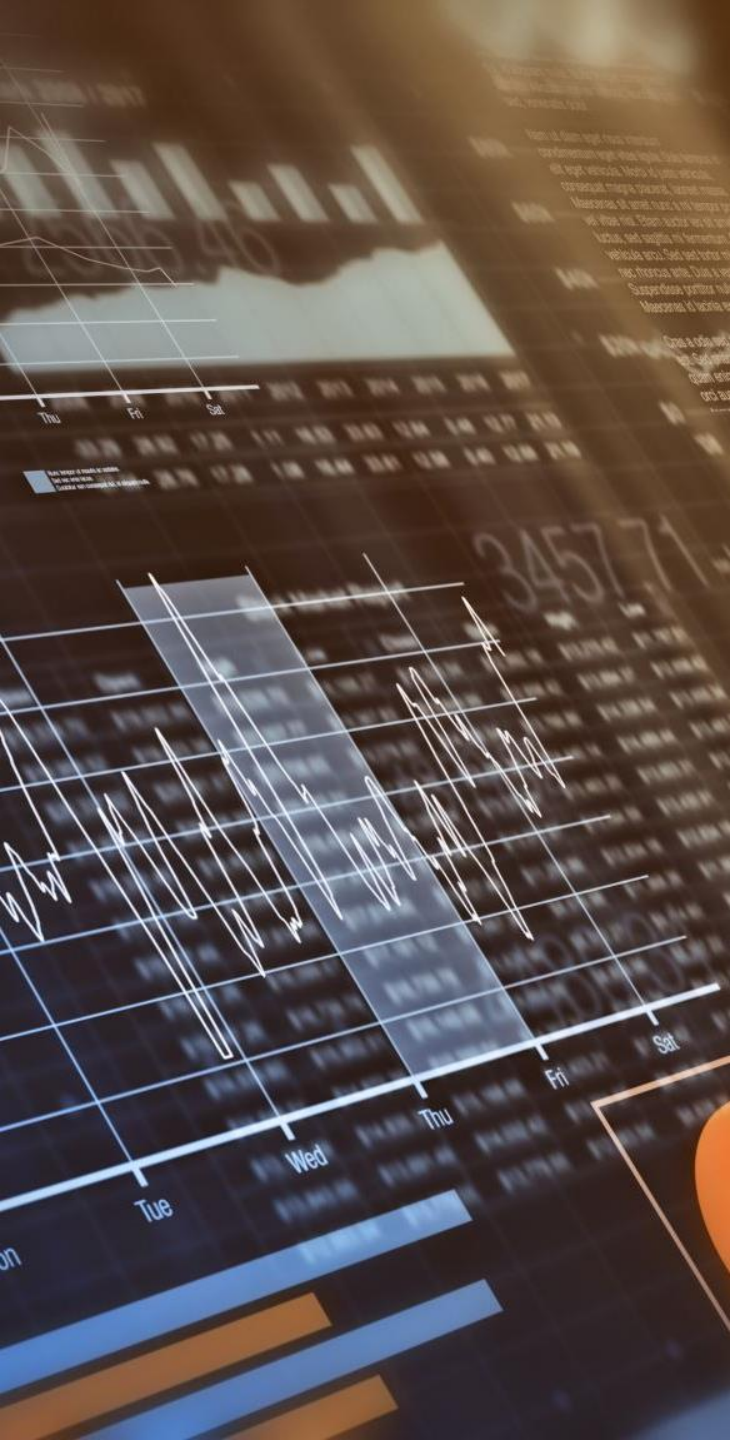


PROGRAMACIÓN AVANZADA

ING. INDUSTRIAL

FAC. DE INGENIERÍA

UNIVERSIDAD NACIONAL DE JUJUY

2DA PARTE

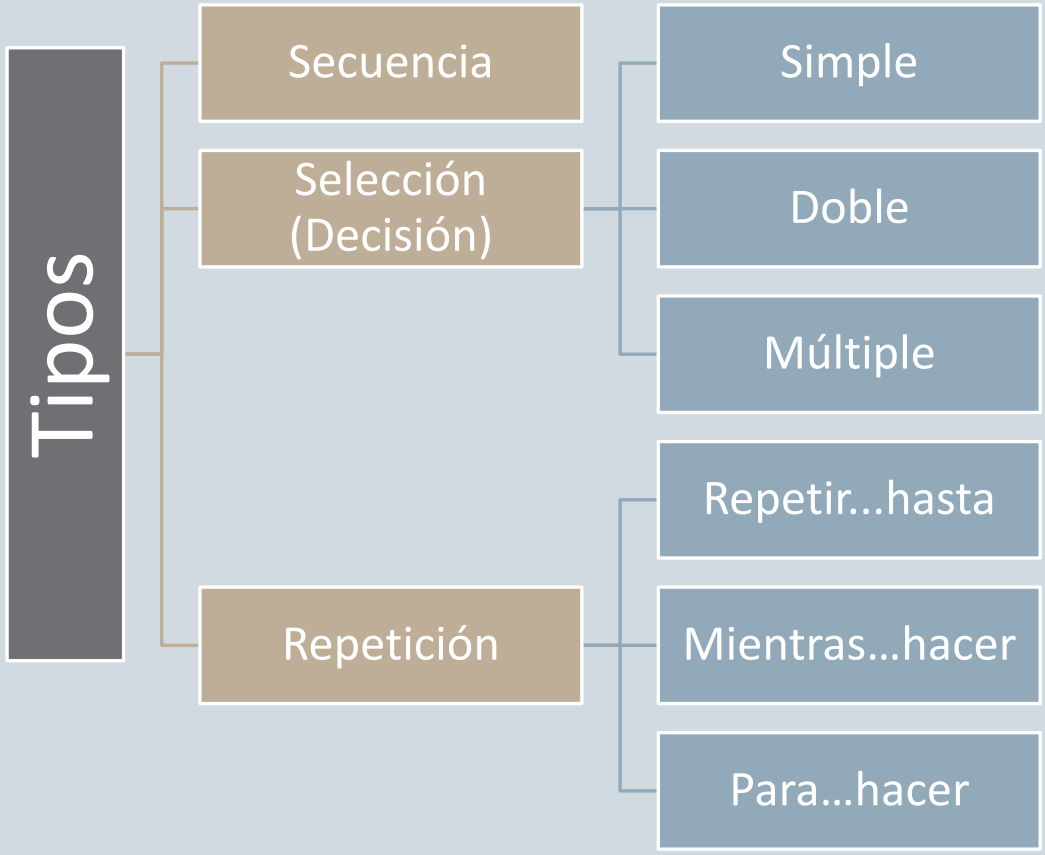
Introducción C++

Estructura de un programa en C++.Elementos de programas C++. Tipos de datos de C++. Expresiones. Entrada y Salida. **Estructuras de control**

