Module 3 - Introduction to OOPS Programming (Theory Exercise)

1 Introduction to C++

1.1 WhatarethekeydifferencesbetweenProceduralProgramming and Object-Oriented Programming (OOP)?

Procedural Programming	Object-Oriented Programming (OOP)
Programisorganizedintofunctions	Programisorganizedintoobjectsand classes
Followsatop-downapproach	Followsabottom-downapproach
Less secure due to lack of data hiding capability	More secure due to inclusion of data hidingcapability
Doesnotsupportsencapsulation	Supportsencapsulation
Inheritanceisnotallowed	Inheritanceisallowed
Doesnotofferaccessspecifiers	Offersaccessspecifiers
Doesnotsupportpolymorphism	Supportspolymorphism
Examples:C,FORTRAN,Pascal	Examples:C++,JAVA,Python

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

- OOP offer encapsulation, which makes code modular as it allow bundling of data and methods into individual classes. This allows developers to keep the code well organized and debug easily.
- It could also potentially reduce development time and cost, as parts of the code that needs to be used across multiple files can be written once and accessed from a single file due to the inheritance feature.
- Encapsulation also allow easy scaling and maintenance in the future.
- Data can be hidden using access specifiers making it more secure.
- Supports polymorphism which enables having functions of same name in same or different classes, making it more flexible and dynamic.

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

Step 1: Install a C++ compiler such as GCC or MinGW which translate the code into an executable program which computer can understand.

Step 2: Install a IDE (Integrated Development Environment) to write, build, execute and debug the code. They automatically detect the type of code and give them different specific colors making it easier to read and understand the code.

Step 3: Configure the IDE for the installed compiler so the IDE can access it for building the code.

Step 4: Write a code, save the file, and click the ”Build and Run” button making the IDE run the compiler with the code file as input, which will then convert it and generate an executable file.

Step 5: If there are no errors, the output will be displayed in a terminal window. But if there are errors, they will be displayed in the IDE.

1.4 Whatarethemaininput/outputoperationsinC++?Provide examples.

1. Standard Output Stream: cout

syntax:

```
cout<<value/variable;
```

example:

```
#include<iostream>
using namespace std;
main(){
    cout<<"a = "<<5;
}
```

2. Standard Input Stream (cin)

syntax:

```
cin<<variable;
```

example:

```
#include<iostream>
using namespace std;
main(){
    cin>>a;
}
```

3. Un-buffered Standard Error Stream (cerr)

syntax:

```
cerr<<string;
```

example:

```
#include<iostream>
using namespace std;
main(){
    cerr>>"Error";
}
```

4. Buffered Standard Error Stream (clog)

syntax:

```
cerr<<string;
```

example:

```
#include<iostream>
using namespace std;
main(){
    cerr>>"Error Log";
}
```

2 Variables,DataTypes,andOperators

2.1 Whatarethedifferentdatatypesavailablein C++? Explain with examples.

1. **bool**: Stores only either 'true' or 'false'
   example:

```
bool b = true;
```

2. **char**: Stores single characters
   example:

```
char c = 'a';
```

3. **int**: Stores integer values
   example:

```
int n =5;
```

4. **float**: Stores decimal values and has a size of 4 bytes
   example:

```
float f = 0.123456;
```

5. **double**: Same as float but has double the size of 8 bytes
   example:

```
double d = 0.123456789012345;
```

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

Implicit type conversion	Explicit type conversion
No loss of data	Data my get lost
Done automatically by the compiler at compile time	Done manually by the programmer when writing the code
Type conversion decided by the compiler	Type conversion specified by the programmer

example,
`int a = 10;`
`double b=a;`
	example,
	`double a = 10.5;`
	`int b = static_cast<int >(a);`

2.3 Whatarethedifferenttypesofoperatorsin C++? Provide examples of each.

• Arithmetic Operators: These operators perform mathematical calculations.

