C++ ASSIGNMENT

Theeory

Introduction to C++

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

ANS:

Feature	Procedural Programming	Object-Oriented Programming (OOP)
Approach	Follows a step-by-step or top-down approach.	Follows a bottom-up approach.
Focus	Focuses on functions (procedures) and sequence of actions.	Focuses on objects that contain both data and functions.
Data Handling	Data is usually not secured; functions can access data freely.	Data is encapsulated inside objects and can be accessed only through methods.
Reusability	Reusability is achieved through functions.	Reusability is achieved through classes and objects (Inheritance).
Complexity	Works well for simple, small programs.	Works well for large and complex software systems.
Example Languages	C, Pascal, Fortran	C++, Java, Python, C#

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

ANS:

1. **: Encapsulation**
    - OOP bundles data and functions together inside objects.
    - This protects data from being accidentally changed and increases security.
2. **Reusability**
    - OOP allows code reusability through classes and inheritance.
    - You can create new classes from existing ones without rewriting code.
3. **Modularity**
    - The program is divided into small, manageable objects.
    - This makes the program easier to understand, maintain, and debug.
4. **Flexibility and Extensibility**
    - OOP supports polymorphism, which allows one interface to be used for different purposes.
    - It is easier to update or extend existing code.
5. **Abstraction**
    - OOP allows hiding unnecessary details and showing only essential features.
    - This makes complex programs easier to work with.
6. **Better Data Security**
    - Access modifiers (public, private, protected) help control data access.
    - Data is more secure compared to POP where data can be accessed freely.

Overall, OOP is more suitable for large and complex software systems, while POP is better for small and simple programs.

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

ANS:

1. Install a C++ Compiler

- A compiler translates your C++ code into machine language.
- Common compilers: GCC, MinGW, or Clang.
- Windows users often install MinGW or TDM-GCC.

2. Install an IDE or Text Editor

- An IDE provides tools like code editing, compiling, debugging, and running programs in one place.
- Popular IDEs for C++: Code::Blocks, Dev C++, Visual Studio, CLion.
- Text editors like VS Code, Sublime Text, or Notepad++ can also be used.

3. Configure Compiler in IDE (if needed)

- Some IDEs auto-detect the compiler.
- If not, go to IDE settings and set the compiler path (e.g., MinGW/bin).

4. Write a C++ Program

- Create a new project or file.
- Write code in a .cpp file.

5. Compile the Program

- Use the Build or Compile option in the IDE.
- This converts the source code into an executable file.

6. Run the Program

- Click Run to execute the compiled program and view the output.

7. Debug if Necessary

- Use the debugger to find and fix errors or bugs.

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

ANS:

Main Input/Output Operations in C++:

- **cout →** Used for output to the screen.
- **cin →** Used for input from the keyboard.
- **cerr →** Used for displaying error messages.
- **clog →** Used for displaying general log or information messages.

Example:

```
cin >> x;

cout << x;
```

Variables, Data Types, and Operators

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

ANS:

1. Built-in (Primitive) Data Types

Data Type	Description	Example
int	Stores whole numbers	int age = 20;
float	Stores decimal numbers (single precision)	float price = 99.5;
double	Stores decimal numbers (double precision)	double marks = 88.75;
char	Stores a single character	char grade = 'A';
bool	Stores true/false values	bool isPass = true;

2. Derived Data Types

Data Type	Example Use
array	int num[5];
pointer	int *ptr;
function	int add(int a, int b);
reference	int &ref = x;

3. User-Defined Data Types

Data Type	Example
struct	struct Student { int id; };
class	class Car { ... };
enum	enum Color { Red, Blue };
union	union Data { int x; float y; };

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

ANS:

Implicit Type Conversion

- Also called automatic type conversion.
- Performed automatically by the compiler.
- Usually occurs when a smaller data type is converted to a larger data type.
- Safe and does not normally cause data loss.

Explicit Type Conversion

- Also called type casting.
- Performed manually by the programmer.
- Usually used when converting a larger data type to a smaller data type.
- May result in data loss if not done carefully.

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

ANS:

1. Arithmetic Operators

- Used to perform mathematical operations like addition, subtraction, multiplication, division, and modulus.
- Examples: + - * / %
- Purpose: Calculate values.

2. Relational (Comparison) Operators

- Used to compare two values.
- Examples: == != > < >= <=
- Purpose: Determine relationships (equal, greater, smaller) and return true/false.

3. Logical Operators

- Used to combine multiple conditions.
- Examples: && (AND), || (OR), ! (NOT)
- Purpose: Check multiple logical conditions in decision-making.

