

Tutorial 1 CS203 (CSE)

Module 1

Topic: Introduction: Principles of OOPs, Basics of C++, Functions in c++, Basic Concepts of OOP, Benefits of OOP, OOP Languages, Applications of OOP

MCQ

Q. (1) Why is reusability a desirable feature?

- (a) Reduces compilation time
- (b) Decreases testing time
- (c) Lowers maintenance cost
- (d) None

Ans: (b) Reusability is a desirable feature as it decreases the testing time.

Q. (2) How do encapsulation and abstraction differ?

- (a) Hiding and Binding
- (b) Binding and Hiding
- (c) Hiding and Hiding
- (d) None

Ans: (b) Encapsulation and abstraction differ on the basis of binding and hiding.

Q. (3) Under which pillar of OOPS do base class and derived class relationships come?

- (a) Encapsulation
- (b) Polymorphism
- (c) Abstraction
- (d) Inheritance

Ans: (d) Base class and derived class relationship come under inheritance.

Q. (4) What is an object in c++?

- (a) It is a function of class
- (b) It is the data type of class
- (c) It is an instance of the class
- (d) It is part of the syntax of class

Ans: (c) An object is an instance of the class.

Q. (5) Identify the feature using which, one object can interact with another object.

- (a) Message passing

- (b) Message reading
- (c) Data binding
- (d) Data transfer

Ans: (a) Message passing is the feature using which, one object can interact with another object.

Descriptive Questions

Q. (1) What is the difference between public, private and protected access modifiers?

Ans:

Name	Accessibility from own class	Accessibility from derived class	Accessibility from world
Public	Yes	Yes	Yes
Private	Yes	No	No
Protected	Yes	Yes	No

Q. (2) What is the application of OOPs?

Ans:

- Real-Time System design: Real-time system inherits complexities and makes it difficult to build them. OOP techniques make it easier to handle those complexities.
- Hypertext and Hypermedia: Hypertext is similar to regular text as it can be stored, searched, and edited easily. Hypermedia on the other hand is a superset of hypertext. OOP also helps in laying the framework for hypertext and hypermedia.
- AI Expert System: These are computer application that is developed to solve complex problems which are far beyond the human brain. OOP helps to develop such an AI expert System
- Office automation System: These include formal as well as informal electronic systems that primarily concerned with information sharing and communication to and from people inside and outside the organization. OOP also help in making office automation principle.
- Neural networking and parallel programming: It addresses the problem of prediction and approximation of complex-time varying systems. OOP simplifies the entire process by simplifying the approximation and prediction ability of the network.
- Stimulation and modeling system: It is difficult to model complex systems due to varying specifications of variables. Stimulating complex systems require modeling and understanding interaction explicitly. OOP provides an appropriate approach for simplifying these complex models.
- Object-oriented database: The databases try to maintain a direct correspondence between the real world and database object in order to let the object retain it identity and integrity

Q. (3) Difference between encapsulation and inheritance.

Ans:

Inheritance	Encapsulation
Inheritance is the process or mechanism by which you can acquire the properties and behaviour of a class into another class.	Encapsulation refers to the winding of data into a single unit which is known as class.
Inheritance indicates that a child class (subclass) inherits all the attributes and methods from a parent class (superclass).	Encapsulation indicates that one class must not have access to the (private) data of another class.

Q. (4) What is the difference between a class and an object?

Ans:

Object	Class
A real-world entity which is an instance of a class	A class is basically a template or a blueprint within which objects can be created
An object acts like a variable of the class	Binds methods and data together into a single unit
An object is a physical entity	A class is a logical entity
Objects take memory space when they are created	A class does not take memory space when created
Objects can be declared as and when required	Classes are declared just once

Q. (5) Give some real-world problem where oops is used.

Ans: Example 1 – We all have been using ATMs. And if we want to withdraw the money from that then we simply interact with the Use Interface Screen where we have been asking for details.

As a user, we only want that I can use that ATM for the services. We are not interested in knowing the internal implementation that what is the process happening inside the ATM Machine like getting card details, validation, etc kind of things. We only have been provided an interface to interact with that and simply ask for the work. And the internal implementation we don't need to know. So that is the Abstraction.

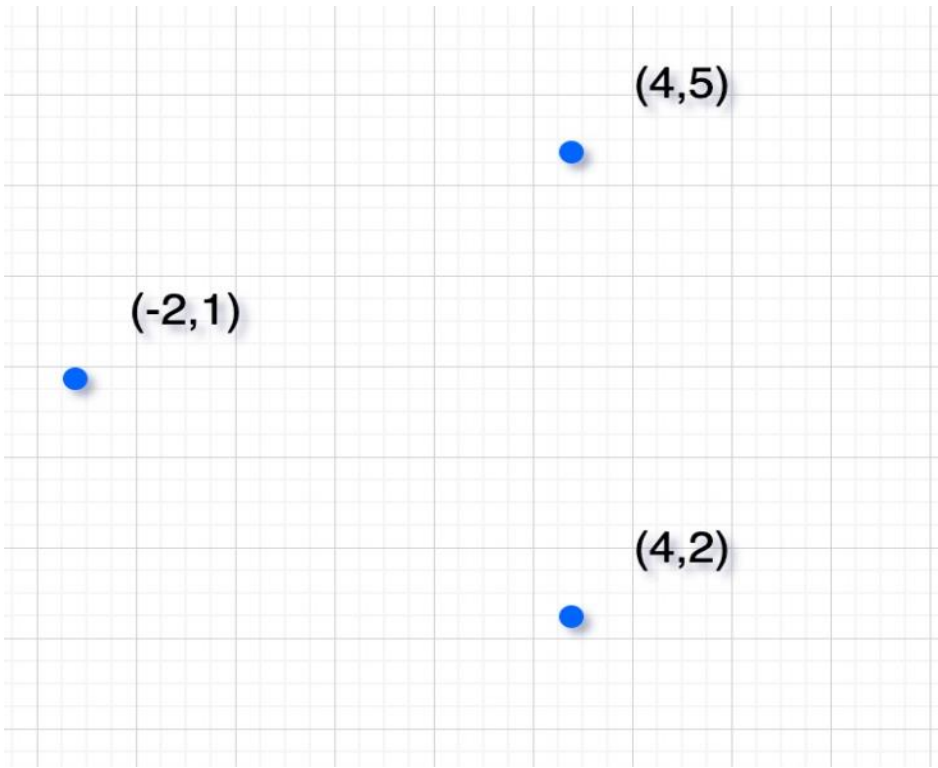
Example 2 – We all have seen the updates which receive on gadgets both in terms of hardware and software. Like firstly mobile phones are their only to talk and then used for media access, Internet, Camera, etc. So these are evolution that arrives by adding some extra feature and making a new product without rebuilding the complete functionality so it is an example of inheritance. Similarly, in software, regular updates are with some new features are also the inheritance.

Example 3 - The most commonly used mobile phones. On a mobile phone, we can perform so many actions like making a call, sending messages, take pictures, download software and etc. We perform a lot of things but here also we don't know the inside process of these things. Which means the implementation parts are hidden.

HOTS Question

Question 1 : There are two friends, Aman and Shubham. Shubham is weak in Euclidian mathematics, on the other hand Aman is good in it. Shubham encounters a problem on Codechef which requires the basic knowledge of Euclidian geometry. While solving the problem, he is stuck in the middle of it, so he decides to take help from Aman. The problem goes as follows:-

In problem there are three coordinates on Cartesian plane naming (x_1, y_1) , (x_2, y_2) and (x_3, y_3) . Find which type of triangle (Equilateral, Isosceles, Scalene) will be formed by these three coordinates. If no triangle is formed then print "Invalid triangle".



