1$$

Algoritmo

Declarar e inicalizar variables

Leer número

Comprobar si número es cero, entonces su factorial es 1

Si número es diferente de cero hallar factorial.

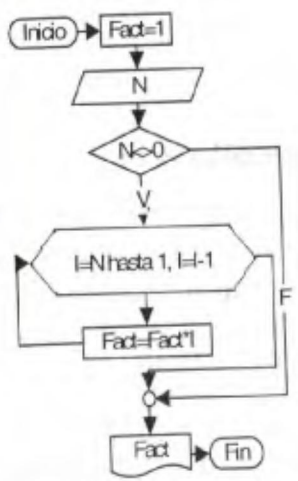
Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        I, N : Entero
        Fact : Real
3. Fact = 1
4. Leer N
5. Si N <> 0 Entonces
     5.1. Desde I = N Hasta 1 con I = I-1 Hacer
          5.1.1. Fact = Fact * I
     5.2. Fin_Desde
6. Fin_Si
7. Escribir 'El factorial es : ', Fact
8. Terminar proceso
```

Nótese en la línea 5.1 que I empieza en N y luego vamos bajando en una unidad hasta llegar a uno. También estamos asumiendo que el número ingresado es un número positivo o cero. A manera de ejercicio puede implementar la validación del número ingresado.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int N;
float Fact;
clrscr();
Fact = 1;
cout<<"Ingrese número:
";cin>>N;
if (N!=0)
 for (int I=N;I>=1;I--)
     Fact*= I;
cout<<endl<<"Factorial es
:"<<Fact;
getch();}
```

Ejemplo 3.21

Escriba un algoritmo que permita invertir un número entero positivo.

Solución

Hemos visto anteriormente como invertir un número, pero esta vez no nos indican cuantas cifras tendrá el número, motivo por el cual debemos cambiar de estrategia, por otro lado debemos asegurarnos que el número ingresado, es positivo.

Algoritmo

Declarar e inicializar variables

Leer y verificar si número ingresado es positivo

Mientras que el número sea diferente de cero

Realizar cálculos necesarios

Mostrar resultado

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          N : Entero
          NI, FI, F : Real
3. Hacer FI = 1
         F = 0.1
         NI = 0
4. Hacer
     4.1. Leer N
5. Mientras(N<=0)
6. Mientras N <> 0 Hacer
     6.1. Calcular NI = NI + (N(MOD)10 * F)
                  N = N DIV 10
                  FI = FI * 10
                  F = F * 0.1
7. Fin_Mientras
8. Calcular NI = NI * FI
9. Escribir 'El número invertido es : ', NI
10. Terminar proceso
```

N, almacena el número ingresado

NI, almacena el número invertido parcial y total

FI, factor que nos indicará por que número debemos multiplicar a NI al final del proceso, para obtener el número invertido final.

F, factor que nos permite ir almacenando los números según sea.

Veamos la ejecución de este algoritmo, para N = 5438

3. FI = 1 Y F=0.1

6. Mientras 5438 <> 0 Hacer (Se cumple la condición)

NI = NI+(N(MOD)10 * F) ==> 0 + (5438(MOD)10*0.1) ==> NI=0.8

N = N DIV 10 ==> 5483 DIV 10 ==> N=543

FI = FI * 10 ==> 1*10 ==> F = 10

F = F * 0.1 ==> 0.1 * 0.1 ==> F = 0.01

Evaluamos nuevamente la condición

6. Mientras 543 <> 0 Hacer (Se cumple la condición)

NI = NI+(N(MOD)10 * F) ==> 0.8 + (543(MOD)10*0.01) ==> NI=0.83

N = N DIV 10 ==> 543 DIV 10 ==> N=54

FI = FI * 10 ==> 10*10 ==> F = 100

F = F * 0.1 ==> 0.1 * 0.1 ==> F = 0.001

Evaluamos la condición

6. Mientras 54 <> 0 Hacer (Se cumple la condición)

NI = NI+(N(MOD)10 * F) ==> 0.83 + (54(MOD)10*0.001) ==> NI=0.834

N = N DIV 10 ==> 54 DIV 10 ==> N=5

FI = FI * 10 ==> 100*10 ==> F = 1000

F = F * 0.1 ==> 0.1 * 0.1 ==> F = 0.0001

Evaluamos la condición

6. Mientras 5 <> 0 Hacer (Se cumple la condición)

NI = NI+(N(MOD)10 * F) ==> 0.834 + (5(MOD)10*0.001) ==> NI=0.8345

N = N DIV 10 ==> 5 DIV 10 ==> N=0

FI = FI * 10 ==> 1000*10 ==> F = 10000

F = F * 0.1 ==> 0.1 * 0.1 ==> F = 0.00001

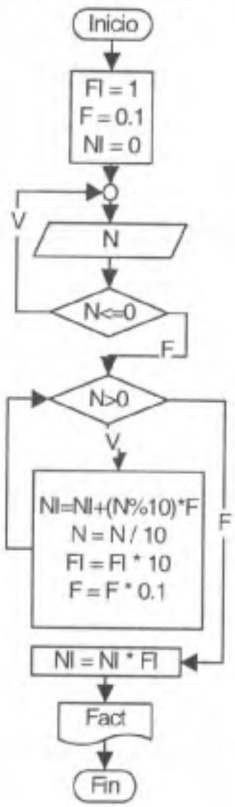
6. Mientras 0 <> 0 Hacer (No se cumple condición fin Mientras)

Finalmente calculamos el número

8. NI = NI * FI = 0.8345 * 10000 = 8345

Note que la lectura de N se ejecutará siempre que ingresemos un valor válido, en este caso un número mayor que cero, en caso contrario el bucle Hacer...Mientras seguirá ejecutándose indefinidamente. Esta es una forma de obligar al usuario a ingresar un valor válido.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
long N;
float NI, FI, F;
clrscr();
FI = 1;
F = 0.1;
NI = 0;
do{
 cout<<endl<<"Ingrese número: ";
 cin>>N;
}while(N<=0);
while (N>0)
{     NI += (N%10)*F;
      N/=10;
      FI*=10;
      F*=0.1;
}
NI*=FI;
cout<<endl<<"Número invertido:"<<NI;
getch();
}
```

Ejemplo 3.22

Diseñar un algoritmo que me permita calcular las 5 primeras parejas de números

primos gemelos.

Solución

Dos números son primos gemelos si además de ser números primos, la diferencia entre ellos es exactamente dos. Entonces primero debemos de ver si ambos números son primos, y luego comprobar si son primos gemelos.

Algoritmo

Declarar variables

Iniciar contador

Repetir

Leer números válidos

Comprobar si números son primos

Si son primos comprobar si son primos gemelos

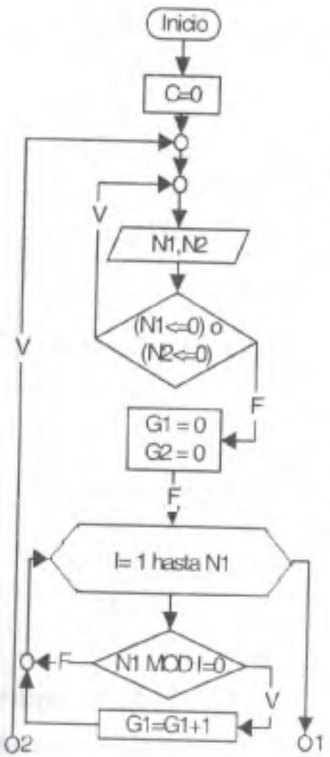
Hasta encontrar 5 pares de números gemelos

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
        C, N1, N2, G1, G2, I : Entero
3. Hacer C = 0
4. Hacer
    4.1. Hacer
        4.1.1. Leer N1, N2
    4.2. Mientras (N1 <= 0) o (N2 <= 0)
    4.3. Hacer G1 = 0
              G2 = 0
    4.4. Desde I = 1 hasta N1 hacer
        4.4.1. Si N1 mod I = 0 Entonces
            4.4.1.1. Hacer G1 = G1 + 1
        4.4.2. Fin_Si
    4.5. Fin_Desde
    4.6. Desde I = 1 hasta N2 hacer
        4.6.1. Si N2 mod I = 0 Entonces
            4.6.1.1. Hacer G2 = G2 + 1
        4.6.2. Fin_Si
    4.7. Fin_Desde
```

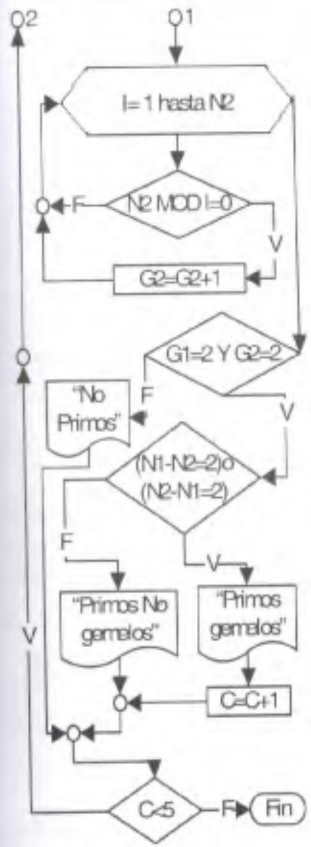
```
    4.8. Si G1 = 2 Y G2 = 2 Entonces
        4.8.1. Si (N1-N2=2)O(N2-N1=2) Entonces
            4.8.1.1. Escribir 'Son primos gemelos'
            4.8.1.2. Calcular C = C + 1
        4.8.2. Si no
            4.8.2.1. Escribir 'Son primos no gemelos'
        4.8.3. Fin_Si
    4.9. Si no
        4.9.1. Escribir 'No son números primos'
    4.10. Fin_Si
5. Mientras (C<5)
6. Terminar proceso
```

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int C, N1, N2, G1, G2;
clrscr();
C=0;
do{
 do{
      cout<<endl<<"Ingrese número
                   1:";cin>>N1;
      cout<<endl<<"Ingrese número
                   2:";cin>>N2;
 }while((N1 <= 0) || (N2<=0));
 G1 = 0;
 G2 = 0;
 for (int I=1;I<=N1;I++)
      if (N1%I==0)
           G1++;
 for (I=1;I<=N2;I++)
      if(N2%I==0)
           G2++;
 if (G1==2 && G2==2)
      if ((N1-N2==2)||(N2-N1==2))
```


```
      {    cout<<"Son primos
                  gemelos";
           C = C + 1 ;
      }else
           cout<<"Son primos
                  no gemelos";
 else
      cout<<"No son números
                  primos";
}while (C<5);
getch();
}
```

El algoritmo lee pares de números y los valida, en caso de no ser números válidos vuelve a pedir un par de números. Al ser los números válidos verifica si son primos, al ser primos comprueba si con primos gemelos.

El algoritmo terminará cuando se encuentre cinco pares de números primos gemelos.

Ejemplo 3.23

Diseñar un algoritmo que me permita calcular los 3 primeros números perfectos.

Solución

Un número es perfecto, cuando la suma de sus divisores, sin incluir al número es exactamente el mismo número. Por ejemplo el 6 es un número perfecto por que sus divisores son 1,2 y 3; y 1+2+3=6

Algoritmo

Declarar variables

Iniciar contador

Repetir

Encontrar y sumar los divisores del número

Comprobar si el número es pefecto

Hasta encontrar 3 números perfectos

Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
          N, P, C, I : Entero
3. Hacer N = 1
        P = 0
4. Repetir
     4.1. C = 0
     4.2. Desde I = 1 Hasta N-1
          4.2.1. Si N MOD I = 0 Entonces
               4.2.1.1. Calcular C = C + I
          4.2.2. Fin_Si
     4.3. Fin_Desde
     4.4. Si C = N Entonces
          4.4.1. Escribir N
          4.4.2. Hacer P = P + 1
     4.5. Fin_Si
     4.6. Hacer N = N + 1
5. Hasta (P = 3)
6. Terminar proceso
```

El contador C se encarga de almacenar la suma de los divisores de N

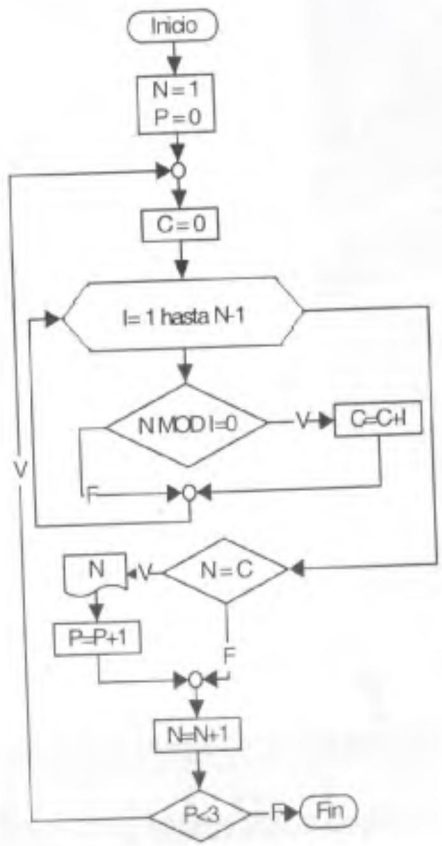
P es el contador que indica el número de números perfectos.

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main(){
int N, P, C;
clrscr();
N = 1;
P = 0;
do{
```



```
C = 0;
for (int I=1;I<=N-1;I++)
     if (N%I==0)
          C+=I;
if (C == N)
{    cout<<endl<<"Perfecto: "<<N;
     P++;
}
N++;
}while(P<3);
getch();
}
```



Capítulo IV

Vectores o Arreglos

Este capítulo contiene:

Lectura y escritura de un vector

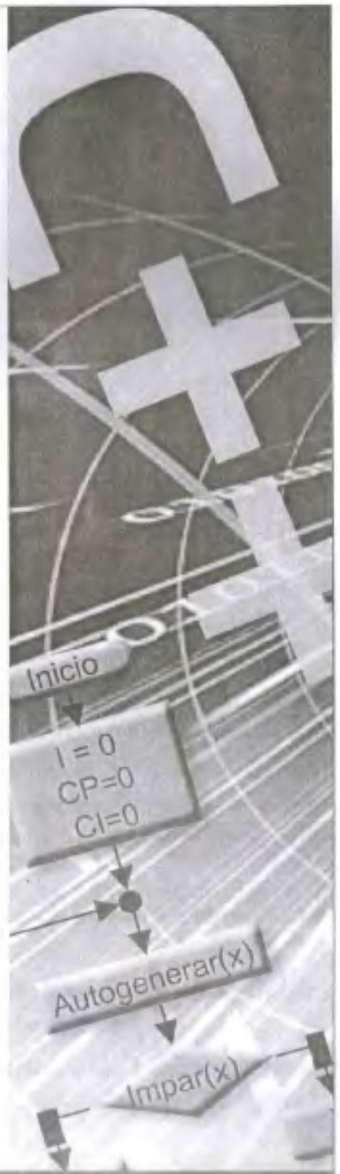
Métodos de ordenamiento de un vector

- Método de la burbuja
- Método de ordenamiento por Inserción
- Método de Selección
- Método de Shell

Búsqueda en un vector

- Búsqueda Lineal
- Búsqueda Binaria

Ejemplos desarrollados



Algoritmos y Diagramas de Flujo aplicados en C++

Vectores

Empezaremos con los tipos de datos estructurados, y con el más sencillo, los arreglos o vectores.

