Tishk International University
Faculty of Applied Science
Information Technology Department

Selection Control Structures

Programming I

Lecture 5

Fall 2025

Course Code: IT117

Grade 1

Islam Abdulazeez
islam.abdulaziz@tiu.edu.iq

January 4, 2026

Outlines

- ✓ Introduction to control structures
- ✓ Relational and logical operators
- ✓ if, if–else, and if–else if statements
- ✓ Nested if statements and flags
- ✓ Variable scope and blocks
- ✓ Switch statement

Learning Outcomes

- **At the end of today's session, you will be able to:**

  - ✓ Identify relational and logical operators used in decision making
  - ✓ Explain how selection control structures work in C++
  - ✓ Apply if, if–else, and switch statements to solve simple problems
  - ✓ Construct C++ programs using appropriate selection structures

Relational Operators

- Relational operators compare **numeric** and **char** values to check if one is greater than, less than, equal to, or not equal to another.
- Comparisons are essential for tasks like analyzing sales figures, calculating profit and loss, checking numerical ranges, and validating user input.

Relational Operators	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

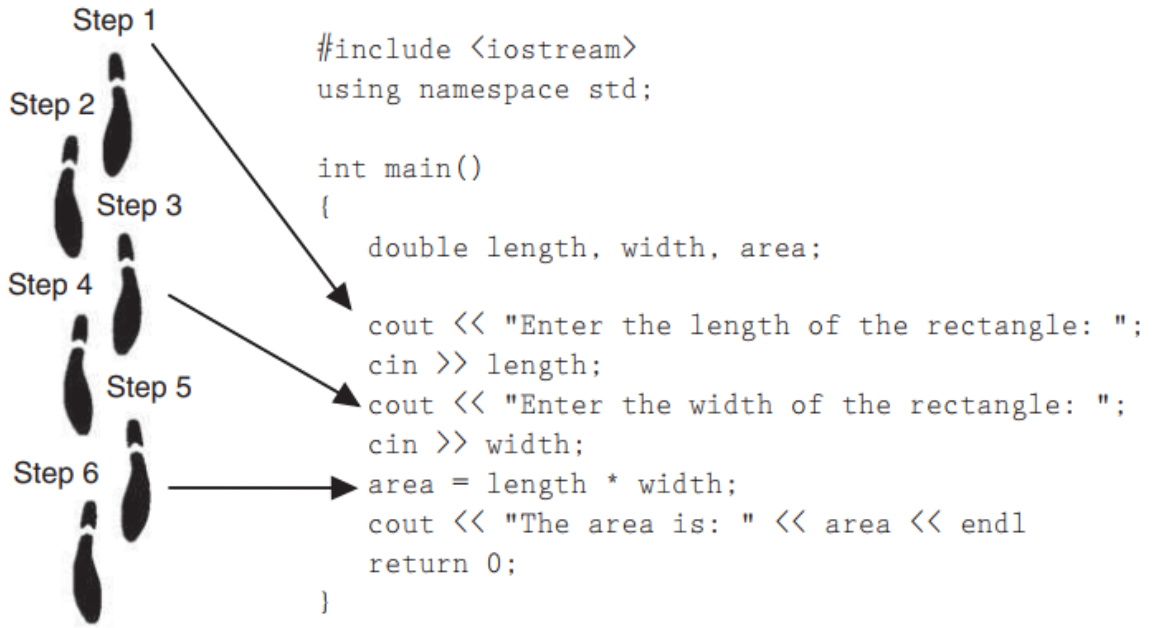
Relational Operators (Ex.)

- Assume **x** is **10** and **y** is **7**.

Expression	Value
`x < y`	false, because `x` is not less than `y`.
`x > y`	true, because `x` is greater than `y`.
`x >= y`	true, because `x` is greater than or equal to `y`.
`x <= y`	false, because `x` is not less than or equal to `y`.
`y != x`	true, because `y` is not equal to `x`.

Control Structures

- We know that program is executed sequentially, unless we give different instructions.
- For the program to not execute sequentially, we need to use a control structure.
- Control Structures provide two basic functions: **selection and repetition (looping)**



Control Structures

- A selection control structure is used to choose among alternative courses of action.
- There must be a condition that determines whether an action occurs.
- C++ provides several selection control structures.