Code :-

```

#include <bits/stdc++.h>
using namespace std;

class Triangle{
private:
    int sideA, sideB, sideC;

public:
    vector<pair<int, int>> pr;
    void coordinate(int x, int y){
        pr.push_back({x, y});
    }

    int a = 0, b = 0, c = 0;

    void find_side(){

        a = sqrt(((pr[0].first -
pr[1].first)*(pr[0].first - pr[1].first)) +
((pr[0].second - pr[1].second)*(pr[0].second -
pr[1].second)));
        b = sqrt(((pr[1].first -
pr[2].first)*(pr[1].first - pr[2].first)) +
((pr[1].second - pr[2].second)*(pr[1].second -
pr[2].second)));
        c = sqrt(((pr[0].first -

```

```

    bool valid_triangle()
    {
        if (sideA + sideB > sideC && sideB + sideC >
sideA && sideC + sideA > sideB)
        {
            return true;
        }
        return false;
    }

    void find_triangle()
    {
        if (!valid_triangle())
        {
            cout << "Invalid Triangle" << endl;
        }
        else
        {
            if (sideA == sideB && sideB == sideC)
                cout << "Equilateral Triangle" <<
endl;
            else if (sideA == sideB || sideB ==
sideC || sideC == sideA)
                cout << "Isosceles Triangle" <<
endl;
            else
                cout << "Scalene Triangle" << endl;
        }
    }
};

```

```

int main(){
    #ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    #endif

    Triangle T;

    // Coordinate of triangle

    cout << "Enter coordinate of triangle" << endl;

    for (int i = 1; i <= 3; i++){
        cout << "Enter " << i << " coordinate of triangle ";
        int x, y;
    }
}

```

```

    cin >> x >> y;
    T.coordinate(x, y);
    cout << x << " " << y << endl;
}

T.find_side();

T.find_triangle();
}

```

Topic: C++ program basics, data types, operators in c++, scope resolution, type cast operators, operator overloading, operator precedence

MCQ

Q. (1) Which of the statements are true or false:

Statement 1 : real destructor exist

Statement 2 : virtual destructor exist

- (a) Statement 1 is true and statement 2 is false
- (b) Statement 1 is false and statement 2 is true
- (c) Both Statements 1 and 2 are true
- (d) Both Statements 1 and 2 are false

Ans: (b) Statement 1 is false and statement 2 is true

Q. (2) What is this operator called "? : " ?

- (a) conditional
- (b) relational
- (c) casting operator
- (d) unrelational

Ans: (a) conditional

Q. (3) Which of the following operators cannot be overloaded?

- (a) +
- (b) ++
- (c) --
- (d) ::

Ans: (d) ::

Q. (4) India is _____ variable to Bihar

- (a) global
- (b) local
- (c) primitive
- (d) none

Ans: (a)global

Q. (5) Identify the user-defined types from the following?

- (a) enumeration
- (b) classes
- (c) both enumeration and classes
- (d) int

Ans: (c) both enumeration and classes

Descriptive Questions

Q. (1) Is it possible for a C++ program to be compiled without the main() function?

Ans: Yes, it is possible. However, as the main() function is essential for the execution of the program, the program will stop after compiling and will not execute.

Q. (2) What is operator overloading?

Ans: Operator overloading is a necessary component for performing operations on user-defined data types. We can change the default meaning of operators like +, -, *, /, =, and so on by overloading them.

For example:

Using operator overloading, the following code adds two complex numbers:

```
class complex{
private:
    float r, i;
public:
    complex(float r, float i){
        this->r=r;
        this->i=i;
    }
    complex(){ }
    void displaydata(){
        cout<<"real part = "<<r<<endl;
        cout<<"imaginary part = "<<i<<endl;
    }
    complex operator+(complex c){
        return complex(r+c.r, i+c.i);
    }
}
```



```
};
int main(){
complex a(2,3);
complex b(3,4);
complex c=a+b;
c.displaydata();
return 0;
}
```

Q. (3) Give a real-life example on private and public access modifier.

Ans: Let us take example of our bank account our personal details can be only seen by you and some trusted bank officers other account holders can't see your personal details in this case your account is private to other account holders.

Let us take example of our college class our classmates roll number the name and roll number is public to our college.

Q. (4) What are the advantages of C++?

Ans:

- Mid-level programming language.
- Portability.
- C++ has the concept of inheritance.
- Multi-paradigm programming language.
- Memory Management.
- C++ is a highly portable language.
- Fast and Powerful.
- C++ contains a rich function library.

Q. (5) Write a C++ program to overload unary operators that is increment and decrement.

Ans: Operator overloading is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. It is used to perform operation on user-defined data type.

Following program is overloading unary operators: increment (++) and decrement (--).

```
#include<iostream>
using namespace std;
class IncreDecre
{
    int a, b;
public:
    void accept()

    {
        cout<<"\n Enter Two Numbers : \n";
```

```

        cout<<" ";
        cin>>a;
        cout<<" ";
        cin>>b;
    }
    void operator--() //Overload Unary Decrement
    {
        a--;
        b--;
    }
    void operator++() //Overload Unary Increment
    {
        a++;
        b++;
    }
    void display()
    {
        cout<<"\n A : "<<a;
        cout<<"\n B : "<<b;
    }
};
int main()
{
    IncreDecre id;
    id.accept();
    --id;
    cout<<"\n After Decrementing : ";
    id.display();
    ++id;
    ++id;
    cout<<"\n\n After Incrementing : ";
    id.display();
    return 0;
}

```

Topic: Main function, function prototyping, call by reference, inline functions, default arguments, constant arguments

MCQ

Q. (1) Which of the following is correct about the second parameter of the main function?

- (a) Second parameter is an array of character pointers
- (b) First string of the list is the name of the program's output file
- (c) The string in the list is separated by space in the terminal

(d) All of the mentioned

Ans: (d)

Explanation: All of the statements about the second parameter are correct. It is the collection of character pointers of which the first represents the name of the program file.

Q. (2) What is the difference between references and pointers?

- (a) References are an alias for a variable whereas pointer stores the address of a variable
- (b) References and pointers are similar
- (c) References stores address of variables whereas pointer points to variables
- (d) Pointers are an alias for a variable whereas references store the address of a variable

Ans: (a)

Explanation: References are an alias/another name for a variable whereas a pointer stores the address of a variable. Pointers need to be dereferenced before use whereas references need not.

Q. (3) Pick the correct statement about references.

- (a) References can be assigned value NULL
- (b) References once assigned cannot be changed to refer another variable
- (c) Reference should not be initialized when created
- (d) Reference is the same as pointers

Ans: (b)

Explanation: References should be initialized during its creation and once assigned cannot be changed to refer to another variable. References cannot be assigned a NULL value. References and pointers are two different concepts.

4. What is an inline function?

- (a) A function that is expanded at each call during execution
- (b) A function that is called during compile time
- (c) A function that is not checked for syntax errors
- (d) A function that is not checked for semantic analysis

Ans: (a)

Explanation: Inline functions are those which are expanded at each call during the execution of the program to reduce the cost of jumping during execution.