Los arreglos o arrays, permiten agrupar datos usando un mismo identificador. Todos los elementos de un array son del mismo tipo, y para acceder a cada elemento se usan subíndices.

Los valores para el número de elementos deben ser constantes, y se pueden usar tantas dimensiones como queramos, limitado sólo por la memoria disponible, al momento de codificar el algoritmo.

Cuando sólo se usa una dimensión se suele hablar de listas o vectores, cuando se usan dos o más, de tablas o matrices.

Luego veremos las cadenas de caracteres que son un tipo especial de arreglos. Se trata en realidad de arreglos de una dimensión de tipo char.

Los subíndices son enteros, y pueden tomar valores desde 1 hasta "número de elementos". Esto es muy importante, y hay que tener mucho cuidado, por ejemplo:

```
Vector : arreglo[1..10] de enteros
```

Creará un arreglo llamado Vector con 10 enteros a los que accederemos como Vector[1], Vector[2]...Vector[10]. Como subíndice podremos usar cualquier expresión entera.

En la ilustración siguiente podemos ver la representación gráfica de un vector, como podemos observar la primera posición de un vector es 1, aunque en ciertos lenguajes de codificación como el C++, un vector empieza en la posición 0 y no en la posición 1, en esta ocasión, usaremos el 1 como principio de un vector.

Posición ⟶	1	2	3	4	5	6	7	8	9	10
Vector ⟶	5	4	14	87	65	25	18	0	32	7

Lectura y escritura de un vector

Como hemos mencionado antes, un vector es una colección de datos y no un dato en particular, por lo tanto, si por ejemplo tenemos el siguiente vector:

Vector : Arreglo [1..10] de enteros

el cual, como podemos ver, es un arreglo de enteros, el cual posee como máximo 10 números. Por tanto sería incorrecto decir:

Vector = 7

esto implica que el proceso de lectura y escritura de un vector no lo podemos realizar en una simple línea de código, pero esto no implica que no sea fácil. Lo más práctico es usar un bucle repetitivo para leer todos los datos del vector, puede usar cualquiera de las estructuras vistas en el capítulo anterior.

Entonces tenemos el siguiente ejemplo de lectura de un arreglo:

```
Desde I = 1 Hasta 5 Hacer
   Leer (Vector[I])
Fin_Desde
```

Por ejemplo, si queremos ingresar los siguientes datos al vector

5, 98, 47, 12, 34

La ejecución del algoritmo sería de la siguiente manera

Primero al declarar el vector este se encuentra vacío

Desde I = 1 Hasta 5 Hacer

Para I = 1, es decir se leerá el valor y se almacenará en la posición 1

Leer (5)

5				

Para I = 2, almacenará en valor leido en la posición 2

Leer (98)

5	98			

Para I = 3, almacenará en valor leido en la posición 3

Leer (47)

5	98	47		

Para I = 4, almacenará en valor leido en la posición 4

Leer (12)

5	98	47	12	

Para I = 5, almacenará en valor leido en la posición 5

Leer (34)

5	98	47	12	34

De esta manera hemos visto como se van ingresando los datos dentro de un vector. De manera análoga se realiza la escritura, de la siguiente manera:

```
Desde I = 1 Hasta 5 Hacer
   Escribir (Vector[I])
Fin_desde
```

Veamos un ejemplo clásico y sencillo de arreglos.

Ejemplo 4.1

Realice un pseudocódigo que calcule la suma de los elementos de un arreglo de 10 elementos.

Solución

Esta es la forma más sencilla de leer un vector, muchas veces al adaptar el algoritmo a un lenguaje de programación específico, se deben de realizar cambios mínimos pero la lógica central es la misma.

Algoritmo

Declarar variables

Iniciar acumulador

Leer vector

Hallar suma de los componentes del vector

Reportar resultado

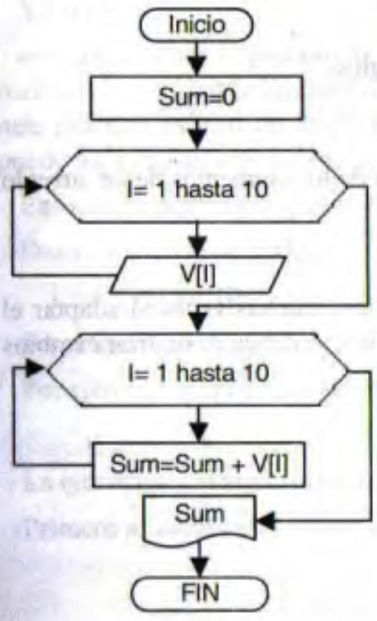
Pseudocódigo

```
1. Iniciar proceso
```

```
2. Declarar Variables
        I, Sum : Entero
        V : Arreglo [1..10] de Enteros
3. Hacer Sum = 0
4. Desde I = 1 Hasta 10 Hacer
    4.1. Leer V[I]
5. Fin_Desde
6. Desde I = 1 Hasta 10 Hacer
    6.1. Sum = Sum + V[I]
7. Fin_Desde
8. Imprimir 'La suma es:', Sum
9. Terminar proceso
```

Como podemos observar antes de realizar cualquier operación con los arreglos primero debemos de ingresar el vector completo, luego podemos usar el vector en cualquier parte del algoritmo.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int Sum;
int V[10];
clrscr();
Sum = 0;
for (int I=0;I<10;I++)
{
 cout<<"Ingrese dato: ";
 cin>>V[I];
}
for (I=0;I<10;I++)
 Sum+=V[I];
cout<<endl<<"La suma es:"<<Sum;
getch();
}
```

Como mencionamos antes en C++ todos lo vectores siempre comienzan en la posición cero debido a eso el contador empieza en I=0 y va hasta I=9, en este ejemplo en particular.

Ejemplo 4.2

Realice un pseudocódigo que calcule la media aritmética de N valores, además reporte los números ingresados en orden inverso.

Algoritmo

Declarar variables

Iniciar acumulador

Leer y sumar elementos del vector

Reportar resultado y reportar vector inverso

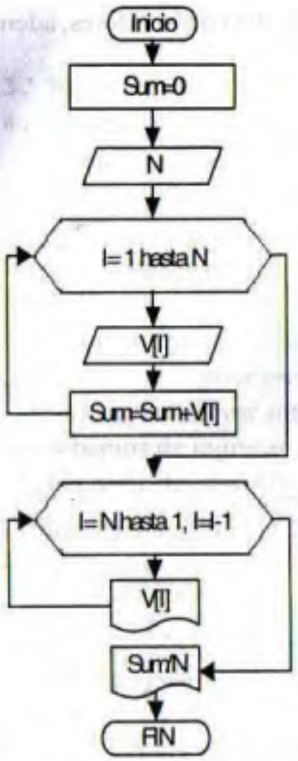
Pseudocódigo

```
1. Iniciar proceso
2. Declarar Variables
      N,I, Sum : Entero
      V : Arreglo [1..100] de Enteros
3. Hacer Sum = 0
4. Leer N
5. Desde I = 1 Hasta N Hacer
      5.1. Leer V[I]
      5.2. Sum = Sum + V[I]
6. Fin_Desde
7. Desde I = N Hasta 1 con [I=I-1] Hacer
      7.1. Escribir V[I]
8. Fin_Desde
9. Imprimir 'La media Aritmética es:', (Sum/N)
10. Terminar proceso
```

Como se puede apreciar luego de ingresado el número a la posición I del arreglo, este es acumulado en la variable Sum, este procedimiento es válido ya que primero leemos el número, por otro lado estamos usando un bucle inverso, para mostrar el arreglo al revés, en verdad lo único que se hace es mostrar el arreglo en orden inverso pero los valores del vector siguen en sus respectivas posiciones, más adelante veremos como cambiar de posiciones los valores de un arreglo.

Otro detalle es que al momento de mostrar el resultado de la media aritmética, lo hacemos con la expresión (Sum/N) es decir calculamos y mostramos. Entendamos que el objetivo de un programa es que sea eficiente y lo más compacto posible, de esto depende la velocidad de procesamiento del mismo.

Diagrama de flujo



Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int Sum,N;
int V[10];
clrscr();
Sum = 0;
cout<<"Ingrese Cantidad de
          números: ";cin>>N;
for (int I=0;I<N;I++)
{
 cout<<"Ingrese dato: ";
 cin>>V[I];
 Sum = Sum + V[I];
}
for (I=N-1;I>=0;I--)
 cout<<endl<<V[I];
 cout<<endl<<"Media:"<<(Sum/N);
getch();
}
```

Ejemplo 4.3

De una lista de 100 números determinar simultáneamente el máximo y mínimo número.

Solución

Este es un ejemplo que ya hemos desarrollado en el capítulo anterior, la diferencia es que ahora debemos usar arreglos.

Algoritmo

Declarar variables

Iniciar acumulador

Leer elementos del vector

Halla el número mayor y menor de la lista

Reportar resultado

Pseudocódigo

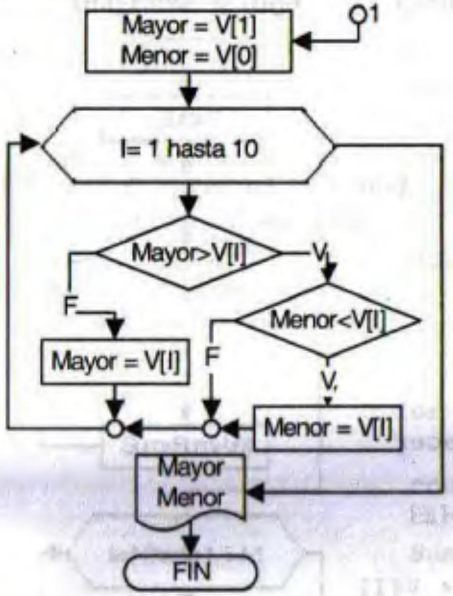
```
1. Iniciar proceso
2. Declarar Variables
                I,Sum, Mayor, Menor : Entero
                V : Arreglo [1..100] de Enteros
3. Desde I = 1 Hasta 100 Hacer
     3.1. Leer V[I]
4. Fin_Desde
5. Hcaer Mayor = V[1]
       Menor = V[1]
6. Desde I = 2 Hasta 100 Hacer
     6.1. Si Mayor > V[I]
          6.1.1. Mayor = V[I]
     6.2. Si no
          6.2.1. Si Menor < V[I]
               6.2.1.1. Menor = V[I]
          6.2.2. Fin_Si
     6.3. Fin_Si
7. Fin_Desde
8. Imprimir 'Número mayor es:', Mayor
            'Número menor es:', Menor
9. Terminar proceso
```

Vea que el bucle Desde inicia I=2, ya que el primer valor del vector, almacenado en la primera posición, es decir V[1], lo usamos como artificio, siendo nuestro primer número mayor y menor a la vez.

