(PROGRAMMING FUNDAMENTALS)

CLASS: BSCS 1ST(MB)

SUBMITTED TO: MAM SADIA

SUBMITTED BY: MAHNOOR SAQIB

ROLL NO. : 2599

(NOTES)

Software:

Software is a set of instructions that tells a computer how to work. These instructions are written in programming languages and executed by the computer's hardware.It is also called **program.**Programs are written in different programming languages.

Programming languages:

A programming language is a formal system used to write instructions that a computer can understand and execute. These languages enable developers to create software, applications, websites, and systems by translating human logic into machine-readable code.

Programming in C++:

To write a program in C++, you need to write code using the correct syntax, save it with a `.cpp` **extension, and compile it using a C++ compiler like GCC or an IDE like Code::Blocks or Visual Studio.**C++ program consists of two parts.First part consists of **library/header files** and second part consists of **main function**.

Use of cout:

In C++, `cout` **is used to display output on the screen. It stands for "character output" and is part of the standard library's** `iostream` **header.**

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!";
    return 0;
}
```

- `#include <iostream>`: Includes the input/output stream library.
- `using namespace std;`: Lets you use standard functions like `cout` without prefixing them.
- `int main()`: The main function where execution starts.
- `cout << "Hello, World!";`: Prints text to the screen.
- `return 0;`: Ends the program successfully.

Alphabets,Symbols and Numbers:

- **Alphabets and Symbols:**
  Alphabets and symbols should be written in double quotes**(“ ”).**
- **Numbers:**
  Numbers should be written without double quotes.If they are written in double quotes,they will be considered as text.
  i.e **cout<< 1+2;** output will be **3.**
  i.e **cout<< “1+2”;** output will be **1+2.**

Relational Operators:

Relational operators are fundamental tools in programming languages that **test or define relationships between two operands**. These relationships include equality, inequality, and comparisons like greater than or less than. The result of a relational operation is usually a Boolean value — either `true` or `false` — which is often used to control the flow of a program (e.g., in `if` statements or loops).

Common Relational Operators

Operator	Meaning	Example	Result
`==`	Equal to	`5 == 5`	`true`
`!=`	Not equal to	`5 != 3`	`true`
`>`	Greater than	`7 > 4`	`true`
`<`	Less than	`3 < 8`	`true`
`>=`	Greater than or equal	`6 >= 6`	`true`
`<=`	Less than or equal	`2 <= 5`	`true`

These operators are used across many programming languages like Python, Java, C++, and JavaScript, though syntax may vary slightly.

Arithmetic Operators:

Arithmetic operators are essential tools in programming languages that allow developers to manipulate numbers. These operators work on operands (values or variables) and return a result based on the specified mathematical operation. They're used in everything from simple calculations to complex algorithms.

🔢 Common Arithmetic Operators

Operator	Name	Example	Result
`+`	Addition	`5 + 3`	`8`
`-`	Subtraction	`9 - 4`	`5`
`*`	Multiplication	`6 * 2`	`12`
`/`	Division	`10 / 2`	`5.0`
`%`	Modulus	`7 % 3`	`1`
`**`	Exponentiation	`2 ** 3`	`8`
`++`	Increment	`x++`	`x = x + 1`
`--`	Decrement	`x--`	`x = x - 1`

Escape Sequence:

Escape sequences allow programmers to include **control characters** or **special symbols** in strings. These sequences are interpreted by the compiler or interpreter to perform specific actions like moving to a new line, inserting a tab space, or printing a quote character.

They are widely used in languages like C, C++, Java, Python, and JavaScript.

📜 Common Escape Sequences

Escape Sequence	Meaning	Example Usage	Output Example
`\n`	New line	`"Hello\nWorld"`	Hello World
`\t`	Horizontal tab	`"Hello\tWorld"`	Hello World
`\\`	Backslash	`"C:\\Program Files"`	C:\Program Files

`\'`	Single quote	`"It\'s fine"`	It's fine
`\"`	Double quote	`"She said \"Hi\""`	She said "Hi"
`\b`	Backspace	`"abc\b"`	ab
`\r`	Carriage return	`"Hello\rWorld"`	World
`\f`	Form feed	`"Page1\fPage2"`	Page1 (form feed) Page2
`\a`	Audible bell (alert)	`"\a"`	(Triggers a beep sound)

Variable:

A variable acts like a container in memory that stores a value — such as a number, text, or more complex data — which can be accessed and modified throughout a program. It has a name (called an identifier),data type **and a value, and it's one of the most fundamental concepts in programming.It stores value temporarily. One variable stores a single value at a time.**

✅ Rules for Writing Variables in C

Here are the essential rules you must follow when naming and declaring variables in C:

1. Start with a Letter or Underscore

- Variable names must begin with an **alphabet (A–Z, a–z)** or an **underscore (_)**.
- Examples: `total`, `_value`, `score1`

2. Use Only Letters, Digits, and Underscores

- After the first character, you can use **letters**, **digits (0–9)**, and **underscores**.
- No special characters like `@`, `#`, `$`, or spaces are allowed.
- Valid: `marks_2025`, `studentID`
- Invalid: `marks@2025`, `student ID`

3. Avoid Reserved Keywords

- You cannot use **C reserved keywords** (like `int`, `return`, `while`, `for`) as variable names.
- Example: `int return = 5;` ❌ (invalid)

4. Case Sensitivity

- C is **case-sensitive**, so `Total`, `total`, and `TOTAL` are treated as **three different variables**.

5. No White Spaces

- Variable names **cannot contain spaces**.
- Example: `student name` ✗ (invalid), use `student_name` ✓

6. Length Limitations

- According to the ANSI C standard, a variable name can be up to **31 characters long**.
- However, some older compilers may only recognize the **first 8 characters** as significant.

7. Meaningful Names

- While not a strict rule, it's good practice to use **descriptive names** that reflect the purpose of the variable.
- Example: Use `averageScore` instead of x

Variable Initiolization:

Variable initialization is the process of assigning an initial value to a variable at the time it is declared. This ensures the variable starts with a known, predictable value instead of containing random or "garbage" data.

Variable Declaration:

Variable declaration is the process of specifying a variable's name and data type so that the compiler can allocate memory for it. It tells the program what kind of data the variable will hold, but does not necessarily assign it a value.

Syntax:

```
data_type variable_name;
```

- You can also declare multiple variables of the same type in one line:

```
int a, b, c;
float temperature, pressure;
```

🧰 Declaration vs Initialization

Concept	Description	Example
Declaration	Introduces the variable and its type	`int x;`
Initialization	Assigns an initial value to the variable	`int x = 10;`

Use of cin:

In C++, `cin` is used to receive input from the user via the keyboard. It stands for "character input" and is part of the `iostream` library.

Operator Precedence:

In programming, **operator precedence** (also called operator hierarchy) is a set of rules that define the sequence in which operations are performed in expressions that contain multiple operators. This concept is crucial for writing correct and predictable code, especially when parentheses are not used.

Operator Precedence Table (C Language)

Below is a simplified hierarchy of common operators in C, from highest

Precedence Level	Operators	Associativity
1 (Highest)	`()` `[]` `.` `->` `++` `--`	Left to Right
2	`+` `-` `!` `~` `++` `--` `*` `&`	Right to Left
3	`*` `/` `%`	Left to Right
4	`+` `-`	Left to Right
5	`<` `<=` `>` `>=`	Left to Right
6	`==` `!=`	Left to Right

7	`&`	Left to Right
8	`^`	Left to Right
9	`	`
10	`&&`	Left to Right
11	`	
12	`?:`	Right to Left
13	`=` `+=` `-=` `*=` `/=` `%=` etc.	Right to Left
14 (Lowest)	`,`	Left to Right

if statement:

An **if statement** is a basic programming construct used to make decisions in code. It lets a program run certain instructions **only when a condition is true**.

General Structure:

```
if (condition) {
    // code runs only if condition is true
}
```

if else Structure in C:

```
if (condition) {

   // code runs if condition is true

} else {

   // code runs if condition is false

}
```

if else if Structure in C :

```
if (condition1) {

   // runs if condition1 is true

}

else if (condition2) {

   // runs if condition2 is true

}

else if (condition3) {

   // runs if condition3 is true

}

else {

   // runs if none of the above conditions are true

}
```

Logical Operators:

Logical operators in C are used to combine or compare conditions inside **if, else-if, while, for**, etc.
They return either **true (1)** or **false (0)**.

✅ Logical Operators in C

Operator	Meaning	Example	True When...
`&&`	Logical AND	`a > 0 && b > 0`	**Both** conditions are true
`		`	Logical OR
`!`	Logical NOT	`!flag`	The condition is **false**

1. Logical AND (&&)

```
if (age >= 18 && age <= 60) {
    cout <<"Eligible\n";
}
```

✔ Runs only if **both** conditions are true.

2. Logical OR (||)

```
if (temp < 0 || temp > 40) {
    cout<<"Extreme weather\n";
}
```

✔ Runs if **either** condition is true.

3. Logical NOT (!)