- Addition ( ): + Addstwooperands.Example:a+b
- Subtraction ( ): - Subtractsthesecondoperandfromthefirst.Example:a-b
- Multiplication ( ): * Multipliestwooperands.Example:a*b
- Division/( ): Dividesthefirstoperandbythesecondandreturnsthequotient. Example: a / b
- Modulus ( )% Dividesthefirstoperandbythesecondandreturnstheremainder. Example: a % b

• Relational Operators: These operators compare two operands and return a boolean value (true or false), represented as 1 or 0.

- Equal to (==): Checks if two operands are equal. Example: a == b
- Not equal to (! =): Checks if two operands are not equal. Example: a != b
- Greater than (>): Checks if the first operand is greater than the second. Example: a > b
- Less than (<): Checks if the first operand is less than the second. Example: a < b
- Greater than or equal to (>=): Checks if the first operand is greater than or equal to the second. Example: a >= b
- Less than or equal to (<=): Checks if the first operand is less than or equal to the second. Example: a <= b

• Logical Operators: These operators compare boolean expressions and return a boolean value.

- Logical AND (&&): Returns true (1) if both operands are true (non-zero). Example: a && b

– Logical OR (||): Returns true (1) if at least one of the operands is true (non-zero). Example: a || b

– Logical NOT (!): Returns true (1) if the operand is false (zero), and false (0) if the operand is true (non-zero). Example: !a

• Assignment Operators: These operators assign values to variables.

– Simple assignment (=): Assigns the value of the right operand to the left operand. Example: a = 10

– Add and assign (+ =): Adds the right operand to the left operand and assigns the result to the left operand. Example: a += 5 (equivalent to a = a + 5)

– Subtract and assign (– =): Subtracts the right operand from the left and assigns the result to the left operand. Example: a -= 5 (equivalent to a = a - 5)

– Multiply and assign (* =): Multiplies the right operand by the left and assigns the result to the left operand. Example: a *= 5 (equivalent to a = a * 5)

– Divide and assign (/ =): Divides the left operand by the right operand and assigns the result to the left operand. Example: a /= 5 (equivalent to a = a / 5)

– Modulus and assign (% =): Performs modulus operation on the operands and assigns the result to the left operand. Example: a %= 5 (equivalent to a = a % 5)

• Increment and Decrement Operators: These operators increase or decrease the value of a variable by one.

– Increment (++): Increases the value of the operand by 1. Can be prefix (++a) or postfix (a++).

– Decrement (––): Decreases the value of the operand by 1. Can be prefix (--a) or postfix (a--).

The prefix form modifies the value before it is used in an expression, while the postfix form uses the current value in the expression and then modifies it.

• Bitwise Operators: These operators perform operations at the bit level. They work on integer operands.

– Bitwise AND (&): Performs a bitwise AND operation. Example: a & b

– Bitwise OR (|): Performs a bitwise OR operation. Example: a | b

– Bitwise XOR (∧): Performs a bitwise exclusive OR operation. Example: a ∧ b

– Bitwise NOT ( ): Performs a bitwise complement (inverts all bits). Example: a

– Left shift (<<): Shifts the bits of the left operand to the left by the number of positions specified by the right operand. Example: a << 2

– Right shift (>>): Shifts the bits of the left operand to the right by the number of positions specified by the right operand. Example: a >> 2

· Conditional Operator (Ternary Operator): This operator is a shorthand for an if-else statement.
Syntax: condition? expression1 : expression2
If the condition is true (non-zero), expression1 is evaluated; otherwise, expression2 is evaluated.

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

• Constants: Constants are used for values that do not need to be changed through- out the code. They help prevent accidental value modification, make it easy to change a value everywhere by changing it only at one place, and make the code run more efficiently, as the compiler could replace the constant's reference throughout the code with the value assigned to it.

• Literals: Literals are fixed direct values written in the code. They can be used without allocating space unlike variables and constants. They are used to assign values to variables or constants, as operands in operations, and pass arguments to functions.

3 Control Flow Statements

3.1 WhatareconditionalstatementsinC++? Explaintheif-else and switch statements.

Conditional statements in C++ are built-in functions which run a block of code only when a given condition is satisfied, otherwise then skip that block of code.