Estructuras de Control

UNIDAD 1

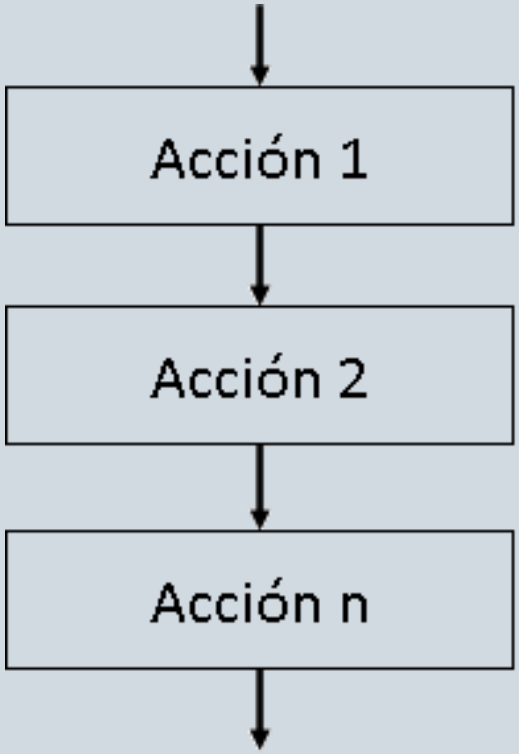
Estructuras de Control



Estructura Secuencial

Una **estructura secuencial** es aquella en la que **una acción** (instrucción) **sigue a otra** en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el final del proceso.

