Ing. María Inés Colombo
Unexpo Núcleo Carora

Estructura Básica de un programa en C++

Estructura basica de programa.cpp

```cpp
// comentario que explica instrucciones del programa
#include <iostream>   //librería de entradas y salidas básicas
using namespace std;

int main() // función principal
{          // inicio de un bloque de instrucciones
   int a;   // declaración de la variable a de tipo entero
   float b; // declaración de la variable b de tipo decimal
   char c;  // declaración de la variable c de tipo caracter
   bool d; // declaración de la variable d de tipo lógico
   float suma; //declaración de la variable suma de tipo decimal

   cout<<"Ingrese el valor de a"<<endl;  //imprime el texto entre "" y salta de línea
   cin>>a;     //El valor que ingresa el usuario lo guarda en la variable a
   cout<<"Ingrese el valor de b"<<endl;  //imprime el texto entre "" y salta de línea
   cin>>b;     //El valor que ingresa el usuario lo guarda en la variable b

   suma = a + b;   //Guarda en la variable suma el resultado de a+b

   cout<<"El resultado de la suma es "<<suma<<endl; //muestra resultado por pantalla

   system("pause");  //Ataja la pantalla hasta que se presione una tecla
   return 0;  //la función principal (main) indica que devolverá un resultado de tipo "int"
}    //fin del bloque de instrucciones de la función principal y del programa
```

Librerías en C++

Son archivos (no siempre externos), que nos permiten llevar a cabo diferentes tareas sin necesidad de saber cómo se hacen sino simplemente entender cómo usarlas. Además, permiten hacer los programas más modulares y reutilizables, facilitando además crear programas con funcionalidades bastante complejas en unas pocas líneas de código.

Librerías en C++

La sintaxis es la siguiente:

#include <nombre de la librería>

o

#include "nombre de la librería"

Cualquiera de las 2 formas es válida en C++, no hay límite para declarar las librerías, sin embargo, no tiene sentido colocarlas si no se van a usar.

La declaración de librerías se debe hacer al principio de todo código, antes de la declaración de cualquier función o línea de código, se debe indicar al compilador qué librerías usar.

Librerías en C++

Librerías Estandar de C++ (Standar Template Library o *STL*):

- **fstream:** Flujos hacia/desde ficheros. Permite la manipulación de archivos desde el programar, tanto leer como escribir en ellos.
- **iosfwd:** Contiene declaraciones adelantadas de todas las plantillas de flujos y sus typedefs estándar. Por ejemplo ostream.
- **iostream:** Parte del a *STL* que contiene los algoritmos estándar, es quizá la más usada e importante (aunque no indispensable).

Librerías en C++

Librerías Estandar de C++ (Standar Template Library o *STL*):

- **math:** Contiene los prototipos de las funciones y otras definiciones para el uso y manipulación de funciones matemáticas.
- **Librería stdio:** Contiene los prototipos de las funciones, macros, y tipos para manipular datos de entrada y salida.
- **Librería stdlib:** Contiene los prototipos de las funciones, macros, y tipos para utilidades de uso general.
- **string:** Parte de la *STL* relativa a contenedores tipo string, para el fácil uso de las cadenas de caracteres.

Librerías en C++

¿Cómo declarar una librería en C++?

```
#include "iostream"
#include "string"
#include <math.h>
using namespace std;
```

*El **using namespace std*** nos ayuda a declarar un espacio de nombre que evita tener que usarlo cada que accedemos a alguna función específica de una librería. Por ejemplo el comando **cout>>**, que pertenece a la librería iostream, sin este namespace sería **std::cout >>**

Variables

Las variables son posiciones en memoria donde estarán guardados los diferentes valores que le damos o que toman durante la ejecución los datos que usamos y normalmente estarán disponibles a lo largo de la ejecución de nuestro programa. La sintaxis para declara una variable es la siguiente: **tipo de dato, el nombre y el punto y coma** al final de la línea. Ejemplo:

```
float promedio;
```

Variables

Normas a seguir al momento de asignar un nombre a una variable:

1. El nombre de una variable nunca debe comenzar con un número.
2. No debes incluir algunos caracteres como símbolos matemáticos, guiones (el guión bajo"_" si es permitido), comas, punto y coma, comillas, signos de interrogación o espacios en blanco.
3. Deben tener un máximo de 40 caracteres
4. No debe ser una palabra reservada.