✓ if

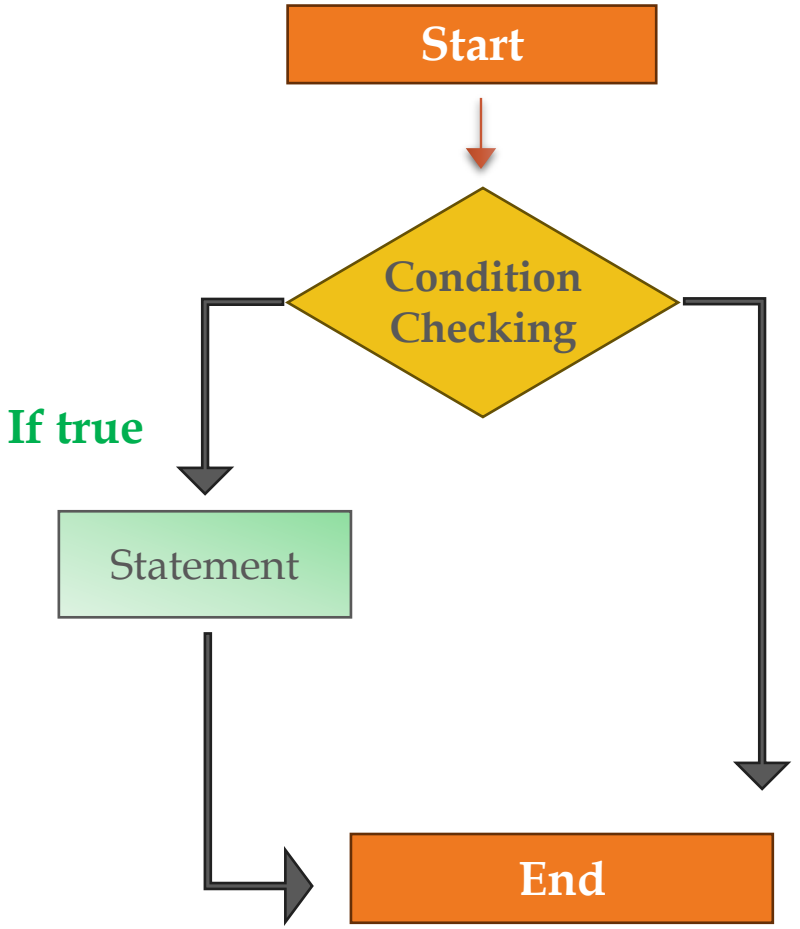
✓ If…else

✓ If…else if

✓ switch

if Statement

- The if statement can cause other statements to execute only under certain conditions.
- The if selection statement is a **single-selection statement**.
- It selects or ignores a single statement (or block of statements) depending on the condition
- Modifies the order of the statement execution.



if Statement

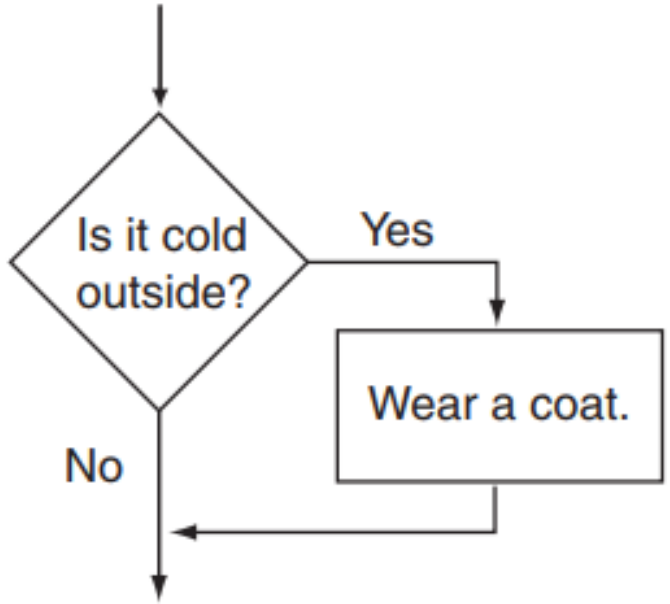
- In the flowchart, the action “Wear a coat” is performed only when it is cold outside. If it is not cold outside, the action is skipped. The action is conditionally executed because it is performed only when a certain condition (cold outside) exists.

