Unidad 4

Flujo de Programa

Estructura Selectiva Simple

Esta estructura se evalúa una condición que al cumplirse efectúa cierta acción, de lo contrario, continúa con la ejecución normal del programa o bien termina su ejecución. Su sintaxis es la siguiente:

```
if (condición) {
//acción a realizar si se cumple la condición
}
```

Ejemplo 1

En una tienda se venden artículos de primera necesidad, a los cuales se les aplica un descuento del 20%, de la compra total, si esta es mayor o igual a $50. Diseñe un programa en C, que a partir del importe total de la compra muestre lo que debe pagar el cliente.

```
#include <iostream>
using namespace std;
int main()
{
 float compra; //declaración de variables
 cout <<"Introduzca el valor de la compra:\n" << endl;
 cin >>compra;
 if (compra>=50) {
compra = compra*0.8;
cout <<"\n\nEl Importe de la compra es de $" << compra << endl;
}

  cout << "Gracias, por su preferencia.\n" << endl;
 }
```

Estructura Selectiva Doble

Esta estructura se evalúa una condición que al cumplirse efectúa cierta acción, de lo contrario, continúa con la ejecución normal del programa o bien termina su ejecución. Su sintaxis es la siguiente:

Esta estructura, se caracteriza por el hecho que ofrece dos caminos a seguir, dependiendo si al evaluar la condición resulta cierta o falsa. Su sintaxis es la siguiente:

If else
If (Condición) {
//Acción 1: Se realiza si la condición se cumple
}
else {
//Acción 2: Se realiza si la condición NO se cumple.
}

Ejemplo 2

Realiza un programa donde obtengas la edad de una persona, para ello debes recordar que debe ser mayor o igual 18, entonces imprimirán Eres mayor de edad, de lo contrario Menor de edad.

```
#include <iostream>
using namespace std;
int main()
{
int edad;
cout <<"Ingresa su edad:\n" << endl;
cin >>edad;
if(edad>=18)
                {
                cout <<"\n\n Eres Mayor de edad." << endl;
                }
else {

                cout <<"\n\n Eres Menor de edad." << endl;
                }
}
```

Estructura Anidada

If Anidado

Esta selectiva nos ayuda comparar diferentes alternativas que se le pueden presentar al usuario.

Ejemplo 3

Realiza un programa donde obtengas un número del 1 al 3, e imprime la opción elegida por el usuario.

```
#include <iostream>
#include <locale.h>
using namespace std;

int main()
{
		setlocale(LC_ALL, "Spanish");
int num;
cout <<"\nIngresa un número del 1 al 3:" << endl;
cin >>num;
if(num==1) //comparar el número igual que 1
		{
		cout <<"\n\nElegiste la primera opción." << endl;
		}
else if(num==2) //comparar el número igual que 2
{
		cout <<"\n\nElegiste la segunda opción." << endl;
		}
		else if(num==3) //comparar el número igual que 3
{
				cout <<"\n\nElegiste la tercera opción." << endl;
		}
		else {
				cout <<"\n\nError, número no valido." << endl;
		}

}
```

Switch (case)

Como su nombre lo indica, permite seleccionar entre varios caminos para llegar al final. En este caso se pueden elegir un camino o acción a ejecutar de entre varios posibles que se debe de evaluar, llamada *selector.* Sintaxis:

switch(selector)
{
case *Etiqueta A:*
Acción A;
break;
case *Etiqueta B:*
Acción B;
break;
case *Etiqueta n:*
Acción n;
break;
default:
Excepción;
break;
}

Ejemplo 4

Realiza un programa donde obtengas un número del 1 al 3, e imprime la opción elegida por el usuario.

```
#include <iostream>
#include <locale.h>
using namespace std;

int main()
{
		setlocale(LC_ALL, "Spanish");
int num;
cout <<"\nIngresa un número del 1 al 3:" << endl;
cin >>num;
switch (num){
		case 1:
				cout <<"\n\nElegiste la primera opción."
<< endl;
		break;
		case 2:
				cout <<"\n\nElegiste la segunda opción."
<< endl;
		break;
		case 3:
				cout <<"\n\nElegiste la tercera opción."
<< endl;
		break;
		default:
				cout <<"\n\nError, número no valido." <<
endl;
		break;
}

}
```

Estructuras Repetitivas

¿Qué es una estructura repetitiva?