• **if-else** statement: It check a given condition. If it is true, it runs one particular block of code, otherwise it runs a different specified block of code.
syntax:

```
if (condition){
    //block of code to be  run if  the condition is  true
}
else{
    //block of code to be  run if  the condition is  false
}
```

example:

```
if (n%2==0){
    cout<<n<<" is even";
}
else{
    cout<<n<<" is odd";
}
```

• **switch** statement: It compares the value of a variable with different values and if the value of the variable is equal to anyone of the specified value, it will run the code under it. And if it is equal to none of the values, the default code block will be ran.
syntax:

```
switch(variable){
    case value1:
        //code to be ran  if  variable=value1
```

```
        break;
    case value2:
        //code to be ran if variable=value2
        break; . . .



    case valuen:
        //code to be ran if variable=valuen
        break;
    default:
        //code to be ran if variable is not equal to any of the
            previous values
}
```

example:

```
switch(marks/10){
    case 10:
        cout<<"A+";
        break;
    case 9:
        cout<<"A";
        break;
    case 8:
        cout<<"B";
    case 7:
        cout<<"C";
        break;
    case 6:
        cout<<"D";
        break;
    default:
        cout<<"F";
}
```

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

for and while loop are entry control loops i.e., they check the condition before executing the block of code, whereas do-while is exit control loop i.e., it checks condition after executing the block of code. Another difference is in for loop, initialization, condition and iteration are written in a single line whereas in while and do-while loops, they are written in different lines. for loop is used when we know how many times we want to run the loop, whereas while loop is used when we want to run the loop as long as a condition is satisfied, similarly do-while loop is used if we want to run the loop at least once irrespective of the condition.

3.3 Howarebreakandcontinuestatementsusedinloops?Provide examples.

break statement: It is used to terminate a loop before it ends if a specific condition is satisfied. It is usually used for finding something or for error handling.

```
for (i=0; i<10; i++){
    if (i==5){
        break;
    }
    cout<<i<<endl;
}
```

continue statement: It is used to skip an iteration of a loop if a specific condition is satisfied. It is usually used for filtering out data from a group of data or if a certain case does not require extra processing.

```
for (i=0; i<10; i++){
    if (i==5){
        continue;
    }
    cout<<i<<endl;
}
```

3.4 Explain nested control structures with an example.

Nested control structures means having control structures inside another allowing to make complex logic and make decisions based on multiple conditions.

```
for (i=0; i<10; i++){
    if (i%2==0){
        cout<<i<<endl;
    }
}
```

4 Functions and Scope

4.1 WhatisafunctioninC++?Explaintheconceptoffunction declaration, definition, and calling.

- Definition: A function is a block code, made usually for a specific purpose which can be used multiple times through out the code and run only when it is called.
- Function Declaration: Function declaration tells the compiler the name of the function, its return type, and its parameters, if any.

  Syntax:

```
return_type function_name(parameter_type1 parameter_name1,
...); parameter_type2 parameter_name2,
```

• Function Definition: Function definition encompasses the block of code that is to be ran when the function is called.

Syntax:

```
return_type function_name(parameter_type1 parameter_name1,
parameter_type2 parameter_name2, ...){
    // Code to be executed
    return return_value;
}
```

• Calling a Function: In order to run the function, it needs to be called, either from the main or from some other function or it could call itself. Function can be called multiple times with same or different arguments passed to it. The type and number of arguments should match that of the parameters in the funciton declaration and definition.

Syntax:

```
function_name(argument1, argument2, ...);
```

4.2 WhatisthescopeofvariablesinC++? Differentiatebe- tween local and global scope.

Scope of a variable is the region in which it can be accessed and used. If a variable is accessed outside its scope, compiler will not be able to find a variable with that name and give error. For example, a variable declared in a function cannot be accessed from another one and the compiler will give error: "'[variable name]' was not declared in this scope". But if a variable is declared outside any class or function, can be accessed from anywhere in the code.