Q. (5) In which of the following cases inline functions may not work?

- i. If the function has static variables.
 - ii. If the function has global and register variables.
 - iii. If the function contains loops
 - iv. If the function is recursive
- (a) i, iv
 - (b) iii, iv
 - (c) ii, iii, iv
 - (d) i, iii, iv

Ans: (d)

Explanation: A function is not inline if it has static variables, loops or the function is having any recursive calls.

Q. (6) Which of the following is the default return value of functions in C++?

- (a) int
- (b) char
- (c) float
- (d) void

Ans: (a)

Explanation: C++ uses int as the default return values for functions.

Q. (7) Which of the following features is used in function overloading and function with default argument?

- (a) Encapsulation
- (b) Polymorphism
- (c) Abstraction
- (d) Modularity

Ans: (b)

Explanation: Both of the above types allow a function overloading which is the basic concept of Polymorphism.

Q. (8) What happens to a function defined inside a class without any complex operations (like looping, a large number of lines, etc)?

- (a) It becomes a virtual function of the class
- (b) It becomes a default calling function of the class
- (c) It becomes an inline function of the class
- (d) The program gives an error

Ans: (c)

Explanation: Any function which is defined inside a class and has no complex operations like loops, a large number of lines then it is made inline.

Q. (9) What will be the output of the following C++ code?

```
#include<iostream>
using namespace std;

int fun(int x = 0, int y = 0, int z)
{ return (x + y + z); }

int main()
{
    cout << fun(10);
    return 0;
}
```

- a) 10
- b) 0
- c) Error
- d) Segmentation fault

Answer: c

Explanation: Default arguments should always be declared at the rightmost side of the parameter list but the above function has a normal variable at the rightmost side which is a syntax error, therefore the function gives an error.

Q. (10) What will be the output of the following C++ code?

```

#include <iostream>
using namespace std;

int fun(int=0, int = 0);

int main()
{
    cout << fun(5);
    return 0;
}
int fun(int x, int y) { return (x+y); }

```

- (a) -5
- (b) 0
- (c) 10
- (d) 5

Ans: (d)

Explanation: C++ allows defining such a prototype of the function in which you are not required to give variable names only the default values. While in function definition you can provide the variable names corresponding to each parameter.

HOTS:

Suggest the most efficient method to handle default arguments out of order in a C++ function with many arguments.

Arguments are copied from left to right. Default arguments are always to the right of the parameter list. The arguments are accessed in the order specified in the Function Prototype.

These restrictions can make it difficult to handle functions with a large number of default arguments.

Consider the following function:

```

void f_name(bool option1 = false, bool option2 = false,
            . . . . .
            bool option10 = false);

```

Let's say we want to change the value of the last option to true. Normally, we have to change it like this in the func call.

```
f_name(false, false . . . . . true);
```

As we can see, just to toggle the value of one argument we had to handle all of those that appeared before it. It becomes more tedious as the number of arguments increases. To mitigate this problem, we can use a derived data structure like struct or class!

For example,

```
struct options
{
    bool optArray[10] = false;
};
```

While using this during function prototyping, we will pass this structure instead of 10 variables.

```
void f_name(options opts);
```

As compared to before to change one parameter we can simply do this.

```
options opts;
opts.optArray[9] = true;
f_name(opts);
```

Descriptive Questions

Q. (1) Do inline functions improve performance?

Ans: Yes and no. Sometimes. Maybe.

There are no simple answers.

inline functions might make the code faster, they might make it slower. They might make the executable larger, they might make it smaller. They might cause thrashing; they might prevent thrashing. And they might be, and often are, totally irrelevant to speed.

inline functions might make it faster: As shown above, procedural integration might remove a bunch of unnecessary instructions, which might make things run faster.

inline functions might make it slower: Too much inlining might cause code bloat, which might cause “thrashing” on demand-paged virtual-memory systems. In other words, if the executable size is too big, the system might spend most of its time going out to disk to fetch the next chunk of code.

Q. (2) What is the difference between arguments and parameters?

Argument	Parameter
The values that are declared within a function when the function is called are known as an argument.	The variables that are defined when the function is declared are known as parameters.
These are used in function call statements to send value from the calling function to the receiving function.	These are used in the function header of the called function to receive the value from the arguments.
During the time of call each argument is always assigned to the parameter in the function definition.	Parameters are local variables which are assigned values of the arguments when the function is called.
They are also known as Actual Parameters.	They are also known as Formal Parameters.

Q. (3) What is the purpose of a function prototype ?

Ans: The Function prototype serves the following purposes –

1. It tells the return type of the data that the function will return.
2. It tells the number of arguments passed to the function.
3. It tells the data types of each of the passed arguments.
4. Also it tells the order in which the arguments are passed to the function. Therefore essentially, the function prototype specifies the input

Q. (4) What is Call by reference in C++?

Ans: The call by reference method in C++ of passing arguments to a function will copy the reference of an argument into the formal parameter. Moreover, inside the function, the reference comes into use for accessing the actual argument used in the call. Meaning to say that the changes made to the parameter impact the passed argument.

Q. (5) Why do we use references in C++?

Ans: C++ adds the so-called reference variables or references in short). The major use of references is that it acts as formal parameters for supporting pass-by-reference. In reference, a variable pass into a function. After that, the function works on the original copy and not a clone copy is pass-by-value.

Topic: function overloading, friend and virtual functions, maths library functions

MCQ

Q. (1) What will be the output of the following C++ code?

```
#include <iostream> using namespace std; class Box
{
double width; public:
friend void printWidth( Box box ); void setWidth( double wid );
};
void Box::setWidth( double wid )
{
width = wid;
}
void printWidth( Box box )
{
box.width = box.width * 2;
cout << "Width of box : " << box.width << endl;
}
int main( )
{
```

```
Box box; box.setWidth(10.0); printWidth( box );  
return 0;  
}
```

- (a) 40
- (b) 5
- (c) 10
- (d) 20

Ans: (d)

Explanation: We are using the friend function for printwidth and multiplied the width value by 2, So we got the output as 20

Q. (2) Pick out the correct statement.

- (a) A friend function may be a member of another class
- (b) A friend function may not be a member of another class
- (c) A friend function may or may not be a member of another class
- (d) None of the mentioned

Ans: (c)

Explanation: A friend function may or may not be a member of another class is the correct statement.

Q. (3) The virtual function is used to tell the compiler to perform ?

- (a) static linkage
- (b) dynamic linkage
- (c) late binding
- (d) Both (b) and (c)

Ans: (d)

Q. (4) Predict output of the following program

```
#include<iostream> using namespace std;  
class Base  
{  
public:  
virtual void show() { cout<<" In Base \n"; }  
};  
class Derived: public Base  
{  
public:  
void show() { cout<<"In Derived \n"; }  
};
```



```
int main(void)
{
Base *bp = new Derived; bp->show();
```

```
Base &br = *bp;
br.show();
return 0;
}
```

In Base

In Base

In Base

In Derived

In Derived In Derived

In Derived In Base

Answer: c

Explanation: Since show() is virtual in base class, it is called according to the type of object being referred or pointed, rather than the type of pointer or reference.

