# Lab Manual: Programming Fundamentals in C++

## 1. What is Programming & Programming Language?

- **Definition**: Programming is the process of creating instructions for computers to perform specific tasks. A programming language is the medium we use to write these instructions.
- **Real-Life Example**: Think of a recipe. Each step is like a programming instruction guiding the computer.
- **Common Languages**: Python, Java, C++, etc.

---

## 2. What is Programming Fundamentals?

- **Definition**: Basic principles and concepts that form the foundation for writing programs, such as variables, control structures, data types, and functions.
- **Purpose**: Helps beginners understand core elements needed to write basic programs in any language.

---

## 3. What is an IDE?

- **Definition**: Integrated Development Environment (IDE) is software that provides tools for writing, testing, and debugging code.
- **Examples**: Visual Studio Code, Code::Blocks, Dev C++.

---

## 4. What is C++?

- **Definition**: C++ is a high-performance, general-purpose programming language often used for systems programming and software development.
- **Real-Life Usage**: Games, operating systems, and large-scale applications.

---

## 5. Basic Structure of a C++ Program

- **Explanation**: Understanding the basic structure is crucial to writing any C++ program.
- **Structure Overview**:

```
#include <iostream>   // Library inclusion for input/output
using namespace std;  // Using standard namespace

int main() {          // Main function - entry point of the program
   // Code goes here
   return 0;          // Indicate program ended successfully
}
```

- **Explanation of Each Part**:
  - `#include <iostream>`: Includes libraries necessary for input/output.
  - `using namespace std;`: Allows using standard functions without prefixing `std::`.
  - `int main() {}`: Main function where the program starts executing.
  - `return 0;`: Signifies successful completion.

---

# 6. What are Variables?

- **Definition**: Variables are named locations in memory that store data values.
- **Naming Conventions**: Use descriptive names, start with a letter, avoid special characters, and follow camelCase.
- **Example**: `int age = 25;` stores the age of a person.

---

# 7. Print Statement

- **Definition**: Used to display output on the screen.
- **Syntax**: `cout << "Your text";`
- **Example**: `cout << "Hello, World!";`

---

# 8. Data Types

- **Definition**: Specify the type of data a variable can hold.
- **Common Types**:
  - `int` (integer)
  - `float` (floating-point number)
  - `char` (character)
  - `bool` (boolean)
- **Example**: `int age = 18;`

---

# 9. User Input

- **Definition**: Allows users to provide data to the program.
- **Syntax**: `cin >> variable;`
- **Example**: `cin >> age;` takes input for the `age` variable.

---

# 10. Operators

- **Definition**: Symbols used to perform operations on variables and values.
- **Types**:
  - **Arithmetic**: `+, -, *, /, %`
  - **Relational**: `==, !=, <, >, <=, >=`
  - **Logical**: `&&, ||, !`
  - **Assignment**: `=, +=, -=, *=, /=, %=`
- **Example**: `int sum = a + b;`

---

# 11. Loops

- **for Loop**: Executes code a fixed number of times.

```
for (int i = 0; i < 10; i++) {
    cout << i << endl;
}
```

- **while Loop**: Repeats code as long as a condition is true.

```
int i = 0;
while (i < 10) {
    cout << i << endl;
    i++;
}
```

- **do-while Loop**: Executes code at least once, then checks the condition.

```
int i = 0;
do {
    cout << i << endl;
    i++;
} while (i < 10);
```

---

# 12. Conditional Statements

- **if-else**: Executes code based on conditions.
- **if-else-if Ladder**: Checks multiple conditions in sequence.
- **Nested if**: if inside another if.
- **Switch Case**: Simplifies multiple condition checks.

---

# 13. Arrays

- **Definition**: Store multiple values of the same data type.
- **Syntax**: dataType arrayName[arraySize];
- **Example**: int numbers[5] = {1, 2, 3, 4, 5};

---

# 14. Functions

- **Definition**: Blocks of code that perform specific tasks.
- **Syntax**:

```
returnType functionName(parameters) {
    // Code
}
```

- **Example**:

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

---

# 15. Recursion

- **Definition**: A function that calls itself to solve a smaller problem.
- **Example**:

```
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

---

# 16. Pointers

- **Definition**: Variables that store memory addresses.
- **Syntax**: dataType* pointerName;

- **Example**: int* ptr = &var;

---

# 17. Dynamic Memory Allocation

- **Definition**: Allocating memory at runtime using new and delete.
- **Syntax**:

```
int* ptr = new int; // Allocation
delete ptr;         // Deallocation
```

---

# 18. Structures, Unions, and Enums

- **Structures**: Group variables of different types under one name.

```
struct Person {
   string name;
   int age;
};
```

- **Unions**: Store different data types in the same memory location.

```
union Data {
   int intValue;
   float floatValue;
};
```

- **Enums**: Assign names to integer constants for readability.

```
enum Days { MON, TUE, WED };
```