Local scope	Global scope
Access is limited to the block of code it is declared in Lifespan is limited to the	Accessible through out the code
block it is declared in Can share the name with other	Lifespan is throughout the execution of the program
variables outside the block it is declared in	Cannot share the name with any other variable through out the program

4.3 Explain recursion in C++ with an example.

Recursion is a technique in which a function is called by itself with modified input(s) until a specific condition(s) is/are met.

Example:

```
#include<iostream>
using namespace std;
int fac( int n){
    if (n==1){
        return 1;
    }
```

```
    return n * fac(n-1);
}
main(){
    cout<<"5! = "<<fac(5);
}
```

In this example, the function fac(int n) is given the number whose factorial is to found. It then multiplies that number with the number just 1 less than it by calling itself recursively until the number becomes 1, then it returns the final multiplied value, which is the factorial of the initial number.

4.4 WhatarefunctionprototypesinC++?Whyaretheyused?

They are declaration of a function with its, name, return type, and number and type of its parameters. They give information of the function to the compiler before it is defined.
Syntax:

```
return_type function_name{parameter1_type,parameter2_type, ...}
```

Uses:

1. They allow functions to be defined after the main function or anywhere in the code to improve readability.

2. They also enable mutual recursion of 2 or more functions which call each other recursively.

3. In large projects, programs are split into multiple files for better readability and debugging. Function prototypes allow developers to put them in one file and define the function in another.

4. They also help detect errors at compile time rather at run time by enabling compiler to compare the number and type of arguments of the function with its parameters along with its return type.

5 ArraysandStrings

5.1 WhatarearraysinC++? Explainthedifferencebetween single-dimensional and multi-dimensional arrays.

It is a derived data type consisting of group data of same data type. They enable storing of multiple data in one single variable.
Example:

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

	One-Dimensional Array	Multi-Dimensional Array
Definition	Linear structure with a single row or column	Array with > 1 dimension with multiple rows and columns

Number of Subscripts	One index	Multiple indices (one per dimension)
Syntax	data_type array_name[array_size];	data_type array_name[size1][size2]...[sizeN];
Example	int a[5]; a[0] = 1; int n1 = a[0];	int matrix[3][5]; matrix[1][2] = 8; int element= matrix [1][2];
Uses	Lists, sequences, vectors	Tables, matrices, images (2D)

5.2 Explain string handling in C++ with examples.

String handing means manipulation of series of characters. In C++, there are two methods to do so:

1. **char** array

   In this method, we simply create an array of char and fill the last index with the null character ('\0'). The null character tells the compiler the end of the string.
   It supports all the C functions for string handling such as strlen(), strcpy(), strcmp(), strcat(), and strchr().
   Example:

```
char s1[99] = "Hello";
char s2[99] = "World";
char s3[99];
strlen(s1);
strcpy(s3,s1);          //gives length of s1
strcmp(s1,s3);          //copies s1 to s3
strcat(s1,s2);          //compares s1 and s3
strchr(s1, s3);         //puts s2 at the end of s1
                        //searches for s3 in s1
```

2. **string** class

   It is part of the string header file which is part of the standard C++ library. It has many advantages over the char array method, for instance, there is no need to specify the maximum length of the string. Strings can be declared and changed just like any other primitive data type.

```
string s1 = "Hello"; string s2 = "World"; string s3; length(s1);


                    //giveslength of s1
s3 = s1;    s1      //copiess1    to s3
s1.find(s3);        //compares s1 and s3
    += s2;          //putss2at    the end of s1
                    //searches for s3 in s1
```

5.3 Howarearraysinitializedin C++? Provideexamplesof both 1D and 2D arrays.

Arrays can be initialized in different ways in C++:

1. Compile time initialization: In this method, the array is initialized at the time of writing the code. There are multiple ways of doing it.

