Sentencias de control. Estructuras de selección

Las estructuras de selección permiten realizar distintas acciones en el programa dependiendo del valor de una condición.

Selección simple.

La estructura alternativa permite seleccionar qué instrucciones se han de ejecutar dependiendo del valor que tengan determinadas variables en un momento dado.

La sintaxis que tiene la *alternativa* es la siguiente:

```
if (condición){
      S;
}
```

Donde **condición** representa una expresión booleana y S la secuencia es un conjunto de instrucciones que solo se ejecutarán si la condición booleana *condición* se evalúa a *cierto*.

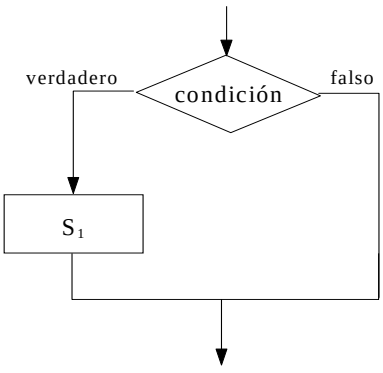
Cuando la secuencia de instrucciones está formada por una única sentencia, no es necesario el uso de llaves. Sin embargo, si la secuencia de instrucciones es compleja, el conjunto debe ir entre llaves.

Los operadores que se pueden emplear en una expresión booleana con el fin de obtener los valores *cierto* y *falso* son los siguientes:

Lógicos**:** **&&** **||** **!**

Relacionales: **<** **<=** **==** **<>** **>=** **>**

En la siguiente figura se puede ver cuál sería la representación mediante un *diagrama de flujo dimensional* de esta sentencia.



Los diagramas de flujo de datos -*dfd*- realizan una representación gráfica de los algoritmos. Como su nombre indica, en ellos se resalta, principalmente, la lógica o el flujo de control del programa, es decir, lo que se hace en cada momento.

Las expresiones condicionales se evalúan en, lo que se denomina, modo '*short-circuit*', es decir, dadas expresiones booleanas como:

- "(condición1 || condición2)", si *condición1* es cierta, entonces no se evalúa *condición2*, ya que el valor de verdad de la expresión será cierto, valga lo que valga *condición2*.
- "(condición1 && condición2)", si *condición1* es falsa, entonces no se evalúa *condición2*, ya que el valor de verdad de la expresión será falso, valga lo que valga *condición2*.

La ventaja de este modo de evaluación de las expresiones booleanas radica en el incremento de velocidad de la ejecución de los programas, ya que no se invierte tiempo haciendo cálculos que no van a influir en el valor de verdad final de la expresión con la que estamos tratando. Sin embargo, plantea ciertas pegas a los programadores a través de lo que se conocen como *efectos laterales* o *side-effects*, es decir, trozos de código que el programador esperaba que fuesen ejecutados, no llegan a serlo debido a la forma de evaluar la expresión.

Es importante saber que no todos los lenguajes de programación evalúan las expresiones booleanas de este modo (El lenguaje C sí lo hace, mientras que otros lenguajes no lo hacen). El usuario de un lenguaje de programación debe conocer el modo de evaluación de las expresiones booleanas que sigue el lenguaje elegido para programar, ya que esto puede repercutir en la correcta ejecución de los algoritmos debido los efectos laterales.

Ejemplo. Expresar mediante sentencias condicionales que:

q = 1 si (n > 0)

0 si (n = 0)

-1 si (n < 0)

```
#include <iostream>

using namespace std;
int main() {
   int q, n;

   cout << "Introduce el valor de n: ";
   cin >> n;

   if (n>0)
      q = 1;
   if (n==0)
      q = 0;
   if (n<0)
      q = -1;
}
```

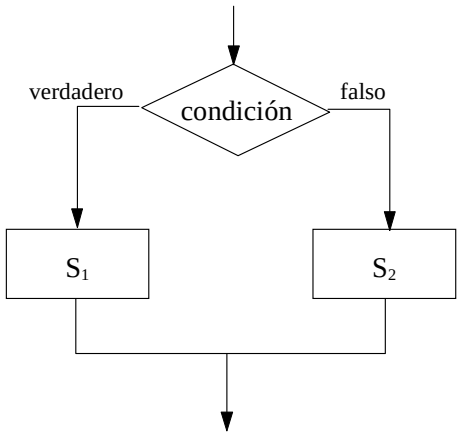
Selección doble.

