**Module #3**

**Introduction to OOPS Programming**

# THEORY EXERCISE SET 1

**1.** What are the key differences between Procedural Programming and Object-Oriented Programming (OOP)?

**A1.** Procedural Programming (POP) is based on functions and follows a top-down approach, while OOP is based on objects and classes and follows a bottom-up approach. POP doesn't support concepts like encapsulation, inheritance, and polymorphism, whereas OOP does, making code more modular, reusable, and easier to maintain.

**2.** List and explain the main advantages of OOP over POP.

**A2.** - Code Reusability through Inheritance  
- Better data security via Encapsulation  
- Easier debugging and maintenance  
- Real-world modeling through Classes and Objects

**3.** Explain the steps involved in setting up a C++ development environment.

**A3.**

1. Install a C++ compiler like GCC or Turbo C++  
2. Use an IDE or text editor (e.g., Code::Blocks, Dev C++, VS Code)  
3. Set up compiler paths in IDE settings  
4. Write, compile, and run programs

**4.** What are the main input/output operations in C++? Provide examples

**A4.** Use `cin` for input and `cout` for output.  
Example:  
int a;  
cin >> a;  
cout << "Value: " << a;

# THEORY EXERCISE SET 2

**1.** What are the different data types available in C++? Explain with examples.

**A1.** int, float, double, char, bool, void. Example: int a = 5;

**2.** Explain the difference between implicit and explicit type conversion in C++.

**A2.** Implicit: Done automatically. Explicit: Done manually using type casting.

**3.** What are the different types of operators in C++? Provide examples of each.

**A3.** Arithmetic: +, -, \*, /  
Relational: ==, !=, >  
Logical: &&, ||  
Assignment: =, +=  
Etc.

**4.** Explain the purpose and use of constants and literals in C++.

**A4.** Constants are fixed values using ‘const’. Literals are direct values like 5, 'A', "Hello".

# THEORY EXERCISE SET 3

**1.** What are conditional statements in C++? Explain the if-else and switch statements.

**A1. ‘**if-else’ runs code based on condition. ‘switch’ chooses among multiple cases.  
Example:  
if(x > 0) {...} else {...}  
switch(day) { case 1: ...; break; }

**2.** What is the difference between for, while, and do-while loops in C++?

**A2.** `for` is used when count is known, `while` checks before executing, `do-while` runs at least once.

**3.** How are break and continue statements used in loops? Provide examples.

**A3.** `break` exits the loop early. `continue` skips current iteration.  
Example:  
if(i==3) break;  
if(i==3) continue;```

**4.** Explain nested control structures with an example.

**A4.** One control structure inside another. Example:  
for(...) { if(...) { ... } }

# THEORY EXERCISE SET 4

**1.** What is a function in C++? Explain the concept of function declaration, definition, and calling.

**A1.** A function in C++ is a block of code designed to perform a specific task. Functions help in making programs modular, reusable, and easier to maintain.

1. Function Declaration (Prototype):  
It tells the compiler about the function’s name, return type, and parameters.  
Example:

int add(int, int);

2. Function Definition:  
It contains the actual code or logic of the function.  
Example:

int add(int a, int b) {

return a + b;

}

3. Function Calling:  
It is the process of invoking the function from another part of the program.  
Example:

int result = add(2, 3);

**Complete Example:**

#include <iostream>using namespace std;

// Declarationint add(int, int);

int main() {

int result = add(2, 3); // Calling the function

cout << "Sum: " << result;

return 0;

}

// Definitionint add(int a, int b) {

return a + b;

}

**2.** What is the scope of variables in C++? Differentiate between local and global scope.

**A2.** Local: Inside functions. Global: Outside all functions. Global can be used anywhere.

**3.** Explain recursion in C++ with an example.

**A3.** A function calling itself. Example:  
int fact(int n) { if(n==0) return 1; return n\*fact(n-1); }

**4.** What are function prototypes in C++? Why are they used?

**A4.** Prototypes declare functions before `main()`, useful for informing compiler and type checking.

# THEORY EXERCISE SET 5

**1.** What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.

**A1.** Array is a collection of similar elements. 1D uses one index, 2D uses two (rows/columns).

**2.** Explain string handling in C++ with examples.

**A2.** Using char arrays or `string` class. Example:  
string name = "John";

**3.** How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

**A3.** 1D: `int a[3] = {1,2,3};` 2D: `int b[2][2] = {{1,2},{3,4}};`

4. Explain string operations and functions in C++.

**A4.** ‘length()’, ‘append()’, ‘substr()’, ‘find()’, etc. used to handle strings easily.

# THEORY EXERCISE SET 6

## **1.** Explain the key concepts of Object-Oriented Programming (OOP).

**A1.** Object-Oriented Programming (OOP) is a programming model based on the concept of “objects”.

The key concepts are:  
- **Class:** A blueprint that defines properties and behaviors (data and functions).  
- **Object:** An instance of a class that can use its properties and functions.  
- **Encapsulation:** Hiding internal data and allowing access only through specific functions.  
- **Inheritance:** Reusing code by allowing one class to inherit properties and behaviors from another.  
- **Polymorphism:** The ability to use a single function or operator in different ways.  
- **Abstraction:** Hiding unnecessary details and showing only essential features to the user.

## **2.** What are classes and objects in C++? Provide an example.

**A2.** A class in C++ is a user-defined data type that holds variables (data members) and functions (member functions). An object is an instance of a class used to access its members.  
  
Example:

#include <iostream>  
using namespace std;  
  
class Car {  
public:  
 void drive() {  
 cout << "Car is driving." << endl;  
 }  
};  
  
int main() {  
 Car myCar;  
 myCar.drive();  
 return 0;  
}

## 3. What is inheritance in C++? Explain with an example.

**A3.** Inheritance is a feature in C++ where one class (called a derived class) can inherit properties and functions from another class (called a base class). It promotes code reuse.  
  
Example:

#include <iostream>  
using namespace std;  
  
class Animal {  
public:  
 void sound() {  
 cout << "Animal makes sound." << endl;  
 }  
};  
  
class Dog : public Animal {  
public:  
 void bark() {  
 cout << "Dog barks." << endl;  
 }  
};  
  
int main() {  
 Dog d;  
 d.sound();  
 d.bark();  
 return 0;  
}

## **4.** What is encapsulation in C++? How is it achieved in classes?

**A4.** Encapsulation is the concept of hiding internal details of a class and exposing only necessary parts. It is achieved by:  
- Making data members private  
- Providing public functions (getters and setters) to access and modify the private data  
  
Example:

#include <iostream>  
using namespace std;  
  
class BankAccount {  
private:  
 int balance;  
  
public:  
 void setBalance(int b) {  
 balance = b;  
 }  
  
 int getBalance() {  
 return balance;  
 }  
};  
  
int main() {  
 BankAccount account;  
 account.setBalance(1000);  
 cout << "Balance: " << account.getBalance();  
 return 0;  
}