Fundamentals of Programming: Control Structure

Abdul Haseeb

Agenda

- Introduction to control structure
- If statement
- If-else statement
- If-else-If statement
- Nested If
- Switch Case Statement
- Loops
  - For Loop
  - While Loop
  - Do While Loop
  - Foreach Loop (After Arrays)
  - Continue, Break, goto statement

Flow of execution

- The order in which a program executes
- **The Normal flow of program execution is sequential**

Control Statements

- Control the flow of execution of a Program
- Determine how instructions will be executed based on a certain criteria/condition

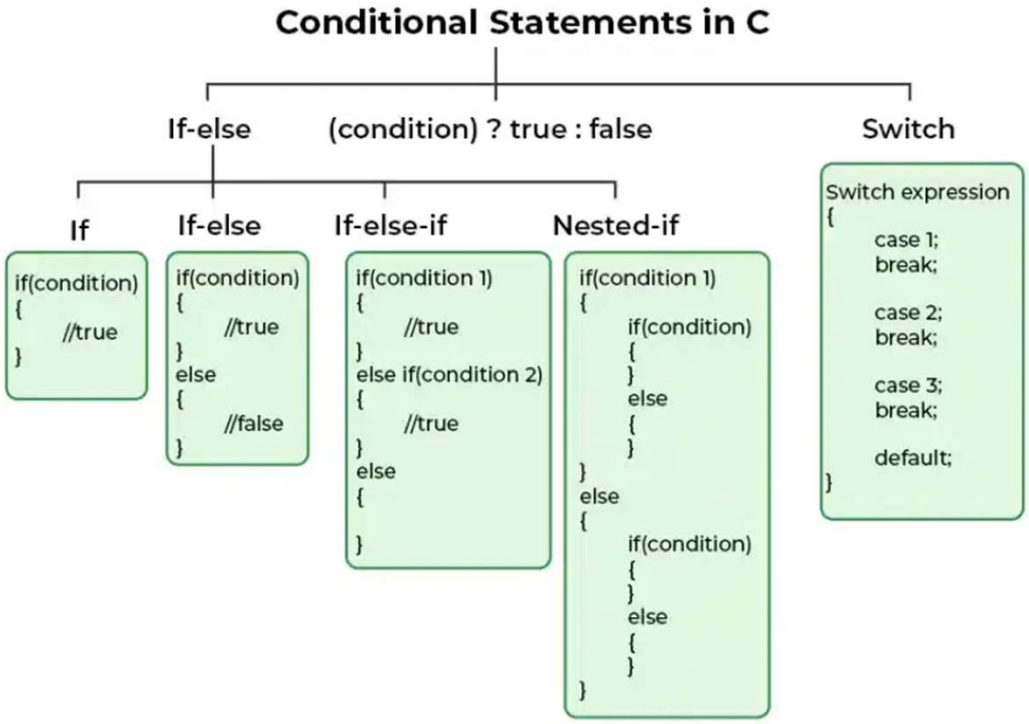
Types of Control Statements

- Sequential:
  - Default flow of program, without any decision making
- Selection/Conditional:
  - Make decisions and choose different path based on conditions/expressions
- Repetition/Loop Statements:
  - Perform a task repeatedly

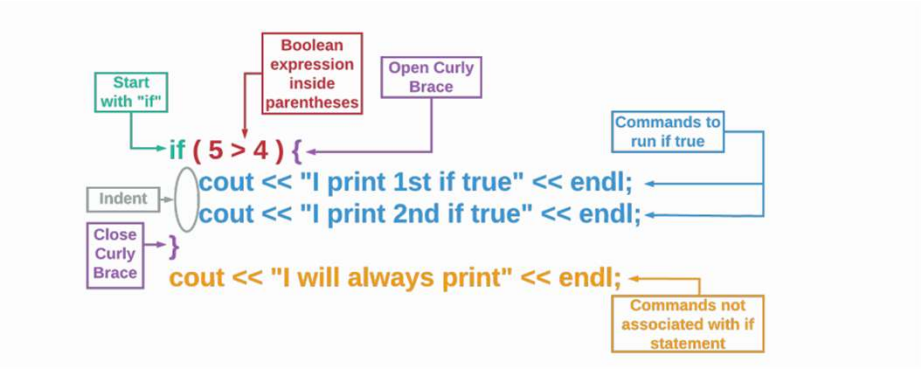
Need of Conditions

- In Real Life we make decisions and take actions accordingly
- Likewise in programming sometimes we may have to take decisions