Diagrama de flujo

Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int Sum, Mayor, Menor;
int V[100];
clrscr();
```




```
for (int I=0;I<100;I++)
{.    cout<<"Ingrese dato:";
      cin>>V[I];
}
Mayor = V[0];
Menor = V[0];
for (I=1;I<100;I++)
 if(Mayor>V[I])
      Mayor = V[I];
 else
      if(Menor<V[I])
            Menor = V[I];
cout<<endl<<"mayor
                  es:"<<Mayor;
cout<<endl<<"menor
                  es:"<<Menor;
getch();}
```

Ordenamiento de un vector

Una operación que se hace muy a menudo con los arrays, sobre todo con los de una dimensión, es ordenar sus elementos.

Método de la burbuja

Este es uno de los métodos de ordenamiento más usados, aunque no de los más eficaces, se trata del método de la burbuja.

Consiste en recorrer la lista de valores a ordenar y compararlos dos a dos. Si los elementos están bien ordenados, pasamos al siguiente par, si no lo están los intercambiamos, y pasamos al siguiente, hasta llegar al final de la lista. El proceso completo se repite hasta que la lista está ordenada.

El algoritmo de ordenación es el siguiente:

```
1. Desde I = 1 hasta N-1 hacer
 1.1. Desde J = 1 hasta N-I hacer
      1.1.1. Si V[J] > V[J+1] entonces
            1.1.1.1. Calcular Aux = V[J]
                     V[J] = V[J+1]
                     V[J+1] = Aux
```



```
      1.1.2. Fin_Si
  1.2. Fin_Desde
2. Fin_Desde
```

En donde:

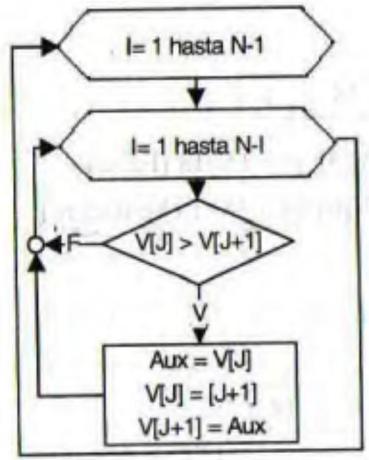
I, J : Variables de tipo entero usados para el control de los bucles.

N representa el tamaño del vector, el cual está dado por el número de elemen que posee.

V : arreglo a ordenar

Aux : Variable de tipo del arreglo, usada para intercambiar los valores.

Diagrama de flujo



Codificación en C++

```
for (I=0;I<N-1;I++)
 for (int J=0;J<(N-1)-I;J++)
      if(V[J]>V[J+1])
      {        Aux=V[J];
           V[J] = V[J+1];
           V[J+1] = Aux;
      }
```

Veamos un ejemplo, ordenando el siguie vector de menor a mayor:

15	3	8	6	18	1

Desde I = 1 hasta N-1 , es decir desde 1 hasta 5

Para i = 1

Desde J = 1 hasta N-I, es decir desde 1 hasta 5

Para J=1 Si V[J] > V[J+1] ==> V[1] > V[2] ==> 15>3 (Verdadero)

Aux = V[J] ==> Aux = 15

V[J] = V[J+1] ==> V[1]=3

V[J+1] = Aux ==>V[2]=15

El vector ahora es de la siguiente manera

3	15	8	6	18	1

Para J=2 Si V[J] > V[J+1] ==> V[2] > V[3] ==> 15>8 (Verdadero)
Aux = V[J] ==> Aux = 15
V[J] = V[J+1] ==> V[2]=8
V[J+1] = Aux ==>V[3]=15

El vector ahora es de la siguiente manera

3	8	15	6	18	1

Para J=3 Si V[J] > V[J+1] ==> V[3] > V[4] ==> 15>6 (Verdadero)
Aux = V[J] ==> Aux = 15
V[J] = V[J+1] ==> V[3]=6
V[J+1] = Aux ==>V[4]=15

3	8	6	15	18	1

Para J=4 Si V[J] > V[J+1] ==> V[4] > V[5] ==> 15>18 (Falso)
Para J=5 Si V[J] > V[J+1] ==> V[5] > V[6] ==> 18>1 (Verdadero)
Aux = V[J] ==> Aux = 18
V[J] = V[J+1] ==> V[5]=1
V[J+1] = Aux ==>V[6]=18

3	8	6	15	1	18

Fin del bucle con el indice J, entonces I aumenta en una unidad, por lo tanto I=2 y vuelve a iniciar J = 1 Hasta 4

Para J=1 Si V[J] > V[J+1] ==> V[1] > V[2] ==> 3>8 (Falso)
Para J=2 Si V[J] > V[J+1] ==> V[2] > V[3] ==> 8>6 (Verdadero)
Aux = V[J] ==> Aux = 8
V[J] = V[J+1] ==> V[2]=6
V[J+1] = Aux ==>V[3]=8

3	6	8	15	1	18

Para J=3 Si V[J] > V[J+1] ==> V[3] > V[4] ==> 8>15 (Falso)
Para J=4 Si V[J] > V[J+1] ==> V[4] > V[5] ==> 15>1 (Verdadero)

Aux = V[J] ==> Aux = 15
V[J] = V[J+1] ==> V[4]=1
V[J+1] = Aux ==>V[5]=15

3	6	8	1	15	18

Ahora I = 3 y J va desde 1 hasta 3

Para J=1 Si V[J] > V[J+1] ==> V[1] > V[2] ==> 3>6 (Falso)
Para J=2 Si V[J] > V[J+1] ==> V[2] > V[3] ==> 6>8 (Falso)
Para J=3 Si V[J] > V[J+1] ==> V[3] > V[4] ==> 8>1 (Verdadero)
Aux = V[J] ==> Aux = 8
V[J] = V[J+1] ==> V[3]=1

3	6	1	8	15	18

V[J+1] = Aux ==>V[4]=15

Ahora I = 4 y J va desde 1 hasta 2

Para J=1 Si V[J] > V[J+1] ==> V[1] > V[2] ==> 3>6 (Falso)
Para J=2 Si V[J] > V[J+1] ==> V[2] > V[3] ==> 6>1 (Verdadero)
Aux = V[J] ==> Aux = 6
V[J] = V[J+1] ==> V[2]=1
V[J+1] = Aux ==>V[3]=6

3	1	6	8	15	18

Ahora I = 5 y J va desde 1 hasta 1

Para J=1 Si V[J] > V[J+1] ==> V[1] > V[1] ==> 3>1 (Verdadero)
Aux = V[J] ==> Aux = 3
V[J] = V[J+1] ==> V[2]=1
V[J+1] = Aux ==>V[3]=3

1	3	6	8	15	18

Al finalizar ambos bucles Desde, el vector ya queda ordenado en forma creciente.

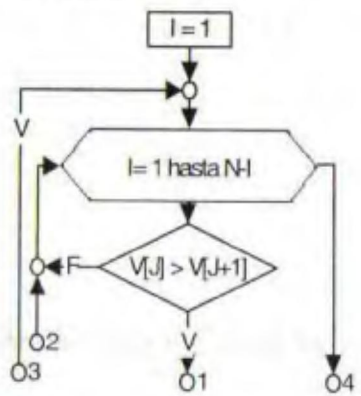
Método de ordenamiento de la burbuja mejorada

Como hemos visto anteriormente, el método de ordenación de la burbuja consiste en la comparación de elementos consecutivos y en donde encuentra dos elementos que cumplan la condición se hace el intercambio, eso quiere decir que si en una corrida no hace ningún intercambio significaría que el vector ya está ordenado, se podría controlar este evento mediante una variable booleana y según el cambio de ésta terminar el proceso de ordenamiento, esto se resume en ahorro de tiempo al momento de ordenar. El algoritmo para esta burbuja sería el siguiente:

```
1. Hacer I = 1
2. Hacer
      2.1. Intercambio = 0
      2.2. Desde J = 1 hasta N-I hacer
           2.2.1. Si V[J]>V[J+1] entonces
                       Aux = V[J]
                       V[J] = V[J+1]
                       V[J+1] = Aux
                       Intercambio = 1
           2.2.2. Fin_Si {Fin del paso 2.2.1}
      2.3. Fin_Desde {Fin del paso 2.2}
      2.4. Hacer I = I + 1
3. Mientras Intercambio = 1
```

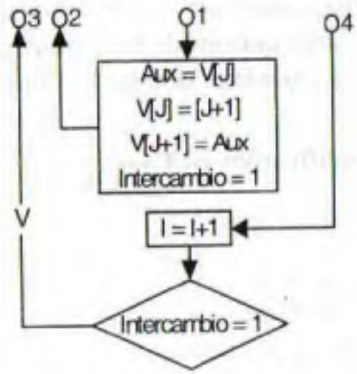
Como podemos observar, estamos usando la variable **Intercambio** como una variable booleana, el bucle Hacer..Mientras se ejecutará mientras que esta variable tenga el valor de 1, si esta variable llegara al final del bucle con un valor 0, significaría que no hay nada que comparar y fin del bucle.

Diagrama de flujo



Codificación en C++

```
I = 1;
Do{
 Intercambio = 0;
 for(int J=0;J<N-I;J++)
     if (V[J]>V[J+1])
     {
          Aux = V[J];
          V[J] = V[J+1];
          V[J+1] = Aux;
          Intercambio = 1;
     }
```


```
     I++;
}while(Intercambio == 1);
```

Método de ordenamiento por Inserción

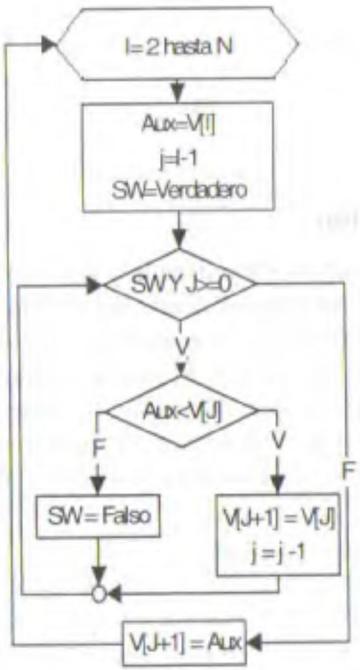
Este método toma cada elemento del arreglo para ser ordenado y lo compara con los que se encuentran en posiciones anteriores a la de él dentro del arreglo. Si resulta que el elemento con el que se está comparando es mayor que el elemento a ordenar, se recorre hacia la siguiente posición superior. Si por el contrario, resulta que el elemento con el que se está comparando es menor que el elemento a ordenar, se detiene el proceso de comparación pues se encontró que el elemento ya está ordenado y se coloca en su posición (que es la siguiente a la del último número con el que se comparó).

El algoritmo en el cual se basa este método es el siguiente:

```
1. Desde I = 2 Hasta N hacer
     1.1. Hacer Aux = V[I]
                j = I-1
                SW = Verdadero
     1.2. Mientras (SW) Y (J>=1) Hacer
          1.2.1. Si Aux < V[j] Hacer
                      V[J+1] = V[J]
                      j = j -1
          1.2.2. Si no
                      SW = Falso
          1.1.3 Fin si
     1.3. Fin_Mientras
     1.4. Hacer V[j+1] = Aux
2. Fin_desde
```

Nótese que en la línea 1.2 se debe cumplir ambas condiciones en caso contrario, se terminará el bucle Mientras. Por otro lado vemos que una de las condiciones solo es (SW), en este caso se debe asumir que es verdadero, también es válido si se pone (SW=Verdadero).

Diagrama de flujo



Codificación en C++

```
for(I=1;I<N;I++)
{
 Aux = V[I];
 J = I-1;
 SW = Verdadero;
 while (SW && J>=0)
 {
      if (Aux<V[J])
      {
           V[J+1] = V[J];
           J--;
      }
      else
           SW = Falso;
 }
 V[J+1] = Aux;
}
```

En C++ no están definidas las variables Verdadero(True) o Falso (False) por lo que debemos hacerlo nosotros mismos, por ejemplo podemos definir Verdadero igual a 1 en todo caso Falso será 0. Entonces debemos incluir las siguientes líneas de código:

```
#define Verdadero 1
#define Falso 0
```

Veamos con un ejemplo el seguimiento de este algoritmo, ordenando en forma creciente el vector mostrado.

15	3	8	6	18	1

Iniciando el algoritmo tenemos I=2 hasta 6

Para I = 2

Aux = V[2] ==> Aux = 3

j = I-1==> J=1

SW = Falso

Mientras

(Verdadero) Y (1>=1) (Se cumplen las condiciones)

Si Aux < V[j] ==> 3<15

V[J+1] = V[J] ==> V[2]=15

j = j-1 ==> j=0

15	15	8	6	18	1

(Verdadero) Y (0>=1) (No se cumplen las condiciones)

V[j+1] = Aux ==> V[1]=3

3	15	8	6	18	1

Para I = 3

Aux = V[3] ==> Aux = 8

j = I-1==> J=2

SW = Verdadero

Mientras

(Verdadero) Y (2>=1) (Se cumplen las condiciones)

Si Aux < V[2] ==> 8<15

V[J+1] = V[J] ==> V[3]=15

j = j-1 ==> j=1

3	15	15	6	18	1

(Verdadero) Y (1>=1) (Se cumplen las condiciones)

Si Aux < V[1] ==> 8<3 (No se cumple)

SW = Falso

(Falso) Y (1>=1) (No se cumplen las condiciones)

V[j+1] = Aux ==> V[2]=8

3	8	15	6	18	1

Para I = 4

Aux = V[4] ==> Aux = 6

j = I-1==> J=3

SW = Verdadero

Mientras

(Verdadero) Y (3>=1) (Se cumplen las condiciones)

Si Aux < V[3] ==> 6<15

V[J+1] = V[J] ==> V[4]=15

j = j-1 ==> j=2

3	8	15	15	18	1

(Verdadero) Y (2>=1) (Se cumplen las condiciones)

Si Aux < V[2] ==> 6<8

V[J+1] = V[J] ==> V[3]=8

j = j-1 ==> j=1

3	8	8	15	18	1

(Verdadero) Y (1>=1) (Se cumplen las condiciones)

Si Aux < V[1] ==> 6<3 (No se cumple)

SW = Falso

(Falso) Y (1>=1) (No se cumplen las condiciones)

V[j+1] = Aux ==> V[2]=6

3	6	8	15	18	1

Para I = 5

Aux = V[5] ==> Aux = 18

j = I-1==> J=4

SW = Verdadero

Mientras

(Verdadero) Y (4>=1) (Se cumplen las condiciones)

Si Aux < V[4] ==> 18<15 (No se cumple)

SW = Falso

(Falso) Y (4>=1) (No se cumplen las condiciones)

Para I = 6

Aux = V[6] ==> Aux = 1

j = I-1==> J=5

SW = Verdadero

Mientras

(Verdadero) Y (5>=1) (Se cumplen las condiciones)

Si Aux < V[5] ==> 1<18

V[J+1] = V[J] ==> V[6]=18

j = j-1 ==> j=4

3	6	8	15	18	18

(Verdadero) Y (4>=1) (Se cumplen las condiciones)

Si Aux < V[4] ==> 1<15

V[J+1] = V[J] ==> V[5]=15

j = j-1 ==> j=3

3	6	8	15	15	18

(Verdadero) Y (3>=1) (Se cumplen las condiciones)

Si Aux < V[3] ==> 1<8

V[J+1] = V[J] ==> V[4]=8

j = j-1 ==> j=2

3	6	8	8	15	18

(Verdadero) Y (2>=1) (Se cumplen las condiciones)

Si Aux < V[2] ==> 1<6

V[J+1] = V[J] ==> V[3]=6

j=j-1 ==>j=1

3	6	6	8	15	18

(Verdadero) Y (1>=1) (Se cumplen las condiciones)

Si Aux < V[1] ==> 1<3

V[J+1] = V[J] ==> V[2]=3

j=j-1 ==>j=0

3	3	6	8	15	18

(Verdadero) Y (0>=1) (No se cumplen las condiciones)

V[j+1] = Aux ==> V[1]=1

1	3	6	8	15	18

Finalmente el vector queda ordenado

Método de Selección

El método de ordenamiento por selección consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo. Su algoritmo es el siguiente:

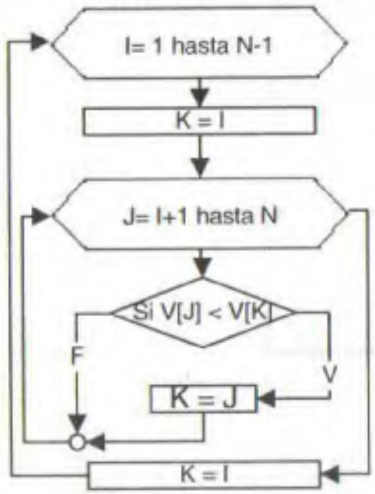
// Buscamos la posición del mínimo en V[I], V[I+1], ..., V[N-1]

