- Unidad 3

INSTRUCCIONES DE REPETICION

(Capitulo 5 bibliografía)

Estructuras Básicas de ciclos – Ciclos While

» Ventre, Luis O.

Intro

- Los programas vistos hasta ahora han ilustrado conceptos de programación implicados con capacidad de **entrada, salida, asignación y selección**.

- Muchos problemas requieren de una capacidad de repetición, donde una sección del programa debe ejecutarse reiteradas veces con distintos conjuntos de datos con los objetivos de **contar, acumular**, **validar, e incluso permitir el constante ingreso de datos y el recalculo de los valores de salida** que solo se detiene al introducir un **valor centinela.**

- Estas secciones de código repetitivas son llamadas generalmente **ciclos**. Y cada repetición del mismo se conoce como **iteración** o paso a través del ciclo.

Estructuras Básicas de ciclos

- Construir una sección de código repetitiva requiere cuatro elementos:

1) Instrucción de repetición
2) Condición de repetición
3) Instrucción Condición de inicio
4) Instrucción de Salida.

1) <u>Instrucción de repetición:</u>

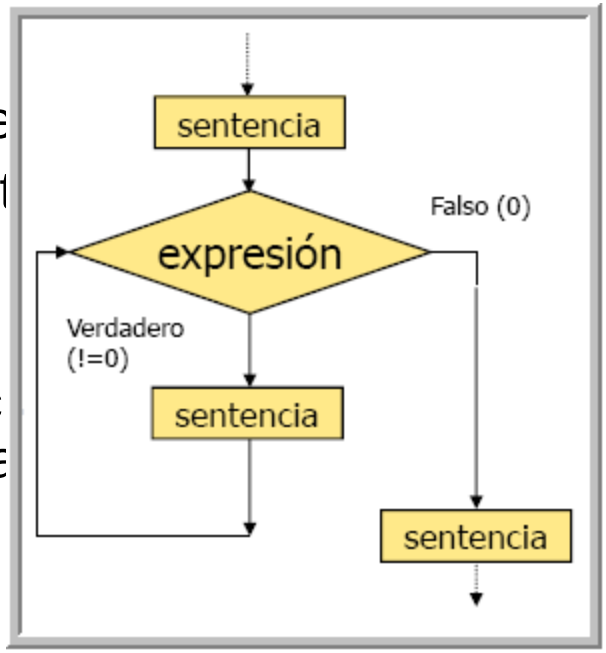
Esta define los **limites** que contienen la *<u>sección de código</u>* repetitiva. Y controla si el código se ejecutara o no. C++ incorpora 3 diferentes:

WHILE **-** **FOR** **-** **DO WHILE**

2) Condición de repetición:

Es la condición que debe evaluarse para
condiciones válidas son iguales a las de las e

Si la condición es verdadera el ciclo es ejecut



3) Instrucción Condición de inicio:

Instrucción que establece la condición al inic
siempre antes de que la condición sea eva
para asegurar la ejecución correcta del ciclo.

4) Instrucción de Salida:

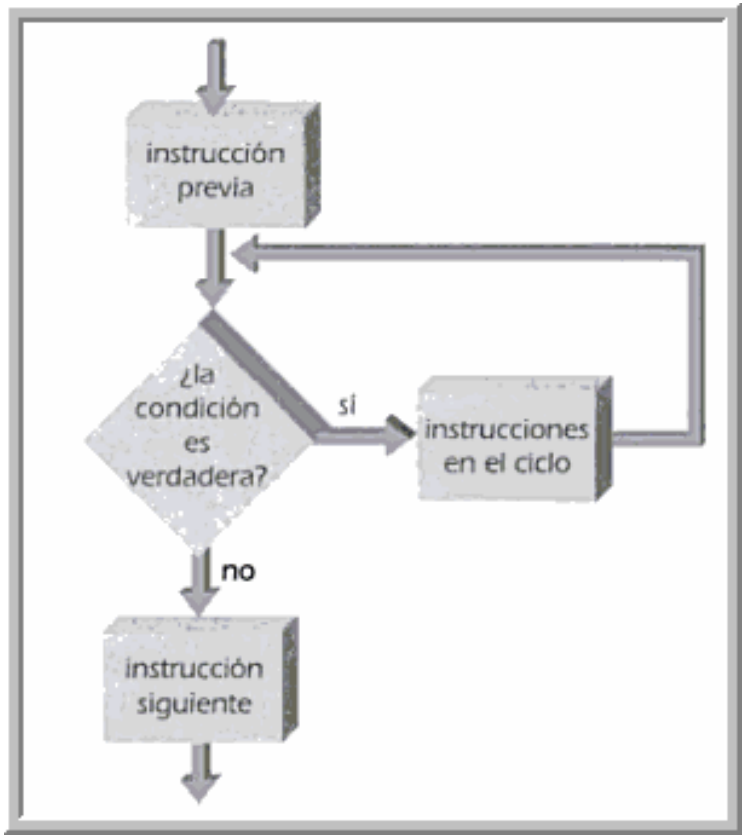
Debe existir en el ciclo, una instrucción que permita volver falsa la condición de ejecución del ciclo. Esto es necesario con el objetivo de poder detener las repeticiones en algún momento.