```
if (!loggedIn) {
    printf("Please log in\n");
}
```

✔ Reverses the condition.

Difference between Logical error and Syntax error:

Syntax error	Logical error
Mistakes in code structure	Mistakes in program logic
Compiler detects it	Compiler **cannot** detect it
Program fails to compile	Program runs but gives wrong output
Easy to find (compiler shows location)	Hard to find (requires testing/debugging)

Loop :

Loops in C allow you to **repeat a block of code multiple times**.
There are **three main types of loops** in C:

1. `for` loop:

Used when you know **exactly how many times** you want to repeat something.

Syntax:

```
for (initialization; condition; increment/decrement)
{
    // code to repeat
}
```

2. `while` loop:

Used when the number of repetitions is **not known**, but depends on a condition.

Syntax:

Initialization;

```
while (condition)
{
    // code to repeat;
Increment/decrement;
}
```

3. `do while` loop:

Similar to `while`, but **executes at least once** even if the condition is false.

Syntax:

Initialization;

```
do {
    // code to repeat
Increment/decrement;
}
 while (condition);
```

4. nested loop

A **nested loop** in C++ is a loop placed **inside another loop**. The inner loop runs completely for **each single iteration** of the outer loop. Nested loops are commonly used when a program needs to work with **tables, matrices, patterns, or repeated comparisons**.

How it works

- The **outer loop** controls how many times the inner loop will repeat.
- For every one iteration of the outer loop, the **inner loop starts from the beginning** and executes fully.
- This continues until the outer loop finishes all its iterations.

Syntax

```
for (initialization; condition; update) {

    for (initialization; condition; update) {

        // statements

    }

}
```

Example

```
#include <iostream>

using namespace std;

int main() {
```

```
    for (int i = 1; i <= 3; i++) {        // outer loop
        for (int j = 1; j <= 4; j++) {    // inner loop
            cout << j << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output

```
1 2 3 4

1 2 3 4

1 2 3 4
```

Uses of Nested Loops

- Printing patterns (stars, numbers)
- Working with 2D arrays or matrices
- Comparing elements in sorting algorithms
- Generating tables

(ARRAY)

In C++, **arrays** are used to store multiple values of the same data type under a single variable name. Based on structure, arrays are mainly classified into **one-dimensional** and **two-dimensional** arrays.

1. One-Dimensional Array (1D Array)

A **one-dimensional array** stores a list of elements in a single row. It is useful when dealing with a simple sequence of related data, such as marks of students or temperatures.

Syntax

```
data_type array_name[size];
```

Example

```
#include <iostream>

using namespace std;

int main() {

    int marks[5] = {85, 90, 78, 88, 92};


    for (int i = 0; i < 5; i++) {

        cout << marks[i] << " ";

    }

    return 0;

}
```

Uses

- Storing lists of data
- Searching and sorting
- Storing student records

2. Two-Dimensional Array (2D Array)

A **two-dimensional array** stores data in the form of **rows and columns**, like a table or matrix.

Syntax

```
data_type array_name[rows][columns];
```

Example

```
#include <iostream>

using namespace std;

int main() {
```

```
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Uses

- Representing matrices
- Storing tabular data
- Image processing
- Games and simulations

Difference Between 1D and 2D Arrays

Feature	1D Array	2D Array
Structure	Single row	Rows and columns
Index	One index	Two indices
Complexity	Simple	More complex
Example	List of numbers	Matrix or table

Feature	1D Array	2D Array

(STRUCTURE)

In **C++**, a **structure** (struct) is a user-defined data type that allows you to **group different types of variables** under one name. Structures are used when you want to represent a real-world entity that has multiple properties, such as a student, employee, or book.

Definition of Structure

A structure is defined using the `struct` keyword.

Syntax

```
struct structure_name {
   data_type member1;
   data_type member2;
   data_type member3;};
```

Example of Structure

```
#include <iostream>using namespace std;

struct Student {
    int rollNo;
    string name;
    float marks;
};
```

```
int main() {
    Student s1;
   s1.rollNo = 101;
    s1.name = "Ali";
    s1.marks = 85.5;
    cout << "Roll No: " << s1.rollNo << endl;
    cout << "Name: " << s1.name << endl;
    cout << "Marks: " << s1.marks << endl;
    return 0;
  }
```

Key Features of Structure

- Can store **different data types** together.
- Helps organize complex data.
- Improves code readability and maintainability.
- Structure variables can be passed to functions.

Array of Structures

Structures can also be used in arrays.