```
1. Desde i= 0 hasta N-1 Hacer
     1.1. Hacer K = I
     1.2. Desde J = I+1 Hasta N Hacer
          1.2.1. Si V[J] < V[K]
               1.2.1.1. Hacer K = J
          1.2.2. Fin_Si
     1.3. Fin_Desde
     // intercambiamos V[I] con V[K]
     1.4. Hacer Aux = V[I]
               V[I]= V[K]
               V[K]= Aux
```



```
2. Fin_Desde
```

Diagrama de flujo



Codificación en C++

```
for(I=0;I<N-1;I++)
{
 K = I;
 for (int J = I+1;J<N;J++)
      if(V[J]<V[K])
           K = J;
 //intercambiamos V[I] con V[K]
 Aux = V[I];
 V[I]= V[K];
 V[K]= Aux;
}
```

Veamos un ejemplo de ordenamiento para el vector siguiente usando el método de selección:

15	3	8	6	18	1

Desde I = 1 hasta N-1, Desde 1 hasta 5

Para I=1

K = I ==> K = 1

Desde J = I+1 Hasta N, es decir desde J=2 hasta 6

Para J=2

Si V[J] < V[K] ==> V[2]<V[1] ==> 3<15 (Se cumple condición)

K = J ==> K=2

Para J=3

Si V[J] < V[K] ==> V[3]<V[2] ==> 8<3 (No se cumple condición)

Para J=4

Si V[J] < V[K] ==> V[4]<V[2] ==> 6<3 (No se cumple condición)

Para J=5

Si V[J] < V[K] ==> V[5]<V[2] ==> 18<3 (No se cumple condición)

Para J=6

Si V[J] < V[K] ==> V[6]<V[2] ==> 1<3 (Se cumple condición)

K = J ==> K=6

// intercambiamos V[I] con V[K] ==> V[1] con V[6]

Aux = V[1] ==> Aux = 15

V[1] = V[6] ==> V[1] = 1

V[6] = Aux => V[6] = 15

Entonces el vector temporalmente seria:

1	3	6	8	15	18

Para I=2

K = I ==> K = 2

Desde J = I+1 Hasta N, es decir desde J=3 hasta 6

Para J=3

Si V[J] < V[K] ==> V[3]<V[2] ==> 8<3 (No se cumple condición)

Para J=4

Si V[J] < V[K] ==> V[4]<V[2] ==> 6<3 (No se cumple condición)

Para J=5

Si V[J] < V[K] ==> V[5]<V[2] ==> 18<3 (No se cumple condición)

Para J=6

Si V[J] < V[K] ==> V[6]<V[2] ==> 15<3 (No se cumple condición)

Técnicamente ahora se realiza el intercambio, pero nótese que en ninguna iteración se ha cumplido con la condición Si, esto sucede cuando el elemento analizado ya está en su posición correcta, cuando la posición del mínimo es K=I. Entonces se intercambia V[I] con V[I]. Una rápida revisión del intercambio permite apreciar que en ese caso V[I] no altera su valor, y por lo tanto no es incorrecto.

Se podría introducir una instrucción Si..Entonces para que no se realice el intercambio cuando K=I, pero esto sería menos eficiente que realizar el intercambio aunque no sirva. En efecto, en algunos casos el Si..Entonces permitiría ahorrar un intercambio, pero en la mayoría de los casos no serviría.

Para I=3

K = I ==> K = 3

Desde J = I+1 Hasta N, es decir desde J=4 hasta 6

Para J=4

Si V[J] < V[K] ==> V[4]<V[3] ==> 6<8 (Se cumple condición)

K = J ==> K=4

Para J=5

Si V[J] < V[K] ==> V[5]<V[4] ==> 18<6 (No se cumple condición)

Para J=6

Si V[J] < V[K] ==> V[6]<V[4] ==> 15<6 (No se cumple condición)

// intercambiamos V[I] con V[K] ==> V[3] con V[4]

Aux = V[3] ==> Aux = 8

V[3] = V[4] ==> V[3] = 6

V[4] = Aux => V[4] = 8

1	3	6	8	18	15

Para I=4

K = I ==> K = 4

Desde J = I+1 Hasta N, es decir desde J=5 hasta 6

Para J=5

Si V[J] < V[K] ==> V[5]<V[4] ==> 18<8 (No se cumple condición)

Para J=6

Si V[J] < V[K] ==> V[5]<V[4] ==> 15<8 (No se cumple condición)

Para I=5

K = I ==> K = 5

Desde J = I+1 Hasta N, es decir desde J=6 hasta 6

Para J=6

Si V[J] < V[K] ==> V[6]<V[5] ==> 15<18 (Se cumple condición)

K = J ==> K=6

// intercambiamos V[I] con V[K] ==> V[5] con V[6]

Aux = V[5] ==> Aux = 18

V[5] = V[6] ==> V[5] = 15

V[6] = Aux => V[6] = 18

1	3	6	8	15	18

El vector finalmente queda ordenado

Método de Shell

A diferencia del algoritmo de ordenación por inserción, este algoritmo intercambia elementos distantes. Es por esto que puede deshacer más de una inversión en cada intercambio, hecho del cual nos aprovechamos para ganar velocidad.

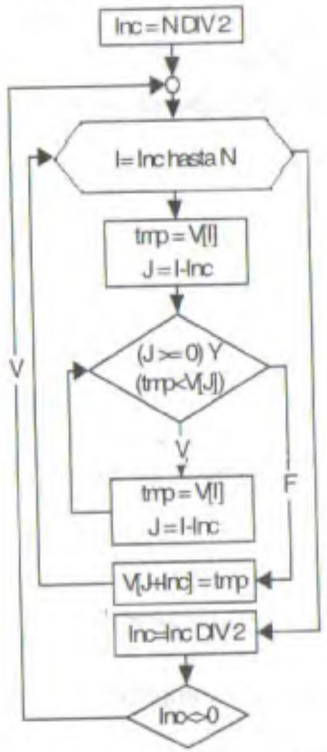
La velocidad del algoritmo dependerá de una secuencia de valores (llamados incrementos) con los cuales trabaja utilizándolos como distancias entre elementos a intercambiar.

Se considera la ordenación de Shell como el algoritmo más adecuado para ordenar entradas de datos moderadamente grandes (decenas de millares de elementos) ya que su velocidad, si bien no es la mejor de todos los algoritmos, es aceptable en la práctica y su implementación (código) es relativamente sencillo.

A continuación damos el algoritmo formal de la ordenación de Shell, en el cual utilizaremos los incrementos propuestos por Shell por simplicidad en el código:

```
Inc = n DIV 2
Hacer
 Desde i = Inc + 1 hasta n hacer
      tmp = v[i]
      J = i-Inc
      Mientras (J >= 0) Y (tmp<v[J])
           v[J+Inc] = v[J]
           J = J - Inc
      Fin_mientras
      v[J+Inc] = tmp
 Fin_Desde
 Inc = Inc DIV 2
Mientras (Inc<>0)
```


Diagrama de flujo



Codificación en C++

```
Inc = N/2;
do{
 for(I=Inc;I<N;I++)
 {
      tmp = V[I];
      J = I-Inc;
      while((J>=0)&&(tmp<V[J]))
      {
           V[J+Inc] = V[J];
           J-=Inc;
      }
      V[J+Inc] = tmp;
 }
 Inc/=2;
}while(Inc!=0);
```

Veamos como el seguimiento de este pseudocódigo mediante un ejemplo, ordenemos el vector siguiente.

15	3	8	6	18	1

Inc = 3

Repetir

Primera iteración del bucle repetir

Desde I = Inc+1 Hasta N Hacer ==> Desde 4 hasta 6

Para i = 4

tmp = v[i] ==> tmp = v[4] ==> tmp = 6

J = i - Inc ==> J = 1

Mientras

(J > 0) Y tmp<v[J] ==>(1> 0) Y (6 < 15) (Se cumplen)

v[J+Inc] = v[J] ==> T[4]=15

J = J - Inc ==> 1-3 ==> J = -2

(J > 0) Y tmp<v[J] ==>(-2 > 0) Y (6 < NE) (No se cumplen)

v[J+Inc] = tmp ==> T[1] = 6

6	3	8	15	18	1

Para I = 5

tmp = v[i] ==> tmp = v[5] ==> tmp = 18

J = i - Inc ==> J = 2

Mientras

(J > 0) Y tmp<v[J] ==>(2 > 0) Y (18 < 3) (No se cumplen)

Para I = 6

tmp = v[i] ==> tmp = v[6] ==> tmp = 1

J = i - Inc ==> J = 3

Mientras

(J > 0) Y tmp<v[J] ==>(3 > 0) Y (1 < 8) (Se cumplen)

v[J+Inc] = v[J] ==> T[6]=8

J = J - Inc ==> 3-3 ==> J = 0

(J > 0) Y tmp<v[J] ==>(0 > 0) Y (1 < NE) (No se cumplen)

v[J+Inc] = tmp ==> T[3] = 1

6	3	1	15	18	8

Fin del bucle desde

Inc = Inc DIV 2 ==> INC = 3 DIV 2 ==> INC = 1

Segunda iteración del bucle repetir ya que no se cumple que INC = 0

Desde I = Inc+1 Hasta N Hacer ==> Desde 2 hasta 6

Para i = 2

tmp = v[i] ==> tmp = v[2] ==> tmp = 3

J = i - Inc ==> J = 1

Mientras

(J > 0) Y tmp<v[J] ==>(1 > 0) Y (3 < 6) (Se cumplen)

v[J+Inc] = v[J] ==> T[2]=6

J = J - Inc ==> 1-1 ==> J = 0

(J > 0) Y tmp<v[J] ==>(0 > 0) Y (6 < NE) (No se cumplen)

v[J+Inc] = tmp ==> T[1] = 3

3	6	1	15	18	8

Para i = 3

tmp = v[i] ==> tmp = v[3] ==> tmp = 1

J = i - Inc ==> J = 2

Mientras

(J > 0) Y tmp<v[J] ==>(2 > 0) Y (1 < 6) (Se cumplen)

v[J+Inc] = v[J] ==> T[3]=6

J = J - Inc ==> 2-1 ==> J = 1

(J > 0) Y tmp<v[J] ==>(1> 0) Y (1 < 3) (Se cumplen)

v[J+Inc] = v[J] ==> T[2]=3

J = J - Inc ==> 1-1 ==> J = 0

(J > 0) Y tmp<v[J] ==>(0 > 0) Y (1 < NE) (No se cumplen)

v[J+Inc] = tmp ==> T[1] = 1

1	3	6	15	18	8

Para i = 4

tmp = v[i] ==> tmp = v[4] ==> tmp = 15

J = i - Inc ==> J = 3

Mientras

(J > 0) Y tmp<v[J] ==>(3 > 0) Y (15 < 6) (No se cumplen)

Para i = 5

tmp = v[i] ==> tmp = v[5] ==> tmp = 18

J = i - Inc ==> J = 4

Mientras

(J > 0) Y tmp<v[J] ==>(4 > 0) Y (18 < 15) (No se cumplen)

Para i = 6

tmp = v[i] ==> tmp = v[6] ==> tmp = 8

J = i - Inc ==> J = 5

Mientras

(J > 0) Y tmp<v[J] ==>(5 > 0) Y (8 < 18) (Se cumplen)

v[J+Inc] = v[J] ==> T[6]=18

J = J - Inc ==> 5-1 ==> J = 4

(J > 0) Y tmp<v[J] ==>(4> 0) Y (8 < 15) (Se cumplen)

v[J+Inc] = v[J] ==> T[5]=15

J = J - Inc ==> 4-1 ==> J = 3

(J > 0) Y tmp<v[J] ==>(3 > 0) Y (8 < 6) (No se cumplen)

v[J+Inc] = tmp ==> T[4] = 8

1	3	6	8	15	18

Fin del bucle Desde

Inc = Inc DIV 2 ==> INC = 1 DIV 2 ==> INC = 0

Fin del bucle Hacer..Mientras ya que no se cumple la condición INC<> 0

Finalmente tenemos el vector ordenado. Nótese que hemos usado en algunas comparaciones **NE**, esto significa que ese valor No Existe. Por ejemplo v[0], cero no es una posición válida, ya que mencionamos que los vectores van desde 1, en lenguajes de programación como el C++, los vectores empiezan en la posición cero, pero el caso práctico, mantendremos la convención de empezar un vector en la posición 1.

Existen otros métodos de ordenación basados en métodos recursivos, por ese motivo los veremos más adelante cuando usemos las funciones o procedimientos

Se ha visto que hay muchos métodos de ordenación para un arreglo, depende del lector la selección de cual de ellos usar.

Búsqueda en un vector

Esta es otra operación muy usada con los arreglos, y consiste en la búsqueda en un conjunto de datos de un elemento específico y la recuperación de alguna información asociada al mismo.

A continuación veremos dos tipos de búsqueda: la búsqueda lineal y la búsqueda binaria.

Búsqueda Lineal

Este método de búsqueda es la más simple, consiste en recorrer todo el vector y devolver el elemento buscado junto con la posición en donde ha sido encontrado, en caso contrario se emitirá un mensaje de una búsqueda infructuosa o se procederá a alguna otra acción.

La forma más fácil de implementar esta búsqueda es a través de un bucle **Desde**, mediante el siguiente algoritmo:

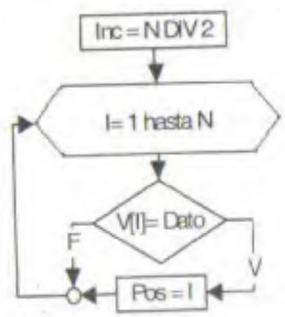
```
1. Hacer Pos = 0
2. Leer Dato
3. Desde I = 1 hasta Fin Hacer
   3.1. Si V[I] = Dato Hacer
      3.1.1. Pos = I
   3.2. Fin_Si
4. Fin_Desde
```

En donde:

Pos: Variable entera que indica posición del dato a encontrar.

Dato: Variable a encontrar.

Nótese en el algoritmo anterior que inicialmente la variable de posición **Pos** tiene un valor 0, esto indica que no corresponde a ninguna posición del vector.



Codificación en C++