La **estructura secuencial** tiene **una entrada** y **una salida**.



Estructura Secuencial

Problema:

Mediante un programa, calcular la suma y el producto de dos números ingresados por el usuario.

Sean los números A y B

La suma S de dos números es S = A+B

El producto P es P = A*B.

El código en C++ es:

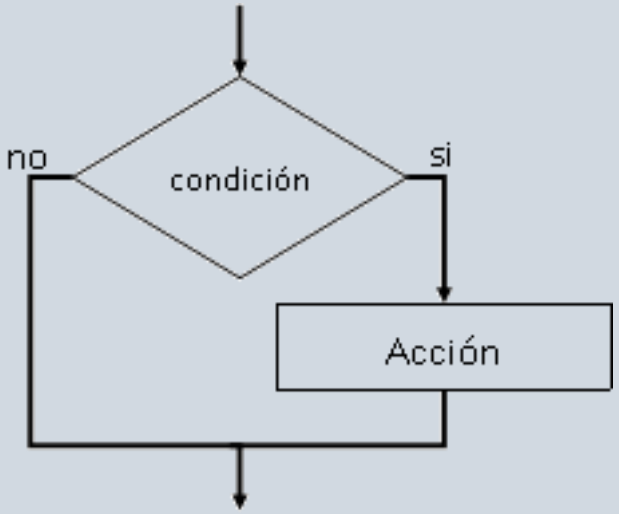
```
#include<iostream>
using namespace std;
int main(){
int A,B,S,P;
cin>>A;
cin>>B;
S=A+B;
P=A*B;
cout<<“La suma de A y B es ”<<S<<endl;
cout<<“El producto de A y B es ” <<P<<endl;
return 0;
}
```

Estructura de Decisión Simple

La **estructura alternativa simple si-entonces** (en inglés if-then) ejecuta una determinada acción cuando se cumple una determinada condición.

La selección **si-entonces** evalúa la condición y

- si la condición es **verdadera**, entonces **ejecuta** la acción o acciones,
- si la condición es **falsa**, entonces no hacer **nada**.



Estructura de Decisión Simple

FORMATO EN C++

if (expresión**)** Acción**;**

O

```
if (expresion){
        Accion1;
        …
        AccionN;
}
```

EJEMPLO

- Dado un número indicar si el mismo es par

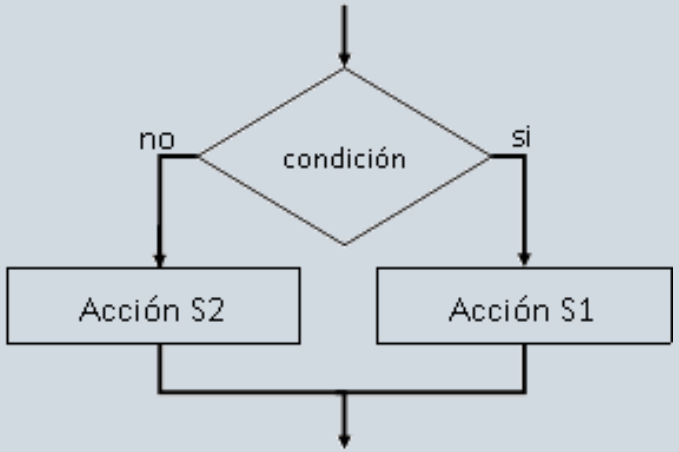
```
#include<iostream>
using namespace std;
int main(){
    int num;
    cin>>num;
    if (num%2==0)
        cout<<num<<" es par"<<endl;
    return 0;
}
```

Estructura de Decisión Doble

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre **dos** opciones o alternativas posibles, en función del **cumplimiento o no** de una determinada condición.

