Module 3 Theory Exercise

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

Ans:

|  |  |
| --- | --- |
| **Procedural Programming** | **Object Oriented Programming** |
| In procedural programming, the program is divided into small parts called **functions.** | In Object-oriented programming, the program is divided into small parts called **objects.** |
| Procedural programming follows a **top-down approach.** | Object-oriented programming follows a **bottom-down approach.** |
| There is no access specifier in procedural programming. | Object-oriented programming has access specifiers like private, public, protected etc. |
| Adding new data and function is not easy. | Adding new data and function is easy. |
| Procedural programming does not have any proper way of hiding data so it is **less secure.** | Object-oriented programming provides data hiding so it is **more secure.** |
| In procedural programming, overloading is not possible. | Overloading is possible in object-oriented programming. |
| In procedural programming, the function is more important than data. | In object-oriented programming data is more important than function. |
| Procedural programming is based on the **unreal world.** | Object-oriented programming is based on the **real world.** |
| Procedural programming uses the concept of procedure abstraction. | Object-oriented programming uses the concept of data abstraction. |
| Code reusability absent in procedural programming. | Code reusability present in object-oriented programming |
| **Examples:** C, FORTAN, PASCAL, Basic etc.. | **Examples:** C++, JAVA, Python, C# etc. |

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

Ans:

1. Modularity and reusability:

* OOP promotes modularity by organizing code in to self-contained objects, which are easier to understand and maintain
* Objects and class can be reused in different parts of program or even in other applications, saving development time and efforts.

1. Code reusability:

* Inheritance allows classes to inherit properties and methods from other classes, reducing redundancy and promoting code reuse.
* This leads to faster development and reduces the need to rewrite code for similar functionalities.

1. Encapsulation and Data-hiding:

* OOP allows data to be hidden within objects, protecting it from unauthorized access and ensuring data integrity.
* This principle of data hiding enhances security and makes code easier to maintain.

1. Improved code maintainability:

* The modular structure of OOP makes it easier to locate and fix errors, as well as modify or upgrade the code without affecting other parts of the program.
* The leads to faster development and reduces the need to rewrite code for similar functionality.

1. Enhanced security:

* Encapsulation and data hiding, along with access modifiers, provide enhanced security by limiting access to data and code to authorized users.
* This helps prevent unauthorized modifications to reduces the risk of security vulnerabilities.

1. Problem solving:

* OOP allows developers to mode real-world systems and problems more accurately, making it easier to solve complex problems.
* By breaking down complex problems into smaller, manageable objects, OOP simplifies the development process.

1. Flexibility and scalability:

* OOP supports polymorphism, allowing objects to be used in different ways depending on the context, enhancing code flexibility.
* This makes it easier to create reusable and extensible code that can be adapted to changing requirements.

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

Ans:\_\_\_\_\_\_\_

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

Ans: \_\_\_\_\_\_\_\_\_\_

Q.5 What are the different data types available in C++? Explain with examples.

Ans: C++ offers varieties of data types, broadly categorized in Primitive, Derived and User-Defined types.

1. Primitive Data Types::

These are the fundamental built-in types.

**int**: stores whole numbers (integers) without decimal points.

**Ex.** int age=30;

**char:** Storessingle character

**Ex.** char grade=’A’;

**float:** Stores single-precision floating-point numbers (numbers with decimal points).

**Ex.** float price =19.99f;

**double:** Stores double-precision floating-point numbers, offering more precision than float.

**Ex.** double pi=3.1415926535;

**boolean:** Stores values either “true” or “false”.

**Ex.** boolean is\_active=true;

**void:** Represents the absence of a type, typically used for functions that don’t return a value or for generic pointers.

**Ex.** void print\_message (){

//function that doesn’t return anything.

}

1. Derived Data Types::

These are built upon primitive data types.

**Arrays:** Collections of elements of same data type stored in contiguous memory locations.

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

**Pointers:** Variables that store memory addresses of other variables.

**Ex.** int a=5;

int&b=a;

**References:**

1. User-Defined Data Types::

These are created by the programmer to represent complex data structures.

**Struct (Structures):** collection of variables of different data types under a single name.

**Ex.** Struct Person

{

Char name [50];

Int age;

};

Person p1;

**class (Classes):** Blu**e**printsfor creating objects, encapsulating data (member variables) and functions (member methods).

**Ex.** class Car {

Public:

Void start () {/\*.........\*/}

} ;

Car mycar;

**enum (Enumerations):** sets of named integer constants.

**Ex.** enum Day {Monday, Tuesday, Wednesday};

Day today=Monday;

**Union (Unions):** Allow different data types to share the same memory location, but only one member can active at a time.

**Ex.** union Data {

Int i;

Float f;

};

Data d;

d.i=10;

Q.6 Explain the difference between implicit and explicit type conversion in C++.

Ans.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implicit type conversion** | **Explicit type conversion** |
| Initiation | Automatic by compiler | Manual by programmer |
| Syntax | No special syntax required. | Uses casting operators (e.g., (type), static\_cast) |
| Data loss risk | Generally low (promotions). | Potential for data loss (e.g., float to int) |
| Control | Less control for programmer | Full control for programmer |
| Purpose | Ensure type compatibility in expressions | Force conversion, even with potential data loss |

Q.7 What are the different type of operators in C++? Provide example of each.