- **Here are some other examples:**

  If the car is low on gas, stop at a service station and get gas.

  If it’s raining outside, go inside.

  If you’re hungry, get something to eat



if Statement (Ex.)

1. You will make people laugh if you are funny.

hypothesis conclusion

If you are funny then you will make people laugh.

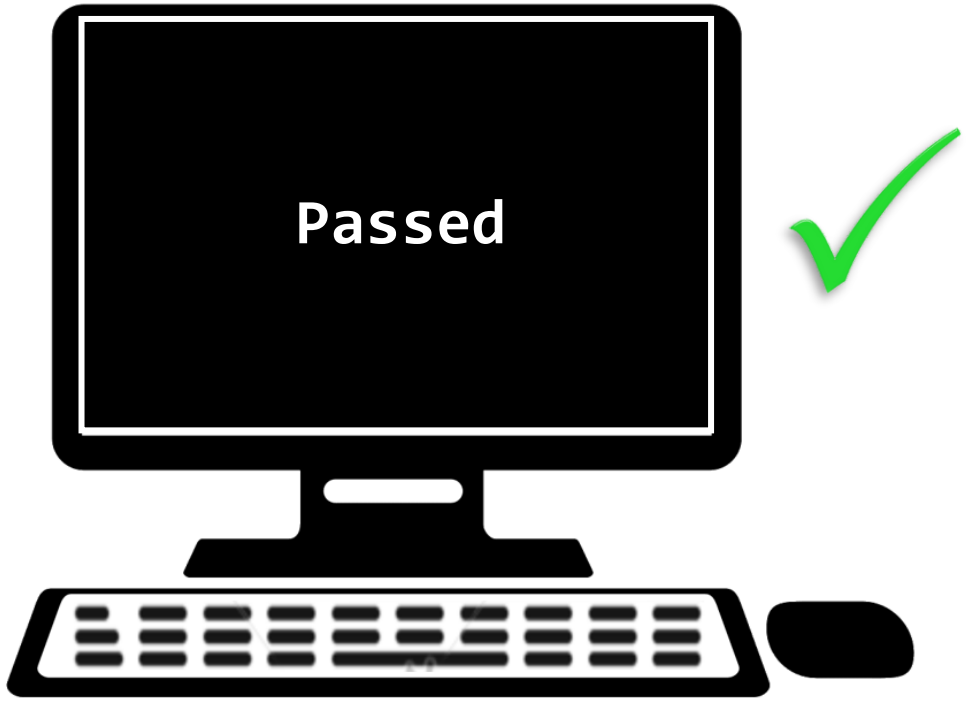
if Statement Syntax

- Evaluates a condition and controls program execution based on its result. If the condition is **true**, the statement executes, if **false**, it is skipped and execution continues with the next statement.

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

if Statement (Ex.)

```
if (92 >= 60) {
   cout<<"Passed";
 }
```



if Statement (Ex.)

```
int mark = 91 ;
if (mark <= 70) {
   cout<<"Passed";
 }
```

if Statement (Ex.)

Q. Write a C++ program that asks the user to input a number, then checks if the number is greater than 10.

```cpp
#include <iostream>
using namespace std;
int main() {

    int num;
    cout << "Enter a number: ";
    cin >> num;

    if (num > 10)
    {
        cout << "Number is greater than 10";
    }

    return 0;
}
```

if Statement (Ex.)

Q. If you want an if statement to execute a group of statements, use a compound statement enclosed in { and }. This allows you to control the execution of multiple statements or control structures.

```
#include <iostream>
using namespace std;
int main() {

    string userNumber = "user21";

    if (userNumber == "user21")
    {
        cout << "Welcome " << userNumber << endl;
        cout << "You have successfully logged in." << endl;
    }

    return 0;
}
```

if Statement (Ex.)