4. Assignment Operators

- Used to assign values to variables.
- Examples: =, +=, -=, *=, /=
- Purpose: Simplify operations while assigning values.

5. Increment/Decrement Operators

- Used to increase or decrease a variable by 1.
- Examples: ++ (increment), -- (decrement)
- Purpose: Often used in loops and counters.

6. Bitwise Operators

- Operate at the bit level of numbers.
- Examples: & | ^ << >>
- Purpose: Perform low-level operations on binary representation of data.

7. Ternary Operator

- A shortcut for if-else statements.
- Example: condition ? value_if_true : value_if_false
- Purpose: Make simple decisions in a single line.

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

ANS:

Constants in C++

- Definition: A constant is a named value that cannot be changed during program execution.
- **Purpose:**
    1. To prevent accidental modification of important values.
    2. To make code readable and maintainable.

Literals in C++

- **Definition:** A literal is a **fixed value written directly in the code**.
- **Purpose:** Represents **constant values** in the program.
- **Types of literals:**
    1. **Integer literals:** 10, 200, -5
    2. **Floating-point literals:** 3.14, 0.5
    3. **Character literals:** 'A', 'b'
    4. **String literals:** "Hello"
    5. **Boolean literals:** true, false
- **Use:** Directly used in expressions, assignments, and calculations.

Control Flow Statements

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

ANS:

Conditional Statements in C++

- Conditional statements allow a program to execute certain code only when a specific condition is true.
- They control the flow of execution based on conditions.

1. if-else Statement

- Evaluates a condition.
- Executes one block of code if the condition is true, and another block if it is false.
- The else part is optional.

2. switch Statement

- Evaluates an expression and chooses among multiple fixed values.
- Executes the block corresponding to the matching value.
- default block executes if no match is found.
- Typically uses break to prevent execution from falling through to the next case.

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

ANS:

1. for Loop

- Used when the number of iterations is known in advance.
- Loop control (initialization, condition, increment/decrement) is in one line.
- Condition is checked before each iteration.

2. while Loop

- Used when the number of iterations is not known in advance.
- Only the condition is specified; initialization and increment are done separately.
- Condition is checked before each iteration.

3. do-while Loop

- Similar to while but the loop executes at least once, because the condition is checked after the loop body.
- Used when you want the code to run at least one time, regardless of the condition.

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

ANS:

1. break Statement

- Purpose: Immediately terminates the loop and exits it.
- Often used to stop a loop when a certain condition is met.

Example:

```
for(int i = 1; i <= 5; i++) {
   if(i == 3)
      break;  // exits the loop when i is 3
   cout << i << " ";
}
```

2. continue Statement

- **Purpose:** Skips the current iteration of the loop and moves to the next iteration.
- Used to ignore certain cases without terminating the loop.

Example:

```
for(int i = 1; i <= 5; i++) {
   if(i == 3)
      continue;  // skips printing when i is 3
   cout << i << " ";
}
```

4. Explain nested control structures with an example.

ANS:

Nested Control Structures

- **Definition:** When one control structure (like if, for, or while) is placed inside another, it is called a nested control structure.
- **Purpose:** Allows more complex decision-making and looping by combining multiple conditions or loops.

Example:

```
int x = 5;
if(x > 0) {
   if(x < 10) {
      cout << "x is between 1 and 9";
   }
}
```

Functions and Scope

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

ANS:

Function in C++

- A function is a block of code that performs a specific task and can be reused in a program.
- Purpose: Makes programs modular, reduces redundancy, and improves readability.

1. Function Declaration (Prototype)

- Tells the compiler the name, return type, and parameters of the function before it is defined.
- Ensures the function can be called before its actual definition in the code.

2. Function Definition

- Provides the actual code of the function that performs the task.

3. Function Calling

- Executes the function wherever needed in the program.
- A function can be called multiple times from different parts of the program.

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

ANS:

Scope of Variables in C++

- **Definition:** The scope of a variable refers to the region of the program where the variable can be accessed or used.
- Determines the lifetime and visibility of a variable.

1. Local Scope

- A variable declared inside a function, block, or loop.
- Accessible only within that block.
- Destroyed when the block ends.

2. Global Scope

- **A variable declared outside all functions.**
- **Accessible from any part of the program after its declaration.**
- **Exists throughout the lifetime of the program.**

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

ANS:

Recursion in C++

- **Definition:** Recursion is a process in which a function calls itself either directly or indirectly to solve a problem.
- **Purpose:** Used to solve problems that can be broken into smaller, similar sub-problems, like factorial, Fibonacci series, or tree traversals.

EXAMPLE:

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

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

ANS:

Function Prototypes in C++

- A function prototype is a forward declaration of a function that specifies the function's name, return type, and parameters, but does not include the function body.
- It tells the compiler about the function before it is actually defined.

