Condition Statement / Decision Making in C

Conditional statements, also referred to as decision control structures like if, if else, switch, etc., are employed for decision-making in C programs.

These are also called Decision-Making Statements and are used to assess one or more conditions, determining whether a specific set of statements should be executed. Such statements guide the flow of program execution based on the conditions evaluated.

Types of Conditional Statements / Decision Making in C

Following are the decision-making statements available in C:

1. if Statement
2. if-else Statement
3. Nested if Statement
4. if-else-if Ladder
5. switch Statement
6. Conditional Operator
7. Jump Statements:

- break
- continue
- goto
- return

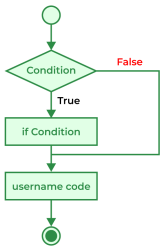
if Statement

The 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 statements is executed otherwise not.

Syntax:

```
if(condition)
{
  // Statements to execute if
  // condition is true
}
```

Flowchart of if Statement



Example:

```
#include <stdio.h>

int main() {
    int number = 10;

    if (number > 0) {
        printf("Number is positive.\n");
    }

    return 0;
}
```

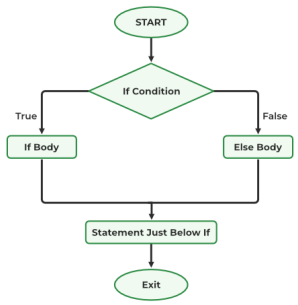
if - else Statement

The if statement by itself tells us that if a condition is true, it will execute a specific block of statements, and if the condition is false, it will not. But what if we want to perform a different action when the condition is false? This is where the else statement in C comes into play. The else statement can be paired with the if statement to run a block of code when the condition evaluates to false. The if-else structure consists of two parts: one for when the condition is true and another for when it is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

Flowchart of if- else statement



Example:

```
#include <stdio.h>

int main() {
    int number = -5;

    if (number > 0) {
        printf("Number is positive.\n");
    } else {
        printf("Number is not positive.\n");
    }

    return 0;
}
```

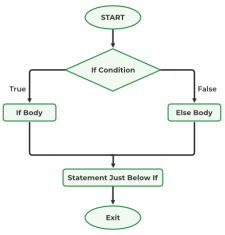
Nested if - else in C

A nested if in C refers to an if statement that is within the scope of another if statement. This means having one if statement inside another. C allows for such nesting, meaning you can place one if statement within another to handle more complex conditions.

Syntax:

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition_2)
  {
    // statement 1
  }
  else
  {
       // Statement 2
   }
}
else {
    if (condition_3)
  {
    // statement 3
  }
  else
  {
       // Statement 4
   }
}
```

Flowchart of Nested if-else



Example:

```
#include <stdio.h>
int main() {
    int number = 0;

    if (number > 0) {
        printf("Number is positive.\n");
    } else if (number < 0) {
        printf("Number is negative.\n");
    } else {
        printf("Number is zero.\n");
    }

    return 0;
}
```

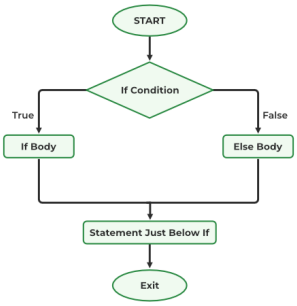
if-else-if Ladder

The if else if statements are utilized when a decision needs to be made between multiple options. In C, these if statements are evaluated from top to bottom. Once a condition is found to be true, the corresponding block of code is executed, and the remaining else-if conditions are skipped. If none of the conditions are met, the final else block is executed. The if-else-if ladder works in a manner similar to the switch statement.

Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

Flowchart of if-else-if ladder



Example:

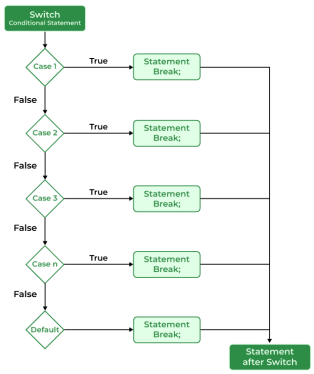
```
#include <stdio.h>
int main() {
    int marks = 85;