Estructuras Básicas de ciclos

- **Ciclos de prueba preliminar y posterior:**

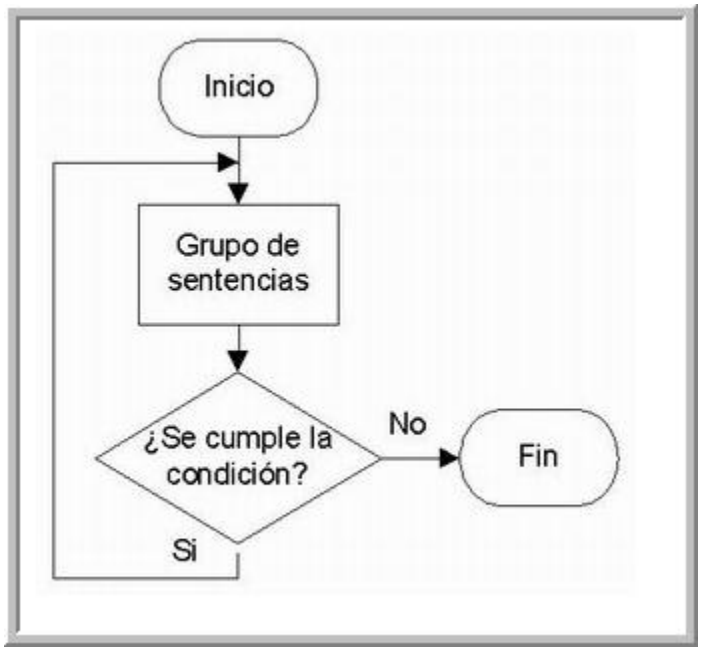
  La condición que se esta probando puede ser evaluada al principio o al final del ciclo.

  Los ciclos de prueba preliminar también conocidos como **ciclos controlados a la entrada**, evalúan la condición antes de comenzar el ciclo, si esta es **falsa, el ciclo no se ejecuta nunca** y sigue la instrucción posterior al bloque de repetición. Las instrucciones **while y for** son ejemplos de estos ciclos.



Estructuras Básicas de ciclos

- Un **ciclo de prueba posterior,** evalúa la condición al final de la ejecución de las instrucciones del bloque repetitivo. Estos ciclos son llamados **ciclo controlados en la salida.**



- Puede observarse que siempre el bloque de instrucciones **se ejecuta al menos una vez.**

- La instrucción **do while** es un ejemplo de este tipo de ciclo.

- **Para evitar ciclos infinitos, la condición debe actualizarse dentro del ciclo……..**

- **Ciclos de cuenta fija vs de condición variable**

  Además del lugar donde se prueba la condición las secciones de código repetitivo también se clasifican según el tipo de condición que se prueba:

  - de cuenta fija - de condición de variable

  En un ciclo de cuenta fija, la condición se usa para dar **seguimiento al numero de repeticiones**, y se ejecuta un numero fijo predefinido de veces. El bucle finaliza al llegar al valor deseado.

  En un ciclo de condición variable, puede no conocerse el limite al inicio, por ende el fin del ciclo no depende de que se alcance una cuenta sino mas bien del cambio de una **variable que puede cambiar en cada paso del ciclo.**

Ciclos WHILE

- La instrucción WHILE se utiliza para armar un ciclo de la siguiente manera:

```
while (expresion)
instrucciónA;
```

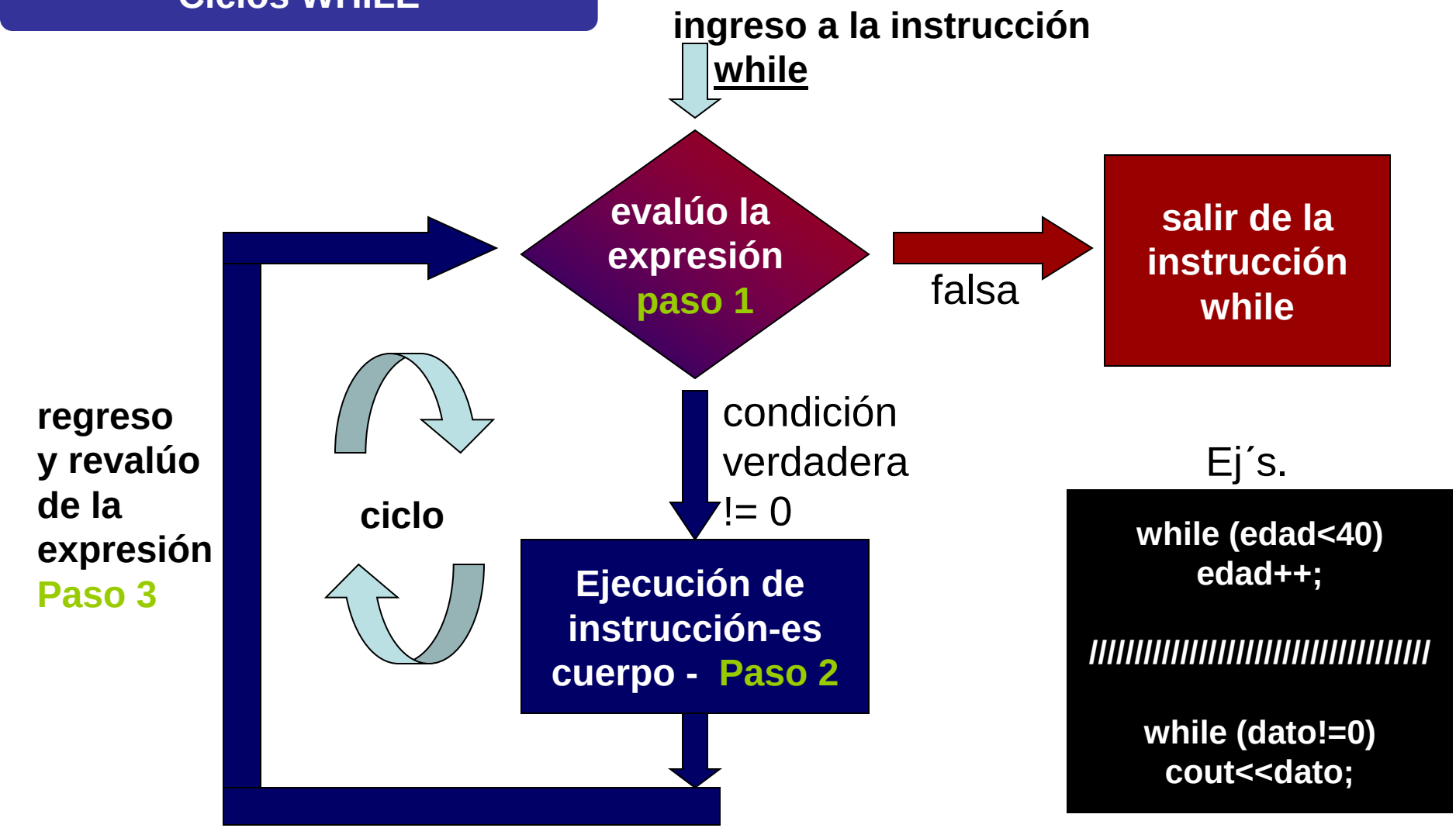
- **EXPRESION**: es la condición probada para ver si se ejecuta la *instrucciónA*.