Q. (5) What will be the output of the following C++ code?

```
#include <iostream> using namespace std;
```

```
int Add(int X, int Y, int Z)
```

```
{
return X + Y;
}
```

```
double Add(double X, double Y, double Z)
```

```
{
return X + Y;
}
```

```
int main()
```

```
{
cout << Add(5, 6); cout << Add(5.5, 6.6); return 0;
}
```

(a) 11 12.1

(b) 12.1 11

(c) 11 12

(d) compile time error

Ans: (d)

Explanation: As one can observe that no function has declaration similar to that of called Add(int, int) and Add(double, double) functions. Therefore, error occurs.

Q. (6) Which of the following permit's function overloading on c++?

- (a) type
- (b) number of arguments
- (c) type & number of arguments
- (d) number of objects

Explanation: Both type and number of arguments permits function overloading in C++, like
int func(int);

float func(float, float)

Here both type and number of arguments are different.

Q. (7) Which of the following in Object Oriented Programming is supported by Function overloading and default arguments features of C++.

- (a) Polymorphism
- (b) Encapsulation
- (c) Inheritance
- (d) None of the above

Explanation: Both of the features allow one function name to work for different parameter

Q. (8) In which of the following we cannot overload the function?

- (a) caller
- (b) called function
- (c) return function
- (d) none of the mentioned

Explanation: While overloading the return function, it will rise a error, So we can't overload the return function.

Q. (9)The C library function double fmod(double x, double y) returns the remainder of x divided by y.

- (a) log10(double x, double y)
- (b) fabs(double x, double y)
- (c) fmod(double x, double y)
- (d) floor(double x, double y)

Ans: (c)

Explanation: double fmod(double x, double y) : The C library function double fmod(double x, double y) returns the remainder of x divided by y.

Q. (10) Which function returns the arc sine in the range $[-\pi/2, +\pi/2]$ radians?

- (a) arcsin()
- (b) asin()
- (c) sin()
- (d) asine()

Descriptive Questions

Q. (1) What is virtual function? Explain with an example.

Ans: A virtual function is a member function that is declared within a base class and redefined by a derived class. To create virtual function, precede the function's declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs.

➤ Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism.

Consider the following program code : Class A

```
{
int a; public:
A()
{
a = 1;
}
virtual void show()
{
cout <<a;
}
};
Class B: public A
{
int b; public:
B()
{
b = 2;
}
virtual void show()
{
```

```

cout <<b;
}
}
int main()
{
A *pA; B oB;
pA = &oB; pA→show(); return 0;
}

```

Output is 2 since pA points to object of B and show() is virtual in base class A.

Q. (2) What are the advantages of the friend function in C++?

Ans: Some of the advantages of the friend function in c++ are

- It enables a non-member function to share confidential class information.
- It makes it simple to access a class's private members.
- It is frequently used when two or more classes include members that are connected to other programme elements.
- It enables the creation of more effective code.
- It offers extra capabilities that the class does not typically use.
- It permits a non-member function to share confidential class information.

1. What are the limitations of friend function?

The major disadvantage of friend functions is that they occupy the maximum size of the memory and can't do any run-time polymorphism concepts.

2. What are the advantages of function overloading?

- The fundamental benefit of function overloading is that it makes code easier to read and reuse.
- Function overloading is used to improve consistency, readability, and memory efficiency.
- The program's execution is accelerated by it.
- Additionally, code upkeep becomes simple.
- Function overloading gives code flexibility.
- The function's versatility avoids the need for many function names to conduct the same set of functions.

Q. (3) What is the difference between virtual function and pure virtual function?

Ans:

Virtual function	Pure virtual function
A virtual function is a member function in a base class that can be redefined in a derived class.	A pure virtual function is a member function in a base class whose declaration is provided in a base class and implemented in a derived class.
The classes which are containing virtual functions are not abstract classes.	The classes which are containing pure virtual function are the abstract classes.
In case of a virtual function, definition of a function is provided in the base class.	In case of a pure virtual function, definition of a function is not provided in the base class.
The base class that contains a virtual function can be instantiated.	The base class that contains a pure virtual function becomes an abstract class, and that cannot be instantiated.
If the derived class will not redefine the virtual function of the base class, then there will be no effect on the compilation.	If the derived class does not define the pure virtual function; it will not throw any error but the derived class becomes an abstract class.
All the derived classes may or may not redefine the virtual function.	All the derived classes must define the pure virtual function.

(4) How do you explain the difference between overloading and overriding?

Ans:

Overloading a method means that multiple methods share the same method name but have different arguments. However, in the case of the overriding, the child class can redefine the implementation of a method by retaining the same arguments. Another difference is that the overloading is resolved at compile-time while overriding is resolved at run time

Q. (5) Given an example of function overloading with parameter with different sequence and parameter with different argument.

Ans:

1.function overloading with parameter with different argument

```
#include <iostream>
using namespace std;

void add(int a, int b)
{
    cout << "sum = " << (a + b);
}

void add(double a, double b)
{
    cout << endl << "sum = " << (a + b);
}

// Driver code
int main()
{
    add(10, 2);
    add(5.3, 6.2);

    return 0;
}
```

2. function overloading with parameter with different sequence

```
#include<iostream>
using namespace std;

void add(int a, double b)
{
    cout<<"sum = "<<(a+b);
}
```

```

void add(double a, int b)
{
    cout<<endl<<"sum = "<<(a+b);
}

// Driver code
int main()
{
    add(10,2.5);
    add(5.5,6);

    return 0;
}

```

HOTS

1. Predict the output of following C++ program?

```

#include<iostream>
using namespace std;
class Test
{
protected:
int x;
public:
Test (int i):x(i) { }
void fun() const { cout << "fun() const " << endl; }
void fun() { cout << "fun() " << endl; }
};
int main()
{
    Test t1 (10);
    const Test t2 (20);
    t1.fun();
    t2.fun();
    return 0;
}

```

Ans: fun()
 fun() const

Explanation:- The two methods ‘void fun() const’ and ‘void fun()’ have same signature except that one is const and other is not. Also, if we take a closer look at the output, we observe that, ‘const void fun()’ is called on const object and ‘void fun()’ is called on non-const object. C++ allows member methods to be overloaded on the basis of const type. Overloading on the basis of const type can be useful when a function return reference or pointer. We can make one function const, that returns a const reference or const pointer, other

non-const function, that returns non-const reference or pointer. See following for more details. Function overloading and const keyword

Q. (2) Explain how this following code works ? and also indicate the output of the code given.

```
#include<iostream>
using namespace std;
void square (int *x)
{
    *x = (*x)++ * (*x);
}
void square (int *x, int *y)
{
    *x = (*x) * --(*y);
}
int main ( )
{
    int number = 30;
    square(&number, &number);
    cout << number;
    return 0;
}
```

Ans: In the following given code the second definition of the square will be overloaded.

Because of square(&number, &number);

function square(int *x, int *y) will be called

.It has:

*x= (*x) * --(*y);

where *x = 30 and *y = 30

-- (decrement) operator has the higher precedence than * (multiplication).

Hence:

*x = 30 * 29s

Module 2

Topic: Classes, objects, constructors and destructors – C structures revisited

MCQ

Q. (1) What will be the output of the following program?

```
#include <iostream>
struct school
{
    double a;
    int b;
```



```

char h;
int c;
char d;
};
int main()
{
    school students;
    printf("size of struct: %d \n",sizeof(students));
    return 0;
}

```

- (a) 18
- (b) 20
- (c) 22
- (d) 24

Ans: (d) 24

Size of Structure is 24. Due to Padding all char variable are also assigned 4 bytes.

