



# Conditional Statements, Loops, functions

CSCI-C212: Class 09-09-2024



# Conditional Statements



- Conditional statements control the flow of a program by executing certain code based on specific conditions.
- Types of conditional statement in C++:
  - if
  - If else
  - If else if
  - Switch
  - Conditional (Ternary) operator

# The **if** Statement

- The **if** statement executes a block of code if a specified condition is true.

- Syntanz:

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

- Example

```
int x = 10;  
if (x > 5) {  
    cout << "x is greater than 5";  
}
```

# The **else** Statements

□ The **else** statement allows executing code if the **if** condition is false.

□ Syntax:

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

□ Example:

```
int x = 3;  
if (x > 5) {  
    cout << "x is greater than 5";  
} else {  
    cout << "x is less than or equal to 5";  
}
```

# The **else if** Statement

- The **else if** statement tests multiple conditions in sequence

- Syntax:

```
if (condition1) {  
    // code if condition1 is true  
} else if (condition2) {  
    // code if condition2 is true  
} else {  
    // code if both conditions are false  
}
```

- Example

```
int x = 5;  
if (x > 10) {  
    cout << "x is greater than 10";  
} else if (x == 5) {  
    cout << "x is equal to 5";  
} else {  
    cout << "x is less than 5";  
}
```

# The **switch** Statement

□ The **switch** statement allows selecting one of many code blocks to execute

□ Syntax:

```
switch (variable) {  
    case constant1:  
        // code to execute if variable == constant1  
        break;  
    case constant2:  
        // code to execute if variable == constant2  
        break;  
    default:  
        // code to execute if none of the cases match  
}
```

□ Example:

```
int day = 3;  
switch (day) {  
    case 1:  
        cout << "Monday";  
        break;  
    case 2:  
        cout << "Tuesday";  
        break;  
    case 3:  
        cout << "Wednesday";  
        break;  
    default:  
        cout << "Invalid day";  
}
```

# Conditional (Ternary) Operator

- The conditional operator (?:) is a shorthand for the if-else statement.

- Syntax:

`condition ? expression_if_true : expression_if_false;`

- Example:

```
int age = 20;  
string result = (age >= 18) ? "Adult" : "Minor";  
cout << result; // Output: Adult
```





# Loops in C++

- Loops are used to repeatedly execute a block of code as long as a specified condition is true.
- Types of Loops in C++:
  - **for** loop
  - **while** loop
  - **do-while** loop



# The **for** Loop

- ▮ The **for** loop is used to iterate over a block of code a specific number of times. Usually, it is used when the number of iterations is known in advance.
- ▮ Syntax: 

```
for (initialization; condition; update) {  
    // code to be executed  
}
```
- ▮ Example: 

```
for (int i = 1; i <= 5; i++) {  
    cout << i << " ";  
}  
// Output: 1 2 3 4 5
```

# The **while** Loop

- ▮ The **while** loop executes a block of code as long as a condition is true. Usually, it is used when the number of iterations is not known in advance

- ▮ Syntax:

```
while (condition) {  
    // code to execute in the loop  
}
```

- ▮ Example

```
int i = 1;           // Output:  
while (i <= 5) {     // i = 1  
    cout << "i = " << i << endl; // i = 2  
    i++;             // i = 3  
}                   // i = 4  
                   // i = 5
```

# The **do-while** Loop

- The **do-while** loop is similar to the while loop, but it ensures that the loop body executes at least once before the condition is checked.

- Syntax:

```
do {  
    // code to be executed  
} while (condition);
```

- Example:

```
int i = 1;  
do {  
    cout << i << " ";  
    i++;  
} while (i <= 5);  
// Output: 1 2 3 4 5
```

```
while (not edge) {  
  run();  
}
```

```
do {  
  run();  
} while (not edge);
```



# Break and Continue Statements

- ▢ The **break** statement exits the loop immediately.
- ▢ The **continue** statement skips the rest of the current loop iteration and jumps to the next iteration.
- ▢ Example:

```
for (int i = 1; i <= 5; i++)  
{  
    if (i == 3) {  
        break;  
    }  
    cout << i << " ";  
}  
// Output: 1 2
```

```
for (int i = 1; i <= 5; i++) {  
    if (i == 3) {  
        continue;  
    }  
    cout << i << " ";  
}  
// Output: 1 2 4 5
```

# Function in C++

- Functions are blocks of code designed to perform specific tasks. They allow code reusability, structure, and modular programming.

- Benefits of Functions:


- Reusability
- Simplicity
- Maintainability

- Syntax:

```
return_type function_name(parameters) {  
    // Function body  
}
```

- Example:


```
int add(int a, int b) {  
    return a + b;  
}
```



# Function Return Types and C++ Data Types

- ▮ Possible Return Types:
  - ▮ int: Integer values
  - ▮ double: Floating-point values
  - ▮ char: Single character
  - ▮ bool: True/False
  - ▮ void: No return value





# Function Definition vs. Function Declaration

- Function Declaration (also known as the function prototype):

- It tells the compiler about a function's name, return type, and parameters.
- It allows the function to be called before it's defined.
- Example:

```
int add(int, int); // Function declaration
```

- Function Definition:

- It contains the actual code for the function, specifying what the function will do when called.
- Must match the declaration.
- Example:

```
int add(int a, int b) { // Function definition  
    return a + b;  
}
```



# Passing Parameters to Functions

- ▮ Functions accept inputs called parameters or arguments, which allow customization of the function's behavior.
- ▮ Example with Multiple Parameters:

```
int multiply(int x, int y) {  
    return x * y;  
}
```

# Two Ways of Parameter Passing

- We can pass parameter to a C++ function in the following ways:
  - Pass by Value
  - Pass by Reference
- Pass by Value:
  - When a parameter is passed by value, a copy of the actual value is passed to the function.
  - Changes made to the parameter inside the function do not affect the original value
  - Example:

```
#include <iostream>

void modifyValue(int num) {
    num = 100; // Changes only the local copy
}

int main() {
    int x = 50;
    modifyValue(x);
    std::cout << "Value of x: " << x << std::endl; // Output: 50
    return 0;
}
```

# Pass by Reference

- When a parameter is passed by reference, the function receives a reference to the original variable.
- Changes made to the parameter inside the function affect the original variable.

Example:

```
#include <iostream>

void modifyValue(int &num) { // Reference to the original variable
    num = 100; // Modifies the original variable
}

int main() {
    int x = 50;
    modifyValue(x);
    std::cout << "Value of x: " << x << std::endl; // Output: 100
    return 0;
}
```

- What will happen if you call the function as `modifyValue(50);`

# Swap Function

- Look at this version of swap function:

```
void swap(int a, int b){  
    int t;  
    t=a;  
    a=b;  
    b=t;  
}
```

- Call it from the main:

```
int main() {  
  
    int p=10, q=5;  
    swap(p,q);  
    std::cout << "Value of p and q: " << p<<"<< q<< std::endl;  
  
    return 0;  
}
```

# New Swap Function

- Look at the new version of swap function:

```
void swap(int &a, int &b){  
    int t;  
    t=a;  
    a=b;  
    b=t;  
}
```

- Call it from the main:

```
int main() {  
  
    int p=10, q=5;  
    swap(p,q);  
    std::cout << "Value of p and q: " << p<<"<< q<< std::endl;  
  
    return 0;  
}
```