Palabras reservadas C++

auto	const	double	float
int	short	struct	unsigned
break	continue	else	for
long	signed	switch	void
case	default	enum	goto
register	sizeof	typedef	volatile
char	do	extern	if
return	static	union	while

Tipos de datos básicos

- **int:** El tipo de dato int, tiene un tamaño de 32 bits y un rango entre -2.147.483.648 y 2.147.483.647. Este tipo de dato, es usado para números enteros
- **float:** El tipo de dato float tiene un tamaño de 32 bits, es usado comúnmente en números con 6 o menos cifras decimales. Tiene un rango entre 1,17549*(e^-38) hasta 3,40282*(e^+38).
- **double:** El tipo de dato double tiene un tamaño de 64 bits, es usado para números de menos de 15 cifras decimales. Tiene un rango entre 2,22507*(e^-308) hasta 1,79769*(e^308).
- **char**: Tipo de dato de un carácter
- **bool:** El tipo de dato bool, tiene un tamaño de 8 bits y un rango entre 0 y 1, es falso o verdadero.

Instrucciones de Asignación

Es una línea de código, que le asigna a una variable cualquiera un valor cualquiera, preferiblemente adecuado al tipo de dato o a las necesidades de dicha asignación. Una asignación tiene la siguiente sintaxis: ***nombre_variable = valor***, con esto le estamos diciendo a nuestro programa que la variable llamada "**nombre_variable**", tendrá ahora como nuevo valor a "**valor**". Ejemplo:

```
float pi = 3,1416
```

Constantes en C++

Las datos constantes tienen un valor fijo durante toda la ejecución del programa, es decir, este valor no cambia ni puede ser cambiado a lo largo de la ejecución de nuestro programa.

Para declarar una constante, se hace después de declarar las librerías y antes de las funciones, la sintaxis es la siguiente:

#define nombre_constante valor

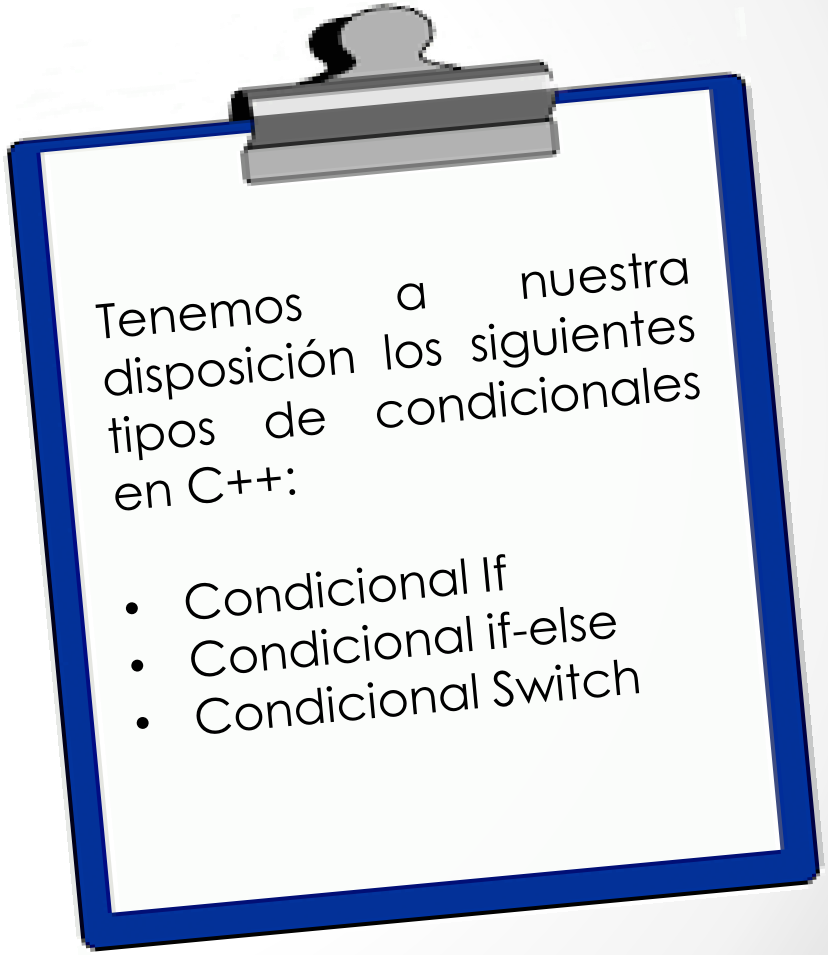
y la segunda es usando la palabra clave ***const***, Ejemplo:

const float PI = 3.1416; //Declaramos una constante llamada PI

#define PI 3.1416 //Definimos una constante llamada PI

Condicionales en C++

Son una estructura de control esencial al momento de programar, dado que permite establecer una serie de condiciones al interior de nuestro programa, que nos ayudan a determinar que acciones llevará cabo dadas ciertas circunstancias.