Q. (2) What will be the output of the following program?

```

#include<bits/stdc++.h>
using namespace std;
class student{
    public:
    student(){
        cout<<sizeof(student)<<endl;
    }
};
int main(){
    student s1;
    return 0;
}

```

- (a) 0 byte
- (b) 1byte
- (c) 2bytes
- (d) 4bytes

Ans: (b) 1bytes

Explanation:

To inform the existence of the object in the class

Q3. By default, access modifier is?

- (a) Public
- (b) Protected
- (c) Private
- (d) All of the above

Ans. (c)

```
Q4) class X
{
public:
void someFunction (X *x)
{
    this = x;
}
}
```

The code above will not compile since assignment to this parameter is not allowed in C++. This parameter cannot be modified, because it always refers to the current object.

Q. (4) What will be the output ?

```
#include <iostream>
using namespace std;
class Employee {
public: int id;
    string name;
    float salary;
    Employee(int id, string name, float salary)
    {
        id = id;
        name = name;
        this->salary = salary;
    }
    void display()
    {
        cout<<id<<" "<<name<<" "<<salary<<endl;
    }
};
int main(void) {
```

```

        Employee e1 =Employee(101, "Sonoo", 890000);
Employee e2=Employee(102, "Nakul", 59000);
e1.display();
e2.display();
return 0;
}
Ans)

```

- (a) 101, "Sonoo", 890000
- (b) 102, "Nakul", 59000
- (c) 101, "Sonoo", 890000
- (d) Garbage value
- (e) none of the above

Ans: (c)

Descriptive Questions

Q. (1) What is this pointer? Explain its use with an example.

Ans: this is a keyword that refers to the current instance of the class. There can be 3 main usages of this keyword in C++.

- It can be used to pass current object as a parameter to another method.
- It can be used to refer current class instance variable.
- It can be used to declare indexers.

C++ this Pointer Example

Let's see the example of this keyword in C++ that refers to the fields of current class.

```

#include <iostream>

using namespace std;

class Employee {

public:

    int id; //data member (also instance variable)

    string name; //data member(also instance variable)

    float salary;

    Employee(int id, string name, float salary)

    {

```

```

        this->id = id;

        this->name = name;

        this->salary = salary;

    }

    void display()

    {

        cout<<id<<" "<<name<<" "<<salary<<endl;

    }

};

int main(void) {

    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee

    Employee e2=Employee(102, "Nakul", 59000); //creating an object of Employee

    e1.display();

    e2.display();

    return 0;

}

```

Output:

101 Sonoo 890000

102 Nakul 59000

Q. (2) Explain the terms Class and Object with the help of real world example

Ans. Consider the Automobile Class. There may be many automobiles with different names and brands, but they will all have some basic characteristics, such as four wheels, a speed limit, and a range of mileage. The class is Car, and the attributes are wheels, speed limitations, and mileage.

A Class is a data type with data members and member functions that is defined by the user.

Data members are data variables, and member functions are functions that manipulate these variables; these data members and member functions together define the properties and behaviour of the objects in a Class.

The data members in the previous example of class Car will be speed limit, mileage, and member methods will be able to apply brake.

Q. (3) Mention the real-world sectors where we use oops.

Ans: The following industries have the most need for OOP developers:

- Services in the financial sector
- Healthcare
- technologically sophisticated
- Services provided by professionals
- Investing in real estate
- E-commerce and retail

Q. (4) Define a class to represent a bank account. Include the following members:

Data members:

- 1) Name of the depositor
- 2) Account number
- 3) Type of account
- 4) Balance amount in the account.

Member functions:

- 1) To assign initial values
- 2) To deposit an amount
- 3) To withdraw an amount after checking the balance
- 4) To display name and balance.

Write a main program to test the program.

Ans:

```
#include<iostream>
#include<stdio.h>
#include<string.h>
```

```
using namespace std;
```

```
class bank
```

```
{
    int acno;
    char nm[100], acctype[100];
    float bal;
public:
    bank(int acc_no, char *name, char *acc_type, float balance) //Parameterized Constructor
    {
        acno=acc_no;
        strcpy(nm, name);
        strcpy(acctype, acc_type);
    }
}
```

```

        bal=balance;
    }
    void deposit();
    void withdraw();
    void display();
};

void bank::deposit() //depositing an amount
{
    int damt1;
    cout<<"\n Enter Deposit Amount = ";
    cin>>damt1;
    bal+=damt1;
}

void bank::withdraw() //withdrawing an amount
{
    int wamt1;
    cout<<"\n Enter Withdraw Amount = ";
    cin>>wamt1;
    if(wamt1>bal)
        cout<<"\n Cannot Withdraw Amount";
    bal-=wamt1;
}

void bank::display() //displaying the details
{
    cout<<"\n -----";
    cout<<"\n Accout No. : "<<acno;
    cout<<"\n Name : "<<nm;
    cout<<"\n Account Type : "<<acctype;
    cout<<"\n Balance : "<<bal;
}

int main()
{
    int acc_no;
    char name[100], acc_type[100];
    float balance;
    cout<<"\n Enter Details: \n";
    cout<<"-----";
    cout<<"\n Accout No. ";
    cin>>acc_no;
    cout<<"\n Name : ";
    cin>>name;
    cout<<"\n Account Type : ";
    cin>>acc_type;
    cout<<"\n Balance : ";
    cin>>balance;
    bank b1(acc_no, name, acc_type, balance); //object is created
    b1.deposit(); //
    b1.withdraw(); // calling member functions

```

```

        b1.display();
        return
    }
//
0;

```

Q. (5) What is padding? Explain its use.

Ans: Suppose we create a user-defined structure. When we create an object of this structure, then the contiguous memory will be allocated to the structure members.

```

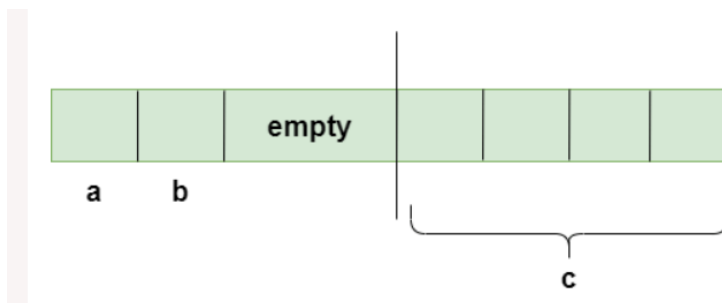
1. struct student
2. {
3.     char a;
4.     char b;
5.     int c;
6. }

```

First, the memory will be allocated to the 'a' variable, then 'b' variable, and then 'c' variable. Now, we calculate the size of the struct student. We assume that the size of the int is 4 bytes, and the size of the char is 1 byte. In the above case, when we calculate the size of the struct student, size comes to be 6 bytes. But this answer is wrong. Now, we will understand why this answer is wrong? We need to understand the concept of structure padding.

The 4-bytes can be accessed at a time as we are considering the 32-bit architecture. The problem is that in one CPU cycle, one byte of char a, one byte of char b, and 2 bytes of int c can be accessed. We will not face any problem while accessing the char a and char b as both the variables can be accessed in one CPU cycle, but we will face the problem when we access the int c variable as 2 CPU cycles are required to access the value of the 'c' variable. In the first CPU cycle, the first two bytes are accessed, and in the second cycle, the other two bytes are accessed.