Q. Write a C++ program that asks the user to input a number, then checks if the number is greater than 10, less than 10, or equal to 10.

```cpp
#include <iostream>
using namespace std;
int main() {

    int num;
    cout << "Enter a number: ";
    cin >> num;

    if (num > 10)
    {
        cout << "Number is greater than 10";
    }

    if (num < 10)
    {
        cout << "Number is less than 10";
    }

    if (num == 10)
    {
        cout << "Number is equal to 10";
    }
    return 0;
}
```

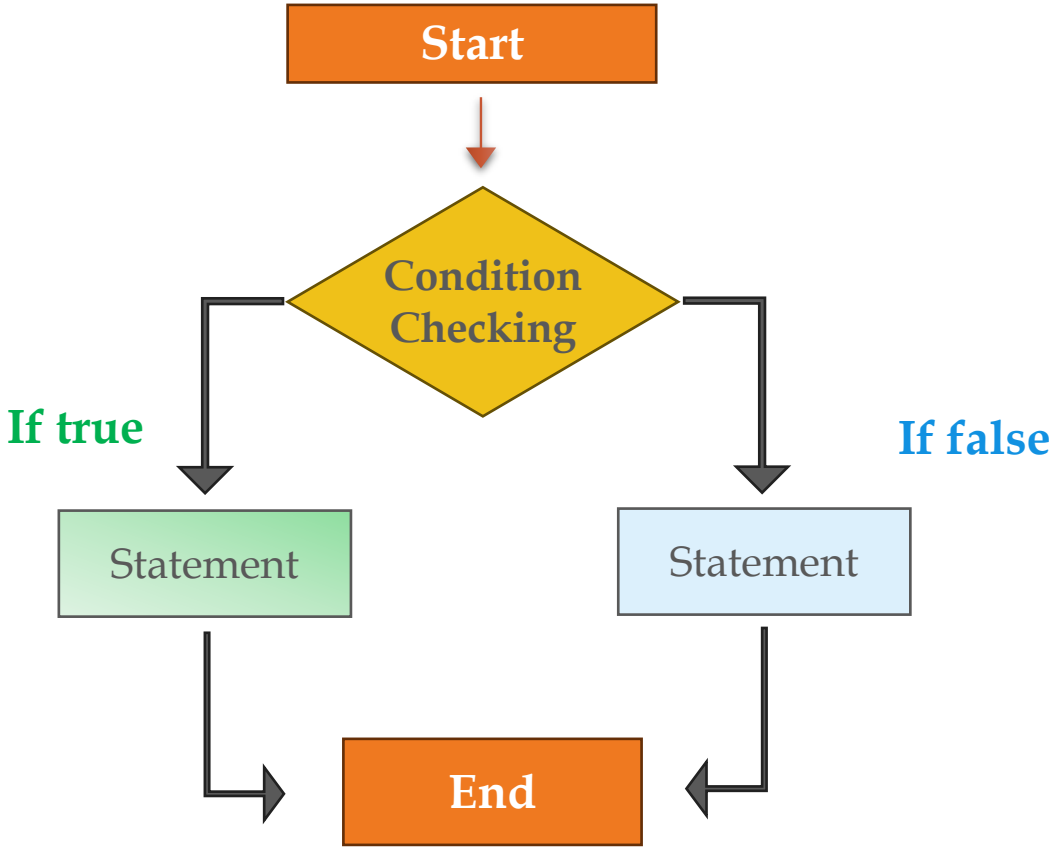
if Statement (Ex.)

Q. Write a C++ program to check whether an input number is even or odd.

```cpp
#include <iostream>
using namespace std;
int main() {

    int num;
    cout << "Enter a number: ";
    cin >> num;

    if (num % 2 == 0)
    {
        cout << "The number is even.";
    }

    if (num % 2 == 1)
    {
        cout << "The number is Odd.";
    }

    return 0;
}
```

if…else Statement

An **if…else** statement in C++ is a decision making statement used to execute one block of code if a condition is true and a different block of code if the condition is false.



if…else Statement Syntax

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

if…else Statement (Ex.)