- Si EXPRESION es verdadero != 0, diferente de 0, se ejecuta instruccionA.

- InstruccionA se ejecuta hasta que EXPRESION evalúe a 0.

- Ver en la próxima grafica los dos puntos de entrada y los dos puntos de salida de la instrucción

Ciclos WHILE



Ej´s.

```
while (edad<40)
edad++;

//////////////////////////////////

while (dato!=0)
cout<<dato;
```

Ciclos WHILE

- **Veamos un ejemplo:**

```
while (cuenta<=10)
cout<<cuenta;
```

- **Que problema tiene esto?**
- **Debo inicializar la variable cuenta:**

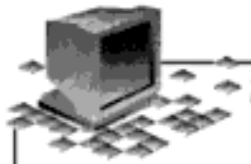
```
cuenta=1;
while (cuenta<=10)
cout<<cuenta;
```

- **Y debo poder hacer falsa la condición**
  para esto puedo utilizar una instrucción compuesta en lugar de una simple

```
cuenta=1;
while (cuenta<=10)
{
cout<<cuenta;
cuenta++;
}
```

Ciclos WHILE

- **El programa anterior**



Programa 5.1

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

int main()
{
  int cuenta;

  cuenta = 1;              // inicializa cuenta
  while (cuenta <= 10)
  {
    cout << cuenta << "   ";
    cuenta++;              // incrementa cuenta
  }

  return 0;
}
```

La salida del programa 5.1 es:

```
1   2   3   4   5   6   7   8   9   10
```

Ciclos WHILE

- **Otro Ejemplo – cuenta fija**

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

// un programa para convertir Celsius a Fahren
int main()
{

  const int MAX_CELSIUS    = 50;
  const int VALOR_INICIAL = 5;
  const int TAMANHO_PASO   = 5;
  int    celsius;
  double fahren;

  cout << "GRADOS    GRADOS\n"
       << "CELSIUS   FAHRENHEIT\n"
       << "-------   ----------\n";

  celsius = VALOR_INICIAL;

  // establece los formatos de salida para numeros de p
  cout << setiosflags(ios::showpoint)
       << setprecision(2);
```

```
  while (celsius <= MAX_CELSIUS)
  {
    fahren = (9.0/5.0) * celsius + 32.0;
    cout << setw(4)  << celsius << fixed
         << setw(13) << fahren << endl;
    celsius = celsius + TAMANHO_PASO;
  }

  system("PAUSE");
  return 0;
}
```

```
GRADOS    GRADOS
CELSIUS   FAHRENHEIT
-------   ----------
   5         41.00
  10         50.00
  15         59.00
  20         68.00
  25         77.00
  30         86.00
  35         95.00
  40        104.00
  45        113.00
  50        122.00
Presione una tecla para continuar . . .
```

Ciclos WHILE Interactivos

- La combinación de un ciclo repetitivo con la capacidad de introducir datos produce programas potentes y adaptables

- El siguiente programa usa el ciclo WHILE para aceptar y luego imprimir en pantalla 4 números introducidos por el

```
Este programa le piede que ingrese 4 numeros.

Ingrese un numero: 100
El numero ingresado es 100
Ingrese un numero: 99
El numero ingresado es 99
Ingrese un numero: 98
El numero ingresado es 98
Ingrese un numero: 97
El numero ingresado es 97
Presione una tecla para continuar . . .
```

```
#include <iostream>

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

int main()
{
  const int MAXNUMS = 4;
  int cuenta;
  double num;

  cout << "\nEste programa le piede que ingrese "
       << MAXNUMS << " numeros.\n";
  cuenta = 1;

  while (cuenta <= MAXNUMS)
  {
    cout << "\nIngrese un numero: ";
    cin  >> num;
    cout << "El numero ingresado es " << num;
    cuenta++;
  }
  cout << endl;

  system("PAUSE");
  return 0;
}
```

Ciclos WHILE Interactivos

- Si el objetivo del programa fuera calcular **suma total de números** ingresados, se debería incluir una instrucción como las vistas en la sección 3.1 (acumulación).

```
int main()
{
  const int MAXNUMS = 4;
  int cuenta;
  double numero, total;

  cout << "\nEste programa le pide que ingrese "
       << MAXNUMS << " numeros.\n";
  cuenta = 1;
  total = 0;

  while (cuenta <= MAXNUMS)
  {
    cout << "\nIngrese numero: ";
    cin  >> numero;
    total = total + numero;
    cout << "Ahora el total es de " << setprecision(7) << total;
    cuenta++;
  }

  cout << "\nEl total final es " << setprecision(7) << total << endl;

  system("PAUSE");
```

```
Este programa le pide que ingrese 4 numeros.

Ingrese numero: 100
Ahora el total es de 100
Ingrese numero: 99
Ahora el total es de 199
Ingrese numero: 98
Ahora el total es de 297
Ingrese numero: 97
Ahora el total es de 394
El total final es 394
Presione una tecla para continuar . . .
```