Suppose we do not want to access the 'a' and 'b' variable, we only want to access the variable 'c', which requires two cycles. The variable 'c' is of 4 bytes, so it can be accessed in one cycle also, but in this scenario, it is utilizing 2 cycles. This is an unnecessary wastage of CPU cycles. Due to this reason, the structure padding concept was introduced to save the number of CPU cycles. The structure padding is done automatically by the compiler.



In order to achieve the structure padding, an empty row is created on the left, as shown in the above diagram, and the two bytes which are occupied by the 'c' variable on the left are shifted

to the right. So, all the four bytes of 'c' variable are on the right. Now, the 'c' variable can be accessed in a single CPU cycle. After structure padding, the total memory occupied by the structure is 8 bytes (1 byte+1 byte+2 bytes+4 bytes), which is greater than the previous one. Although the memory is wasted in this case, the variable can be accessed within a single cycle.

Topic: Specifying a class, defining a member function, private member functions, memory allocation for objects

MCQ

Q. (1) In How many ways we can define a member function?

- (a) Only outside the class definition.
- (b) Only inside the class definition.
- (c) Both outside and inside the class definition.
- (d) In the main body of the program.

Ans: (c)

Q. (2) Which of the following keywords is/are also known as visible labels?

- (a) Private
- (b) Public
- (c) Struct
- (d) Class

Ans: Both (a) and (b)

Q. (3) Which type of member function can only be used by another function that is a member of its class?

- (a) Private Member Function
- (b) Public Member Function
- (c) Both (a) and (b)
- (d) None of the Above

Ans: (a)

Q. (4) Where is the memory allocated for the objects?

- (a) Hard Disk
- (b) Cache
- (c) RAM
- (d) ROM

Ans: (c)

Q. (5) Which of the following keyword can be used to free the allocated memory for an object?

- (a) Free
- (b) Delete
- (c) Either delete or free
- (d) Only delete

Ans: (c)

Descriptive Questions

Q. (1) What is Class. How do we specify a Class in C++?

Ans:

A class is a way to bind the data and its associated functions together. It allows the data (and functions) to be hidden, if necessary, from external use. When defining a class, we are calling a new abstract data type that can be treated like any other built-in data types.

Generally a class specification has two parts:

1. Class declaration

2. Class function definition

The class declaration describes the type and scope of its members. The class function definition describes how the class function are implemented.

The general form of a class declaration is:

Class class_name

```
{  
    Private:  
        Variable    declaration  
        Function    declaration  
    Public:  
        Variable declaration  
        Function declaration  
}
```

Q. (2) What is the difference between an object and a class?

Ans: Class is

- (a) A Class is static.
- (b) A class combination of data (called data members) and functions (called member functions). Class is a user defined datatype with data members and member functions.

- (c) Classes defines object.
- (d) One class can define any no of Object.
- (e) Class is a template(type) or blueprint its state how objects should be and behave.

Objects are:

- (a) Object is an instance of a class while a class can have many objects.
- (b) Object is a dynamic.
- (c) Objects are the basic runtime entities in an object-oriented system.
- (d) Data and function which combine into a single entity is called object.
- (e) Objects are instances of classes which identifies the class properties.
- (f) Object can't define classes.
- (g) Object can be created and destroyed as your necessity.
- (h) Object is defined as a s/w construct which binds data and logic or methods(functions) both.

Q. (3) What are the two methods of creating an Object?

Ans:

Method 1

Declare objects just like declaring normal variables.

Syntax,

Class Name Object1, Object2..., Object N.

Class Name is a user-defined data type, and the object list is the valid identifier names for objects created.

For example: - employee obj1, obj2.

The objects can only be declared after the prototype or definition of the class.

Method 2

Declaring objects during the definition of the class itself.

The object list can be written after the body of the class and before the semi colon.

Syntax,

class

```
ClassName {
    // body of the class
} object list;
```

Q. (4) Why defining a member function inside the class is not a Good Practice?

Ans:

It is not a good programming practice to define the member functions of a class inside the class because it increases the size of the class definition and hence it does not justify the abstraction property of object-oriented programming.

The reason we define the member functions outside the class is to reduce the complexity of the class definition and make it a bit more abstract in nature. But defining the class outside slows down the speed of execution as there is jumping between the function calls.

We can overcome this problem and still define the functions outside the class. We can define the member function outside the class as inline member function. We can define the member function to be inline using the inline keyword.

An inline member function is one whose declaration is replaced by its definition when the program is compiled, and its final executable is formed. We can define a member function as inline using the following statement:

inline Return Type ClassName:: MemberFunction (Argument list);

Q. (5) How can we Dynamically allocate memory to class objects using pointer variable?

Ans:

A pointer variable can also be used to allocate memory to class objects dynamically at run time. It provides greater flexibility to create N number of objects at run time. The memory can also be de-allocated dynamically. We can allocate memory dynamically to class object using the new operator.

For example, `student * ptr; ptr = new (student);`

The first statement is used to create pointer object of the base class. The pointer object is then used to allocate memory at run time. New operator is used to achieve the same and the amount of memory to be allocated is the amount of memory taken by the object of the student class.

We can also de-allocate the memory dynamically allocated to the pointer object using the delete operator as follows:

`delete (ptr);`

HOT

Q. (6) How can we say that inline is one of the important features of C++. In what circumstances Compiler can ignore the request for inlining?

Ans:

C++ provides an inline function to reduce the function call overhead. Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

Remember, inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining. Compiler may not perform inlining in such circumstances like:

- 1) If a function contains a loop. (for, while, do-while)
- 2) If a function contains static variables.
- 3) If a function is recursive.
- 4) If a function return type is other than void, and the return statement doesn't exist in function body.
- 5) If a function contains switch or goto statement.

Q. (7) Explain the importance of private member function and Data Hiding using a real-life example.

Ans:

Access modifiers are used to implement an important aspect of Object-Oriented Programming known as Data Hiding. Consider a real-life example:

The Research and Analysis Wing (R&AW), having 10 core members, has come into possession of sensitive confidential information regarding national security. Now we can correlate these core members to data members or member functions of a class, which in turn can be correlated to the R&A Wing.

These 10 members can directly access the confidential information from their wing (the class), but anyone apart from these 10 members can't access this information directly, i.e., outside functions other than those prevalent in the class itself can't access the information (that is not entitled to them) without having either assigned privileges (such as those possessed by a friend class or an inherited class, as will be seen in this article ahead) or access to one of these 10 members who is allowed direct access to the confidential information (similar to how private members of a class can be accessed in the outside world through public member functions of the class that have direct access to private members). This is what data hiding is in practice.

Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members, i.e., they set some restrictions on the class members so that they can't be directly accessed by the outside functions.

Q. (8) Explain the purpose of the Classes and Objects with a real-life example.

Ans:

Consider an ATM. An ATM is a class. It's machine which is pretty much useless until you insert your debit card. After you insert your debit card, the machine has information about you and your bank account and the balance in it, so at this point it is an object.

You have used a class and created an object, now you can perform operations on it like withdrawal of money or checking you balance or getting statement of your account, these operations will be methods belonging to that class (ATM) but you cannot use them until you create an object out of it.

And when you did perform whatever operation, you wanted to perform and clicked exit/cancel and removed your card, you just destroyed the object. Now it is not an object, it has methods

and all (the functions an ATM can perform) but you cannot use them until you insert your card again and create an object.