La estructura alternativa permite un enunciado más general, el cual da la posibilidad de expresar una serie de sentencias que se ejecutan si no se cumple la condición. La sintaxis es esta otra:

```
if (condición)
   S1;
else
   S2
```

Donde, *condició*n representa una expresión booleana, y S_1 y S_2 representan conjuntos de instrucciones. Las primeras solo se ejecutarán si la condición booleana *condición* se evalúa a *cierto*, y las segundas si se evalúa a *falso*.

La representación mediante un *dfd* de esta sentencia se puede ver en la siguiente figura.



Al igual que con la selección simple, las secuencias de instrucciones deben ir entre llaves cuando se trate de más de una sentencia.

Ejemplo. Escribir un programa que calcule el mínimo de dos enteros introducidos por teclado:

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

int main(){
   float a, b, min;

   cout << "Introduce el primer número: ";
   cin >> a;
   cout << ”Introduce el segundo número: ";
   cin >> b;

   if (a<=b)
      min = a;
   else
      min = b;

   cout << "El mínimo es " << min;
}
```

Ejemplo. Escribir un programa que calcule el mínimo de tres enteros introducidos por teclado

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

int main() {
   float a, b, c, min;

   cout << "Introduce el primer número: ";
   cin >> a;
   cout << "Introduce el segundo número: ";
   cin >> b;
```

```
    cout << "Introduce el tercer número: ";
    cin >> c;

    if (a<=b)
       min = a;
    else
       min = b;
    if (min>c)
       min = c;

    cout << "El mínimo es " << min;
 }
```

Selección múltiple.

`if-else` anidada

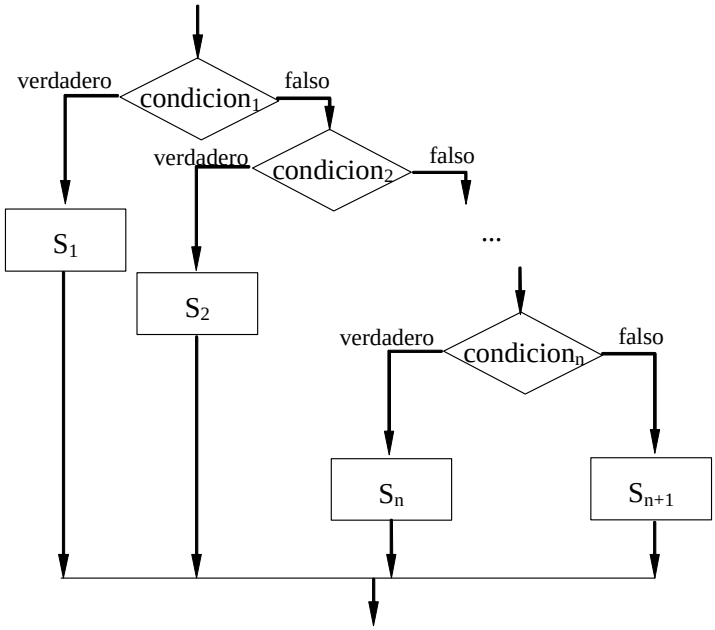
Consiste en usar estructuras if-else de manera **anidada,** es decir, unas estructuras if-else dependientes de otras. La sintaxis es:

```
if (expresión_lógica_1) {
      S1;
} else if (expresión_lógica_2) {
           S2;
       } else if (expresión_lógica_3) {
                  S3;
              }
```

Donde S_1, S_2 y S_3, corresponden a conjuntos de instrucciones.