Ciclos WHILE Interactivos

- Si el objetivo ahora fuera calcular el **promedio de números** ingresados, se debería determinar donde calcular el promedio y como?

```
{
  const int MAXNUMS = 4;
  int cuenta;
  double numero, total, promedio;

  cout << "\nEste programa le pide que ingrese "
       << MAXNUMS << " numeros.\n";
  cuenta = 1;
  total = 0;

  while (cuenta <= MAXNUMS)
  {
    cout << "Enter a numero: ";
    cin >> numero;
    total = total + numero;
    cuenta++;
  }

  cuenta--;
  promedio = total / cuenta;
  cout << "\nEl promedio de los numeros es " << promedio << endl;
```

```
Este programa le pide que ingrese 4 numeros.
Enter a numero: 100
Enter a numero: 99
Enter a numero: 98
Enter a numero: 97

El promedio de los numeros es 98.5
Presione una tecla para continuar . . .
```

Ciclos WHILE Interactivos

- En los programas vistos, todos los **ciclos son de cuenta fija**, ya que se utiliza un **contador** para determinar el fin del ciclo

- En muchos programas es necesario construir **ciclos de condición variable.**

- Existen casos donde es necesario ingresar datos de manera continua, en ellos se ingresan datos hasta que un determinado valor, denominado "**centinela**" es ingresado y este detiene el ciclo.

- Evidentemente un requisito fundamental es que el valor centinela no pueda confundirse con los valores esperados en el programa.

- Veamos un ejemplo

Ciclos WHILE Interactivos

```
int main()
{
  const int CALIFICACIONMAYOR = 100;
  double calificacion, total;

  calificacion= 0;
  total = 0;
  cout << "\nPara detener el ingreso de cali
       << "teclee cualquier numero";
  cout << "\n mayor a 100.\n\n";

  while (calificacion <= CALIFICACIONMAYOR)
  {
    total = total + calificacion;
    cout << "Ingese una calificacion: ";
    cin  >> calificacion;
  }

  cout << "\nEl total de las calificaciones es " << total << endl;

  system("PAUSE");
  return 0;
```

```
Para detener el ingreso de calificaciones,
 mayor a 100.

Ingese una calificacion: 100
Ingese una calificacion: 99
Ingese una calificacion: 98
Ingese una calificacion: 100
Ingese una calificacion: 50
Ingese una calificacion: 101

El total de las calificaciones es 447
Presione una tecla para continuar . . .
```

Centinela

Ciclos WHILE Interactivos

- Dos instrucciones útiles relacionadas con las instrucciones de repetición son:

- BREAK - CONTINUE.

- Una es la opuesta a la otra, la instrucción BREAK, obliga a una interrupción inmediata o salida del ciclo. Como se vio en switch. Se usa en while, for, do while.

- **Esta instrucción viola los principios puros de programación estructurada porque proporciona una segunda salida no estándar de un ciclo.**

- **Es útil cuando se detecta una condición inusual.**

- La instrucción CONTINUE, indica que la siguiente iteración del ciclo comienza de inmediato. Útil cuando no se deben evaluar condiciones.

Ciclos WHILE Interactivos

Ej de instrucción break:

```
while(cuenta<=10)
{
   cout<<“introduzca un numero : “;
   cin>>num;
   if (num > 76)
   {
      cout << “Perdiste!\n”;
      break;       //interrumpe el ciclo y salta
    }
    else
       cout << “Sigue intentandolo!\n”;
   cuenta++;
}
                              //break salta hasta aquí.
```

Ej. instrucción continue

```
while(cuenta<30)
{
   cout<<“introduzca una calificacion: “;
   cin>>calificacion;
   if (calificacion<0 || calificacion>100)
       continue;
   total=total +calificacion;
   cuenta++;
}
```

- La instrucción nula, es una instrucción que no hace nada y se utiliza donde se requiere una instrucción desde el punto de vista sintáctico. Son usadas de manera típica en instrucciones del tipo while o for.

Ciclos **FOR**

- Como hemos visto la construcción de ciclos en C++ puede realizarse de varias formas. Una de estas es utilizando una **instrucción FOR**, cuya traducción puede interpretarse como **"para" o "para cada".**

- En muchas situaciones, en particular aquellas que se necesita de una **cuenta fija,** o sea se conoce en un comienzo la cantidad de iteraciones a realizar es mas fácil el **formato de la instrucción FOR.**

  - **La sintaxis de la instrucción FOR tiene la siguiente forma:**

```
for (lista de inicialización; expresión; lista de alteración)
    instrucción;
```

Ciclos FOR

- Aunque esta instrucción **parezca compleja**, es bien simple si analizamos sus **componentes por separado**.

- La sintaxis de esta instrucción muestra entre paréntesis 3 elementos OPCIONALES, **separados por ";";** aunque los elementos pueden faltar, los ";" deben estar SIEMPRE.

```
for (lista de inicialización; expresión; lista de alteración)
      instrucción;
```

- **lista de inicialización:** en su forma mas común, consiste de una sola instrucción usada para **establecer el valor inicial de un contador**.

Ciclos FOR

```
for (lista de inicialización; expresión; lista de alteración)
instrucción;
```

- **expresión**: contiene el valor **máximo o mínimo** que puede tener el contador y determina **cuando se termina el ciclo**.

- **lista de alteración:** proporciona el valor de incremento que se **suma o resta del contador cada vez que se ejecuta el ciclo**.