Why Function Prototypes Are Used

1. Compiler Awareness: Ensures the compiler knows about the function's existence, type, and parameters in advance.
2. Supports Early Function Calls: Allows functions to be called before their definitions appear in the source code.
3. Type Checking: Helps the compiler verify that function calls match the expected return type and parameters, preventing errors.
4. Modularity and Readability: Improves code organization, especially in large programs, by separating function declarations from function definitions.

Arrays and Strings

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

ANS:

Arrays in C++

- Definition: An array is a collection of elements of the same data type, stored in contiguous memory locations.
- Purpose: Allows storing and accessing multiple values using a single variable name with an index.

1. Single-Dimensional Array

- Stores elements in a single linear row.
- Accessed using a single index.
- Example concept: array[index]

Characteristics:

- Fixed size defined at declaration.
- Easier to traverse with a single loop.

2. Multi-Dimensional Array

- Stores elements in more than one dimension, like a table or matrix.
- Accessed using multiple indices.
- Example concept: array[row][column] for 2D arrays.

Characteristics:

- Can represent complex structures like matrices, tables, or grids.
- Traversal requires nested loops.

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

ANS:

Strings in C++

- **Definition:** A string is a sequence of characters used to store text.
- **Two main types in C++:**
    1. C-style strings: Arrays of characters ending with a null character \0.
    2. C++ string class: Provided by the <string> library for easier manipulation.

Example: #include <iostream>

#include <string>

using namespace std;

```
int main() {
   string name = "Tootie";
   cout << name.length();
   return 0;
}
```

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

ANS:

Array Initialization in C++

- **Definition:** Assigning values to an array at the time of declaration or later in the program.

1D Array

- **Definition:** A 1D array is a collection of elements of the same type stored in a single row.
- **Example:**

  int numbers[5] = {1, 2, 3, 4, 5};

2D Array

- **Definition:** A 2D array is a collection of elements arranged in rows and columns.
- **Example:**

  ```
  int matrix[2][3] = {
     {1, 2, 3},
     {4, 5, 6}
  };
  ```

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

ANS:

String Operations in C++

Strings are used to store and manipulate text. C++ provides two ways to handle strings:

1. C-style strings (character arrays)
2. C++ string class (from <string> library)

Functions

Operation / Function	Purpose
length() / size()	Finds the length of the string
append() or +	Joins (concatenates) two strings
compare() or ==	Compares two strings
substr()	Extracts part of a string
find()	Searches for a character or substring
erase()	Removes part of the string

Introduction to Object-Oriented Programming

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

ANS:

Key Concepts of Object-Oriented Programming (OOP)

1. **Class**
   A class is a blueprint or template that defines the structure and behavior (data and functions) of objects.
2. **Object**
   An object is an instance of a class. It represents a real-world entity and has data and behavior.
3. **Encapsulation**
   Encapsulation means binding data and functions together and restricting direct access to the data to protect it.
4. **Abstraction**
   Abstraction means showing only the essential features and hiding unnecessary details.
5. **Inheritance**
   Inheritance allows a class to acquire properties of another class, promoting code reusability.
6. **Polymorphism**
   Polymorphism means the same function or operator can behave differently depending on the context.

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

ANS:

Classes in C++

- A class is a user-defined data type that contains data members (variables) and member functions (methods).
- It acts as a blueprint for creating objects.

Objects in C++

- An object is an instance of a class.
- It represents real-world entities and uses the data and functions defined in the class.

Example:

```
#include <iostream>

using namespace std;


class Student {

public:

   string name;

   int age;


   void display() {

      cout << "Name: " << name << ", Age: " << age;

   }

};


int main() {

   Student s;

   s.name = "Tootie";

   s.age = 20;

   s.display();

   return 0;

}
```

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

ANS:

Inheritance in C++

- Inheritance is a feature in C++ where a **new class (derived class)** is created from an **existing class (base class)**.
- The **derived class** inherits the **data members and functions** of the base class.
  It helps in **code reusability** and creating hierarchical relationships.

Example:

```
class A {       // Base class
public:
   void display() {
      cout << "This is class A";
   }
};

class B : public A {
};

int main() {
   B obj;
   obj.display();
}
```

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

ANS:

Encapsulation in C++

Encapsulation is the process of wrapping data and functions together into a single unit (class) and restricting direct access to the data.
It helps in protecting data from unauthorized access and misuse.

How Encapsulation is Achieved

Encapsulation is achieved by using access specifiers in classes:

- **private:** Data members are hidden and cannot be accessed directly from outside.
- **public:** Functions are accessible from outside and used to access or modify private data.