```
Student s[3];
```

Uses of Structure

- Storing student or employee records
- Database management
- File handling
- Representing real-world entities

(Continue and Break)

In **C++**, the `break` and `continue` statements are used to control the flow of loops. They help change the normal execution of a loop based on certain conditions.

---

1. Use of `break` Statement

The `break` statement is used to **immediately terminate** a loop or switch statement when a specific condition is met. After `break`, the control moves outside the loop.

Example

```
#include <iostream>using namespace std;
int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 5) {
            break;   // exits the loop
        }
        cout << i << " ";
    }
    return 0;
}
```

Output

```
1 2 3 4
```

Use of `break`

- To stop a loop early
- To exit a loop when a condition is satisfied
- Commonly used in searching and menu-driven programs

---

2. Use of `continue` Statement

The `continue` statement is used to **skip the current iteration** of a loop and move to the next iteration without executing the remaining code in the loop body.

Example

```
#include <iostream>using namespace std;
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue;   // skips this iteration
        }
        cout << i << " ";
    }
    return 0;
}
```

Output

```
1 2 4 5
```

Use of `continue`

- To skip unwanted values
- To avoid executing certain code under specific conditions
- Useful in filtering data

Difference Between `break` and `continue`

Feature	break	continue
Function	Terminates the loop	Skips current iteration
Loop execution	Stops completely	Continues with next iteration
Control flow	Moves outside the loop	Moves to loop condition

Conclusion

The `break` statement is used to stop a loop entirely, while the `continue` statement is used to skip a particular iteration. Both are useful for controlling loop execution in C++ programs.

(Comments in C++)

In **C++**, **comments** are used to explain the code and make it easier to understand. They are not executed by the compiler and do not affect the program's output. Comments are especially helpful for students, programmers, and anyone who reads or maintains the code later.

Uses of Comments in C++

Improve Code Readability
Comments help explain what a program or a specific part of code does, making it easier to understand.

Documentation
They are used to describe the purpose of variables, functions, classes, and logic in the program.

Easy Maintenance

Comments help programmers understand the code when updating or debugging it in the future.

Debugging

Parts of code can be temporarily commented out to test or debug a program.

Team Communication
In group projects, comments help team members understand each other's work.

Types of Comments in C++

1. Single-line Comment

Used for short explanations.

```
// This is a single-line comment
```

2. Multi-line Comment

Used for longer explanations.

```
/*
This is a multi-line comment
It can span multiple lines
*/
```

Example

```
#include <iostream>using namespace std;
int main() {
    // This program prints a message
    cout << "Hello, World!" << endl;  // Output statement
    return 0;
}
```

Conclusion

Comments play an important role in C++ programming by improving clarity, understanding, and maintainability of code. Writing meaningful comments is considered a good programming practice.

(Switch statement)

In **C++**, the `switch` **statement** is a control structure used to execute **one block of code from multiple options** based on the value of a variable or expression. It is an alternative to using multiple `if-else` statements when checking a single variable against many constant values.

Syntax of Switch Statement

```
switch (expression) {
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    case value3:
        // statements
        break;
    default:
        // statements
}
```

Example

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

```
    cout << "Enter a number (1-3): ";
    cin >> choice;

    switch (choice) {
        case 1:
            cout << "You selected option 1";
            break;
        case 2:
            cout << "You selected option 2";
            break;
        case 3:
            cout << "You selected option 3";
            break;
        default:
            cout << "Invalid choice";
    }
    return 0;
}
```

Important Rules

- Case values must be **constant expressions** (integers or characters).
- Duplicate case values are not allowed.
- The `break` statement is optional, but without it, **fall-through** occurs.
- `switch` does not work with floating-point numbers or strings (in basic C++).

Use of Switch Statement

- Menu-driven programs
- Simple decision-making
- Replacing long if-else chains

Conclusion

The `switch` statement makes programs cleaner and easier to understand when dealing with multiple choices based on a single variable. It improves readability and efficiency in decision-making logic.