Estructuras de decisión

Decisión simple

La decisión simple permite que el programa elija entre dos posibles caminos de ejecución. La ejecución se determina luego de evaluar la proposición lógica que contiene la decisión simple: si es verdadera el flujo de ejecución toma un camino, y si es falsa otro.

Gracias a esta estructura de programación es posible hacer que nuestros programas ejecuten o no instrucciones a partir de una o más condiciones.

La forma general es la siguiente:

```
if (proposicion_logica){
  // Instrucciones a ejecutar si la proposición lógica fue verdadera
}
else{
  // Instrucciones a ejecutar si la proposición lógica fue falsa
}
```

Código 1 · Estructura base de la decisión simple.

Por ejemplo, si dado un número entero ingresado queremos saber si es positivo. Necesitamos hacer uso de una decisión simple.

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

int main(){
  int numero;
  cout << "Ingresar número: ";
  cin >> numero;
  if (numero > 0){
    cout << "El número es positivo";
  }
  return 0;
}
```

Código 2 · Ingresa por teclado un número y determina e informa si es positivo.

Como se puede observar el código anterior, no hay garantía de que el mensaje "El número es positivo" se muestre siempre que ejecutemos nuestro programa. Esto se debe a que depende del número ingresado por el usuario. Si ingresa un número 0 o negativo, la proposición lógica de nuestra decisión simple no se evaluará como verdadero y, por lo tanto, no tenemos acciones por ejecutar.
Con esto además, podemos observar que una decisión simple puede no tener instrucciones si la condición no se evalúa como verdadera. En ese caso, no escribiremos su correspondiente else.

Si retomamos el ejemplo anterior y queremos que se muestre por pantalla un mensaje cuando el número ingresado no sea positivo entonces haremos uso del *else*.

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

int main(){
  int numero;
  cout << "Ingresar número: ";
  cin >> numero;
  if (numero > 0){
    cout << "El número es positivo";
  }
  else {
    cout << "El número NO es positivo";
  }
  return 0;
}
```

Código 3 · Ingresa por teclado un número y determina e informa si es positivo o no.

Una aclaración importante a tener en cuenta y que puede observarse en el código anterior es que el else de un if no tiene condición. Un else propiamente dicho no puede tener condición ya que es el camino que debe ejecutarse si la condición de la estructura de decisión no es evaluada como verdadera.
Si queremos que en el lado falso, o else, de una condición se realice otra decisión entonces necesitaremos hacer uso de otra estructura de decisión. Es decir, debemos escribir otro if.

Supongamos ahora que deseamos que dado un número entero ingresado por el usuario, queremos saber si el número ingresado es positivo, negativo o cero. No puede resolverse con una sola decisión simple ya que tenemos tres posibles resultados. Pero sí es posible combinar varias decisiones simples entre sí.

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

int main(){
  int numero;
  cout << "Ingresar número: ";
  cin >> numero;
  if (numero > 0){
    cout << "El número es positivo";
  }
  else{
    if (numero == 0){
      cout << "El número es cero";
    }
    else{
      cout << "El número es negativo";
    }
  }
  return 0;
}
```

Código 4 · Ingresa por teclado un número y determina e informa si es positivo, negativo o cero.

En el código de arriba podemos ver como luego de ingresar el número se procederá a evaluar si es mayor a cero. Si lo es, muestra por pantalla que es positivo y luego no tiene más instrucciones por ejecutar por lo que finaliza el programa con valor de retorno cero.
En caso de no ser mayor a cero, continúa ejecutando el lado falso (o else) aquí tiene otra decisión simple por realizar preguntando si el número es igual a cero (nótese el == para comparar), en caso de serlo, muestra por pantalla que el número es cero y luego finaliza el programa ya que no tiene más instrucciones por ejecutar. Caso contrario, muestra por pantalla que es negativo y finaliza el programa.

Operadores relacionales

Para poder realizar proposiciones lógicas es necesario hacer uso de operadores relacionales. Los mismas se detallan a continuación:

Símbolo	Operación
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
==	Igual que
!=	Distinto que

Tabla 1 · Operadores relacionales.

Operadores lógicos

También es posible que necesitemos hacer una decisión simple que evalúe más de una proposición lógica a la vez. Para ello utilizaremos los operadores lógicos. Existen varios operadores lógicos pero los más utilizados son And y Or.

Operador en C/C++	Inglés	Castellano
&&	and	Y
||	or	O
!	not	NO

Tabla 2 · Operadores lógicos.

Operador And

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

int main(){
  int numero;
  cout << "Ingresar número: ";
  cin >> numero;
  if (numero > 0 && numero < 10){
      cout << "El número está entre 1 y 9";
  }
  return 0;
}
```

Código 5 · Ingresa un número y determina e informa si está entre 1 y 9.

En el ejemplo anterior podemos ver que disponemos de una decisión simple que se hace dos preguntas para ejecutar el código de su "lado verdadero". En este caso se ha utilizado el operador &&. Este operador lógico nos permite combinar proposiciones lógicas y su resultado será verdadero si todas las proposiciones lógicas son verdaderas.

Ejemplo

numero	numero > 0	&&	numero < 10	Resultado de la proposición lógica
50	verdadero	&&	falso	falso
- 90	falso	&&	verdadero	falso
6	verdadero	&&	verdadero	verdadero

Tabla 3 · Demostración del código anterior.

Podemos resumirlo a:

A	B	A && B
verdadero	verdadero	verdadero
falso	verdadero	falso
verdadero	falso	falso
falso	falso	falso

Tabla 4 · Tabla de verdad del operador and.