Si la condición es **verdadera**, se ejecuta la acción **S1** y,

si es **falsa**, se ejecuta la acción **S2**



Estructura de decisión doble

FORMATO EN C++

```
if (expresión)
        AccionV;
else
        AccionF;
```

o

```
if (expresion) {
                AccionV1;
                …
                AccionVN;}
else {
        AccionF1;
        …
        AccionFN;}
```

FORMATO EN C++

```
if (expresión)
        AccionV;
else {
        AccionF1;
        …
        AccionFN;}
```

o

```
if (expresion) {
                AccionV1;
                …
                AccionVN;}
else
        AccionF;
```

Estructura de decisión doble

EJEMPLO

Dado un número evaluarlo y mostrar si se trata de un número par o impar

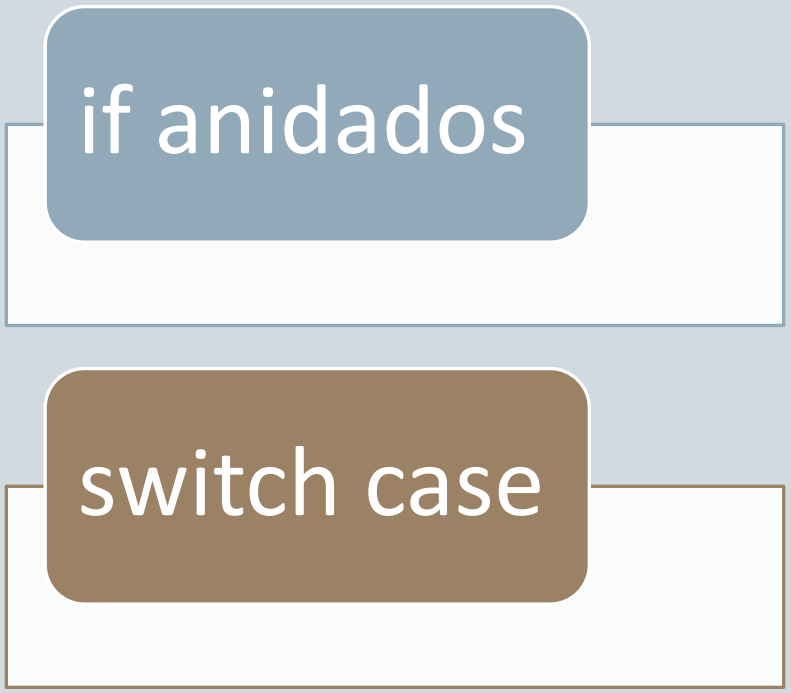
PROGRAMA EN C++

```
#include<iostream>
using namespace std;
int main(){
   int num;
   cin>>num;
   if (num%2==0)
        cout<<num<<" es par"<<endl;
   else
        cout<<num<<" es impar"<<endl;
   return 0;
}
```

Estructuras de Decisión Múltiple

Las **estructuras de decisión múltiple** en programación son mecanismos que permiten tomar decisiones basadas en **múltiples condiciones o alternativas**.

Estas estructuras ayudan a dirigir el flujo de ejecución del programa según los valores de las variables y las condiciones especificadas.



