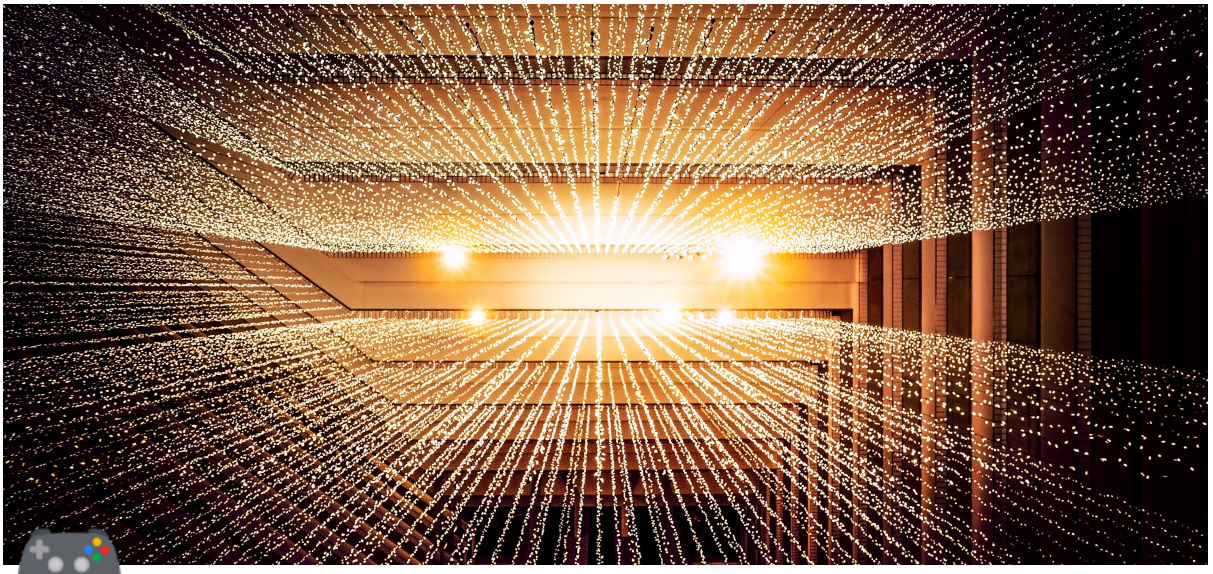

# Control Flow in C++

## Introduction

Control flow is the way a program decides which statements to execute and in what order. Control flow statements are special keywords or symbols that can alter the normal sequence of execution. Control flow is important for programming because it allows us to create complex and dynamic behaviors based on different conditions, inputs, or events.

Some examples of common tasks that require control flow are:

- **Validating user input:** We can use control flow statements to check if the user entered valid data, such as a number, a password, or a choice. If the input is invalid, we can display an error message and ask for a new input.
- **Performing calculations:** We can use control flow statements to perform different calculations based on different values, such as finding the area of a circle, a square, or a triangle. We can also use control flow statements to handle possible errors, such as division by zero or overflow.
- **Looping through data:** We can use control flow statements to iterate over a collection of data, such as an array, a vector, or a string. We can also use control flow statements to break out of the loop or skip some iterations based on certain conditions.

The main types of control flow statements in C++ are:

- **Conditional statements:** These are statements that execute a block of code only if a condition is true, or choose between different blocks of code based on the value of an expression. The most common conditional statements in C++ are `if`, `else`, `else if`, and `switch`.
- **Loops:** These are statements that execute a block of code repeatedly as long as a condition is true, or for a fixed number of times based on an initial value, a condition, and an update expression. The most common loops in C++ are `while`, `do-while`, `for`, and ranged-for.
- **Jumps:** These are statements that transfer control to another part of the program identified by a label. The most common jump statement in C++ is `goto`, which is generally considered bad practice and should be avoided if possible.
- **Function calls:** These are expressions that invoke functions with arguments and return values. Functions are reusable blocks of code that perform specific tasks. Function calls are a form of control flow because they transfer control to the function definition and back to the caller.
- **Exceptions:** These are special objects that represent abnormal or unexpected situations that occur during program execution. Exceptions can be thrown and caught using special keywords and blocks. Exceptions are a special kind of control flow structure designed for error handling.

In this document, we will explain each type of control flow statement in detail and give some examples of how to use them in C++.

# Conditional Statements

Conditional statements are statements that execute a block of code only if a condition is true, or choose between different blocks of code based on the value of an expression.