Ans: C++ provides various types of operators to perform operations on data. The main types of operators are:

1. **Arithmetic operators**

Used for mathematical calculation.

* + - Addition (+)
    - Subtraction (-)
    - Multiplication (\*)
    - Division (/)
    - Modules (%)

1. **Assignment operators**

Used to assign values to variables.

* + Simple assignment (=)
  + Add and assign (+=)
  + Subtract and assign (-=)
  + Multiply and assign (\*=)
  + Divide and assign (/=)
  + Modulus and assign (%=)

1. **Relational operators**

Used to compare two values, resulting in a boolean (true/false) outcome.

* + Equality (=)
  + Inequality (! =)
  + Greater than (>)
  + Less than (<)
  + Greater than or equal to (>=)
  + Less than or equal to (<=)

1. **Logical operators**

Used to combine or modify boolean expressions.

* + Logical AND (&&)
  + Logical OR (||)
  + Logical NOT (!)

1. **Bitwise operators**

Used to perform operations on individual bits of integer types.

* + Bitwise AND (&)
  + Bitwise OR (|)
  + Bitwise XOR (^)
  + Bitwise NOT (~)
  + Left shift (<<)
  + Right shift (>>)

1. **Increment or Decrement operators**

Used to increase or decrease the values of a variable by one

* + Increment (++)
  + Decrement (--)

1. **Conditional (ternary) operators**

A Shorthand for an if-else statement.

* condition? expression 1 : expression 2;
* int max = (a>b)?a:b;

Q.8 Explain the purpose and use of constants and literals in C++.

Ans:

**Literals:** Literals are fixed values directly represented in the source code. They are the raw data values that the compiler understands without any further interpretation.

* Purpose: To represent specific, immutable data values directly within expressions, assignments, or arguments to functions.
* Use:

Integer literals: 10, 0xFF (hexadecimal), 012 (octal)

Floating-point literals: 3.14, 1.23e-5

Character literals: ‘A’, ‘\n’

String literals: “Hello World”

Boolean literals: “true” or “false”

**Constants:** Constants are named entities whose values are fixed and cannot be changed after initialization during program execution. In C++, they are typically declared using the const keyword.

* Purpose: To provide meaningful names for fixed values, enhance code readability, prevent accidental modification of important data, and enable compile-time checks for immutability.
* Use:

Declaring fixed values

Function parameters

Constant member functions

Q.9 What are the conditional statements in C++? Explain if-else and switch statements.

Ans: Conditional statements in C++ allow a program to make decisions and execute different blocks of code based on whether certain conditions are true or false. The primary conditional statements in C++ are **if-else** and **switch**.

* **if-else statement:** The if-else statement is used to execute one block of code if a specified condition is true, and other block of code if the same condition is false
* if statements: Executes a block of code if the given condition evaluates to true.
* else statements: Used with an if statement, it executes a block of code if the if condition evaluates false.
* Else-if statements: Used to specify a new condition to test if the preceding if or else if conditions are false. This allows for multiple conditions to be checked sequentially.
* **switch statements:** The switch statement is a multi-way branch statement that provides an alternative to a log if-else if ladder, especially when comparing a single variable or expression against multiple constant values.
* The switch statement evaluates an expression and compares its value against a series of case labels.
* If a match is found, the code block associated with that case is executed.
* The break statement is crucial within each case to exit the switch block once a match is found and executed, preventing “fall-through” to subsequent case blocks.
* The default case is optional and executes if none of the case labels match the expression’s value.

Q.10 What is difference between for, while, and do-while loops in C++?

Ans:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **For loop** | **While loop** | **Do-while loop** |
| **Purpose** | Primarily used when the number of iterations is known or can be determined before the loop begins | Used when the numbers of iterations is unknown and loop continues as long as condition remains true. | Similar to a while loop, but guarantees that the loop body executes at least once, regardless of the initial condition. |
| **Structure** | Consists three parts within parentheses: initialization, condition, and increment/decrement | Consists of condition within parentheses | The do keyword precedes the loop body followed by the while keyword and the condition in parentheses. |
| **Execution** | The condition is checked before each iteration. If true, the loop body executes, followed by the increment/decrement. If false, the loop terminates. | The condition is checked before each iteration. If true, the loop body executes, If false, the loop terminates and the body is never executed. This means the while loop body might not execute at all if the initial condition is false. |  |

Q.11 How are break and continue statements used in loops? Provide example.

Ans: In loops, the break and continue statements alter the normal flow of execution. Break terminates the loop entirely, while continue skips the current iteration and proceeds to the next one.

Break statements:

* The break statement is used to exit a loop prematurely, regardless of whether the loop’s condition has been met.
* When break is encountered inside a loop, the loop is immediately terminated, and the program control jumps to the statement following the loop.
* It’s often used with conditional statements to exit a loop when a specific condition is met.