Estructura de decisión múltiple

Sentencias **if-else** anidadas

- Una sentencia **if** es anidada cuando la sentencia de la rama verdadera o la rama falsa, es a su vez es una sentencia **if.** Una sentencia **if anidada** se puede utilizar para implementar decisiones con varias alternativas o multi-alternativas.

```
if (condición1)
     sentencia1;
else if (condición2)
     sentencia2;
          .
          .
          .
else if (condiciónn)
     sentencian;
else
     sentencia;
```

Estructura de decisión múltiple

Instrucción **switch … case**

- La instrucción **switch** se utiliza cuando se requiere tomar una decisión entre múltiples posibilidades, de acuerdo con el valor de una variable o expresión ordinal.

```
switch (expresion)
{
   case valor_1:
      sentencias_1
   case valor_2:
      sentencias_2
   //Resto de bloques case
   case valor_n:
      sentencias_n
   default:
      sentencias_default;
}
```

Estructuras de decisión múltiple

¿Cuál elegir?

if anidados

Son más flexibles y permiten evaluar condiciones más complejas, pero pueden volverse difíciles de leer y mantener si hay muchas condiciones anidadas

switch case

Son más fáciles leer y escribir cuando se tienen múltiples opciones con valores específicos, pero son menos flexibles para evaluar condiciones más complejas.

Estructuras de decisión múltiple

Ejemplo:

Realizar un programa que permita mostrar cuantos días tiene un mes cuyo número se ingresa.

- if anidado V1 (condición simple)
- if anidado V2 (condición compleja)
- switch case

Estructuras de decisión múltiple

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Dame el mes: (1, 2, ..., 12): ";
   int mes;
   cin >> mes;

   if (mes > 0 && mes < 13)
      if (mes == 1)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 2)
         cout << "El mes tiene 28 o 29 dias.\n";
      else if (mes == 3)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 4)
         cout << "El mes tiene 30 dias.\n";
      else if (mes == 5)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 6)
         cout << "El mes tiene 30 dias.\n";
      else if (mes == 7)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 8)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 9)
         cout << "El mes tiene 30 dias.\n";
      else if (mes == 10)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 11)
         cout << "El mes tiene 30 dias.\n";
      else if (mes == 12)
         cout << "El mes tiene 31 dias.\n";
      else
         cout << "¡Imposible!\n";
   else
      cout << "El valor introducido no es válido\n";
}
```

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Dame el mes: (1, 2, ..., 12): ";
   int mes;
   cin >> mes;

   if (mes > 0 && mes < 13)
      if (mes == 1 || mes == 3 || mes == 5 || mes == 7 ||
          mes == 8 || mes == 10 || mes == 12)
         cout << "El mes tiene 31 dias.\n";
      else if (mes == 2)
         cout << "El mes tiene 28 o 29 dias.\n";
      else if (mes == 4 || mes == 6 || mes == 9 || mes == 11)
         cout << "El mes tiene 30 dias.\n";
      else
         cout << "¡Imposible!\n";
   else
      cout << "El valor introducido no es válido.\n";
}
```

Estructuras de decisión múltiple

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Dame el mes: (1, 2, ..., 12): ";
   int mes;
   cin >> mes;

   if (mes > 0 && mes < 13)
      switch(mes)
      {
         case 1:
         case 3:
         case 5:
         case 7:
         case 8:
         case 10:
         case 12:
            cout << "El mes tiene 31 dias.\n";
            break;
         case 4:
         case 6:
         case 9:
         case 11:
            cout << "El mes tiene 30 dias.\n";
            break;
         case 2:
            cout << "El mes tiene 28 o 29 dias.\n";
            break;
         default:
            cout << "¡Imposible!\n";
      }
   else
      cout << "El valor introducido no es válido.\n";
}
```

Práctica #2

- ***Evaluación de Costos de Producción***
- ***Problema:*** *Una fábrica produce tres tipos de productos: A, B y C. El costo de producción varía según el tipo de producto y el volumen de producción. La empresa desea determinar el* ***costo total de producción*** *en función del tipo de producto y la cantidad producida.*
- ***Instrucciones para resolver:***
  - *Definir el tipo de producto y la cantidad producida.*
  - *Calcular el costo basado en el tipo de producto y el costo unitario.*

Expresiones condicionales

El operador **? :**

Una **expresión condicional** tiene el formato

C ? A : B

y es una operación **ternaria** (tres operandos) en el que C, A y B son los tres operandos y **? :** es el operador.

Sintáxis:

condición **?** expresión1 **:** expresion2

condición *es una expresión lógica*

expresion1 y expresion2 *son expresiones compatibles de tipos*

Expresiones condicionales

EJEMPLO

Dados dos números enteros mostrarlos de mayor a menor.

PROGRAMA