Types of Selection/Conditional Statements



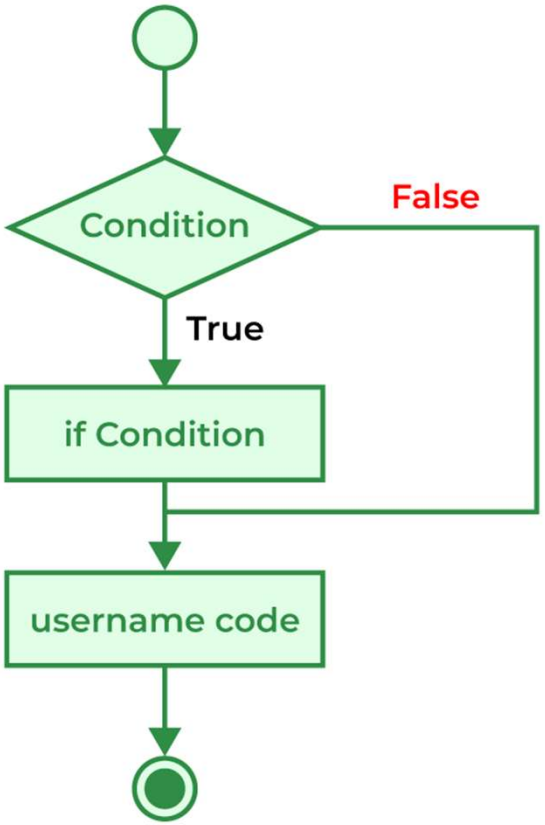
If-Statement



If statements in C++ must contain the following items:

* The keyword `if`.
* A boolean expression in parentheses, `()`.
* Curly braces, `{}`, surrounding all lines of code that will run if the boolean expression is true.

Flowchart of If-Statement



```
if (7 != 10) {
  cout << "The above statement is true" << endl;
  cout << "The above statement is still true" << endl;
}
cout << "This is not related to the if statement" << endl;
```

Example:

Challenge

What happens if you:

- Change `!=` in the code above to `==`?
- Change `7 != 10` in the code above to `true`?
- Change `7 != 10` in the code above to `false`?
- Remove the curly braces `{}` with the condition set to `if (false)`?

Testing Multiple Cases:

Sometimes you will need to test your variable for multiple times

Be sure that you have set up your conditional to test for all possible values

Example: Testing Multiple Cases

```
int grade = 90;

if (grade > 70) {
  cout << "Congrats, you passed the class" << endl;
}

if (grade < 70) {
  cout << "Condolences, you did not pass the class" << endl;
}
```

What happens if you:

- Assign `int grade` to `60`?
- Assign `int grade` to `70`?
- Change `grade > 70` in the code above to `grade >= 70`?

Challenge

Compound Conditional Statement

- Two or more than two conditions are to be tested

```
int num = 16;

if (num % 2 == 0 && num > 10) {
  cout << "Even and greater than 10" << endl;
}
```

Challenge

What happens if you:

- Assign `num` to `8`?
- Change `&&` in the code above to `||`?
- Change `==` in the code above to `!=`?