Q. Write a C++ program that checks whether a number is even or odd using an if–else statement.

```
#include <iostream>
using namespace std;
int main() {

    int number = 7;

    if (number % 2 == 0) {
        cout << "The number is even.";
    }
    else {
        cout << "The number is odd.";
    }

    return 0;
}
```

if…else if Statement

- The if/else if statement tests a series of conditions. It is often simpler to use an if/else if statement than a set of nested if/else statements when checking multiple conditions.
- Use multiple if → when more than one condition may be true.
- Use if…else if → when only one condition should be true.

```
if (expression_1)
{
   statement
   statement                    } If expression_1 is true these state-
   etc.                           ments are executed, and the rest of the
}                                 structure is ignored.
else if (expression_2)
{
   statement
   statement                    } Otherwise, if expression_2 is true these
   etc.                           statements are executed, and the rest of the
}                                 structure is ignored.
Insert as many else if clauses as necessary
else
{
   statement
   statement                    } These statements are executed
   etc.                           if none of the expressions above
}                                 are true.
```

if...else if (Ex.)

```cpp
#include <iostream>
using namespace std;
int main() {

    int number;
    cout << "Enter a number: ";
    cin >> number;

    if (number > 0) {
        cout << number << " is positive.";
    }
    else if (number < 0) {
        cout << number << " is negative.";
    }
    else {
        cout << number << " is zero.";
    }

    return 0;
}
```

if...else if (Ex.)

```cpp
#include <iostream>
using namespace std;
int main() {

    int score;
    cout << "Enter your score: ";
    cin >> score;

    if (score >= 90) {
        cout << "Your grade is A\n";
    }
    else if (score >= 80) {
        cout << "Your grade is B\n";
    }
    else if (score >= 70) {
        cout << "Your grade is C\n";
    }
    else {
        cout << "Your grade is D";
    }

    return 0;
}
```

Comparison

```cpp
#include <iostream>
using namespace std;
int main() {

    int score = 85;

    if (score >= 50) {
        cout << "Passed" << endl;
    }

    if (score >= 70) {
        cout << "Good" << endl;
    }

    if (score >= 90) {
        cout << "Excellent" << endl;
    }

    return 0;
}
```

Output | Passed
Good ✓

```cpp
#include <iostream>
using namespace std;
int main() {

    int score = 85;

    if (score >= 50) {
        cout << "Passed" << endl;
    }
    else if (score >= 70) {
        cout << "Good" << endl;
    }
    else if (score >= 90) {
        cout << "Excellent" << endl;
    }

    return 0;
}
```

Output | Passed ✓

Nested if Statement

- A **nested if statement** is an if statement placed inside another if statement.

  - Syntax

```
if (condition1) {
    if (condition2) {
        }
  }
```

Nested if Statement (Ex.)

Q. Write a C++ program that checks if a number is greater than 10 and then checks if it is less than 20.

```
#include <iostream>
using namespace std;
int main() {

    int number = 15;

    if(number > 10) {
        if (number < 20) {
            cout << "The number is between 10 and 20." << endl;
        }
        cout << "The number is greater than 10.";
    }
    return 0;
}
```

Output

```
The number is between 10 and 20.
The number is greater than 10.
```

Nested if Statement (Ex.)

```
#include <iostream>
using namespace std;
int main() {

    int number = 25;

    if(number > 10) {
        if (number < 20) {
            cout << "The number is between 10 and 20." << endl;
        }
        cout << "The number is greater than 10.";
    }
    return 0;
}
```

?

Output | The number is greater than 10.

Nested if Statement (Ex.)

```
string username, password;
cout << "Enter your username: ";
cin >> username;    //Admin
cout << "Enter your password: ";
cin >> password;    //pass@122

if (username == "Admin") {
    if (password == "pass@123") {
        cout << "Logged in successfully.";
    }
    else {
        cout << "Invalid password.";
    }
}
else {
    cout << "Invalid username.";
}
```

Nested if Statement (Ex.)

```
string username, password;
cout << "Enter your username: ";
cin >> username;   //admin
cout << "Enter your password: ";
cin >> password;   //pass@123

if (username == "Admin") {
    if (password == "pass@123") {
        cout << "Logged in successfully.";
    }
    else {
        cout << "Invalid password.";
    }
}
else {
    cout << "Invalid username.";
}
```

Logical Operators

- Logical operators connect two or more relational expressions into one or reverse the logic of an expression.

Operator	Meaning	Effect
&&	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
||	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
!	NOT	Reverses the "truth" of an expression. It makes a true expression false, and a false expression true.

AND (&&) Operator

- AND (&&) → true if all conditions are true.