## The `if` statement

The `if` statement is the simplest form of conditional statement in C++. It executes a block of code only if a condition is true. The syntax of the `if` statement is:

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

The condition is an expression that evaluates to either true or false. The code inside the curly braces `{}` is called the body of the `if` statement. The body can contain one or more statements, or be empty. If the condition is true, the body is executed; otherwise, it is skipped.

For example, suppose we want to check if a number entered by the user is positive, negative, or zero. We can use an `if` statement like this:

```
// ask the user to enter a number
cout << "Enter a number: ";
// declare a variable to store the number
int number;
// read the number from standard input
cin >> number;
// check if the number is positive
if (number > 0) {
    // print "The number is positive" if the condition is true
    cout << "The number is positive" << endl;
}
```

If the user enters a positive number, such as 5, the condition `number > 0` will be true, and the program will print “The number is positive”. If the user enters a negative number, such as -3, or zero, the condition will be false, and the program will skip the body of the `if` statement.

## The `else` keyword

The `else` keyword can be used to provide an alternative block of code when the condition of an `if` statement is false. The syntax of using `else` with an `if` statement is:

```
if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if condition is false
}
```

The `else` keyword must be placed after the closing curly brace `}` of the `if` statement. The code inside the curly braces `{}` after the `else` keyword is called the `else` block. The `else` block can contain one or more statements, or be empty. If the condition of the `if` statement is false, the `else` block is executed; otherwise, it is skipped.

For example, suppose we want to modify the previous program to also print “The number is negative” or “The number is zero” depending on the value of the number entered by the user. We can use an `else` keyword like this:

```
// ask the user to enter a number
cout << "Enter a number: ";
// declare a variable to store the number
int number;
// read the number from standard input
cin >> number;
// check if the number is positive
if (number > 0) {
    // print "The number is positive" if the condition is true
    cout << "The number is positive" << endl;
} else {
    // check if the number is negative
    if (number < 0) {
        // print "The number is negative" if the condition is true
        cout << "The number is negative" << endl;
    } else {
        // print "The number is zero" if the condition is false
        cout << "The number is zero" << endl;
    }
}
```

If the user enters a positive number, such as 5, the condition `number > 0` will be true, and the program will print “The number is positive”. If the user enters a negative number, such as -3, the condition will be false, and the program will execute the `else` block. Inside the `else` block, there is another `if` statement that checks if the number is negative. The condition `number < 0` will be true, and the program will print “The number is negative”. If the user enters zero, both conditions will be false, and the program will execute the `else` block of the inner `if` statement and print “The number is zero”.

## The `else if` keyword

The `else if` keyword can be used to chain multiple conditions and execute different blocks of code depending on which one is true. The syntax of using `else if` with an `if` statement is:

```
if (condition1) {
    // code to execute if condition1 is true
} else if (condition2) {
    // code to execute if condition2 is true
} else if (condition3) {
    // code to execute if condition3 is true
```

```
} ...
else {
    // code to execute if none of the conditions are true
}
```

The `else if` keyword must be placed after an `if` statement or another `else if` statement. The code inside the curly braces `{}` after each `else if` keyword is called an `else if` block. Each `else if` block can contain one or more statements, or be empty.

The conditions are evaluated from top to bottom until one of them is true, and then the corresponding block is executed; otherwise, all blocks are skipped. The final `else` block is optional and provides a default block of code when none of the conditions are true.

For example, suppose we want to modify the previous program to also print “The number is even” or “The number is odd” depending on whether the number entered by the user is divisible by 2 or not. We can use an `else if` keyword like this:

```
// ask the user to enter a number
cout << "Enter a number: ";
// declare a variable to store the number
int number;
// read the number from standard input
cin >> number;
// check if the number is positive
if (number > 0) {
    // print "The number is positive" if the condition is true
    cout << "The number is positive" << endl;
} else if (number < 0) {
    // print "The number is negative" if the condition is true
    cout << "The number is negative" << endl;
} else {
    // print "The number is zero" if none of the above conditions are true
    cout << "The number is zero" << endl;
}

// check if the number is even
if (number % 2 == 0) {
    // print "The number is even" if the condition is true
    cout << "The number is even" << endl;
} else {
    // print "The number is odd" if the condition is false
    cout << "The number is odd" << endl;
}
```

If the user enters a positive even number, such as 4, both conditions `number > 0` and `number % 2 == 0` will be true, and the program will print “The number is positive”

and "The number is even