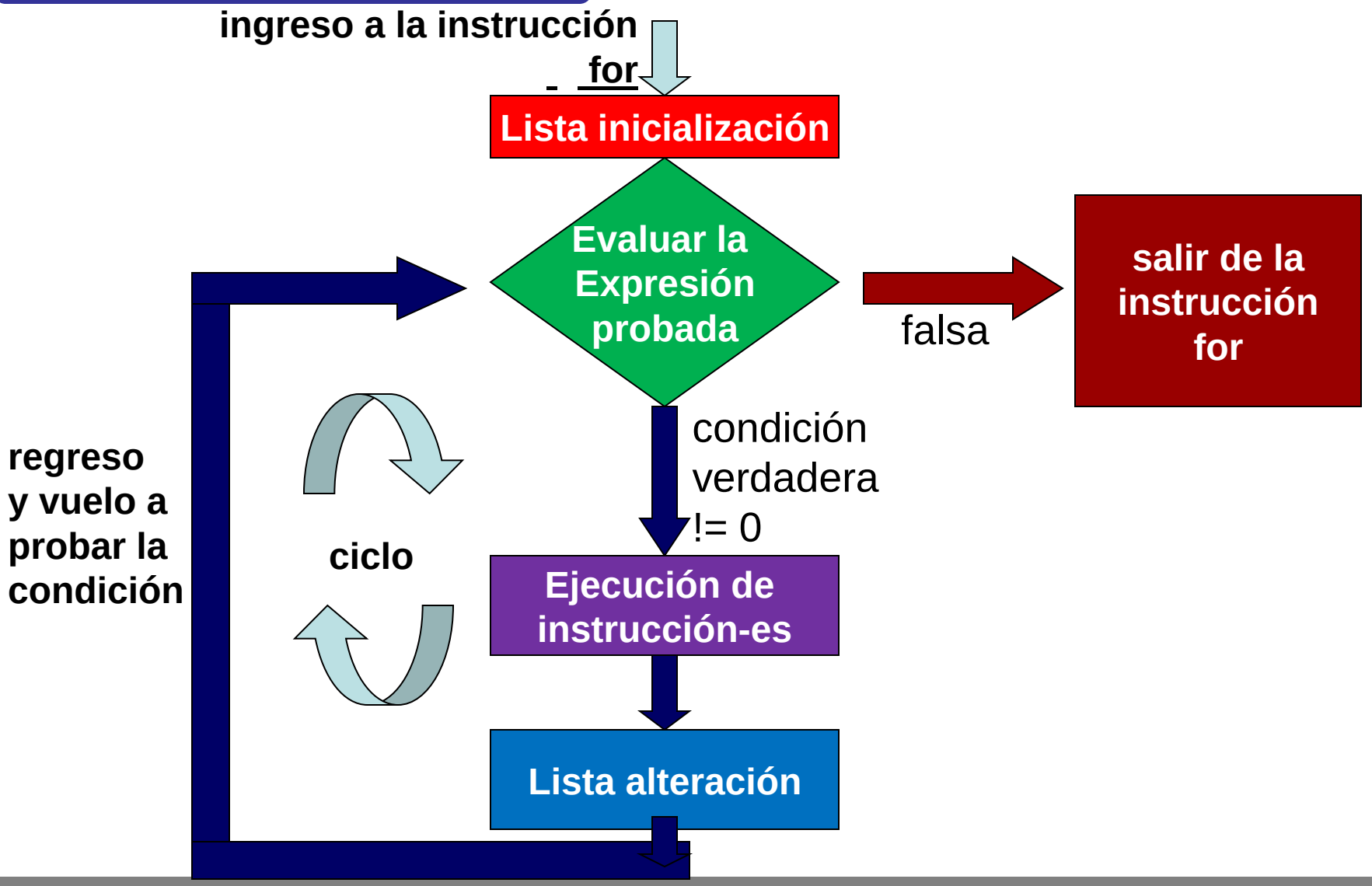
- **Ej. simples de esta instrucción son:**

```
for (cuenta=1   ; cuenta <10   ; cuenta = cuenta +1)
      cout<<cuenta;
```

```
for (i=1; i <=15; i = i +2)
      cout<<i;
```

```
for (lista de inicialización; expresión; lista de alteración)
    instrucción-es;
```

Diagrama de Flujo - FOR



Ciclos FOR

```
for (cuenta=1; cuenta <10; cuenta = cuenta +1)
cout<<cuenta;
```

- En esta instrucción, la **variable contadora debe haber estado DECLARADA anteriormente**, y se llama cuenta. La misma es inicializada en valor 1, y se repite el ciclo incrementándose en 1 siempre y cuando sea menor que 10.

```
for (i=5; i <=15; i = i +2)
cout<<i;
```

- **En esta instrucción, la variable contadora debe haber estado DECLARADA anteriormente, y se llama i. La misma es inicializada en valor 5, y se repite el ciclo incrementándose en 2 siempre y cuando su valor sea menor o igual a 15.**

Ciclos FOR

- **Ejemplo:**

```
// PGM5-09
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
  const int MAXCUENTA = 5;
  int cuenta;

  cout << "NUMERO     RAIZ CUADRADA\n";
  cout << "------     -------------\n";

  cout << setiosflags(ios::showpoint);

  for (cuenta = 1; cuenta <= MAXCUENTA; cuenta++)
     cout << setw(4)  << cuenta
          << setw(15) << sqrt(double(cuenta)) << endl;


  system("PAUSE");
  return 0;
}
```

```
NUMERO     RAIZ CUADRADA
------     -------------
   1         1.00000
   2         1.41421
   3         1.73205
   4         2.00000
   5         2.23607
Presione una tecla para continuar . . .
```

