PROGRAMACIÓN AVANZADA

ING. INDUSTRIAL

FAC. DE INGENIERÍA

UNIVERSIDAD NACIONAL DE JUJUY

2da parte

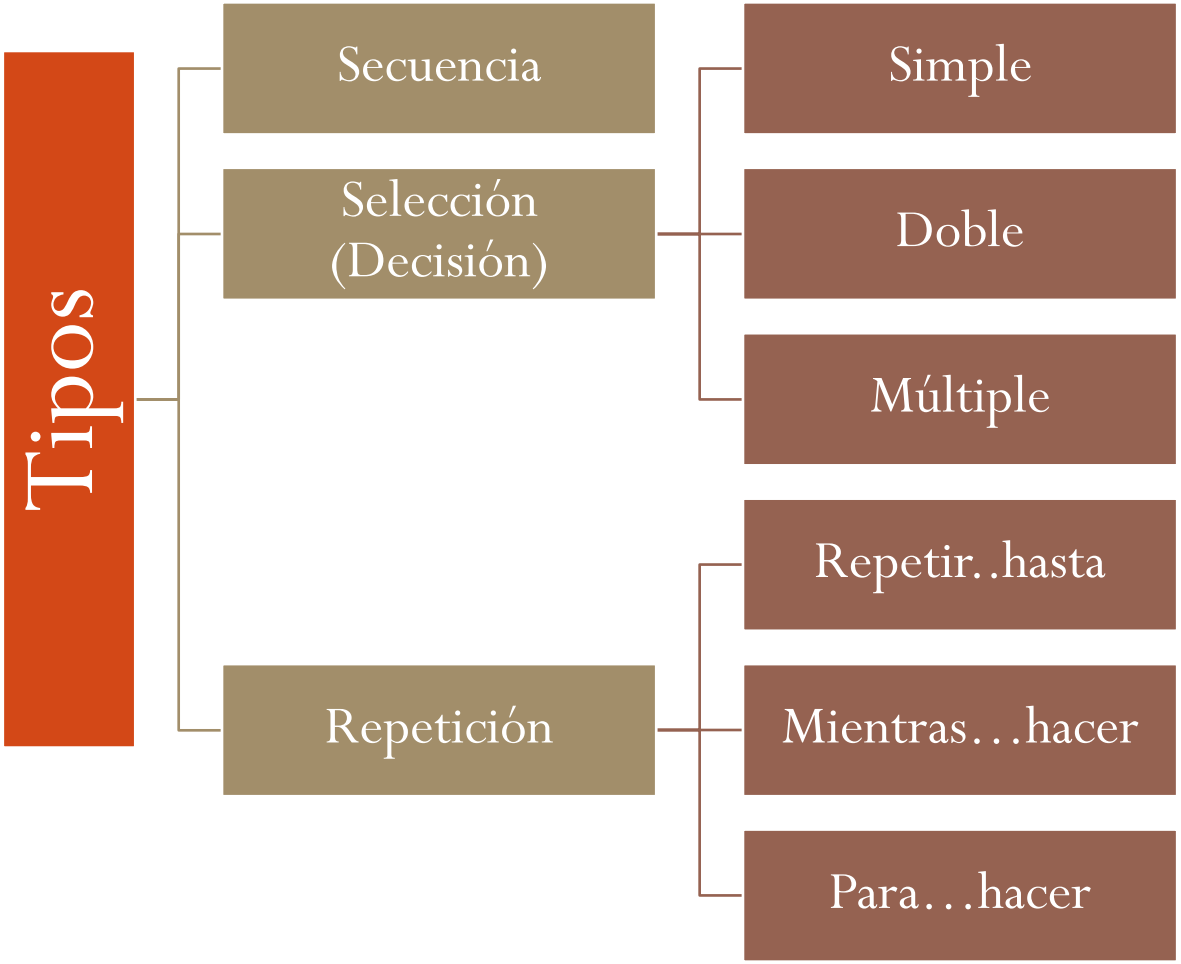
Introducción C++

- 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

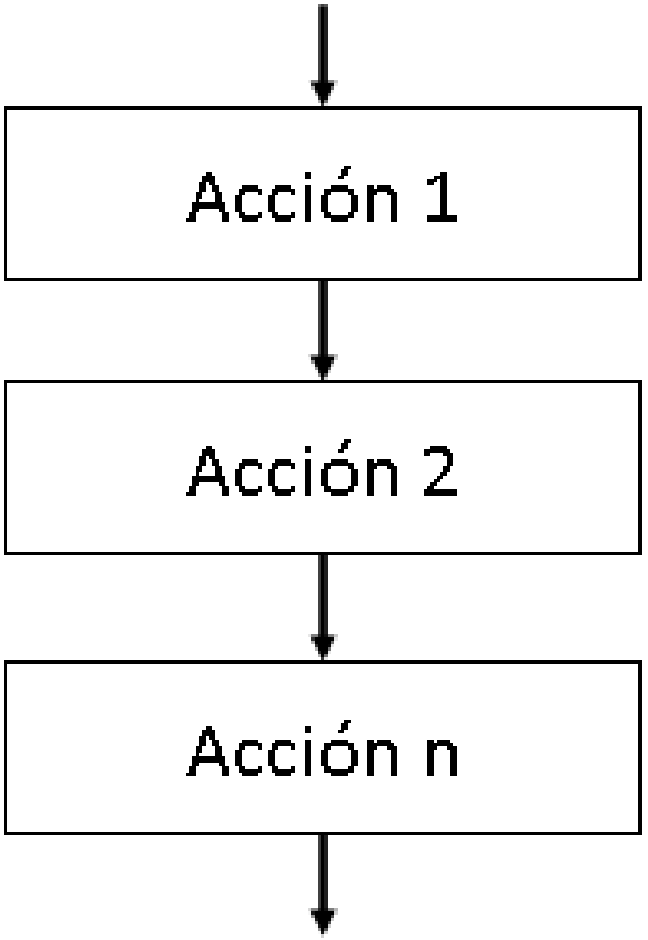
Unidad 1

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

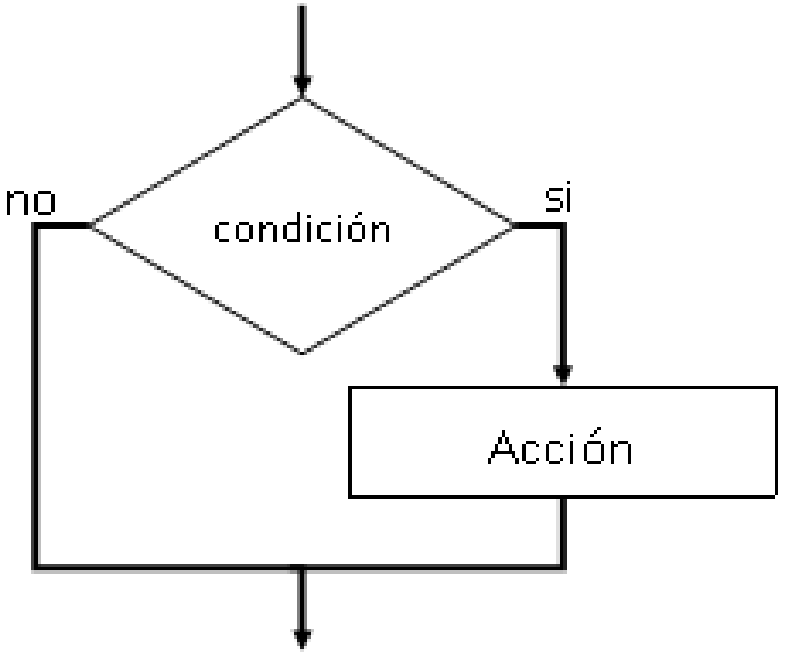
Problema:

- Cálculo de la suma y producto de dos números.
- 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){
  Acciones;
  }