Condicional if else C++

Sintaxis de un condicional if-else:

```
if(condición a evaluar) //Ejemplo num <= 10
{
      ....
   //Instrucciones SI se cumple la condición....
       ....
 }
 else
 {
      ....
    //Instrucciones si NO se cumple la
 condición....
      ....
 }
```

Es una estructura que nos posibilita definir las acciones que se deben llevar a cabo si se cumple cierta condición y también determinar las acciones que se deben ejecutar en caso de que no se cumpla; generando así una separación o bifurcación en la ejecución del programa.

Condicional if else C++

Ejemplo: Escriba un programa que le indique al estudiante si aprobó o no la evaluación que tuvo sobre 20 puntos:

SumaPares.cpp | Psuma.cpp | [*] ifelse.cpp

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

int main()
{
    int valor=0;
    cout<<"Ingrese su calificacion en la ultima evaluacion sobre 20"<<endl;
    cin>>valor;
    if (valor >= 10)
      {
       cout<<"Usted aprobo el examen"<<endl;
      }
    else
       {
         cout<<"Usted salio aplazado en el examen"<<endl;
       };

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

Condicional switch C++

Son una estructura de control condicional, que permite definir múltiples casos que puede llegar a cumplir una variable cualquiera, y qué acción tomar en cualquiera de estas situaciones, incluso es posible determinar qué acción llevar a cabo en caso de no cumplir ninguna de las condiciones dadas.

Sintaxis:

```
switch(opción) //donde opción es la
               //variable a comparar
{
   case valor1:
        //Bloque de instrucciones 1;
        break;
   case valor2:
        //Bloque de instrucciones 1;
        break;
   default:
        //Bloque de instrucciones
        //por defecto;
}
```

Condicional switch C++

Detalles sobre el condicional switch en C++

- En C++, sólo se puede usar números en cada case y caracteres representados como números enteros.
- La sentencia default es opcional. Sin embargo, es recomendable usarla, para así controlar todas las opciones posibles y que el programa sepa como responder ante cualquier acción del usuario.
- Dentro de cada case se pueden utilizar varias líneas de código. Sin embargo, es preferible mantener el código ordenado. Preferiblemente una sola instrucción.
- Para hacer tareas complejas al interior de un switch en C++, posiblemente sea mejor hacer esas tareas en una función y llamar a esa función desde el switch o simplemente usar un if-else.