   (a) Complete array initialization: In this method, all the elements of the array are initialized.
   1D example:

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

   2D example:

```
int a[2][5] = { {1, 2,
    3, 4, 5} {6, 7, 8,
    9, 0}
};
```

   (b) Partial array initialization: In this method, only some of the elements of the array are initialized.
   1D example:

```
int a[5] = {1, 2, 3};
```

   2D example:

```
int a[2][5] = {
    {1, 2}
    {6, 7, 8}
};
```

   (c) Initialization without array size: In this method, the array is initialized without specifying its size. The compiler sets its size based on the number of elements it is initialized with.
   1D example:

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

   2D example:

```
int a[][5] = { {1, 2, 3,
    4, 5} {6, 7, 8, 9,
    0}
};
```

2. Run time initialization: In this method, the array is initialized by the user at the time of running the program. This allows to have different values for each element of the array without making changes in the code.
   1D example:

```
int a[5];
for (int i=0;i<5;i++){
    cin>>a[i];
}
```

2D example:

```
int a[2][5];
for (int i=0;i<2;i++){
    for (int j=0;i<5;i++){
    cin>>a[i][j];
    }
}
```

5.4 Explain string operations and functions in C++.

1. String Operations

(a) **=**: Used to assign a literal or another string's value to a string.

```
string s1 = "hello";          //assigns "hello" to s1
```

(b) **+**: Used to combine two strings.

```
string s1 = "hello" +     " world"; //stores "hello world" in s1
```

(c) **+=**: Used to add a string at the of the one stored in the string variable.

```
string s1 += " world";       //puts " world" at the end of string
   stored in s1
```

(d) Comparison operators: Used to compare two strings and returns either 1 or 0 based on the operator.

```
"hello" == "world"; //retuns 0
"hello" != "world"; //retuns 1
"hello" < "world"; //retuns 1
"hello" > "world"; //retuns 0
"hello" <= "world"; //retuns 1
"hello" >= "world"; //retuns 0
```

(e) Array index **[]** : Used to access or modify a character of a string at the specified index

```
s1[3]='a';              //changes the 4th letter of the string stored
   in s1 to 'a'
```

2. String functions

(a) **length()**: Used to find the length of a string stored in a string variable. Returns a non-negative integer.

```
s1.length();        //returns the length of the string stored in
    s1
```

(b) **at()** : Used for the same purpose as the array index operator [].

```
s1.at(3);       //retunrs the 4th character of the string stored
    ins1
```

(c) **append()**: Used for the same purpose as the + op erator.

```
s1.append("world"); //appends "world" at the end of the string
    stored in s1
```

(d) **compare()**: Used for the same purpose as the comparison operators. Returns 0 if both string are equal, a positive number if the first string is greater than the second, and a negative number if the first string is smaller than the second.

```
s1.compare("world");      comparesstring stored in s1 with "
    world"
```

(e) **find()** : Used to find a specified character in a string stored in a string variable.

```
s1.find('a');       //returns the index where the specified
    character is found in the string stored in s1
```

(f) **substr()**: Used to generate a new string from a given string from a specified index number and of specified length.

```
s1.substr(3,5);         //returns a ne string formedfrom the
    string stored in s1 from the w character to the 8th one
                                    4t
                                    h
```

(g) **insert()**: Used to add a string at a specified index to a string stored in a string variable.

```
s1.insert(3, "abc");            //adds "abc" at the 3rd index of s1
```

(h) **replace()**: Used to replace part of a string stored in a string variable at a specified index and of specified length with a specified string.

```
      s1.replace(1,3,"abc");    //replaces the 2nd to the 4th
    characters in s1 with "abc"
```

(i) **erase()**: Used to remove part of a string stored in a string variable starting from a specified position and of specified length.

```
s1.erase(1, 3); //removes     2nd to 4th characters in s1
```

(j) **c_str()** : Used to convert a string object to a character array.

```
s1.c_str(); //returns     string in s1 as a character array
```

6 Introduction to Object-Oriented Programming

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

1. Class: It is a user defined data type which encompasses data members (variables) and member functions (methods). It act as a template for a group of object which have same properties and behaviors.

2. Object: It is a variable of a class. It has specific values for the data members of the class and run the functions defined in the class.

3. Inheritance: It is a property of class which allows one class access the variables and functions of other class so we do not have to rewrite them in the new class if we need to use them again in it. A single class can be inherited by multiple classes and a single class can inherit multiple classes. Also, a class which has inherited some other class, can also be inherited by a third class.