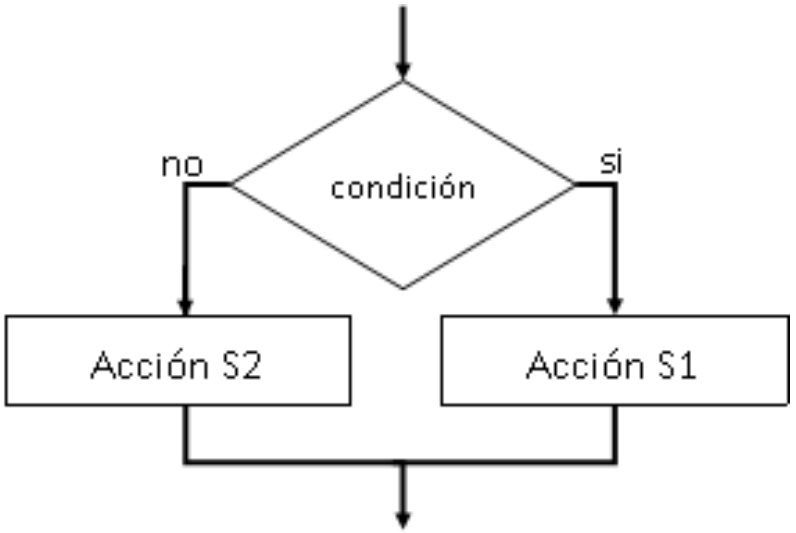
Ejemplo

- 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*) *Acción1;* **else** *Accion2;*

- **if** (expresion) {
  Acciones 1;
  } **else** {
  Acciones 2;
  }

Ejemplo

- Dado un número mostrar si el mismo es par o impar.

```
#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;
}
```

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

- Una de las principales aplicaciones es la elaboración de un "menú de opciones" dentro de un programa.

