Decision-making statements available in C or C++ are:

* **if statement**
* **if..else statements**
* **nested if statements**
* **if-else-if ladder**
* **switch statements**
* **Jump Statements:**

1. **break**
2. **continue**
3. **goto**
4. **return**

**if statement in C/C++:**

if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

**Syntax**:

if(condition)

{

   // Statements to execute if

   // condition is true

}

Here, the **condition** after evaluation will be either true or false. C if statement accepts Boolean values – if the value is true then it will execute the block of statements below it otherwise not. If we do not provide the curly braces ‘{‘ and ‘}’ after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

**if-else in C/C++:**

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

**Syntax**:

if (condition)

{

    // Executes this block if

    // condition is true

}

else

{

    // Executes this block if

    // condition is false

}

**Nested-if :**

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement. **Syntax**:

if (condition1)

{

   // Executes when condition1 is true

   if (condition2)

   {

      // Executes when condition2 is true

   }

}

**if-else-if ladder in C/C++**

Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.

**Syntax**:

 if (condition)

    statement;

else if (condition)

    statement;

.

else

    statement;

**Jump Statements in C/C++**

These statements are used in C orC++ for the unconditional flow of control throughout the functions in a program. They support four types of jump statements:

1. **C break**: This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

**Syntax:**

break;

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.

1. **C continues**: This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

**Syntax:**

continue;

If you create a variable in if-else in C/C++, it will be local to that if/else block only. You can use global variables inside the if/else block. If the name of the variable you created in if/else is as same as any global variable then priority will be given to ***LocalVariable.***

1. **C goto**: The goto statement in C/C++ also referred to as unconditional jump statement can be used to jump from one point to another within a function.

Syntax:

Syntax1      |   Syntax2

----------------------------

goto label;  |    label:

.            |    .

label:       |    goto label

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here label is a user-defined identifier that indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement in the above syntax.

1. **C return**: The return in C or C++ returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and return the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value is must be returned.

Syntax:

return[expression];

**Switch Statement in C/C++**

Switch case statement evaluates a given expression and based on the evaluated value(matching a certain condition), it executes the statements associated with it. Basically, it is used to perform different actions based on different conditions(cases).

* Switch case statements follow a selection-control mechanism and allow a value to change control of execution.
* They are a substitute for long if statements that compare a variable to several integral values.
* The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

**Syntax:**

switch (n)

{

    case 1: // code to be executed if n = 1;

        break;

    case 2: // code to be executed if n = 2;

        break;

    default: // code to be executed if n doesn't match any cases

}