Sólo se ejecuta la primera de las secuencias de sentencias cuya expresión lógica asociada se evalúe a verdadero. Si todas las expresiones lógicas se evalúan a falso, entonces se ejecuta la secuencia de sentencias situada a continuación del `if-else` anidado a menos que la última alternativa este asociada a una parte `else`, en cuyo caso se ejecutará esta rama.

La siguiente figura representa el dfd.



switch

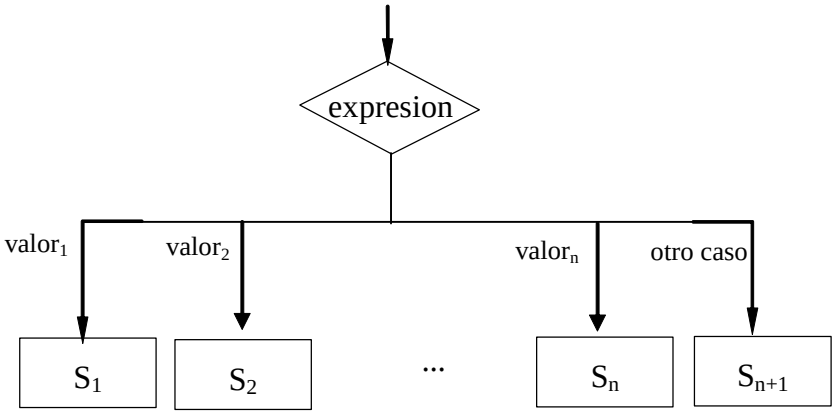
En ocasiones es interesante poder distinguir entre diversas alternativas sin tener que emplear múltiples sentencias `if...else`, las cuales producen el efecto deseado, pero oscurecen el código escrito. En estos casos podemos usar la sentencia `switch`, la cual tiene la siguiente sintaxis:

```
switch (expresión) {
    valor_1:  S1;
              break;
    valor_2:  S2;
              break;
    valor_n:  Sn;
              break;
    default:  Sn+1;
}
```

Como se observa, se pregunta por el valor actual de la variable, y dependiendo de cuál sea éste, se selecciona una u otra secuencia de instrucciones S_i a ejecutar. Si ocurre que el valor de la variable no se corresponde con ninguno de los valores '**valor**' que esperábamos, entonces se ejecuta la secuencia de instrucciones asociadas a la palabra reservada **`default`**, la cual representa cualquier otro valor de la variable no tenido en cuenta previamente por la sentencia **`switch`**. Una vez ejecutada esta secuencia de instrucciones, el flujo del programa se transmite automáticamente al final de la sentencia **`switch`**, es decir, a la primera sentencia que siga a la llave final.

El tipo de la expresión debe ser escalar (entero, carácter, booleano). Las etiquetas case no pueden estar repetidas y deben ser del mismo tipo que el selector. Cada sentencia debe estar precedida por una o más etiquetas **`case`**. La sentencia **`break`** hace que se transfiera el control fuera de la estructura **`switch`**. En caso de no existir break, una vez encontrada la etiqueta case que coincide con el valor del selector, se ejecutarían la sentencia asociada a la etiqueta en cuestión y todas las sentencias que hay hasta el final del **switch**.

La representación mediante un *dfd* de esta sentencia la podemos ver en la siguiente figura



Un ejemplo de uso habitual es la selección de una opción de un menú.

Ejemplo

Implementa un algoritmo en C que realice las cuatro operaciones básicas de una calculadora (suma, resta, multiplicación y división). El programa debe leer los dos números y la operación y devolver el resultado.

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

int main() {
   float a, b, resultado;
   char op;

   cout << "Introduce el primer término: ";
   cin >> a;
   cout << "Introduce el segundo término: ";
   cin >> b;
   cout << "Introduce la operación ";
   cin >> op;

   switch (op){
      case '+': resultado = a + b;
                break;
      case '-': resultado = a - b;
                break;
      case '*': resultado = a * b;
                break;
      case '/': resultado = a / b;
                break;
   }
   cout << "El resultado es: " << resultado << endl;
}
```