```
Pos = -1;
cou<<endl<<"Ingrese dato a
      buscar";
cin>>Dato;
for (I=0;I<N;I++)
{     if (V[I]== Dato)
      Pos = I;
}
```

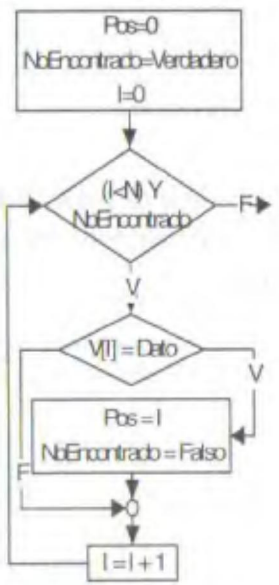
Este método no es muy eficiente, puesto que si el elemento buscado se encuentra en las primeras posiciones, igual se tiene que terminar de recorrer todo el vector, en un vector grande esto sería una perdida de tiempo.

Debido a este inconveniente resulta mas óptimo el uso de una variable de comprobación de tipo booleano, de tal modo que si el elemento es encontrado la variable cambie de estado, lo cual permita terminar la búsqueda, sin necesidad de terminar de recorrer el vector.

El pseudocódigo a usar es el siguiente:

```
1. Hacer Pos = 0
        NoEncontrado = Verdadero
        I = 1
2. Mientras (I<=N) Y (NoEncontrado) hacer
   2.1 Si V[I] = Dato Hacer
      2.1.1. Hacer  Pos = I
                    NoEncontrado = Falso
   2.2 Fin_Si
   2.3 Hacer I = I +1
3. Fin_Mientras
```

Diagrama de flujo



Codificación en C++

```
Pos = 0;
NoEncontrado = Verdadero;
I = 0;
cout<<endl<<"Ingrese dato a
buscar:";cin>>Dato;
while ((I<N) && NoEncontrado)
{
 if(V[I] == Dato)
 {
      Pos = I;
      NoEncontrado = Falso;
 }
 I++;
}
```


Veamos la ejecución de este algoritmo con un ejemplo, al busca el numero 18 dentro del arreglo siguiente

45	4	7	18	17	53	43	74

N=8

Dato = 18

Pos = 0

NoEncontrado = Verdadero

I = 1

Mientras

Primera iteración

(I<=N) Y (NoEncontrado) ==> (1<=8) Y (Verdadero) (Se cumplen)

Si V[I] = Dato ==> V[1] = 18 ==> 45 = 18 (No se cumple condición)

45	4	7	18	17	53	43	74

I = I +1 ==> I = 2

Segunda iteración

(I<=N) Y (NoEncontrado) ==> (2<=8) Y (Verdadero) (Se cumplen)

Si V[I] = Dato ==> V[2] = 18 ==> 4 = 18 (No se cumple condición)

45	4	7	18	17	53	43	74

I = I +1 ==> I = 3

Tercera iteración

(I<=N) Y (NoEncontrado) ==> (3<=8) Y (Verdadero) (Se cumplen)

Si V[I] = Dato ==> V[3] = 18 ==> 7 = 18 (No se cumple condición)

45	4	7	18	17	53	43	74

I = I +1 ==> I = 4

Cuarta iteración

(I<=N) Y (NoEncontrado) ==> (4<=8) Y (Verdadero) (Se cumplen)

Si V[I] = Dato ==> V[4] = 18 ==>18 = 18 (Si se cumple condición)

45	4	7	18	17	53	43	74

Pos = I ==> Pos = 4

NoEncontrado = Falso

I = I +1 ==> I = 5

Quinta iteración

(I<=N) Y (NoEncontrado) ==> (5<=8) Y (Falso) (No se cumple condiciones)

Fin del algoritmo elemento fue encontrado en la posición Pos = 4

Búsqueda Binaria

Este método es muy útil en arreglos muy grandes, el único problema es que se requiere de un vector ordenado. Su algoritmo es el siguiente:

Algoritmo

Se compara el dato buscado con el elemento central del vector

Si coinciden hemos encontrado el dato buscado

si el dato es mayor que el elemento central del vector, buscar el dato en la parte superior del vector

Por el contrario si el dato es menor, busque en la parte inferior del vector

La idea en que se basa el algoritmo es bien conocida: si el vector V está ordenado, la localización del valor X puede hacerse de manera que en cada paso se vaya reduciendo el intervalo de búsqueda. Para ello, en lugar de comenzar por un extremo, se considera en primer lugar el valor central V [Cen], de modo que se tienen las tres posibilidades siguientes:

V [Cen] = Dato. Es decir, ya se ha encontrado el elemento buscado; no es por tanto necesario seguir buscando. Esto resulta bastante improbable, al menos en la primera iteración.

V [Cen] > Dato. Al estar el vector ordenado, que el punto medio quede por encima implica que X, si se encuentra en el vector, ha de estar en la mitad inferior. Por tanto, se restringe la búsqueda al intervalo [Inf; Cen -1].

V [med] < Dato. De modo análogo, puede ubicarse ahora al valor X en la mitad

superior del vector. Por tanto, la búsqueda se restringe al intervalo [Cen + 1; Sup].

Así pues, comenzando con el intervalo inicial i..f , se aplica esta técnica de manera sucesiva; en cada paso cambian los límites del intervalo, que se irá haciendo cada vez más pequeño, hasta que el valor X se encuentra (o se determina que no está en el vector).

Dado que la variante mostrada está desestructurada, es fundamental lograr una versión estructurada. De hecho, no hay una, sino varias propuestas en este sentido: la propia importancia del algoritmo ha propiciado esta diversificación. En definitiva, todas las variantes parten de la misma idea, y son igualmente válidas, dado que las diferencias entre unas y otras son mínimas.

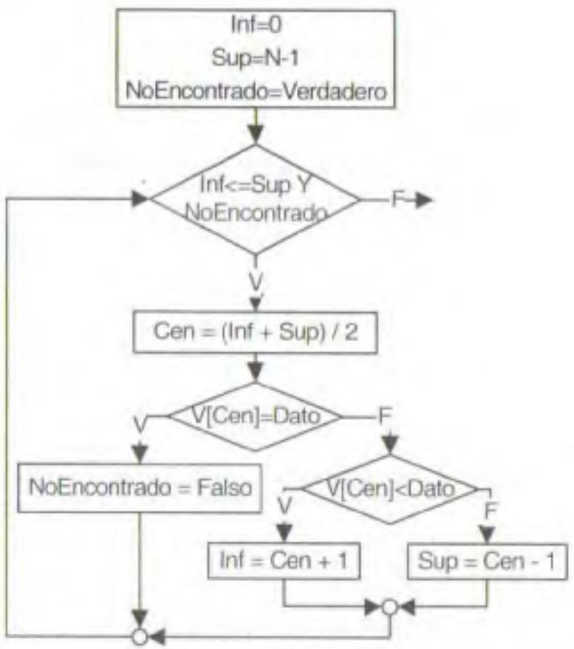
Pseudocódigo

```
1. Hacer Inf = 1
        Sup = N
      NoEncontrado = Verdadero
2. Mientras (Inf <= Sup) Y (NoEncontrado) hacer
     2.1. Hacer Cen = (Inf + Sup) / 2
     2.2. Si V[Cen] = Dato entonces
          2.2.1. NoEncontrado = Falso
     2.3. Si no
          2.3.1. Si V[Cen] < Dato entonces
               2.3.1.1. Hacer Inf = Cen + 1
          2.3.2. Si no
               2.3.2.1. Hacer Sup = Cen - 1
          2.3.3. Fin_Si
     2.4. Fin_Si
3. Fin_Mientras
4. Si NoEncontrado hacer
     4.1. Hacer Escribir "Dato no encontrado"
5. Si no
     5.1. Hacer Escribir "Dato encontrado en posición",Cen
6. Fin_Si
```

En donde:

P, U, Cen : Variables de tipo entero, que indican la primera posición, la última posición y la posición central respectivamente

```
Inf = 0;
Sup = N-1;
NoEncontrado = Verdadero;
while(Inf<=Sup && NoEncontrado)
{Cen = (Inf + Sup) / 2;
 if(V[Cen]==Dato)
      NoEncontrado = Falso;
 else
      if(V[Cen]<Dato)
           Inf = Cen + 1;
      else
           Sup = Cen - 1;
}
if (NoEncontrado)
 cout<<endl<<"No encontrado";
else
 cout<<endl<<"Encontrado en:"<<Cen;
```



Veamos un ejemplo de búsqueda con este algoritmo, usando el siguiente vector.

4	7	17	18	43	45	53	74

Suponiendo que el dato a buscar es 53, N=8 y Dato=53 entonces tenemos:

Inf = 1

Sup = N ==> U = 8

NoEncontrado = Verdadero

Mientras

Primera iteración Mientras

(Inf<=Sup) Y (NoEncontrado) ==> (1<=8) Y (Verdadero) (Se cumplen)

Cen = (Inf + Sup) / 2 ==> (1+8)/2 ==> Cen = 4

Si V[Cen] = Dato ==> V[4] = 53 ==> 18 = 53 (No se cumple)

Si V[Cen] < Dato ==> 18<53 (No se cumple condición)

Inf = Cen + 1 ==> 4+1 ==> Inf = 5

4	7	17	18	43	45	53	74

Segunda iteración Mientras

(Inf<=Sup) Y (NoEncontrado) ==> (5<=8) Y (Verdadero) (Se cumplen)

Cen = (Inf + Sup) / 2 ==> (5+8)/2 ==> Cen = 6

Si V[Cen] = Dato ==> V[6] = 53 ==> 45 = 53 (No se cumple)

Si V[Cen] < Dato ==> 45<53 (Si se cumple condición)

Inf = Cen + 1 ==> 5+1 ==> Inf = 6

4	7	17	18	43	45	53	74

Tercera iteración Mientras

(Inf<=Sup) Y (NoEncontrado) ==> (6<=8) Y (Verdadero) (Se cumplen)

Cen = (Inf + Sup) / 2 ==> (6+8)/2 ==> Cen = 7

Si V[Cen] = Dato ==> V[7] = 53 ==> 53 = 53 (Si se cumple)

NoEncontrado = Falso

Inf = Cen + 1 ==> 5+1 ==> Inf = 6

4	7	17	18	43	45	53	74

Cuarta iteración Mientras

(Inf<=Sup) Y (NoEncontrado) ==> (7<=8) Y (Falso) (No se cumplen)

Fin del bucle mientras

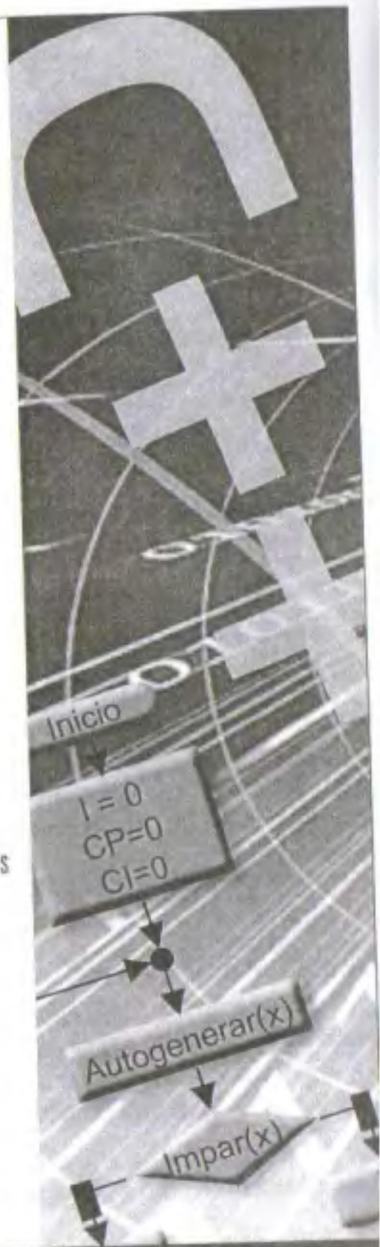
Si NoEncontrado ==> Falso

Capítulo V

Uso de funciones para desarrollar programas

Este capítulo contiene:

- Funciones
- Procedimientos
- Semejanzas y diferencias entre funciones y procedimientos
- Ámbito de las variables
- Paso de parámetros
- Funciones y Procedimientos como parámetros
- Ejemplos desarrollados



Algoritmos y Diagramas de Flujo aplicados en C++

Introducción

La resolución de problemas complejos se facilita considerablemente si se dividen en pequeños (subproblemas).

La solución de estos subproblemas se realiza con **subprogramas**. Su uso permite al programador desarrollar programas más complejos utilizando un método de diseño **descendente.** Se denomina descendente ya que se inicia en la parte superior con un problema general y luego abajo el diseño específico de las soluciones en los subproblemas. Normalmente las partes en que se divide un programa deben poder desarrollarse independientemente entre si, planteando los requerimientos de variables, el cuerpo del programa principal y luego el código de los subprogramas.

El subprograma, por ser un algoritmo, debe cumplir con las mismas características de éste y hacer tareas similares como aceptar datos, escribir datos y hacer cálculos; sin embargo, es utilizado para un propósito específico. El subprograma recibe datos del algoritmo o subalgoritmo que lo invoca y éste le devuelve resultados.

Los subprogramas pueden ser de dos tipos: funciones y procedimientos.

Funciones

Una función va a ser un subprograma que nos devuelve un dato de tipo estándar,

es un subprograma que puede tomar uno o varios parámetros como entrada y devuelve a la salida un único resultado. Su sintaxis es la siguiente:

Sintaxis:

```
Función nombrefuncion (P1: tipo, P2: tipo, ...): Tipo
 Declaraciones locales
 Inicio
      Cuerpo de la función
      Retornar (valor)
 Fin_Función
```

Donde P1, P2,... son parámetros, los parámetros son la información que se le tiene que pasar a la función. Los parámetros luego dentro de la función los podemos utilizar igual que si fueran variables locales definidas en la función y para cada parámetro hay que poner su nombre y tipo.

El nombre de la función lo da el usuario y tiene que ser significativo.

En las variables locales se declaran las variables que se pueden usar dentro de la función.

La línea:

```
retornar valor
```

indica que la función devuelve el valor del proceso final por lo tanto se debe de colocar cuando finalice la función, por otro lado **valor** debe ser del mismo tipo de la función.

Una función es llamada por medio de su nombre, en una sentencia de asignación o en una sentencia de salida.

Se puede llamar a una función en cualquiera de las siguientes formas:

```
nombrefunción o nombrefunción(par)
```

La primera es cuando no se necesita ningún parámetro y en la segunda obligatoriamente se necesita un parámetro.

Como las funciones devuelven un valor específico la forma más usual de utilizarlas es por medio de asignaciones de una variable a la función. Por ejemplo:

```
idVar = nombrefunción
```

No se permiten funciones que no devuelvan nada.

Ejemplo de función

Una función que calcule el cuadrado de un valor que le pasa parámetro. Suponemos que es un valor entero.


```
Función Cuadrado (n: entero): real
 Declarar variables
       C : real
 Inicio
     C = n*n
     Retornar (C)
 Fin_Función_Cuadrado
```

Procedimientos

Un procedimiento es un subprograma o un subalgoritmo que ejecuta una determinada tarea, pero que tras ejecutar esa tarea no tienen ningún valor asociado a su nombre como en las funciones, sino que si devuelve información, lo hace a través de parámetros.

Al llamar a un procedimiento, se le cede el control, comienza a ejecutarse y cuando termina devuelve el control a la siguiente instrucción a la de llamada. Su sintaxis es la siguiente:

Sintaxis:

```
Procedimiento Nombre_Proc (P1: tipo, P2: tipo, ...)
 Declaraciones locales
 Inicio
     Cuerpo del procedimiento
 Fin_Procedimiento
```

La cabecera va a estar formada por el nombre del procedimiento que será un identificador y que debe de ser significativo, y luego entre paréntesis los parámetros o la información que se le pasa al procedimiento. Para cada parámetro hay que indicar el tipo de paso de parámetro. Veamos un ejemplo de procedimientos.

```
Procedimiento Cuadrado (num:entero,M:real)
 Inicio
     M = num/2
 Fin_Procedimiento_Cuadrado
```

Diferencias entre funciones y procedimientos

- Una función devuelve un único valor y un procedimiento puede devolver 0,1 o N.

- Ninguno de los resultados devueltos por el procedimiento se asocian a su nombre como ocurría con la función.
- Mientras que la llamada a una función forma siempre parte de una expresión, la llamada a un procedimiento es una instrucción que por sí sola no necesita instrucciones.

Semejanzas entre Procedimientos y Funciones.

- La definición de ambos aparece en la sección de subprogramas de la parte de declaraciones de un programa y en ambos casos consiste en una cabecera, una parte de declaraciones una parte de instrucciones.
- Ambos son unidades de programa independientes. Los parámetros, constantes y variables declarados en una función o procedimiento son locales a la función o al procedimiento, solamente son accesibles dentro del subprograma.
- Cuando se llama a una función o a un procedimiento, el número de los parámetros reales debe ser el mismo que el número de los parámetros formales y los tipos de los parámetros reales deben coincidir con los tipos de los correspondientes parámetros formales, con una excepción: se puede asociar un parámetro real de tipo entero con un parámetro formal por valor de tipo real.

Ambito de las variables

La parte del programa principal o función en que una variable se define y se puede utilizar o alterar su contenido se conoce como **Ambito.**

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos:

- variables locales
- variables globales

Variable local

Una **variable local** es aquella que está declarada y definida dentro de un subprograma, en el sentido de que está dentro de ese subprograma y es distinta de las variables con el mismo nombre declaradas en cualquier parte del programa principal. El significado o visibilidad de una variable se confina a la función en la que está declarada.

Esto quiere decir que la variable no tiene ningún significado, no se conoce y no se puede acceder a ella desde fuera del subprograma y que tiene una posición de

memoria distinta a la de cualquier otra, incluso si es de una variable que tiene el mismo nombre pero que está definida fuera del subprograma.

Las variables locales a un subprograma se definen en la parte de la definición de variables del mismo. Los parámetros formales que se le ponen a un subprograma se comportan dentro de él como si fueran también variables locales a él.

Las variables locales al finalizar la función o el procedimiento, desaparecen de la memoria. Si dos variables, una global y una local, tienen el mismo nombre, la local prevalecerá sobre la global dentro del módulo en que ha sido declarada. Dos variables locales pueden tener el mismo nombre siempre que estén declaradas en funciones o procedimientos diferentes.

Variable global

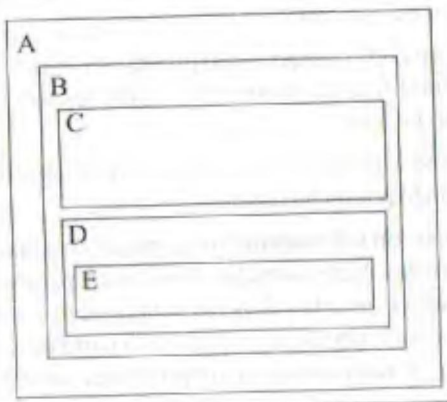
Una **variable global** es aquella que está declarada para el programa principal.

A esta variable podemos acceder desde cualquiera de los subprogramas y el programa principal, salvo que alguno de esos subprogramas tenga definida una variable local con el mismo nombre que la variable global, en este caso si utilizo el nombre de esa variable me referiré a la local, nunca a la global (ya que tienen 2 zonas de memoria distintas).

Las variables permanecen activas durante todo el programa. Se crean al iniciarse éste y se destruyen de la memoria al finalizar.

Las variables globales tienen la ventaja de compartir información de diferentes subprogramas sin una correspondiente entrada en la lista de parámetros.

La figura siguiente muestra un esquema de un programa con diferentes subprogramas con variables locales y otras globales, aquí se muestra el ámbito de cada definición.



En donde:

Variables definidas en:	Son accesibles desde:
A	A, B, C, D, E
B	B, C, D, E
C	C
D	D, E
E	E

Paso de parámetros

Existen diferentes métodos para la transmisión o *paso de los parámetros* a subprogramas. Es preciso conocer el método adoptado por cada lenguaje, ya que no siempre son los mismos. Dicho de otro modo, un mismo programa puede producir diferentes resultados bajo diferentes **sistemas de paso de parámetros**.

Cuando llamamos a una función o procedimiento, le pasamos a través de los parámetros la información que necesita, y en el caso de un procedimiento también devolvemos a través de sus parámetros los resultados. Para ello definiremos el tipo del parámetro a principio del subprograma, que es lo que conocemos como parámetros formales, y al hacer la llamada pasamos la información a través de los parámetros reales.

Los parámetros pueden ser clasificados como:

- Entradas: Solo proporcionan valores desde el programa que los llama y que se utilizan dentro del subprograma. En el caso de una función, todos sus parámetros son de este tipo.

  Como solo sirven como entrada, solo pueden ser leídos, pero no modificados, y aunque se modificasen dentro de un subprograma, fuera no va a tener efecto esa modificación.

- Salidas: las salidas producen los resultados del subprograma, este devuelve un valor calculado por dicha función.

- Entradas/Salidas: un solo parámetro se utiliza para mandar argumentos a un programa y para devolver resultados. El valor del parámetro tiene importancia tanto a la entrada como a la salida del subprograma, nos aporta información cuando llamamos al subprograma y por otra parte devolvemos a través de él resultados cuando terminamos el subprograma, en este caso, tiene sentido

tanto leer como actualizar el parámetro.

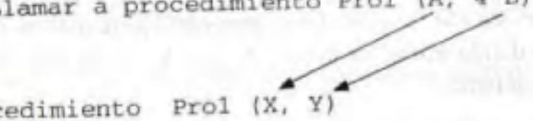
Teniendo en cuenta los tipos de parámetros, los métodos más empleados para realizar el paso de parámetros son:

Paso de parámetros por valor

Son los parámetros que pueden recibir valores pero que no pueden devolverlos. Es una variable global que se conecta con una variable local mediante el envío de su valor, después de lo cual ya no hay relación. Lo que le sucede a la variable local no afectará a la global. Cuando un parámetro actual se pasa por valor, el subprograma hace una copia del valor de éste en una posición de memoria idéntica en tamaño pero distinta en ubicación a la del parámetro actual y la asigna al parámetro formal correspondiente. Como el subprograma trabaja a partir de sus parámetros formales, si durante la ejecución se modifica el valor de un parámetro formal correspondiente a un paso por valor, el contenido de la posición de memoria del parámetro actual no se verá alterado.

A continuación mostramos el mecanismo de paso por valor de un procedimiento con dos parámetros.

```
1. A = 3
2. B = 9
3. Llamar a procedimiento Pro1 (A, 4*B)


Procedimiento  Pro1 (X, Y)
```



Aquí podemos ver que se realiza una **copia** de los valores de los argumentos en la llamada al procedimiento en las variables X e Y , pero en ningún caso los valores de A y B se verán modificados.

Por lo tanto X toma el valor de A es decir X = 3, y Y toma el valor de la expresión 4*B, es decir Y=36

```
Por ejemplo:
Función F (M : Entero) : Entero
 Declarar variables
      X : entero
 Inicio
      X = M
      X = X * X
```

```
        M = X
        Retornar(M)
    Fin_Función_F

    Programa principal
    Inicio
     1. Declarar variables
          X, Y : Entero
     2. Leer(X)
     3. X = X + 1
     4. Y = F(X)
     5. Escribe (X,Y)
    Fin_Programa_Principal
```

Si en la ejecución de este programa damos a X el valor 2, la siguiente línea lo incrementa en 1 (línea 3) y llamamos a la función con f(3). En la llamada a la función se hace una **copia** del valor de la variable **X** del programa principal en la variable **M** de la función. Una vez dentro de la función se declara otra variable **X** que es distinta a la del programa principal, siendo después asignada al valor de M, luego hacemos el cuadrado, lo asignamos a M y retornamos el valor.

Aquí finaliza la llamada a la función. En el programa principal escribimos los valores de X e Y, y dando como resultado X=3 e Y=9. Es muy importante comprender este mecanismo.

Paso por referencia (dirección) o variable

Son los que pueden recibir y devolver valores. Son variables globales que se conectan con una local a través de su contenido; al establecerse dicha conexión las variables se convierten en sinónimos, lo que afecte a la variable local le sucederá a la variable global.

Se utiliza el paso por variable cuando el subprograma debe modificar el contenido de una variable del programa principal o devolver algún valor más, recordemos que una función solo devuelve un valor directamente.

Por ejemplo si la función anterior la modificamos así:

```
Función F (M : Entero) : Entero
 Declarar variables
      X : entero
 Inicio
```



```
      X = M
      X = X * X
      M = X
      Retornar(2X)
 Fin_Función_F

Programa principal
Inicio
 1. Declarar variables
      X, Y : Entero
 2. Leer(X)
 3. X = X + 1
 4. Y = F(X)
 5. Escribe (X,Y)
Fin_Programa_Principal
```

Si llegamos a la llamada con f(3), al ser un paso por variable o dirección, significa que la variable **M** de la función f tiene la dirección de memoria de la variable **X** lo que significa que cualquier cambio del valor de **M** se verá reflejado en **X.** Por tanto dentro de la función hacemos X=3, X al cuadrado (9), M=9 y devolvemos 2x, con lo que en la programación principal escribiremos X=9 e Y=18.

Funciones y Procedimientos como parámetros

En la mayor parte de los lenguajes se permite también que una función o procedimiento pueda ser pasado como parámetro de otro subprograma. En este caso, es decir, cuando el parámetro formal es de tipo función o procedimiento, pasaremos como parámetro real funciones o procedimientos que tengan la misma definición que el que hemos puesto como parámetro formal, y en nuestro caso para indicar que se pasa como parámetro real una función o procedimiento.

Desde el subprograma al que se pasa como parámetro esa función o procedimiento podemos llamar en cualquier momento a esa función pasada como parámetro que en cada momento podrá ser una distinta dependiendo del parámetro formal asociado.

El paso de funciones y procedimientos como parámetros, nos va a permitir que desde un subprograma podamos llamar a otros, pero teniendo en cuenta que el subprograma llamado no va a ser uno determinado, sino que va a depender en cada momento del subprograma pasado como parámetro real, de esta manera el

subprograma puede llamar a un conjunto de n subprogramas que cumplan unas determinadas características, pero solo uno en cada momento.

Ejemplos de Funciones y Procedimientos

Veamos algunos ejemplos de los conceptos antes mencionados:

Ejemplo 5.1

Realizar un programa que pida dos números enteros por teclado y muestre por pantalla el siguiente menú:

MENÚ

1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Salir

Elija una opción:

El usuario deberá elegir una opción y el programa deberá mostrar el resultado por pantalla y finalizar.

Solución

Este es un ejemplo realizado en el capítulo 2, ahora usaremos funciones y procedimientos y veremos la diferencia de la resolución del ejemplo. En esta ocasión hemos agregado al menú la opción salir, de manera que se ejecutará este programa mientras que no se ingrese el número 5, el cual indica salida. Cuando se ejecuta un pseudocódigo o un programa basado en

Algoritmo

Función Leer_Número
  Leer y retornar número leido
Fin_función
Función Menú
  Mostrar opciones
  Leer y retornar opción leida
Fin_función

Procedimiento Operación
  Realizar la operación según sea el caso y mostrar resultado
Fin Procedimiento
Procedimiento Principal
  Hacer
    Mostrar menú
    Leer dos número y ejecutar operación
  Mientras opcion del menú sea 5
Fin Procedimiento principal

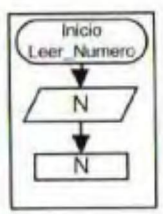
Pseudocódigo