Expression	Value of the Expression
`false && false`	`false (0)`
`false && true`	`false (0)`
`true && false`	`false (0)`
`true && true`	`true (1)`

AND (&&) Operator (Ex.)

```cpp
#include <iostream>
using namespace std;
int main() {

    string username, password;
    cout << "Enter your username: ";
    cin >> username;
    cout << "Enter your password: ";
    cin >> password;

    if (username == "Admin" && password == "pass@123") {
        cout << "Logged in successfully.";
        } else {
            cout << "Invalid username or password.";
        }

    return 0;
}
```

AND (&&) Operator (Ex.)

- Create a C++ program that determines whether a person is eligible to vote. The program should check if the person is 18 years or older and is a citizen.

```
int age;
char citizenship;

cout << "Enter your age: ";
cin >> age;

cout << "Are you a citizen? (Y/N): ";
cin >> citizenship;

if (age >= 18 && citizenship == 'Y') {
    cout << "You are eligible to vote!";
}
else {
    cout << "Sorry, you are not eligible to vote.";
}
```

OR (||) Operator

- OR (||) → true if at least one condition is true.

Expression	Value of the Expression
`false || false`	`false (0)`
`false || true`	`true (1)`
`true || false`	`true (1)`
`true || true`	`true (1)`

OR (||) Operator (Ex.)

```cpp
#include <iostream>
using namespace std;
int main() {

    int temperature = -5;
    string weather = "Cold";

    if (temperature > 0 || weather == "Cold") {
        cout << "The weather is cold.";
    }

    return 0;
}
```

OR (||) & AND (&&) Operators (Ex.)

- Write a C++ program that takes two numbers as input and checks whether both numbers are positive, at least one number is positive, or both numbers are non-positive.

```
int number1, number2;

cout << "Enter two numbers: ";
cin >> number1 >> number2;

if (number1 > 0 && number2 > 0) {
    cout << "Both numbers are positive." << endl;
}
else if (number1 > 0 || number2 > 0) {
    cout << "At least one of the numbers is positive." << endl;
}
else {
    cout << "Both numbers are non-positive." << endl;
}
```

NOT (!) Operator

- NOT (!) → reverses the condition

Expression	Value of the Expression
`!false`	`true (1)`
`!true`	`false (0)`

NOT (!) Operator (Ex.)

```cpp
#include <iostream>
using namespace std;
int main() {

    int balance = 0;
    string membership = "expired";

    if (!(balance > 0)) {
        cout << "Insufficient balance.\n";
    }

    if (!(membership == "active")) {
        cout << "Membership is not active.\n";
    }

    return 0;
}
```

Flags

- A flag in C++ is a variable that shows whether something has happened or not.
- Flags are usually Boolean or integer variables.

```cpp
int number;
bool flag = false;

cout << "Enter a number: ";
cin >> number;

if (number >= 0) {
    flag = true;
}

if (flag) {
    cout << "The number is positive or zero.";
}
else {
    cout << "The number is negative.";
}
```

```cpp
int number;
int flag = 0;

cout << "Enter a number: ";
cin >> number;

if (number >= 0) {
    flag = 1;
}

if (flag == 1) {
    cout << "The number is positive or zero.";
}
else {
    cout << "The number is negative.";
}
```