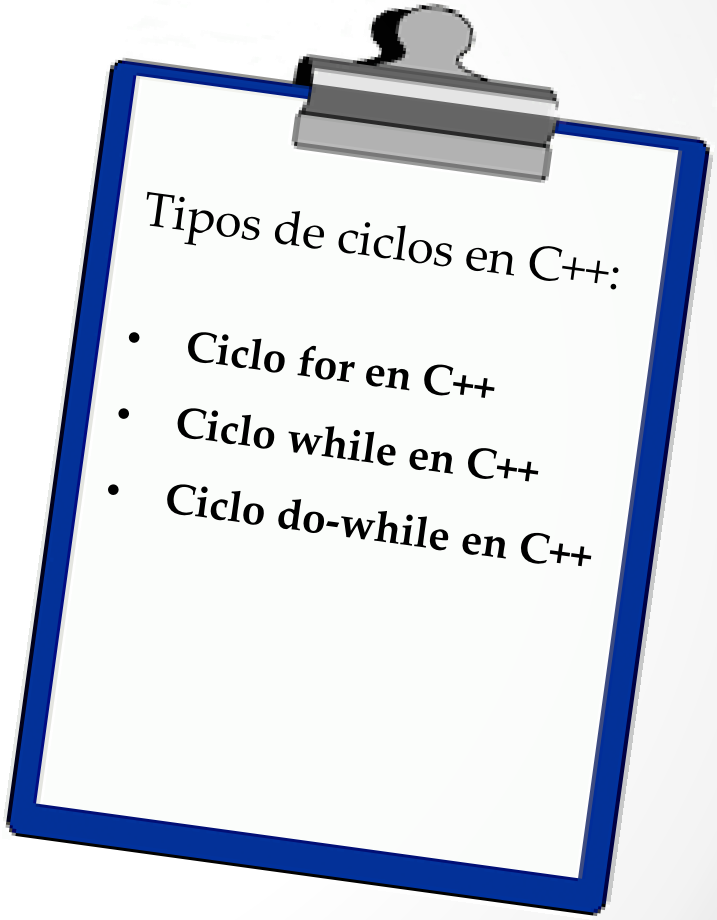
Condicional switch C++

Ejemplo: Realice un programa que permita al usuario seleccionar una opción de las 3 opciones de un menú.

```
1  #include "iostream"
2  using namespace std;
3  int main()
4  {
5      char opcion;
6      cout << "Ingrese la Opción a ejecutar: ";
7      cin >> opcion;
8      switch(opcion)
9      {
10         case 'a': cout << "Usted ha seleccionado la opcion a"<<endl;
11         break;
12         case 'b': cout << "Usted ha seleccionado la opcion b"<<endl;
13         break;
14         case 'c': cout << "Usted ha seleccionado la opcion c"<<endl;
15         break;
16         default: cout << "Usted ha ingresado una opcion incorrecta"<<endl;
17     }
18     system("pause");
19     return 0;
20 }
```

Bucles o Ciclos en C++

Un ciclo o bucle permite repetir una o varias instrucciones cuantas veces se necesite, por ejemplo, si se requiere escribir los números del uno al cien no tendría sentido utilizar cien variables, para esto y para muchas otras cosas es útil un ciclo, permitiendo hacer una misma tarea en una cantidad de líneas muy pequeña y de forma prácticamente automática.

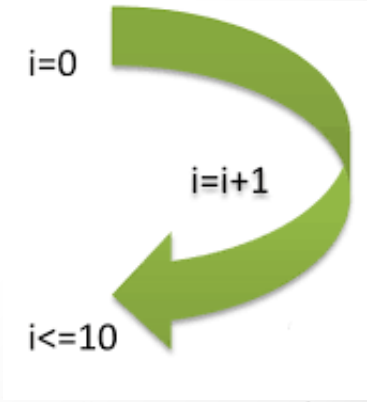


Ciclo for en C++

Sintaxis de un ciclo for en C++:
for (inicio, final y tamaño de paso)
tenemos prácticamente todo hecho:

```
for(int i = valor inicial; i <= valor
final; i = i + paso)
{
    ....
    Bloque de Instrucciones
        ....
}
```

Es una estructura de control iterativa, que permite ejecutar de manera repetitiva un bloque de instrucciones, conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo.



Ciclo for en C++

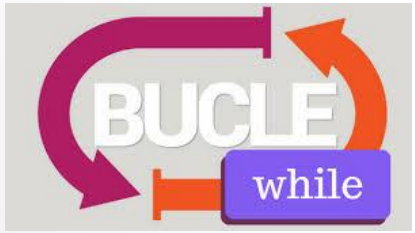
Ejemplo: Elabore un programa que sume los números pares que existen entre 10 y 60

SumaPares.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int sumapar=0;
7
8      for(int i=10; i<=60; i+=2)
9      {
10       sumapar = sumapar + i;
11     }
12     cout<<"La suma de los valores pares es "<<sumapar<<endl;
13
14     system("pause");
15     return 0;
16 }
```

Ciclo while en C++

Son una estructura cíclica, que permite ejecutar una o varias líneas de código de manera repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuando se va a dar el valor final que permite concluir el ciclo while.



Sintaxis:

```
while(condición de finalización)
//ejemplo mientras numero < 100
{
      ....
   //Bloque de Instrucciones....
      ....
}
```

Ciclo while en C++

Ejemplo: Escriba que un programa que solicite un número al usuario, el ciclo se repetirá mientras el número sea diferente de 50

```
1   #include "iostream"
2   using namespace std;
3   int main()
4   {
5       int numero;
6       cout <<  "Ingrese un numero ";
7       cin >> numero;
8       while(numero != 50)
9       {
10          cout <<  "Ingrese un numero ";
11          cin >> numero;
12      }
13      system("PAUSE");
14      return 0;
15  }
```

Ciclo do-while en C++

Es similar al ciclo while, sin embargo, el ciclo do-while permite añadir cierta ventaja adicional y esta consiste que da la posibilidad de ejecutar primero el bloque de instrucciones antes de evaluar la condición necesaria, de este modo los ciclos do-while, son más efectivos para algunas situaciones.

Sintaxis:

```
do
{
    ....
   //Bloque de Instrucciones....
    ....
}
while(condición de finalización);
//por ejemplo numero != 23
```

Ciclo do-while en C++

Ejemplo: Escriba un programa que lea números menores o iguales a 50

```
1   #include "iostream"
2
3   using namespace std;
4   int main()
5   {
6       int numero;
7       do
8       {
9           cout <<  "Ingrese un numero ";
10          cin >> numero;
11      }
12      while(numero <= 50);
13      system("pause");
14      return 0;
15  }
```

Gracias!