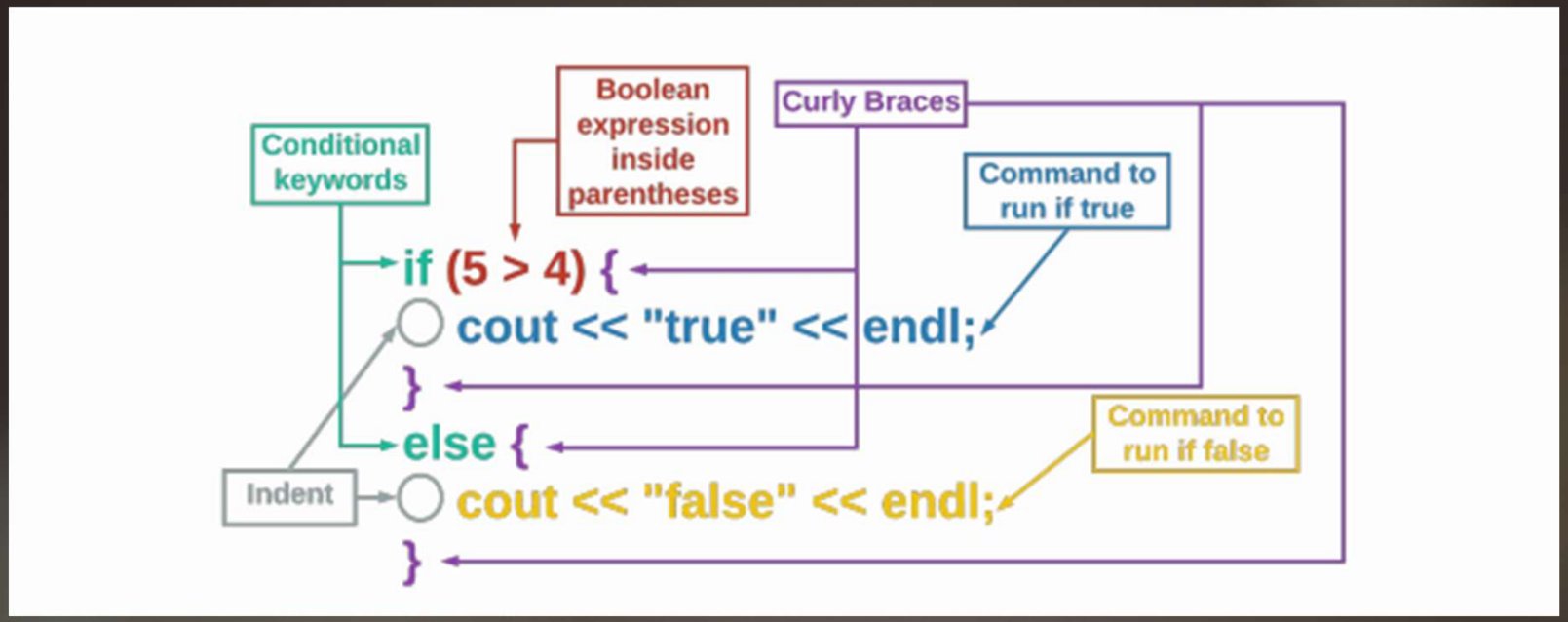
Why use Compound Conditionals

```
int my_var = 19;

if (my_var > 15) {
    if (my_var < 20) {
        cout << my_var << endl;
    }
}
```

```
int my_var = 19;

if (my_var > 15 && my_var < 20) {
    cout << my_var << endl;
}
```

The code on the left is a **nested** `if` statement - which means an `if` statement is *inside* another `if` statement.

The code with the **compound conditional** (on the right) has fewer lines of code, and is easier for a human to read. In fact, it almost reads like a sentence.



If-Else Syntax

Indentation is the best practice

```
if (5 > 4) {
  cout << "Print me if true" << endl;
}
else {
  cout << "Print me if false" << endl;
}
```

Challenge

challenge

What happens if you:

- Change `4` in the code above to `6`?
- Remove all the curly braces `{}`?
- Add `cout << "False" << endl;` under `cout << "Print me if false" << endl;` *without* any curly braces `{}` in the code?
- Add `cout << "True" << endl;` under `cout << "Print me if true" << endl;` *without* any curly braces `{}` in the code?