Comparing Characters

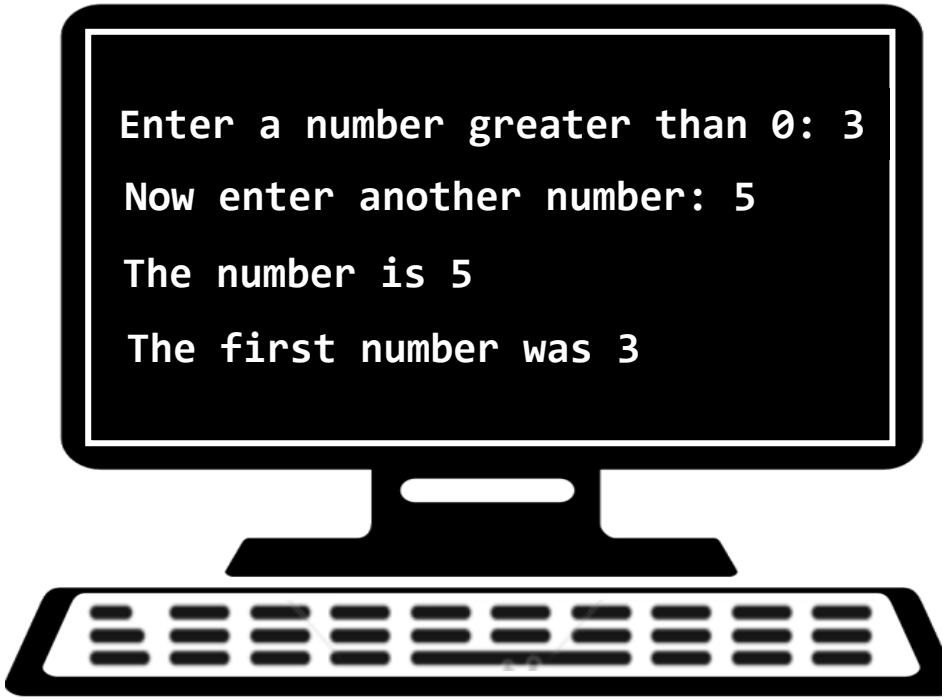
```
char ch;

cout << "Enter a digit or a letter: ";
cin >> ch;

if (ch >= '0' && ch <= '9') {
    cout << "You entered a digit.\n";
}
else if (ch >= 'A' && ch <= 'Z') {
    cout << "You entered an uppercase letter.\n";
}
else if (ch >= 'a' && ch <= 'z') {
    cout << "You entered a lowercase letter.\n";
}
else {
    cout << "That is not a letter or a digit.\n";
}
```

Character	ASCII Value
'0'–'9'	48–57
'A'–'Z'	65–90
'a'–'z'	97–122
blank	32
period	46

Blocks and Variable Scope

- The scope of a variable is limited to the block in which it is defined. C++ allows you to create variables almost anywhere in a program.

```cpp
#include <iostream>
using namespace std;
int main() {

    int number;
    cout << "Enter a number greater than 0: ";
    cin >> number;

    if (number > 0) {
        int number; // Another variable named number.
        cout << "Now enter another number: ";
        cin >> number;
        cout << "The number is " << number << endl;
    }
    cout << "The first number was " << number << endl;

    return 0;
}
```



Blocks and Variable Scope (Ex.)

```cpp
#include <iostream>
using namespace std;
int main() {

    int x = 10;   // x is declared in main(), so it is visible everywhere inside main

    if (x > 5) {
        int y = 20;   // y is declared inside this block
        cout << "x = " << x << endl; // valid
        cout << "y = " << y << endl; // valid
    }

    cout << "x = " << x << endl; // valid
    cout << "y = " << y << endl; // ERROR: y is out of scope here

    return 0;
}
```

Switch Statement

- A **switch** statement in C++ is a control structure used to execute one block of code from many possible options, based on the value of a single variable or expression.

- Works with **int**, **char**, and **enum** (not strings in basic C++).

```
switch (expression) {
   case value1:
      // code to execute
      break;
   case value2:
      // code to execute
      break;
   default:
      // code if no case matches
}
```