Las estructuras repetitivas se utilizan cuando se quiere que un conjunto de instrucciones se ejecuten un cierto número finito de veces, por ejemplo, escribir algo en pantalla cierta cantidad de veces, mover un objeto de un punto a otra cierta cantidad de pasos, o hacer una operación matemática cierta cantidad de veces.

¿Qué es un ciclo?

Período temporal en el que se observan una serie de acontecimientos, acciones o instrucciones, una vez que finaliza, vuelve realizar el mismo proceso.

¿Cuál es el funcionamiento de un ciclo?

Un ciclo, funciona de la siguiente manera: Evalúa una condición de resultar cierta, realiza una acción o bloque de acciones, luego vuelve a evaluar la condición y si nuevamente resulta cierta, realiza la (s) acción (es). Cuando la condición de cómo resultado falso se sale del ciclo y continúa con la ejecución normal del programa.

Acumulador

Es una variable, que, como su nombre lo indica se encarga de acumular valores. Esto se vuelve muy útil, por ejemplo, cuando queremos encontrar la suma de los números del 0 al 9, en el acumulador, vamos guardando los valores de dichas cifras. Puede ser tanto real como entera. Su valor inicial, en la mayoría de los casos es cero.

Contador

Es una variable de tipo entero, que nos ayuda, en el programa a contabilizar el número de ejecuciones de una misma acción, de un grupo de alumnos etc.

While

La sentencia while (condición o expresión) instrucciones; es seguramente la más utilizada. La sentencia, o grupo de sentencias o instrucciones se ejecutan mientras la evaluación de la expresión sea verdadera.

Funciona de la siguiente manera: primero evalúa la condición, si da como resultado cierto realiza la acción, luego vuelve a evaluar la condición, si su resultado es falso, se sale del ciclo y continúa con la ejecución del programa. Su sintaxis es la siguiente:

```
while(condición){
//acciones
}
```

Ejemplo

Diseñe un programa que imprima los primeros 10 números.

```
#include <iostream>
#include <locale.h>
using namespace std;
int main()
{
int i=1; /*Declaramos nuestro contador con su Valor Inicial*/
while(i<=10) /*Mientras i sea menor o igual a 10:*/
{
cout <<"\t" << i <<endl;/*Imprimir el valor de i*/
i=i+1; /*Aumentar el contador en 1*/
}
}
```

Do While

Es una estructura de control cíclica que permite ejecutar de manera repetitiva un bloque de instrucciones sin evaluar de forma inmediata una condición específica, sino evaluándose justo después de ejecutar por primera vez el bloque de instrucciones. Su sintaxis es la siguiente:

```
do{
…
//acciones
…
} while (condición);
```

Ejemplo
Diseña un diagrama de flujo, donde realices la sumatoria de dos números e imprime su resultado.

```
#include <iostream>
#include <locale.h>
using namespace std;

int main()
{
setlocale(LC_ALL, "Spanish");
int suma, a, b, regresar;
do{ //inicio de while
cout <<"Ingresa el primer valor a sumar:\n" << endl;
cin >>a;
cout <<"Ingresa el segundo valor a sumar:\n" << endl;
cin >>b;
suma = a + b;
cout <<"El resultado de los dos números es: " << suma <<endl;
cout <<"\n\n\n¿Deseas ingresar otra edad? \n(1)Si\n(0)No\t:"<< endl;
cin >>regresar;

/* Verifica que solo se acepte el número 0 ó 1, si se ingresa uno número
diferente de 0 ó 1, imprimirá el mensaje de Error... */
while(regresar<0||regresar>1)
{
cout <<"\n Error\n¿Deseas ingresar otra edad? \n(1)Si\n(0)No\t: " << endl;
cin >>regresar;
}

} while(regresar==1);//termino de do...while


cout << "Gracias, por su preferencia.\n" << endl;
}
```