Topic: static data members and member functions, array of objects, objects as function arguments

MCQ

Q. (1) Which is correct syntax to access the static member functions with class name?

- (a) className . functionName;
- (b) className -> functionName;
- (c) className : functionName;
- (d) className :: functionName;

Explanation: The scope resolution operator must be used to access the static member functions with class name. This indicates that the function belongs to the corresponding class.

Q. (2) Which among the following is not applicable for the static member functions?

- (a) Variable pointers
- (b) void pointers
- (c) this pointer
- (d) Function pointers

Explanation: Since the static members are not property of objects, they doesn't have this pointer. Every time the same member is referred from all the objects, hence use of this pointer is of no use.

Q. (3) Which is the correct syntax for declaring static data member?

- (a) static memberName dataType;
- (b) dataType static memberName;
- (c) memberName static dataType;
- (d) static dataType memberName;

Explanation: The syntax must firstly be mentioned with the keyword static. Then the data type of the member followed by the member's name should be given. This is general form of declaring static data members.

Q. (4) Predict the output of following C++ program.

```
#include <iostream>
using namespace std;
class Test
{
```

```

    static int x;
public:
    Test() { x++; }
    static int getX() {return x;}
};
int Test::x = 0;
int main()
{
    cout << Test::getX() << " ";
    Test t[5];
    cout << Test::getX();
}

```

- (a) 0 0
- (b) 5 5
- (c) 0 5
- (d) Compiler Error

Explanation: Static functions can be called without any object. So the call "Test::getX()" is fine. Since x is initialized as 0, the first call to getX() returns 0. Note the statement x++ in constructor. When an array of 5 objects is created, the constructor is called 5 times. So x is incremented to 5 before the next call to getX().

Q. (5) Which of the following is true?

- (a) Static methods cannot be overloaded.
- (b) Static data members can only be accessed by static methods.
- (c) Non-static data members can be accessed by static methods.
- (d) Static methods can only access static members (data and methods)

Explanation: A static function is a special type of function which is used to access only static data, any other normal data cannot be accessed through static function. Just like static data, static function is also a class function, it is not associated with any class object. Static method overloaded and static method can access only static members.

Descriptive Questions

Q. (1) Define Static Data Members and Static Member Functions.

Ans:

Static data members:

Static Data Members are those which are declared by using the Static Keyword in front of the

Data Members. Means Put the Static Keyword in front of the Variables. And Static Data Members always have Default values as\ 0 for int and Null for Strings. So that they will Never Stores any Garbage values. Always remember that Static Data Members are always used in the Static Member Functions. Means if a Member Functions wants to use a Static Data then we must have to declare that Member Function as Static. And the Static Data Members are always Assigned Some values from the outside from the Class.

Static Member Functions: The Static Member Functions are those which are declared by using the Static in Front of the Member Function. And in this we use the Static Data Members. Any Method can be converted into the Static just by Using the Static in front of the Member Function.

For Accessing the Static data Member Function we doesn't need to Create an Object of Class and we will call the Function with the name of Class and Scope resolution Operator. So that there will be no object and no Extra Memory is required to Store the Data of a Class.

Q. (2) What are the properties of static member functions?

Ans: Below are the properties of static member functions:

- “static” is the keyword used to declare static member function.
- “this” pointer will not be present in static member functions.
- Static member functions can be accessed without creating an object. It can be called using scope resolution operator. “<class_name>::<static_member_function_name> ()”.
- Static member function can only access static data members and static member functions.
- You cannot have static and non-static member functions with same name.

Q. (3) What are the properties of static Data members ?

Ans: Properties of Static Data Members:

- The static data member gets memory only once, which is shared by all the objects of its class.
- The static data members are associated with the class and not with any object, so they are also known as class variables.
- The static data members must be defined outside the class; otherwise, the linker will generate an error. Also, there is only one definition of a static data member.
- The static data member follow the public, private and protected access rules. The private static data members are accessed within the class in the same way as normal private data members. The public static data members can be accessed throughout the program using the name of the class, followed by the scope resolution operator and then the static data member itself.

Q. (4) By how many types objects can be passed in arguments of a function explain in brief?

Ans: The objects of a class can be passed as arguments to member functions as well as nonmember functions either by value or by reference. When an object is passed by value, a copy of the actual object is created inside the function. This copy is destroyed when the function terminates. Moreover, any changes made to the copy of the object inside the function are not reflected in the actual object. On the other hand, in pass by reference, only a reference to that object (not the entire object) is passed to the function. Thus, the changes made to the object within the function are also reflected in the actual object.

Q. (5) Difference B/W Static And Non-Static

Ans:

1) A static member function can access only static member data, static member functions and data and functions outside the class.

- A non-static member function can access all of the above including the static data member.

2) A static member function can be called, even when a class is not instantiated,

- A non-static member function can be called only after instantiating the class as an object.

3) A static member function cannot be declared virtual ,

- A non-static member functions can be declared as virtual

HOTS QUESTION:

Q1) Give an example with the help of code to show you are Passing objects as argument

Ans:

```
#include <iostream.h>
class time
{
private:
    int hours;
    int minutes;

public:
    void gettime(int h, int m)
    {
        hours = h;
        minutes = m;
    }
    void puttime()
    {
        cout << hours << " hours and ";
```



```

        cout << minutes << " minutes " << endl;
    }
    void sum(time, time); // function prototype
                          // Objects as arguments
};

void time ::sum(time t1; time t2) // t1 and t2 are objects
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes / 60;
    minutes = minutes % 60;
    hours = hours + t1.hours + t2.hours;
}

void main()
{
    time T1, T2, T3;
    T1.gettime(2, 45); // get T1
    T2.gettime(3, 30); // get T2

    T3.sum(T1, T2); // get T3 = T1 + T2

    cout << "T1 = " << T1.puttime(); // display T1
    cout << "T2 = " << T2.puttime(); // display T2
    cout << "T3 = " << T3.puttime(); // display T3
}

```

Topic: friendly functions, returning objects, pointers to members

MCQ

Q. (1) Which of the followings is/are not false about friend function?

1. It can be called / invoked with class object.
2. It has objects as arguments.
3. It can have built-in types as arguments.

4. It must declare only in public part of a class.
 5. It does not have this pointer as an argument.
- (a) 2, 4.
(b) 1, 2, 5.
(c) 2, 3, 5.
(d) All of these.

Ans: (c)

Q. (2) What will be output for the following code?

```
#include <bits/stdc++.h>
```

```
class A {
```

```
private:
```

```
    int a;
```

```
public:
```

```
    A() { a = 0; }
```

```
    friend class B;
```

```
};
```

```
class B {
```

```
private:
```

```
    int b;
```

```
public:
```

```
    void showA(A& x)
```

```
    {
```

```
        cout << "A::a=" << x.a;
```

```
    }
```

```
};
```

```
int main()
```

```
{  
  
    A a;  
  
    B b;  
  
    b.showA(a);  
  
}
```

- (a) A::a=0.
- (b) A.
- (c) A:0
- (d) a=0.

Ans: (a)

Q. (3) What is referred by pointers to members?

- (a) Static members of class objects.
- (b) Non-static members of class objects
- (c) Referring to whole class.
- (d) Dynamic members of class objects.