Sintaxis

```
switch (expresion)  {
    case  c1:          // código a ejecutar en caso de que expresion == c1
                       break;
    case c2:           // código a ejecutar en caso de que expresion == c2
                       break;
    …

    default:           // código (opcional) a ejecutar en caso de que no se cumpla
                       // ninguna de las condiciones anteriores
}
```

* Es importante recordar que c1, c2, etc. deben ser constantes enteras!

Ejemplo: menú de opciones

Usando switch

```
int a, b, r, opcion;
cout << "1.- Suma" << endl;
cout << "2.- Resta" << endl;
cout << "3.- Producto" << endl;
cout << "4.- División" << endl;
cout << "Elige una opción: " << endl;
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; break;
    default: cout << "Opcion invalida" << endl;
}
cout << "El resultado es " << r << endl;
```

Usando if

```
int a, b, r, opcion;
cout << "1.- Suma" << endl;
cout << "2.- Resta" << endl;
cout << "3.- Producto" << endl;
cout << "4.- División" << endl;
cout << "Elige una opción: " << endl;
cin << opcion;
if (opcion == 1) {
        r = a + b;
} else {
        if (opcion == 2) {
                r = a – b;
        } else {
            if (opcion == 3) {
                r = a * b;
            } else {
                if (opcion == 4) {
                    r = a / b;
                } else {
                    cout << "Opcion invalida" << endl;
                }
            }
        }
}
cout << "El resultado es " << r << endl;
```

Práctica #2

- En la música occidental, las notas Do, Re, Mi, Fa, Sol, La, y Si suelen representarse mediante las letras C, D, E, F, G, A, y B, respectivamente.

  Elabore un programa que tome como entrada la letra (mayúscula o minúscula) correspondiente a una nota e imprima su nombre. Si la letra dada no corresponde a una nota, el programa debe indicarlo mediante un mensaje de error.

  Sugerencia: *utilice el tipo char para almacenar caracteres y recuerde que las constantes de este tipo se escriben entre apóstrofes (e.g., 'a', 'z', '*', etc.).*

ESTRUCTURAS DE ITERACION

Estructuras de iteración

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

- Todas las estructuras de iteración se basan en la ejecución de un bloque de código de manera repetida mientras el valor de una expresión booleana de control sea verdadero.

- Por lo tanto, es muy importante asegurarse de que la condición se vuelve falsa en algún momento, de manera que el programa no quede atascado en un ciclo infinito.

Instrucción while

- Sintaxis:

```
while (expresion) {
  // Código a ejecutar de manera repetida
  // mientras la expresión sea verdadera
}
```

Ejemplo

- El siguiente codigo calcula el factorial de *n*, donde *n* es dado por el usuario:

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

Instrucción do...while

- La instrucción while siempre verifica primero que la condición de control sea verdadera antes de ejecutar el bloque de código.

- Si se desea ejecutar el bloque de código al menos una vez antes de verificar la condición, se puede usar la instrucción do…while:

```
do {
  // bloque de código a ejecutar al menos una
  vez
} while (expresion);
```

La instrucción for

- La otra estructura en C/C++ para la realización de ciclos es proporcionada por la instrucción for, cuya sintaxis es:

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

Donde *inicializacion* es una expresión que se ejecuta antes de iniciar el ciclo. El ciclo se ejecuta mientras la *condicion* sea verdadera, y en cada iteración (después de ejecutar el bloque de instrucciones) se evalúa la expresión *incremento*.

Equivalencia entre for y while

- Las siguientes estructuras son equivalentes:

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

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

Ejemplo

- La instrucción for se usa típicamente para realizar ciclos controlados por contador, ya que permite ver todos los parámetros del ciclo en una sola línea:

```
int main() {
  int i, n, f = 1;
  cout << "Dame un numero entero: ";
  cin >> n;
  for (i = 1; i <= n; i++) {
      f = f * i;
  }
  cout << n << "! = " << f << endl;
}
```

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.

- Una manera de entender el funcionamiento de los ciclos anidados es comparándolos con el odómetro de un coche:

Ejemplo

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

using namespace std;

int main() {
    int i, j, k;
    int t0;
    for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
            for (k = 0; k < 10; k++) {
                cout << "\r" << i << j << k;
                cout.flush();
                t0 = time(0);
                while (t0 == time(0)) {}
            }
        }
    }
    return 0;
}
```

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 promedio de los valores dados por el usuario e imprimirlos al final.

Práctica #4

- Realice un programa que pida al usuario una serie de números enteros no negativos utilizando un ciclo do…while. Una vez que el usuario introduzca un número negativo, el ciclo deberá terminar y el programa deberá imprimir el mayor, menor, y promedio de todos los números ingresados (excepto el último).

Práctica #5

a) Escriba un programa que, dado un número entero ***n***, determine si ***n*** es primo o no. Sugerencia: verifique si ***n*** es divisible entre algún entero entre 2 y ***n/2***.

b) Modifique el programa anterior para que pida al usuario un entero positivo ***N***, y luego imprima los primeros ***N*** números primos.

Ejemplo: si el usuario ingresa 10, el programa debe imprimir: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.