For

En algunas ocasiones, sabemos a ciencia cierta el número de veces que se tiene que repetir una misma acción o bloque de acciones. Y para ello es que nos sirve, esta estructura. Su sintaxis es la siguiente:

for(valor inicial; condición; incremento){
//acciones
}

Diseña un programa que imprima los sueldos totales de 15 empleados, e imprime el resultado.

```
#include <iostream>
using namespace std;
int main()
{
int i, sueldo, total=0;
for(i=1; i<=15; i++){
cout <<"\nIngresa el sueldo: " << i << endl;
cin >> sueldo;
total = total + sueldo;
}
cout <<"\nIngresa el sueldo: " << total << endl;
}
```

Unidad 5
Manejo de Arreglos y Cadenas

Un arreglo es un conjunto de datos del mismo tipo ordenados de forma lineal uno después de otro. Los componentes de un arreglo se han de referenciar por medio del nombre del arreglo y un índice de desplazamiento para indicar el componente deseado.

Los arreglos son usados extensamente por los programadores para contener listas de datos en la memoria, por ejemplo el de un menú de opciones que se desplegarán dentro de una ventana para que el usuario pueda elegir una de éstas, en tales casos y cuando las opciones son numerosas, solamente se ponen unas cuantas, de ellas dentro de la ventana, pero se le da al usuario la oportunidad de poder subir y bajar a su antojo para ver el resto de opciones que, aunque no se vean en la ventana, forman parte del menú o arreglo de opciones

Los índices son números que se utilizan para identificar a cada uno de los componentes de un arreglo. Es decir, si creamos un arreglo de 5 elementos los índices quedarían de la siguiente manera, tomando en cuenta que el elemento siempre comienza desde la posición 0:

Arreglo
[5] [3 [7] [1] [2]

Elemento	Índice
5	[0]
3	[1]
7	[2]
1	[3]
2	[4]

```
#include<stdio.h>
#include<conio.h>
#include<windows.h>

main()
{
		int i, j, arre[2][3];
		for(i=0;i<2;i++)
		{
				for(j=0;j<3;j++)
				{
						printf("Escribe el valor de
la posicion [%d][%d]\n",i,j);
						scanf("%d",&arre[i][j]);
				}
		}
		system("cls");
		printf("Su arreglo quedo: \n\n\n");
		for(i=0;i<2;i++)
		{
				for(j=0;j<3;j++)
				{
						printf("[%d]",arre[i][j]);
				}
				printf("\n");
		}
		getch();
}
```

Arreglos Unidimensionales

Un arreglo unidimensional es aquel en donde los componentes son accesibles por medio de uno y solamente un índice que apunte al componente requerido. Los arreglos de este tipo son conocidos también con el nombre de vectores. Conceptualmente, podemos pensar en un arreglo unidimensional como una lista compuesta de datos, donde para referirnos a uno de ellos emplearemos un número para indicar la posición del mismo dentro de la lista.

Por ejemplo, supongamos que elaboremos un arreglo llamado ventas, la cual está pensada para registrar las ventas de cada uno de los días de la semana. De manera conceptual podemos ver el arreglo de la siguiente manera:

Si en el arreglo queremos que el elemento 4 contenga el valor de 8987 lo podemos lograr con la instrucción: Ventas [4] = 8900; y el estado del arreglo quedaría así:

Empezando el primer elemento del Arreglo como lunes y el último como domingo, el código quedaría de la siguiente manera:

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

Using namespace std;

int main ()
{
                    int ventas[6];
                    ventas[4]=4900;
                    cout<<"Las ventas del dia viernes fueron de:  "<<ventas[4];
   getch();
  return 0;
}
```

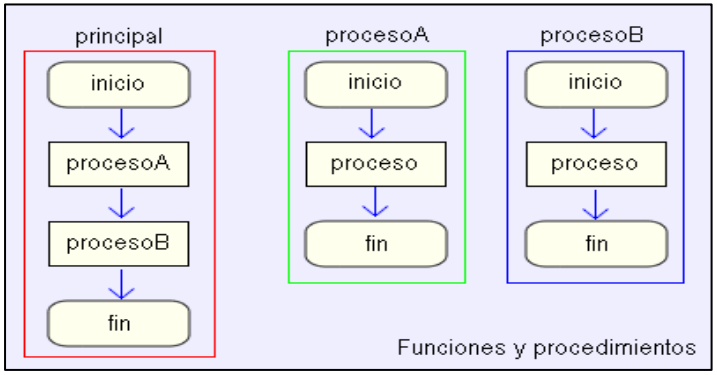
Arreglos Bidimensionales

Un arreglo bidimensional es aquel en donde los componentes son accesibles por medio de una pareja de índices que apunten a la fila y a la columna del componente requerido. Los arreglos de este tipo son conocidos también con el nombre de *matrices*.
Conceptualmente, podemos pensar en un arreglo bidimensional como en una lista compuesta de filas y columnas, en donde para referirnos a una de ellas emplearemos un número para indicar la posición de fila y otro número para indicar la posición de la columna del componente deseado

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

using namespace std;

int main(){
Intventas[2][7]={{2800,7500,2300,4300,4900,1300,1600},{4300,1350,
2250,5200,4200,5000,3000}};

  ventas[1][5]=9000;

                cout<<"Las ventas del dia viernes fueron de:
"<<ventas[1][5];

                getch();
                return 0;
}
```

Unidad 6
Funciones

Las funciones son un conjunto de instrucciones que realizan una tarea específica. En general toman ciertos valores de entrada, llamados parámetros y proporcionan un valor de salida o valor de retorno; aunque en C++, tanto unos como el otro son opcionales, y pueden no existir.



Acerca de los argumentos o parámetros

Hay algunos detalles respecto a los argumentos de una función, veamos:

- Una función o procedimiento pueden tener una cantidad cualquier de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más parámetros. Aunque habitualmente no suelen tener más de 4 o 5.
- Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.
- Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo de la función.

Consejos acerca de return

Debes tener en cuenta dos cosas importantes con la sentencia return:

Cualquier instrucción que se encuentre después de la ejecución de return NO será ejecutada. Es común encontrar funciones con múltiples sentencias return al interior de condicionales, pero una vez que el código ejecuta una sentencia return lo que haya de allí hacia abajo no se ejecutará.

El tipo del valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara int, el valor retornado debe ser un número entero.

Llamar o invocar a una función.

Para que una función se ejecute es necesario llamarla o invocar desde alguna parte del programa.

La llamada a una función está formada por su nombre seguido de una lista de argumentos entre paréntesis y separados por comas (cero o más argumentos) que son los datos que se le envían a la función.

Los argumentos que aparecen en la llamada a una función se llaman **parámetros actuales**, porque son los valores que se pasan a ésta en el momento de la ejecución.

Los parámetros actuales y los formales deben coincidir en número, orden y tipo. Si el tipo de un parámetro actual no coincide con su correspondiente parámetro formal, el sistema lo convertirá al tipo de este último, siempre que se trate de tipos compatibles. Si no es posible la conversión, el compilador dará los mensajes de error correspondientes.

EJEMPLO:

```
// Programa que lee un año y muestra si es o no bisiesto
#include <iostream>
using namespace std;
int bisiesto(int);   //declaración o prototipo de la función
int main()
{
  int anio;
  cout<<"Introduce a"<<(char)164<<"o: "; //164 ascii de ñ
  cin >> anio;
  if(bisiesto(anio))  //llamada a la función
    cout << "Bisiesto" << endl;
  else
    cout << "No es bisiesto" << endl;
  system("pause");
}
int bisiesto(int a)   //definición de la función
{
   if(a%4==0 and a%100!=0 or a%400==0)
      return 1;
   else
      return 0;
}
```