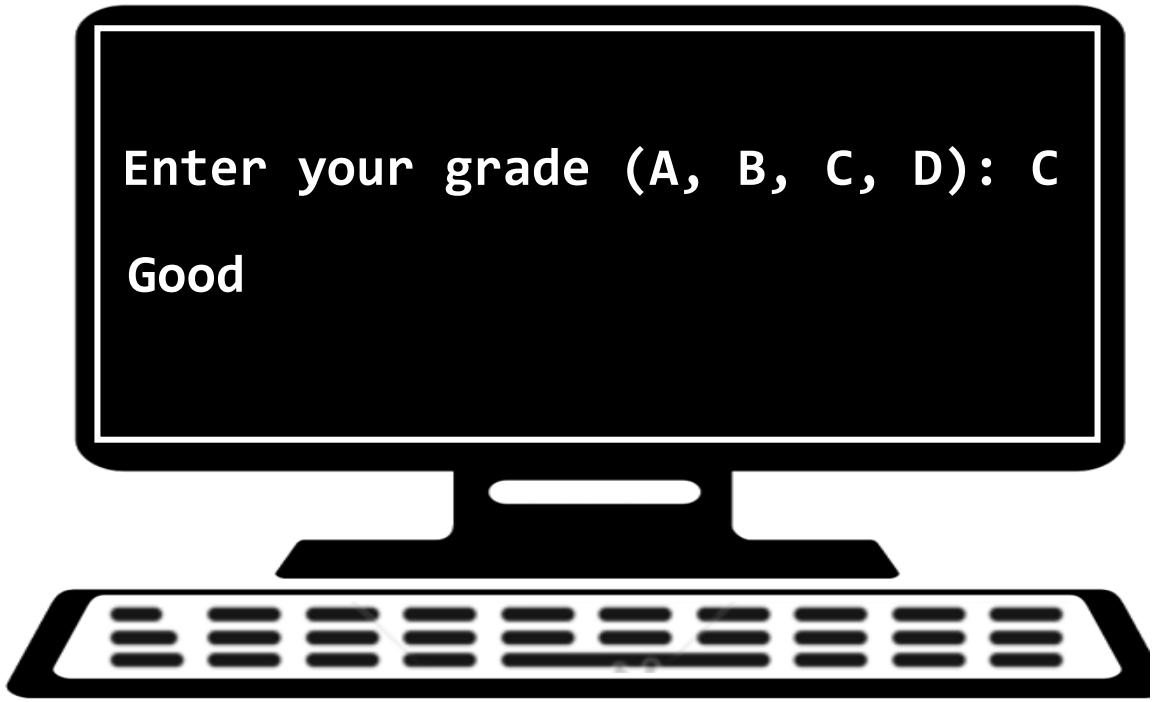
Switch Statement (Ex.)

```cpp
int day = 3;

if (day == 2) {
    cout << "Monday";
}
else if (day == 3) {
    cout << "Tuesday";
}
else if (day == 4) {
    cout << "Wednesday";
}
else {
    cout << "Invalid day";
}
```

```cpp
int day = 3;

switch (day) {
case 2:
    cout << "Monday";
    break;
case 3:
    cout << "Tuesday";
    break;
case 4:
    cout << "Wednesday";
    break;
default:
    cout << "Invalid day";
}
```

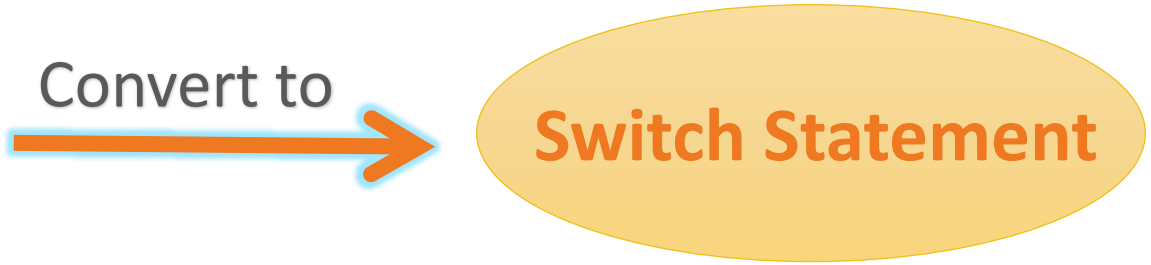
Switch Statement (Ex.)

```
char grade;
cout << "Enter your grade (A, B, C, D): ";
cin >> grade;

switch (grade) {
case 'A':
    cout << "Excellent";
    break;
case 'B':
    cout << "Very Good";
    break;
case 'C':
    cout << "Good";
    break;
case 'D':
    cout << "Pass";
    break;
default:
    cout << "Invalid grade";
}
```



Switch Statement (Homework)

```
char op;
double num1, num2;

cout << "Enter first number: ";
cin >> num1;

cout << "Enter an operator (+, -): ";
cin >> op;

cout << "Enter second number: ";
cin >> num2;

if (op == '+') {
    cout << "Result: " << num1 + num2 << endl;
}
else if (op == '-') {
    cout << "Result: " << num1 - num2 << endl;
}
else {
    cout << "Invalid operator!" << endl;
}
```



Activities and Next Lecture's Topic

Activities

- Review this lecture note
- Practice

Next Lecture's Topic

- Iteration Control Structures

References

- Gaddis, T. (2014). Starting out with C++: Early objects (7th ed.). Pearson Education.

Thank You!