```
#include<iostream>
using namespace std;
int main(){
   int n1,n2;
   cout<<“ingrese dos números ”;
   cin>>n1>>n2;
   n1>n2?cout<<n1<<“,”<<n2:cout<<n2<<“,”<<n1;
   return 0;
}
```

ESTRUCTURAS REPETITIVAS

MIENTRAS…HACER – REPETIR…HASTA – PARA…HACER

Estructuras repetitivas

- Las **estructuras repetitivas** se utilizan cuando es necesario **repetir** un cierto **bloque de código** un número determinado o indeterminado de veces.

while ()

- Repetir mientras

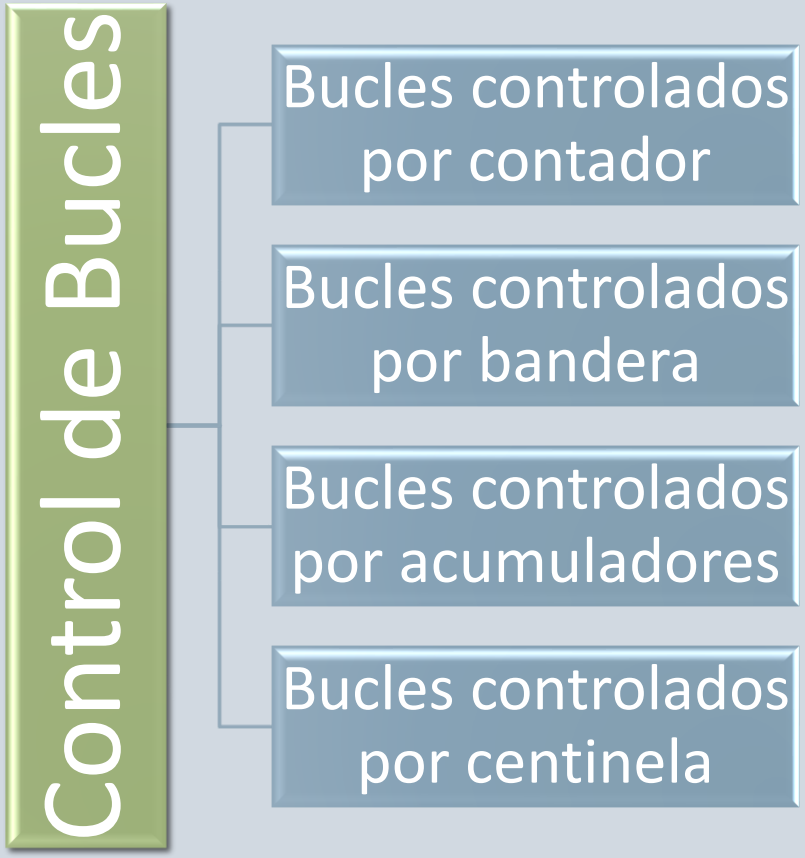
do { } while ();

- Hacer mientras

for(; ;)

- Para … hacer

Control del bucle



Bucle while

Un bucle **while** tiene asociado un bloque de sentencias que se ejecutarán secuencialmente **mientras** la **condición de entrada** al bucle sea **cierta**.

La condición se evalúa al comienzo de la estructura. Esto supone que el bloque de instrucciones **puede no ejecutarse** ninguna vez si la condición es inicialmente falsa.