4. Polymorphism: It allows different functions with the same name to be declared and called without having any errors. This is done either by having different types or numbers of parameters for each function with the same name, or by having them in different classes and using access the scope resolution operator (::) when calling a function from a specified class.

5. Encapsulation: It is combining data and the methods that use it in a single unit called class.

6. Abstraction: This allows to hide the inner workings of the program from the user and only display the essential things. This is achieved by using access specifiers.

6.2 WhatareclassesandobjectsinC++?Provideanexample.

A class is a user-defined data type which is similar to structure but it can also contain functions. To access a class's data members and member functions one needs to create the class's object. An object is basically an instance of a class. Without object, the data members and the member functions of a class cannot be outside the class. To access them, direct member access operator (.) is used. Data members and members functions in a class need to be under public access specifier to be able to accessed outside the class.
#include #include

```
1          <iostream>
2          <string>
3  usin    namespace std;
4  g       car{
5  class private :
6          string name;
7          int  year;
8      public:
9           display(string n,  int  y){
10              name = n;
11               year = y;
```

```
            cout<<"Car name: "<<name<<endl;
            cout<<"Release year:  "<<year<<endl;
        }
};
main(){
car     obj;
    obj.display("Toyota", 1999);
}
```

6.3 WhatisinheritanceinC++?Explainwithanexample.

Inheritance is a property of a class in C++ to access data members and member function from another class(es) so the developer do not have to rewrite them again. The class which is being inherited by another class is called the base class and the class which inherits another class is called the derived class. Inheritance is achieved by using colon operator (:) followed by access specifier and the base class.

Syntax:

```
class derived_class: access_specifier   base_class{
    //derived_class's data members and memberfunctions
};
```

There are three modes of inheritance based on access specifiers:

1. Public: All the public and protected data members and member functions are accessible in the derived class and any class that inherits the derived class but only public ones are accessible outside the derived class.

2. Private: All the public and protected data members and member functions are accessible in the derived class and any class that inherits the derived class but none are accessible outside the derived class.

3. Protected: All the public and protected data members and member functions are accessible in the derived class but none are accessible by any class that inherits the derived class or anywhere outside the derived class.

There are five types of inheritance in C++:

1. Single inheritance: In this type, only one class is inherited by only one other class.
Example:

```
class derivedClass: public baseClass{
    //derived class members
}
```

2. Multilevel inheritance: In this type, a derived class is inherited by another class.
Example:

```
class derivedClass: public baseClass{
    //derived class members
}
class secondLevelDerivedClass: public derivedClass{
```

```
    //second level derived   class  members
}
```

3. Multiple inheritance: In this type, a class inherits from multiple classes.
   Example:

```
class derivedClass: public baseClassA, public baseClassB{
    //derived class members
}
```

4. Hierarchic inheritance: In this type, a class is inherited by multiple classes.
   Example:

```
class derivedClass1: public baseClass{
    //derived class 1's members
}
class derivedClass2: public baseClass{
    //derived class 2's members
}
```

5. Hybrid inheritance: This type is a combination of any of the other types of inheritances.
   Example:

```
class derivedClass1: public baseClass{
    //derived class 1's members
}
class derivedClass2: public baseClass{
    //derived class 2's members
}
class secondLevelDerivedClass: public derivedClass1,   public
    derivedClass1{
    //second level derived   class  members
}
```

6.4 What is encapsulation in C++? How is it achieved in classes?

Encapsulation is a concept that allows binding together of data and functions that manipulate said data into a single entity called class. It also allows to hide them from rest of the code if needed.
It is achieved through the use of classes. A class allows the bundling of data members and member functions. Access specifiers such as private, protected, and public control the visibility of class members. Typically, data members are declared private and can only be accessed or modified through public member functions, called getters and setters.

Example:

```
class Employee{
private :
```

```
    int salary;
public:
    setSalary(int s) {
        salary = s;
    }
    int getSalary(){
        return salary;
    }
};
```

In this example, the data member salary is encapsulated within the Employee class. Direct access to salary is restricted, and it can only be accessed or modified via the public member functions setSalary() and getSalary().