Ciclos FOR

- **Ciclo WHILE equivalente**

```
cuenta=1;
while(cuenta<=MAXCUENTA)
{cout<<setw(4)<<cuenta
      <<setw(15)<<sqrt(double(cuenta)<<endl;
cuenta++;
}
```

```
for( [expre1]; [expre2]; [expre3] )
        sentencia;

for( [expre1]; [expre2]; [expre3] )
{
      sentencia 1;
       .....
      sentencia n;
}
```

Obsérvese que esto equivale a:

```
      expre1;
      while (expre2)
      {
        sentencia;
        expre3;
      }
```

Ciclos FOR

- **Puede determinar la salida del siguiente programa?**

```
// PGM5-10
#include <iostream>
using namespace std;

int main()
{
  int cuenta;

  for (cuenta = 2; cuenta <= 20; cuenta = cuenta + 2)
    cout << cuenta << "  ";

  system("PAUSE");
  return 0;
}
```

```
2  4  6  8  10  12  14  16  18  20  Presione una tecla para continuar . . .
```

Ciclos FOR

- Como se menciono anteriormente, las expresiones dentro del paréntesis **son opcionales**. La instrucción FOR solo asegura que todas las expresiones en la lista de **inicialización se ejecutaran una vez**, antes de la evaluación de la expresión, y que todas las expresiones en la **lista de alteración se ejecutan al final del ciclo** antes que se vuelva a verificar la expresión probada.
- Por lo tanto el programa anterior tiene los siguientes equivalentes:

```
// PGM5-10C
#include <iostream>
using namespace std;

int main()    // todas las expresiones dentro del parentesis del for
{
  int cuenta;

  for (cuenta = 2; cuenta <= 20; cout << cuenta << "  ", cuenta = cuenta + 2);

  system("PAUSE");
  return 0;
}
```

Ciclos FOR

- Como pudo observarse en los distintos casos, **faltaban algunos de los elementos en la expresión** entre paréntesis de la instrucción FOR; estando reemplazados en el **lugar adecuado en el programa**.

- Es importante destacar que en la ultima versión, se encontraron separados por “,”, **dos elementos en la lista de alteración**. El uso de **comas para separar elementos en las listas de inicialización y de alteración se requiere si cualquiera de estas dos listas contiene mas de un elemento.**

- **Agregar elementos distintos a las variables de control del ciclo y sus condiciones de actualización dentro del ciclo tiende a CONFUNDIR su legibilidad. Por lo tanto mantener “limpia” la estructura es una buena practica de programación.**

- **Ciclo FOR o WHILE?,** depende del estilo de programación, al ser ambos **prueba preliminar** una alternativa es **FOR para cuenta fija y WHILE para cond. vble.**

Ciclos FOR

- La instrucción **CONTINUE** en los ciclos FOR, transfiere el control a la **lista de alteración**, no a la expresión como en while.
- La instrucción **BREAK** opera igual que en los ciclos while.

- TECNICAS DE PROGRAMACION CON CICLOS

- Es de importancia destacar que con la instrucción FOR, se pueden utilizar todas las composiciones de instrucciones de selección, acumulación, ingreso de datos, etc tal cual la instrucción while.

- A continuación veremos un ejemplo, que nos permite dentro de un bucle FOR de cuenta fija, **ingresar números y calcular el promedio al salir del mismo.**

Ciclos FOR

```
// Este programa calcula el promedio de MAXCUENTA
// numeros entrados por el usuario
int main()
{
  const int MAXCUENTA = 4;
  int cuenta;
  double numero, total, promedio;

  total = 0.0;

  for (cuenta = 0; cuenta < MAXCUENTA; cuenta++)
  {
    cout << "Ingrese un numero:
    cin  >> numero;
    total = total + numero;
  }

  promedio = total / MAXCUENTA;
  cout << "El promedio de los datos ingresados es "
       << promedio << endl;
```

```
Ingrese un numero: 150
Ingrese un numero: 685
Ingrese un numero: 542
Ingrese un numero: 658
El promedio de los datos ingresados es 508.75
Presione una tecla para continuar . . .
```

Ciclos FOR

- Otro caso es **la selección** dentro de un ciclo for.

- Esta implica hacer un ciclo en el cual através de un conjunto de números es posible seleccionar aquellos que satisfagan uno o mas criterios.

- A continuación se vera un ejemplo donde se desea encontrar **la suma positiva y negativa** de un conjunto de números. Los criterios aquí son si el numero es positivo o negativo y el pseudocódigo seria.

```
mientras la condición se cumpla
introducir un numero
      si es mayor que cero
                  sumar a la suma positiva
       de lo contrario
                  sumar a la suma negativa
fin
```

Ciclos FOR

```
int main()
{
  const int MAXNUMS = 5;
  int i;
  double usenum, postot, negto

  postot = 0; // esta inicializacion se puede hacer en la declaracion

  negtot = 0; // esta inicializacion se puede hacer en la declaracion

  for (i = 1; i <= MAXNUMS; i++)
  {
    cout << "Ingrese un numero (positivo or negativo) : ";
    cin  >> usenum;
    if (usenum > 0)
      postot = postot + usenum;
    else
      negtot = negtot + usenum;
   }
   cout << "EL total positivo es " << postot << endl;
   cout << "El total negativo es " << negtot << endl;
```

```
Ingrese un numero (positivo or negativo) : 54
Ingrese un numero (positivo or negativo) : -68
Ingrese un numero (positivo or negativo) : 40
Ingrese un numero (positivo or negativo) : -33
Ingrese un numero (positivo or negativo) : -21.5
EL total positivo es 94
El total negativo es -122.5
Presione una tecla para continuar . . .
```

Ciclos FOR

Evaluación de funciones de una variable:

No limitados a enteros en variable contador.