    if (marks >= 90) {
        printf("Grade: A\n");
    }
    else if (marks >= 80) {
        printf("Grade: B\n");
    }
    else if (marks >= 70) {
        printf("Grade: C\n");
    }
    else if (marks >= 60) {
        printf("Grade: D\n");
    }
    else {
        printf("Grade: F\n");
    }

    return 0;
}
```

Switch Case Statement:

The switch case statement is an alternative to the if-else-if ladder and is used to run conditional code depending on the value of the variable specified in the switch statement. The switch block contains multiple cases, each executed according to the value of the switch variable.

```
switch (expression) {
    case value1:
        statements;
    case value2:
        statements;
    ....
    ....
    ....
    default:
        statements;
}
```

Flowchart:



Example:

```
#include <stdio.h>
int main() {
    int day = 3;

    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        case 4:
            printf("Thursday\n");
            break;
        default:
```

```
        printf("Invalid day\n");
    }

    return 0;
}
```

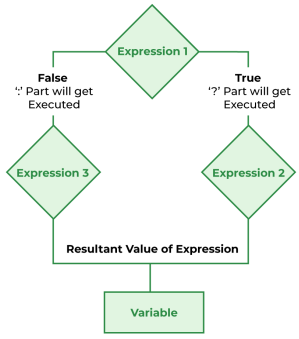
Conditional Operator

The conditional operator is used to include conditional logic in a program. It functions similarly to the if-else statement. Known as the ternary operator, it operates on three operands.

Syntax:

(condition) ? [true_statements] : [false_statements];

Flowchart in Conditional Operator



Example:

```
#include <stdio.h>
int main() {
    int a = 10, b = 20;
    int max;

    // Conditional operator to find the maximum value
    max = (a > b) ? a : b;
     printf("The maximum value is: %d\n", max);
    return 0;
}
```

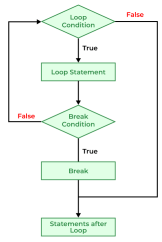
Jump Statements

These statements in C are used for the unconditional flow of control within the functions of a program. They encompass four types of jump statements:

Break in c

This loop control statement is used to end the loop. Once the break statement is encountered inside a loop, the loop's execution halts, and control is immediately passed to the first statement following the loop.
Syntax: break;

Flowchart of break



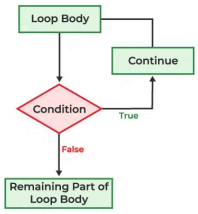
Example

```
#include <stdio.h>
int main() {
    int i;
  // Loop from 1 to 10
    for (i = 1; i <= 10; i++) {
        // If i equals 5, break the loop
        if (i == 5) {
            break;
        }
        printf("%d\n", i);
    }

    return 0;
}
```

Continue

This loop control statement functions similarly to the break statement but with a different purpose. Unlike the break statement, which terminates the loop, the continue statement skips the current iteration and moves to the next one. As its name implies, the continue statement causes the loop to proceed with the next iteration. When it is encountered, the remaining code within the loop after the continue statement is bypassed, and the loop starts the next iteration.
Syntax: continue;

Flowchart of Continue



Example:

```
#include <stdio.h>
int main() {
    int i;
   // Loop from 1 to 10
    for (i = 1; i <= 10; i++) {
        // If i is 5, skip the current iteration
        if (i == 5) {
            continue;
        }
        printf("%d\n", i);
    }
    return 0;
}
```

Goto Statement

The goto statement in C, also known as an unconditional jump statement, allows jumping from one location to another within a function.

Syntax:

```
Syntax1      |   Syntax2
---------------------------
goto label;  |    label:
.            |    .
.            |    .
.            |    .
label:       |    goto label;
```

Example:

```
#include <stdio.h>
int main() {
    int num = 1;
   // Using goto to jump to a label
    if (num == 1) {
```

```
        goto label;
    }

    printf("This will not be printed.\n");

label:
    printf("Jumped to the label using goto.\n");

    return 0;
}
```

Return

The return in 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 returns 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 must be returned.

Flowchart of Return