Continue statements:

* The continue statement is used to skip the rest of the current iteration of a loop and move on to the next iteration.
* When continue is encountered, the remaining code within the current loop iteration is skipped, and the loop’s condition is checked again to determine if another iteration should be performed.
* It’s useful to skipping certain iterations based on specific conditions.

Q.12 Explain nested control structures with an example.

Ans: Nested control structures involve placing one control statement (like if, for, while, or switch) inside another. This allows for more complex logic and decision-making or multi-level iteration. The inner control structure executes fully for each iteration or evaluation of the outer control structure.

Ex.

#include<iostream>

Int main ()

{

For (int i=0;i<=3;i++)

{

cout<<”outer loop...”<<i<<endl;

For (int i=0; i<=2; i++)

{

cout<<”inner loop...”<<i<<endl;

}

}

}

Q.13 What is a function in C++? Explain the concept of function declaration, definition, and calling.

Ans: A function in C++ is a self-contained block of code designed to perform a specific task. Functions enhance code reusability, modularity, and readability by encapsulating operation into distinct units that can be invoked multiple times throughout a program. They can accept input values, known as parameters and can return a result.

Function Declaration:

This informs the compiler about a function’s existence before its actual definition. It specifies the function’s the return type, name, and the types and order of its parameters. The declaration does not contain the function’s body. It acts as signature, allowing the compiler to verify correct function calls.

Function Definition:

This contains the actual implementation of the function, including the statements that perform the designated task. The definition provides the body of the function, which executed when the function is called.

Function Calling:

This is the act of invoking or executing a function. When a function is called, the program control transfers to the function’s body and the statements within it are executed. Arguments (actual values) are passed to the function’s parameters, and if the function has a return type other than void, it can return a value to the calling location.

Q.14. What is the scope of variables in C++? Differentiate between local and global scope.

Ans. In C++, the scope of a variable defines the region of the program where that variable is accessible and can be used. Variables can have different scopes, the most common being local and global.

**Differentiation:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Local scope** | **Global scope** |
| Declaration | Inside a function or a code block | Outside of all functions. |
| Access | Only within declaring function or block | Anywhere throughout the entire program |
| Memory | Typically allocated on the stack | Typically allocated in the data segment. |
| Sharing | No direct data sharing between functions | Facilitates data sharing between functions |

Q.15 Explain recursion in C++ with an example.

Ans: In C++ recursion is a technique which a function calls itself repeatedly until given condition is satisfied. It is used for solving a problem by breaking it down into smaller, simple sub-problems. Then finding the solution to find the global solution.

Ex.

#include <iostream>

using namespace std;

void print Hello (int n) {

if (n == 0) return;

cout << "Hello" << endl;

print Hello (n - 1);

}

int main () {

print Hello (5);

return 0;

}

Q.16 What are function prototypes in C++? Why are they used?

Ans: A function prototype in C++ is a declaration of a function that satisfies its name, return type, and the types of parameters, but does not include the function’s body. It essentially provides the function’s “signature” to the compiler.

* Forward Declarations
* Type checking
* Code organization and Readability
* Handling circular dependencies

Q.17 What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.

Ans: Arrays in C++ are data structures used to store a collection of elements of the same data type in contiguous memory locations. Each element in an array can be accessed using an index, which represents its position within the array.

Single-dimensional Array:

* + Represent a linear list or sequence of elements.
  + Elements are arranged in single row or column.
  + Accessed using single index.

Multi-dimensional Array:

* + Represent a collection of arrays, forming a grid-like structure.
  + Can have two or more dimensions.

Q.18 Explain string handling in C++ with examples.

Ans: In C++ strings can handled by using two primary methods:

C-style strings and the std::string class.

1. C-style Strings:

C-style strings are character arrays terminated by null character (\0).

1. Std::string Class:

The std::string class from the <string> header provides a more robust and flexible way to handle strings. It manages memory dynamically, eliminating the need for manual memory handling.

Ex.

#include <iostream>

#include <cstring> // For C-style string functions

int main() {

char greeting[] = "Hello"; // Declaring and initializing a C-style string

std::cout << greeting << std::endl; // Output: Hello

// Concatenation (using strcat)

char name[20] = "World";

strcat(greeting, name); // Appends "World" to "Hello"

std::cout << greeting << std::endl; // Output: HelloWorld

// Length (using strlen)

std::cout << "Length: " << strlen(greeting) << std::endl; // Output: 10

return 0;

}

Q.19 Explain key concept of Object Oriented Programming (OOP).

Ans:

* + Objects
  + Class
  + Encapsulation
  + Abstraction
  + Inheritance
  + Polymorphism

Q.20 what are classes and objects in C++? Provide an example.

Ans:

**Class:**

* A class is a user-defined data type that groups related data and functions together
* It defines the structure and behaviour of objects that belong to that class.
* In C++, the class keyword is used to define a class.

**Objects:**

* An object is an instance of a class, created during program execution.
* It has its own set of data values based on the class definition.

Q.21 What is inheritance? Explain with an example.

Ans: Inheritance, a fundamental concept in object-oriented programming (OOP), allows a class to inherit properties and methods from another class. This promotes code reusability and establishes a hierarchical relationship between classes.