```
while (condicion)
{
    bloque de sentencias;
}
```

Bucle while

PROBLEMA

- Calcular el factorial de un valor entero n, donde n está dado por el usuario.

PROGRAMA

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main() {
6      int i, n, factorial;
7      cout <<"Dame el valor de n: ";
8      cin >> n;
9      factorial = 1;
10     i = 1;
11     while (i <= n) {
12         factorial = factorial * i;
13         i = i + 1;
14     }
15     cout << "El factorial de " << n << " es " << factorial << endl;
16     return 0;
17 }
18
19
20
```

Bucle do…while

Es una **variante** especial del bucle **while**. Al contrario que el bucle while, que comprueba la condición antes de entrar en el bucle, el bucle **do - while** la evalúa al **final** del bucle. Esto implica que el bucle se ejecutará **al menos una vez**.

```
do
{
   bloque de sentencias;
}
while (condicion);
```

Bucle do…while

PROBLEMA

- Calcular el factorial de un valor entero n, donde n está dado por el usuario.

PROGRAMA

```
#include<iostream>

using namespace std;

int main() {
    int i, n, factorial;
    cout <<"Dame el valor de n: ";
    cin >> n;
    factorial = 1;
    i = 1;
    do {
        factorial = factorial * i;
        i = i + 1;
    }while (i <= n);
    cout << "El factorial de " << n << " es " << factorial << endl;
    return 0;
}
```

Bucle for

El bucle **for** está concebido fundamentalmente para ejecutar sus sentencias asociadas un **número fijo** de veces.

Por tanto, cuando el número de iteraciones necesarias para realizar una tarea en una parte del programa se conoce de antemano, lo más lógico es el empleo de una estructura tipo **for**, que controla automáticamente el número de repeticiones.

El diseño sintáctico de la sentencia compuesta for es la de integrar un **contador**, con su valor **inicial**, su valor **final** y un **valor fijo de incremento**.

```
for (inicializacion; condicion; incremento)
{
   bloque de sentencias;
}
```

Bucle for

PROBLEMA

- Calcular el factorial de un valor entero n, donde n está dado por el usuario.

PROGRAMA

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main() {
6      int i, n, factorial;
7      cout <<"Dame el valor de n: ";
8      cin >> n;
9      factorial = 1;
10
11     for (i=1;i<=n;i++)
12         factorial = factorial * i;
13
14     cout << "El factorial de " << n << " es " << factorial << endl;
15     return 0;
16 }
17
18
19
```

Equivalencia entre for y while

Las siguientes estructuras son equivalentes:

```
for (inicializacion;  condicion;  incremento) {
   // bloque de instrucciones
}
```

```
inicializacion;
while (condicion) {
   // bloque de instrucciones
   incremento;
}
```

Ciclos anidados

En muchas aplicaciones se requiere utilizar dos o más ciclos anidados, de manera que un ciclo exterior no complete una iteración hasta que los ciclos interiores hayan finalizado.

Los bucles anidados constan de un **bucle externo** con uno o más **bucles internos**. Cada vez que se repite el **bucle externo**, los **bucles internos** se repiten, se vuelven a evaluar los componentes de control y se ejecutan todas las iteraciones requeridas.

Ejemplo

```c
#include <stdio.h>
/* constantes globales */
const int num_lineas = 5;
const char blanco =' ';
const char asterisco = '*';

int main(){
    int fila, blancos, cuenta_as;
    puts(" "); /* Deja una línea de separación */
    /* bucle externo: dibuja cada línea */
    for (fila=1; fila<=num_lineas; fila++){
        putchar('\t');

        /*primer bucle interno: escribe espacios */
        for (blancos = num_lineas-fila; blancos > 0; blancos--)
            putchar (blanco);

        for (cuenta_as = 1; cuenta_as < 2*fila; cuenta_as++)
            putchar(asterisco);

        /* terminar línea */
        puts (" ");
    } /* fin del bucle externo */
return 0;
}

```

options | compilation | execution

```
    *
   ***
  *****
 *******
*********
```

Triángulo isósceles

Práctica #3

Elabore un programa que pida al usuario un entero positivo *n*. Luego, el programa debe pedir al usuario *n* enteros usando un ciclo. El programa deberá encontrar el mayor, menor, y el promedio de los valores dados por el usuario e imprimir los resultados al final.

Fin Unidad 1