```
Función Leer_Numero() : Entero
 Inicio
     1. Declarar variables
          N : Entero
     2. Leer N
     3. Retornar N
 Fin Función

Función Menu()
 Inicio
     1. Declarar variables
          opcion : Entero
     2. Escribir    "MENU"
                    "1. Sumar"
                    "2. Restar"
                    "3. Multiplicar"
                    "4. Dividir"
                    "5. Salir"
     3. Hacer
          3.1. Leer Opcion
     4. Mientras (opcion<1 Y opcion>4)
     5. retornar opcion
 Fin Función
```

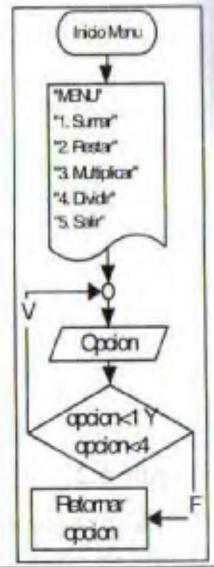
```
Procedimiento Operacion(A:entero, B:entero,
                        Opcion:entero)
 Inicio
      1. Declarar Variables
           R : Real
      2. En caso de (Opcion)
           2.1. caso 1 : R = A + B
           2.1. caso 2 : R = A - B
           2.3. caso 3 : R = A * B
           2.4. caso 4 : R = A / B
      3.Fin caso
      4. Mostrar R;
 Fin Procedimiento

Procedimiento Principal()
 Inicio
      1.Declarar variables
              A, B, Op: entero
      2. Hacer
           2.1 Op=Menu()
           2.2. Si(Op<>5)
                2.2.1. A=Leer_Numero()
                2.2.2. B=Leer_Numero()
                2.2.3. Operacion(A,B,Op)
           2.3. Fin_Si
      3. Mientras(Op<>5)
 Fin_Procedimiento_Principal
```

Codificación en C++

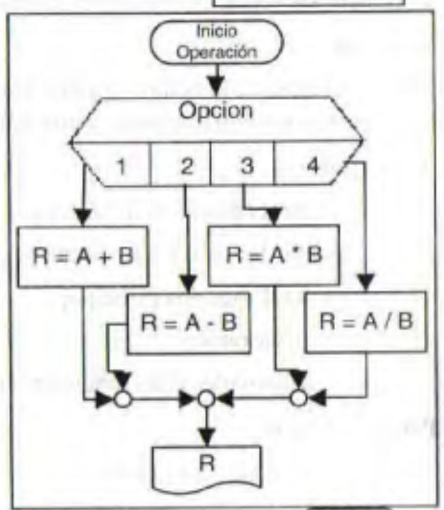
```
#include<iostream.h>
#include<conio.h>
int Leer_Numero()
{int N;
 cout<<"Ingrese número: ";
 cin>>N;
 return N;
}
```



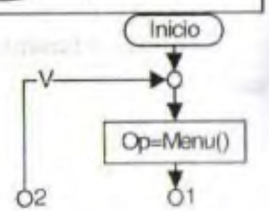

```
int Menu()
{clrscr();
 int opcion;
 cout<<endl<<"MENU";
 cout<<endl<<"1. Sumar";
 cout<<endl<<"2. Restar";
 cout<<endl<<"3. Multiplicar";
 cout<<endl<<"4. Dividir";
 cout<<endl<<"5. Salir";
 do{
      cout<<endl<<"Elija opción:
";cin>>opcion;
 }while(opcion<1 && opcion<4);
 return opcion;
}
```



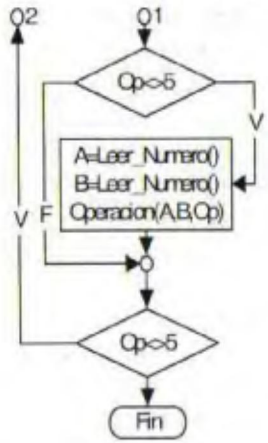

```
void Operacion(int A,
        int B,int Opcion)
{
 float R;
 switch(Opcion)
 {
 case 1:R = A + B;break;
 case 2:R = A - B;break;
 case 3:R = A * B;break;
 case 4:R = A / B;
 }
 cout<<"resultado:"<<R;
 getch();
}
```




```
void main()
{int A, B, Op;
 do{
      Op=Menu();
      if(Op!=5)
      {    A=Leer_Numero();
           B=Leer_Numero();
```



```
    Operacion(A,B,Op);
    }
 }while(Op!=5);
}
```



Ejemplo 5.2

Escriba un pseudocódigo, que dados como datos N números enteros, obtenga el número de ceros que hay entre estos números.

Solución

Este ejercicio ya lo hemos resuelto anteriormente, por lo que ahora veremos su implementación con funciones y procedimientos.

Algoritmo

Construir procedimiento de lectura de datos del vector

Implementar función para contar los ceros del vector

Implementar Programa principal

Declarar variables

Realizar llamados a las funciones correspondientes

Pseudocódigo

```
1. Iniciar proceso

2. Procedimento Leer_Vector(
                V : Arreglo [1..100] de Enteros
                N : Entero)
 2.1. Declarar variables
        I : Entero
```



```
 2.2. Inicio
      2.2.1. Desde I = 1 hasta N Hacer
           2.2.1.1. Leer V[I]
      2.2.2. Fin_Desde
 2.3. Fin_Procedimento Leer_Vector

3. Función Contar_Ceros(
              V : Arreglo [1..100] de Enteros
              N : Entero) : Entero
 3.1. Declarar variables
        C, I : Entero
 3.2. Inicio
      3.2.1. Hacer C = 0
      3.2.2. Desde I = 1 hasta N Hacer
           3.2.2.1. Si V[I]=0 Entonces
                3.2.2.1.1. Hacer C = C + 1
           3.2.2.2. Fin_Si
      3.2.3. Fin_Desde
      3.2.4. Retornar (C)
 3.3. Fin_Procedimiento Contar_Ceros

4. Procedimiento principal
 4.1. Declarar variables
        Ceros : Arreglo [1..100] de Enteros
        Cantidad : Entero
 4.2. Leer Cantidad
 //Llamada al procedimiento Leer_Vector
 4.3. Leer_Vector(Ceros,Cantidad)
 //Obtener cantidad de ceros llamando a Contar_Ceros
 4.4. CC = Contar_Ceros(Ceros,Cantidad)
 4.5. Escribir CC
5. Terminar proceso
```

Veamos como se ejecuta este algoritmo para Cantidad=5, tome en cuenta que siempre se ejecuta primero el procedimiento principal.

4. **Procedimiento principal**

4.1. Declarar variables

Ceros : Arreglo [1..100] de Enteros

N : Entero

Cuando declaramos el arreglo Ceros, vemos que tiene un máximo de 100 caracteres y N = 5, esto quiere decir que se ha solicitado más almacenamiento de memoria del que se requiere. Siguiendo con el ejemplo al ser declarado el arreglo Ceros, este se encuentra vacio, de modo que gráficamente lo representaremos así:

4.2. Leer Cantidad ==> Leer 5 ==> Cantidad = 5

4.3. Leer_Vector(Ceros,Cantidad)

Al realizar la llamada al procedimiento Leer_Vector, se le cede temporalmente el mando de la ejecución del procedimiento, hasta que este se termine de ejecutar.

2. **Procedimiento** Leer_Vector (V , N)

Nótese que el procedimiento recibe el contenido de Ceros, el cual se encuentra vacío, y es almacenado en la variable V, el cual es una arreglo; de la misma manera la variable N recibe el contenido de Cantidad, por lo tanto N=5

2.1.1. Desde I = 1 hasta N ==> Desde 1 hasta 5

2.1.1.1. Leer V[I]

2.2.2. Fin_Desde

Luego de leer los 5 valores (1, 0, 9, -4, 0), el vector sería el siguiente

Al terminar la lectura de los números se finaliza el procedimiento Leer_Vector y devuelve el mando al procedimiento principal, además el contenido del vector Ceros es el mismo del vector V, de manera que:

4.3. CC = Contar_Ceros(Ceros,N)

Se hace el llamado a la función Contar_Ceros en cual debe devolver un valor de tipo entero, el cual será almacenado en la variable C.

3. **Función** Contar_Ceros(V,N)

3.2.1. Hacer c = 0

3.2.2. Desde I = 1 hasta N ==> Desde 1 a 5 Hacer

3.2.2.1. Si V[I]=0 Entonces

3.2.2.1.1. Hacer C = C + 1

No haremos la ejecución de este procedimiento, ya que es sencillo, pero si

quiere ver como se ejecuta, revise el ejercicio 3.3 en el capítulo 3. El resultado final de C es 2, ya que hay dos ceros en el vector, entonces tenemos:

3.2.4. Retornar (C) ==> Retornar (2)

Al retornar el valor de la función también devolvemos el mando a la función principal, por lo tanto tenemos que:

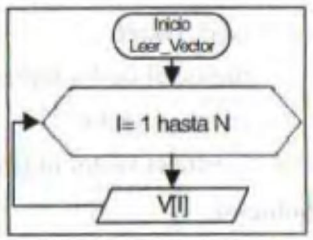
4.3. CC = Contar_Ceros(Ceros,N) ==> CC = 2

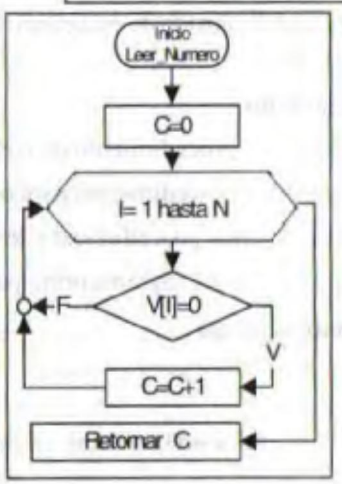
4.4. Escribir CC ==> Escribir 2

5. Terminar proceso

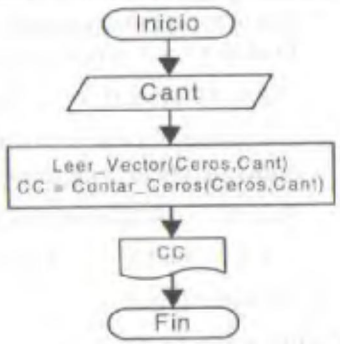
Codificación en C++

```
#include<iostream.h>
#include<conio.h>
void Leer_Vector(int V[ ], int N)
{ for (int I = 0;I<N;I++)
  {cout<<endl<<"Ingrese dato:";
            cin>>V[I];
  }
}
int Contar_Ceros(int V[ ], int N)
{ int C;
  C = 0;
  for (int I = 0;I<N;I++)
            if (V[I]==0)
                  C++;
  return C;
}
void main()
{
  int Ceros[100];
  int Cant,CC;
  clrscr();
```





```
    cout<<"Ingrese cantidad de datos:";
    cin>>Cant;
    Leer_Vector(Ceros,Cant);
    CC = Contar_Ceros(Ceros,Cant);
    cout<<endl<<"Cantidad de ceros: "<<CC;
    getch();
}
```



Ejemplo 5.3

Diseñe un algoritmo que dado un vector de N elementos, realice las siguientes acciones:

- Lea el vector
- Muestre el vector ingresado
- Ordene el vector
- Muestre el vector ordenado

Solución

Para el desarrollo de este ejercicio usaremos el método de selección para ordenar el vector.

Algoritmo

Construir procedimiento de lectura de datos del vector

Construir procedimiento para escribir vector

Implementar procedimiento de ordenamiento por selección.

Implementar Programa principal

Pseudocódigo

```
1. Iniciar proceso

2. Procedimiento Leer_Vector(
                V : Arreglo [1..100] de Enteros
                N : Entero)
 2.1. Declarar variables
          I : Entero
 2.2. Inicio
```

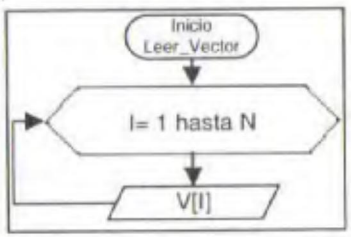


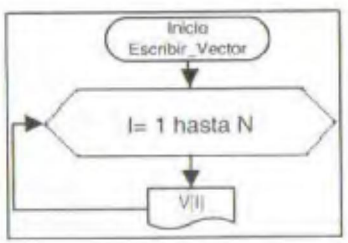
```
      2.2.1. Desde I = 1 hasta N Hacer
          2.2.1.1. Leer V[I]
      2.2.2. Fin_Desde
 2.3. Fin_Procedimiento Leer_Vector