Operador Or

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

int main(){
  int edad;
  cout << "Ingresar edad: ";
  cin >> edad;
  if (edad < 10 || edad > 65){
      cout << "La edad es menor a diez o mayor a sesenta y cinco";
  }
  return 0;
}
```

Código 6 · Ingresa un número y determina e informa si es menor a 10 o mayor a 65.

En el ejemplo anterior podemos ver que disponemos de una decisión simple que se hace dos preguntas para ejecutar el código de su "lado verdadero". En este caso se ha utilizado el operador || u or. Este operador lógico nos permite combinar proposiciones lógicas y su resultado será verdadero si alguna de las proposiciones lógicas son verdaderas.

Ejemplo

edad	edad < 10	||	edad > 65	Resultado de la proposición lógica
6	verdadero	||	falso	verdadero
90	falso	||	verdadero	verdadero
40	falso	||	falso	falso

Tabla 5 · Demostración de la lógica del código anterior.

Podemos resumirlo a:

A	B	A || B
verdadero	verdadero	verdadero
falso	verdadero	verdadero
verdadero	falso	verdadero
falso	falso	falso

Tabla 6 · Tabla de verdad del operador or.

Operador Not

El operador Not no es utilizado como el operador And o el operador Or. Es decir, no nos sirve para combinar varias proposiciones lógicas. Lo que nos permite el operador Not es negar un valor de verdad obteniendo su valor contrario. Recordemos que un valor de verdad puede tomar únicamente dos posibles valores **verdadero** o **falso.** Por lo que, aplicar el operador not a verdadero nos dará falso y viceversa.

A	!A
verdadero	falso
falso	verdadero

Tabla 7 · Tabla de verdad del operador not.

En el lenguaje C++ utilizaremos las siguientes palabras reservadas para referirnos a los valores de verdad verdadero y falso.
verdadero → **true**
falso → **false**

Decisión múltiple

La decisión múltiple, en C/C++, la utilizaremos cuando deseemos evaluar entre un conjunto de datos y éstos sean valores o "casos" específicos. Por cada conjunto de casos podremos especificar qué acciones queremos que nuestro programa realice.

Cabe destacar que, según lo explicado en el párrafo anterior, la decisión múltiple no podrá ser utilizada para establecer casos donde especifiquemos rangos.

```
switch (var){
  case 1:
   // A) instrucciones a ejecutar si variable vale 1
   cout << "A";
  break;

  case 5:
  case 10:
   // B) instrucciones a ejecutar si variable vale 5 o 10
   cout << "B";
  break;
}
```

Código 6 · Estructura básica de la decisión múltiple

Como se puede observar en el código anterior de ejemplo, la decisión múltiple evalúa la variable *var*.

Si la variable var tiene el valor 1, se ejecuta el código que se puede observar en el primer case. A modo de ejemplo se escribió la leyenda "A".

En cambio, si la variable var tiene el valor 5 o el valor 10, se ejecuta el código que se puede observar en el segundo case. A modo de ejemplo se escribió la leyenda "B".

El programa no realiza ninguna acción si la variable var contiene un valor distinto a 1, 5 o 10.

Caso por omisión

La decisión múltiple cuenta con un caso denominado default y que se ejecutará cuando ninguno de los casos del case se hayan comprobado. En el ejemplo anterior, si la variable contiene algo distinto a 1, 5 o 10 podemos contar con un caso que ejecute código ante tal situación. En el siguiente ejemplo mostraremos la leyenda "C" en el caso por omisión.

```
switch (var){
  case 1:
   // A) instrucciones a ejecutar si variable vale 1
   cout << "A";
  break;
  case 5:
  case 10:
   // B) instrucciones a ejecutar si variable vale 5 o 10
   cout << "B";
  break;
  default:
  // C) instrucciones a ejecutar por omisión a los casos previos
   cout << "C";
  break;
}
```

Código 7 · Estructura básica de la decisión múltiple con caso por omisión.

Veamos un ejemplo donde se ingrese un código de las tres primeras categorías de monotributo y muestre el valor mensual a abonar en el año 2024.

```
#include <iostream>
using namespace std;
#include <ctype>
int main(){
  char categoria;
  float importe;
  cout << "Ingresar categoría de monotributo: ";
  cin >> categoria;
  categoria = toupper(categoria); // Convierte a mayúsculas el caracter
  switch(categoria){
    case 'A':
     importe = 12128.39;
    break;
    case 'B':
     importe = 13561.75;
    break;
    case 'C':
     importe = 15503.51;
    break;
  }
  cout << "El importe a abonar es: " << importe << endl;
  return 0;
}
```

Código 8 · Demostración del funcionamiento de la estructura de decisión múltiple evaluando una variable de tipo carácter.

Veamos otro ejemplo donde se ingrese la posición en una competencia de atletismo y muestre por pantalla el tipo de medalla que se obtiene por dicha posición.

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

int main(){
  int posicion;
  string medalla;

  cout << "Ingresar posición en la competencia: ";
  cin >> posicion;

  switch(posicion){
```

```
    case 1:
     medalla = "Oro";
    break;
    case 2:
     medalla = "Plata";
    break;
    case 3:
     medalla = "Bronce";
    break;
  }
  cout << "La medalla obtenida es de " << medalla << endl;
  return 0;
}
```

Código 8 · Demostración del funcionamiento de la estructura de decisión múltiple evaluando una variable de tipo entera.