Saltos fraccionarios.

```
//Programa 5-15

#include<iost
#include<ioma
#include<cmat

using namespa

int main()
{
    double x,

    cout<<"valor x          valor y\n"
        <<"-------          ----------\n";

    cout<<fixed<<setprecision(5);

    for(x = 2.0; x <= 6.0; x = x + 0.5)
    {
          y=10.0 * pow(x,2.0) + 3.0 * x - 2.0;
          cout<<setw(7)<< x
              <<setw(20)<<y<<endl;
              }

system("pause");
return 0;

}
```

```
valor x          valor y
-------          ----------
2.00000            44.00000
2.50000            68.00000
3.00000            97.00000
3.50000           131.00000
4.00000           170.00000
4.50000           214.00000
5.00000           263.00000
5.50000           317.00000
6.00000           376.00000
Presione una tecla para continuar . . .
```

Ciclos FOR

Control interactivo de un Ciclo:

Se pueden utilizar variables para controlar un ciclo, y su valor puede ser ingresado interactivamente.

```
//Programa 5-16
#include<iostream>
#include<iomanip>
//Este programa des
//sus respectivos
//de la tabla es i
using namespace st

int main()
{
     int num, final;

     cout<<"Introduzca el numero final para la tabla: ";
     cin>>final;

     cout<<"NUMERO CUADRADO CUBO\n";
     cout<<"------ -------- ----\n";

     for(num = 1; num <= final; num++)
     {
          cout<<setw(3)<<num;
          cout<<setw(8)<<num*num;
          cout<<setw(7)<<num*num*num<<endl;
          }

system("pause");
return 0;

}
```

```
Introduzca el numero final para la tabla: 6
NUMERO CUADRADO CUBO
------ -------- ----
  1        1       1
  2        4       8
  3        9      27
  4       16      64
  5       25     125
  6       36     216
Presione una tecla para continuar . . .
```

DO WHILE

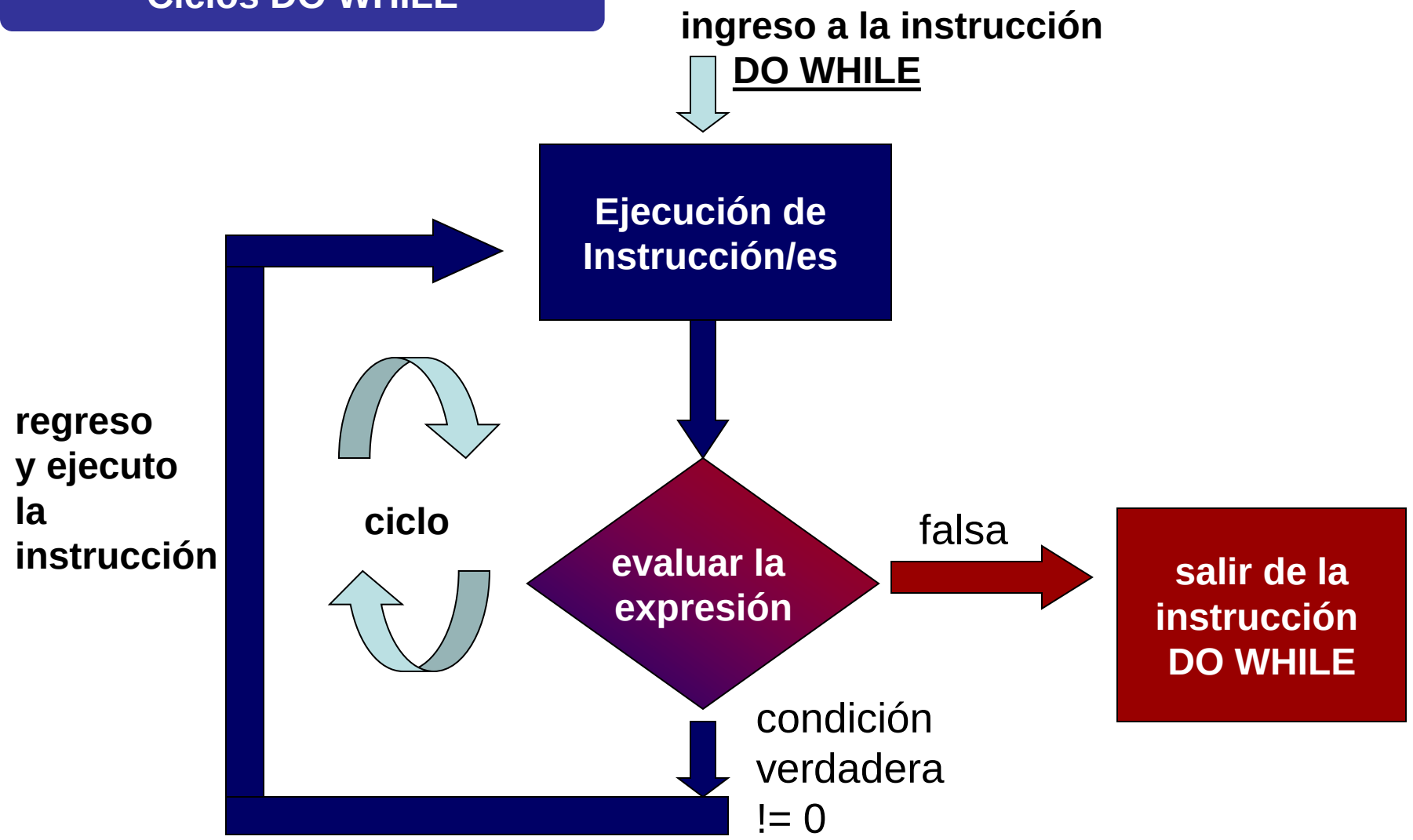
- Traducción: HACER MIENTRAS.
- Como puede deducirse, la gran diferencia con la instrucción WHILE, radica en que esta instrucción produce un ciclo de repetición **CONTROLADO A LA SALIDA** o DE PRUEBA POSTERIOR, y al menos se **EJECUTA una vez** el código dentro del cuerpo de la instrucción.