3. Procedimiento Escribir_Vector(
                V : Arreglo [1..100] de Enteros
                N : Entero)
 3.1. Declarar variables
          I : Entero
 3.2. Inicio
      3.2.1. Desde I = 1 hasta N Hacer
          3.2.1.1. Escribir V[I]
      3.2.2. Fin_Desde
 3.3. Fin_Procedimiento Leer_Vector

4. Procedimiento Ordenar_Vector(
                V : Arreglo [1..100] de Enteros
                N : Entero)
 4.1. Declarar variables
          I, J, k, Aux : Entero
 4.2. Inicio
      4.2.1. Inc = N DIV 2
      4.2.2. Hacer
          4.2.2.1. Desde I = Inc + 1 hasta N hacer
              4.2.2.1.1. Hacer tmp = V[i]
                                 J = I-Inc
              4.2.2.1.2. Mientras (J>=0) Y (tmp<V[J])
                  4.2.2.1.2.1. Hacer V[J+Inc] = V[J]
                                     J = J - Inc
              4.2.2.1.3. Fin_mientras
              4.2.2.1.4. Hacer V[J+Inc] = tmp
          4.2.2.2. Fin_Desde
          4.2.2.3. Hacer Inc = Inc DIV 2
      4.2.3.. Mientras (Inc<>0)
 4.3. Fin_Procedimiento Ordenar_Vector
```

```
5. Procedimiento principal
 5.1. Declarar variables
          Vector : Arreglo [1..100] de Enteros
          M : Entero
 5.2. Leer M
 //Llamada al procedimiento Leer_Vector
 5.3. Leer_Vector(Vector,M)
 //Mostramos el vector ingresado
 5.4. Escribir_Vector(Vector,M)
 //Ordenamos el vector
 5.5. Ordenar_Vector(Vector,M)
 5.6. Escribir_Vector(Vector,M)
6. Terminar proceso
```

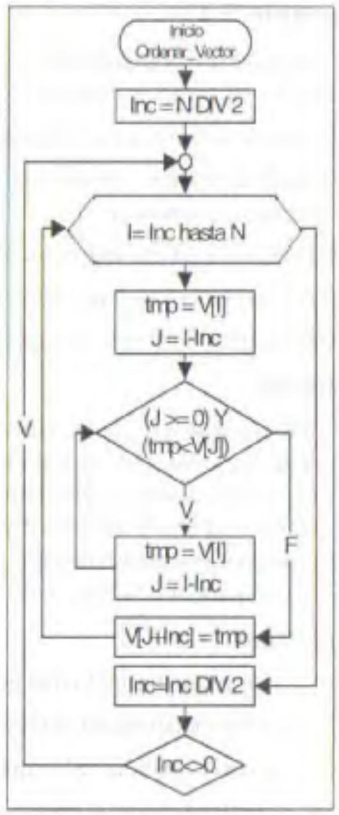
En la línea 5.4 llamamos al procedimiento Escribir_Vector, quien nos muestra el contenido del vector recién ingresado en la línea 5.3. Pero hacemos un nuevo llamado en la línea 5.6 del procedimiento Escribir vector, esta vez ya nos mostrará un vector ordenado.

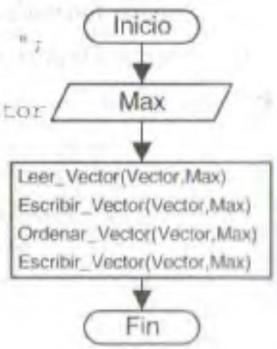
```
#include<iostream.h>
#include<conio.h>
void Leer_Vector(int V[], int N)
{clrscr();
 cout<<"Ingrese valores del
          vector"<<endl;
 for(int I=0;I<N;I++)
      cin>>V[I];
}

void Escribir_Vector(int V[], int N)
{
 clrscr();
 for(int I=0;I<N;I++)
      cout<<V[I]<<endl;
 cout<<"Pulse una tecla para
          continuar...";
 getch();
}
```






```
void Ordenar_Vector(int V[],
                    int N)
{
 int Inc,tmp, J, I;
 Inc = N/2;
 do
 {
 for(I=Inc;I<N;I++)
 {
      tmp = V[I];
      J = I-Inc;
      while((J>=0)&&
                (tmp<V[J]))
      {
           V[J+Inc] = V[J];
           J-=Inc;
      }
      V[J+Inc] = tmp;
 }
 Inc/=2;
 }while(Inc!=0);
}

void main()
{
 int Vector[100];
 int Max;
 cout<<"Ingrese número de elementos: ";
 cin>>Max;
 //Llamada al procedimiento Leer_Vector
 Leer_Vector(Vector,Max);
 //Mostramos el vector ingresado
 Escribir_Vector(Vector,Max);
 //Ordenamos el vector
 Ordenar_Vector(Vector,Max);
 //Mostramos el vector ordenado
 Escribir_Vector(Vector,Max);
}
```





Ejemplo 5.4

Construya un programa que realice las siguientes acciones. Considere un vector ordenado en forma ascendente

- ❖Genere un vector de 100 números aleatorios entre 0 y 100
- ❖Dado un número ingresado por el usuario, de como resultado la posición de la primera ocurrencia.
- ❖Dado un número ingresado ver cuantas veces se repite en el vector.
- ❖Almacene en vectores diferentes, los números pares y los impares.
- ❖Muestre el vector las veces que el usuario lo requiera.

Solución

Como es obvio este ejercicio tienes que ser desarrollado por medio de un pequeño menú de opciones. La generación de un vector de número aleatorios es muy fácil y por lo general todos los programadores tienen comandos especiales q nos ayudan en esta tarea. Para el caso del Pseudocódigo y el diagrama de flujo solo quedará indicado como GenerarNumero(), veremos en la programación el uso del comando Random() para la generación de número aleatorios

Algoritmo

Construir procedimiento que genere números aleatorios

Construir procedimiento para escribir vector

Implementar función de búsqueda de un elemento en el vector.

Implementar función q cuente el número de repeticiones

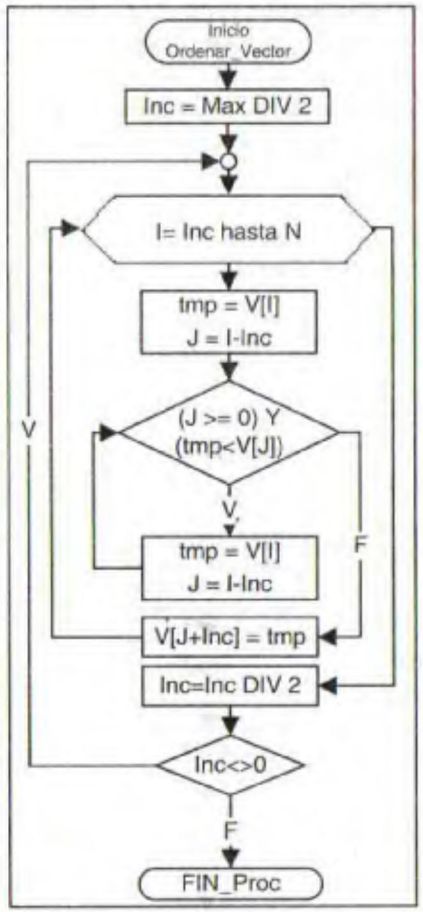
Implementar procedimiento de número pares e impares

Implementar funciones de menú

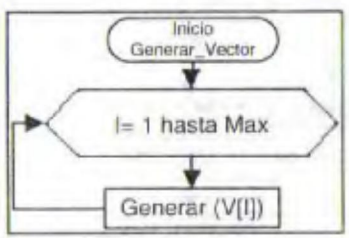
Implementar Programa Principal.

Pseudocódigo

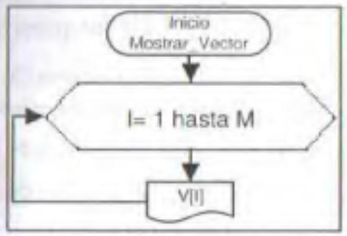
```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<ctype.h>
#define Verdadero 1
#define Falso 0
#define Max 10
```


```
void OrdenarVector(int
V[])
{
 int Inc,tmp,j;
 Inc=Max/2;
 do{
      for(int
i=Inc;i<Max;i++)
      {
            tmp=V[i];
            j=i-Inc;
            while((j>=0)&&

                 (tmp<V[j]))
            {

V[j+Inc]=V[j];
                 j-=Inc;
            }
            V[j+Inc]=tmp;
      }
      Inc/=2;
 }while(Inc!=0);
}
```




```
void Generar(int V[])
{     randomize();
      for(int i=0;i<Max;i++)
           V[i]=random(101);
      cout<<endl<<"Vector
                 generado...";
      cout<<endl<<"Presione una
         tecla para continuar";
      OrdenarVector(V);
      getch();
}
```



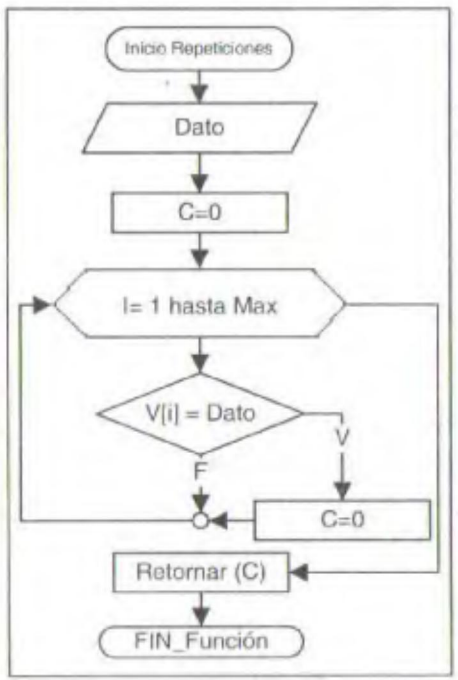

```
void MostrarVector(int V[], int M)
{     for(int i=0;i<M;i++)
           cout<<endl<<V[i];
      cout<<endl<<"Presione una
           tecla para continuar";
      getch();
}
```

```
int ParImpar(int V[],int VA[],int Caso)
{int c,i;
 c=0;
 switch(Caso)
 {    case 1: for(i=0;i<Max;i++)
              if(V[i]%2==0)
              {    VA[c]=V[i];
                   c++;
              }
              break;
      case 2: for(i=0;i<Max;i++)
              if(V[i]%2!=0)
              {    VA[c]=V[i];
                   c++;
              }
 }
 return c;
}
```

```
int Busqueda(int V[])
{int Inf,Sup,Cen,Dato;
 int NoEncontrado;
 cout<<endl<<"Ingrese dato a buscar: ";
 cin>>Dato;
 Inf=0;
 Sup=Max-1;
 NoEncontrado=Verdadero;
 while(Inf<=Sup && NoEncontrado==Verdadero)
```



```
{     Cen=(Inf+Sup)/2;
      if(V[Cen]==Dato)
           NoEncontrado=Falso;
      else
           if(V[Cen]<Dato)
                Inf=Cen+1;
           else
                Sup=Cen -1;
}
if(NoEncontrado)
     return -1;
else
     return Cen;
}
```




```
int Repeticiones (int V[])
{
      int Dato,c;
      cout<<"Ingrese dato a
      buscar repeticiones:";
      cin>>Dato;
      c=0;
      for(int i=0;i<Max;i++)
           if(V[i]==Dato)
                c++;
      return c;
}
char Menu()
```

```
{     char R;
      clrscr();
      cout<<endl<<"          Menú          ";
      cout<<endl<<"------------------------";
      cout<<endl<<"[G]. Generar vector";
      cout<<endl<<"[M]. Mostrar Vector";
      cout<<endl<<"[B]. BÉsqueda";
      cout<<endl<<"[R]. Repeticiones";
      cout<<endl<<"[P]. Pares/Impares";
      cout<<endl<<"[S]. Salir";
      cout<<endl<<"Opcion: ";
      do{
           cin>>R;
           R=toupper(R);
      }while(R!='G' && R!='M' && R!='B' && R!='R' &&
               R!='P' && R!='S');
      return R;
}

int MenuParImpar()
{     char R;
      clrscr();
      cout<<endl<<"      Menú Par/Impar      ";
      cout<<endl<<"--------------------------";
      cout<<endl<<"[P]. Ejecutar Pares";
      cout<<endl<<"[I]. Ejecutar impares";
      cout<<endl<<"[V]. Ver Vector pares";
      cout<<endl<<"[W]. Ver vector impares";
      cout<<endl<<"[S]. Salir";
      cout<<endl<<"Opcion: ";
      do{
           cin>>R;
           R=toupper(R);
      }while(R!='P' && R!='I' && R!='V' && R!='W' &&
```



```
               R!='S');
      return R;
}

void main()
{int V[Max],Par[Max],Impar[Max];
 int NI, NP,x;
 char R, R2;
 do{
      R=Menu();
      switch(R)
      {
           case 'G' : Generar(V);break;
           case 'M' : MostrarVector(V,Max);break;
           case 'B' : x=Busqueda(V);
                      if(x>=0)
                        cout<<endl<<"Encontrado en
                                    posición: "<<x;
                      else
                        cout<<"No Encontrado";
                      cout<<endl<<"Presione una tecla para
                                    continuar";
                      getch();
                      break;
           case 'R' : x=Repeticiones(V);
                      cout<<endl<<"Se repite "<<x<<"
                                    veces";
                      cout<<endl<<"Presione una tecla para
                                    continuar";
                      getch();
                      break;
           case 'P' : do{
                        R2=MenuParImpar();
                        switch(R2)
                        {
```

```
                        case 'P' :
                                NP=ParImpar(V,Par,1);
                                  break;
                        case 'I' :
                                NI=ParImpar(V,Impar,2);
                                  break;
                        case 'V' :
                                MostrarVector(Par,NP);
                                  break;
                        case 'W' :
                                MostrarVector(Impar,NI);
                                  break;
                    }
                  }while(R2!='S');
        }
    }while(R!='S');
}
```