Decision Making in C (if , if..else, Nested if, if-else-if)

In C, programs can choose which part of the code to execute based on some condition. This ability is called **decision making** and the statements used for it are called **conditional statements.** These statements evaluate one or more conditions and make the decision whether to execute a block of code or not.

For example, consider that there is a show that starts only when certain number of people are present in the audience. So, you can write a program like as shown:

```
{...}
  // Number of people in the audience
  int num = 100;

  // Conditional code inside decision making statement
  if (num > 50) {
    printf("Start the show");
  }

{...}
```

Output

```
Start the show
```

In the above program, the show only starts when the number of people is greater than **50**. It is specified in the **if statement** (a type of conditional statement) as a condition **(num > 50)**. You can decrease the value of **num** to less than **50** and try rerunning the code.

Types of Conditional Statements in C

In the above program, we have used if statement, but there are many different types of conditional statements available in C language:

1. if in C

The if statement is the simplest 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.

A **condition** is any expression that evaluates to either a true or false (or values convertible to true or flase).

Example

```
{...}
  int i = 10;


  // If statement
  if (i < 18) {
    printf("Eligible for vote");
  }


{...}
```

Output

```
Eligible for vote
```

The expression inside **() parenthesis** is the **condition** and set of statements inside **{} braces** is its **body**. If the condition is true, only then the body will be executed.

*If there is only a single statement in the body, **{} braces** can be omitted.*

2. if-else in 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 when the condition is false? Here comes the C **else**statement. We can use the **else**statement with the **if**statement to execute a block of code when the condition is false. The if-else statement consists of two blocks, one for false expression and one for true expression.

Example

```
{...}
  int i = 10;
  if (i > 18) {
    printf("Eligible for vote");
  }
  else {
    printf("Not Eligible for vote");
  }

{...}
```

Output

```
Not Eligible for vote
```

The block of code following the *else* statement is executed as the condition present in the *if* statement is false.

3. Nested if-else in C

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, C allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Example

```
{...}
  int i = 10;

  if (i == 10) {
    if (i < 18)
      printf("Still not eligible for vote");
    else
      printf("Eligible for vote
");
```

```
  }
  else {
    if (i == 20) {
      if (i < 22)
        printf("i is smaller than 22 too
");
      else
        printf("i is greater than 25");
    }
  }

{...}
```

Output

```
Still not eligible for vote
```

4. if-else-if Ladder in C

The if-else-if ladder are used when the user has to 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 is true, then the final else statement will be executed. if-else-if ladder is similar to the switch statement.

Example

```
{...}
  int i = 20;

  // If else ladder with three conditions
  if (i == 10)
    printf("Not Eligible");
  else if (i == 15)
```

```
    printf("wait for three years");
  else if (i == 20)
    printf("You can vote");
  else
    printf("Not a valid age");


{...}
```

Output

```
You can vote
```

5. switch Statement in C

The switch statement is an alternative to the if else if ladder that can be used to execute the conditional code based on the value of the variable specified in the switch statement. The switch block consists of cases to be executed based on the value of the switch variable.

Example

```
{...}
  // declaring switch cases
  switch (var) {
  case 15:
    printf("You are a kid");
    break;
  case 18:
    printf("Eligible for vote");
    break;
  default:
    printf("Default Case is executed");
    break;
```

```
{...}
```

Output

```
Eligible for vote
```

Note: *The switch expression should evaluate to either integer or character. It cannot evaluate any other data type.*

6. Conditional Operator in C

The conditional soperator is used to add conditional code in our program. It is similar to the if-else statement. It is also known as the ternary operator as it works on three operands.

Example:

```
{...}
  var = flag == 0 ? 25 : -25;
  printf("Value of var when flag is 0: %d
", var);

{...}
```

Output

```
Value of var when flag is 0: 25
```

7. Jump Statements in C

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

A) 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.

Example

```
{...}
    if (arr[i] == key) {
      printf("Element found at position: %d",
           (i + 1));
      break;
    }


{...}
```

Output

```
Element found at position: 3
```

B) continue

This loop control statement is just like the break statement. The conditional 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.

Example:

```
{...}
    if (i == 6)
      continue;


{...}
```

Output

```
1 2 3 4 5 7 8 9 10
```

C) goto

The goto statemin C also referred to as the unconditional jump statement can be used to jump from one point to another within a function.

Examples:

```
{...}
```

```
label:
  printf("%d ", n);
  n++;
  if (n <= 10)
    goto label;


{...}
```

Output

```
1 2 3 4 5 6 7 8 9 10
```

D) 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.

Example:

```
{...}
int SUM(int a, int b) {
  int s1 = a + b;
  return s1;
}


{...}
```

Output

```
20
```