- La sintaxis de esta instrucción es:

```
do
   instrucción;
while (expresion);
```

- **La instrucción simple puede ser reemplazada por una compuesta, NO OLVIDAR LAS {}**
- **Puede ser reemplazada por un ciclo WHILE o FOR equivalente, generalmente es mas usado el ciclo WHILE por claridad.**

Ciclos DO WHILE



Ciclos DO WHILE

- En el ultimo programa visto puede cambiarse el ciclo WHILE a:

```
while (calificacion <= CALIFICACIONMAYOR)
{
  total = total + calificacion;
  cout << "Ingese una calificacion: ";
  cin  >> calificacion;
}

cout << "\nEl total de las calificaciones es
```

```
Para detener el ingreso de calificaciones,
 mayor a 100.

Ingese una calificacion: 100
Ingese una calificacion: 99
Ingese una calificacion: 98
Ingese una calificacion: 100
Ingese una calificacion: 50
Ingese una calificacion: 101

El total de las calificaciones es 447
Presione una tecla para continuar . . .
```

```
//while (calificacion <= CALIFICACIONMAYOR)
do
{
  total = total + calificacion;
  cout << "Ingese una calificacion: ";
  cin  >> calificacion;
}
while(calificacion <= CALIFICACIONMAYOR);

cout << "\nEl total de las calificaciones es
```

```
Para detener el ingreso de calificaciones,
 mayor a 100.

Ingese una calificacion: 100
Ingese una calificacion: 99
Ingese una calificacion: 98
Ingese una calificacion: 100
Ingese una calificacion: 50
Ingese una calificacion: 101

El total de las calificaciones es 447
Presione una tecla para continuar . . .
```

REPASO CLASE IF ELSE SWITCH

- La instrucción SWITCH permite comparar, el valor de una expresión con los distintos CASOS. Si no coincide ninguno se ejecuta el DEFAULT si esta presente.

```
switch(edad)
{
    case 10:
    case 15:
    case 16: cout<<“Usted es demasiado JOVEN para estudiar
                    << informatica”;
     case 17:
     case 18: cout<<“Es el momento de empezar”;

      default: cout<<“ Usted ya deberia SABER C++”;
}
```

REPASO CLASE IF ELSE SWITCH

- Resolución Ejercicio 2c con SWITCH

c. En un año que no es bisiesto, febrero tiene 28 días, los meses de enero, marzo, mayo, julio, agosto, octubre y diciembre tienen 31 días y todos los demás meses tienen 30 días. Usando esta información, modifique el programa escrito en el ejercicio 2a para desplegar un mensaje cuando se introduzca un día inválido para un mes introducido por un usuario. Para este programa ignore los años bisiestos.

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

int main()
{
    int mes=0,dia=0;

    cout<<"Ingrese un mes (utilize 1 para enero, 2 para febrero y asi sucesivamente)"<<endl;
    cin>>mes;
    cout<<"Ahora ingrese un dia del mes!"<<endl;
    cin>>dia;
```

```
switch(mes)
{
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: if(dia<1 || dia>31)
                     cout<<"El dia ingresado es INVALIDO\n";
                     else
                     cout<<"La fecha ingresada es valida y es: "<< dia<<"
                     break;
            case 4:
            case 6:
            case 9:
            case 11: if(dia<1 || dia>30)
                     cout<<"El dia ingresado es INVALIDO\n";
                     else
                     cout<<"La fecha ingresada es valida y es: "<< dia<<"
                     break;
            case 2:  if(dia<1 || dia>28)
                     cout<<"El dia ingresado es INVALIDO\n";
                     else
                     cout<<"La fecha ingresada es valida y es: "<< dia<<"
                     break;
            default:
                    cout<<"Usted esta seguro de lo que hace?\n";
                    }
```

REPASO

EJERCICIO 4 - página 229 Bronson.

*Todos los años que se dividen **exactamente** entre 400 "o" que son divisibles **exactamente** entre 4 "y" no son divisibles **exactamente** entre 100 son años bisiestos. Por ejemplo en vista que 1600 es divisible exactamente entre 400, el año 1600 fue un año bisiesto. Del mismo modo, en vista que 1988 es divisible exactamente por 4 y no es divisible exactamente por 100, el año 1988 también fue un año bisiesto.*

Usando esta información escriba un programa en C++, que acepte el año como una entrada del usuario, determine si el año es un año bisiesto y despliegue un mensaje apropiado que le indique al usuario si el año introducido es o no bisiesto.

Ayuda: revise la funciones aritméticas de la biblioteca cmath vistas en clase

Repaso Clase IF ELSE SWITCH

Primero debo OBSERVAR que EXISTEN DOS CONDICIONES QUE ME DA EL ENUNCIADO que determinan si un año es bisiesto. Esto es debido a que el tiempo necesario para que la tierra gire una vuelta alrededor del sol es 365.25635 días. Por lo tanto no solo cada cuatro años puede darse una año bisiesto debe considerarse la fracción completa.

Que se cumpla UNA U OTRA CONDICION es SUFICIENTE para decir que el año ingresado es bisiesto.

```
int main()
{
    int age;//declaro variable a usar

    cout<<"Ingrese un a\244o (ej: 1924) "<<"\n\n"; //indicador de comandos
    cin>>age;//ingreso por el usuario del año

    if((age%400==0) || (age%4==0 && age%100!=0)) //estructura de seleccion
    cout<<"\nEl a\244o "<<age<<" ingresado es bisiesto"<<endl; //impresion de resultados
    else
    cout<<"\nEl año ingresado no es bisiesto!!"<<"\n\n";
```