Ans: (b)

Q. (4) Which symbols are used in pointer to members?

- (a) .*
- (b) ->*
- (c) Both .* and ->*
- (d) None

Ans: (c)

Q. (5) What should be used to point to a static class member?

- (a) Smart pointer
- (b) Dynamic pointer.
- (c) Normal pointer
- (d) Static pointer

Ans: (c)

Descriptive Questions

Q. (1) Merits and Demerits of using friend function.

Ans:

Merits:

- 1) A friend function is able to access members without the need of inheriting the class.
- 2) Friend function acts as a bridge between two classes by accessing their private data.
- 3) It can be declared in any part of the class, i.e. in public/private/protected/anywhere in the code.
- 4) A friend function can be friend of more than one class.

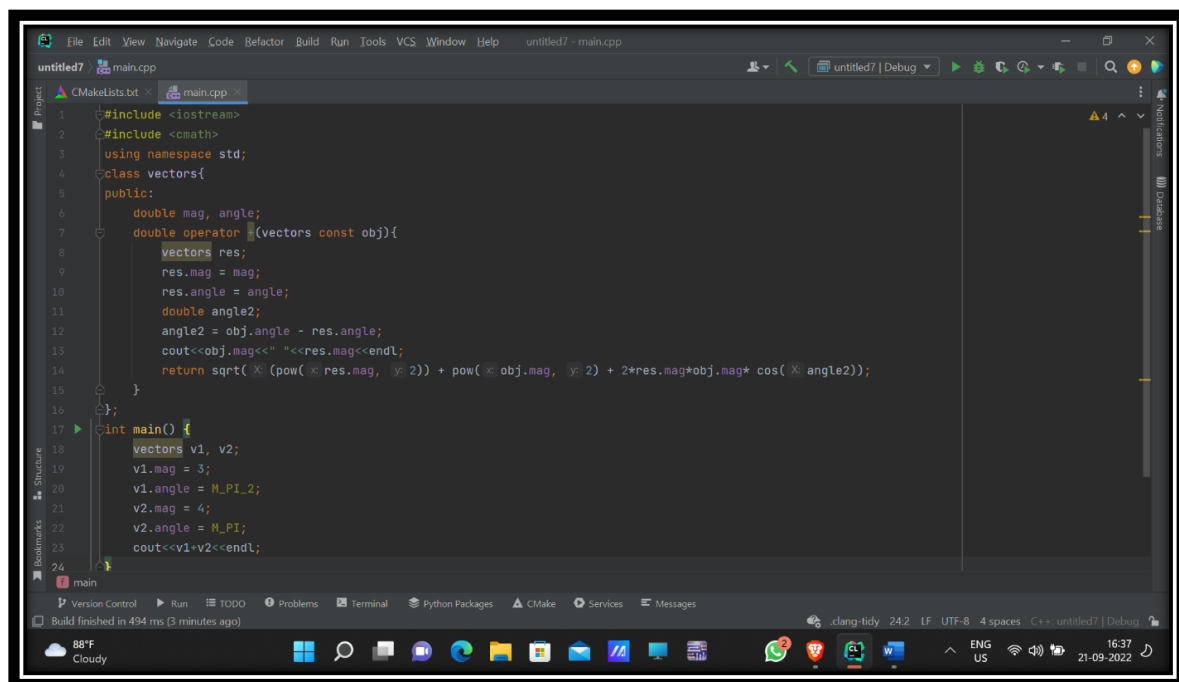
Demerits:

- 1) The use of friend function eliminates the concept of data hiding.
- 2) Friend functions cannot be specified as static or extern because they lack a storage class specifier.

HOTS:-

Q. (1) How can operation overloading can be used to find resultant of vectors?

Ans:



```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 class vectors{
5 public:
6     double mag, angle;
7     double operator +(vectors const obj){
8         vectors res;
9         res.mag = mag;
10        res.angle = angle;
11        double angle2;
12        angle2 = obj.angle - res.angle;
13        cout<<obj.mag<<" "<<res.mag<<endl;
14        return sqrt( (pow( res.mag, 2)) + pow( obj.mag, 2) + 2*res.mag*obj.mag* cos( angle2));
15    }
16 };
17 int main() {
18     vectors v1, v2;
19     v1.mag = 3;
20     v1.angle = M_PI_2;
21     v2.mag = 4;
22     v2.angle = M_PI;
23     cout<<v1+v2<<endl;
24 }
```

:

One can use to add the vectors and find the resultant magnitude as following

Q. (2) Why some operators can't be overloaded?

Answer

Various factors prevent why some of the operators can't be overloaded, like if an operator is evaluated by the compiler, and cannot be evaluated in run time these types can't be overloaded

one of its kind is sizeof() operator. Some of the other reasons, include like the operator int, double like operators shouldn't be changed as with this help only compiler evaluates the data type. The scope operator as "::" shouldn't be also overloaded as it provides the reference to the keyword where it's actually referring.

Q. (3) What do one understand by run time polymorphism and also explain why friendly functions can't do it?

Answer

Run time polymorphism occurs when a function in the derived class has definition of at least one of the member function of the base class. Friend functions can't do run time polymorphism as we know friend functions are inherited, so it would not be counted in run time polymorphism. Although, friend functions can be doing compile time polymorphism.

Q. (4) What do you understand by overridden functions and how it's different from overloaded functions?

Answer

It's pretty evident that, in both the things we have multiple definitions of functions, but the question arises what makes them different? Well the only difference occurs is when we relate it to classes, and in that they do different types of polymorphism. Overloading functions inside a class is compile time polymorphism whereas when we, define the function with the same name as in the parent class, then it's called overriding the function, and this is an example of run time polymorphism.

Q. (5) What is early binding? Explain the way it occurs and why it affects compilation and how it affects the running time of a program.

Answer

Whenever a polymorphism is resolved during the compilation process, we say it's static polymorphism or compile time polymorphism or early binding as it's resolved in compilation only. It can occur in two ways

1)Method overloading

2)Operator overloading

In method overloading, a function is overloaded by changing the number of arguments, data types or both. One should be beware that compilation will not happen, if one has changed the return type of functions and everything else is same, as this is compile time polymorphism which will not let the program to be compiled as the polymorphism wouldn't be resolved.

As static polymorphism is resolved in compile time only, hence the running time of this type of polymorphism will be less compared to run time polymorphism.

Topic: constructors, Parametrized constructors, Multiple constructors, Copy constructor, Destructors.

MCQ

Q. (1) What is the role of a constructor in classes?

- (a) To modify the data whenever required
- (b) To destroy an object
- (c) To initialize the data members of an object when it is created
- (d) To call private functions from the outer world

Ans: (c)

Q. (2) Why constructors are efficient instead of a function `init()` defined by the user to initialize the data members of an object?

- (a) Because user may forget to call `init()` using that object leading segmentation fault
- (b) Because user may call `init()` more than once which leads to overwriting values
- (c) Because user may forget to define `init()` function
- (d) All of the mentioned

Ans: (d)

Q.3). What happens if a user forgets to define a constructor inside a class?

- (a) Error occurs
- (b) Segmentation fault
- (c) Objects are not created properly
- (d) Compiler provides a default constructor to avoid faults/errors

Ans: (d)

Q. (4) How many parameters does a default constructor require?

- (a) 1
- (b) 2
- (c) 0
- (d) 3

Ans: (c)

Q. (5). How constructors are different from other member functions of the class?

- (a) Constructor has the same name as the class itself
- (b) Constructors do not return anything
- (c) Constructors are automatically called when an object is created
- (d) All of the mentioned

Ans: (d)

Q. (6) How many types of constructors are there in C++?

- (a) 1
- (b) 2
- (c) 3
- (d) 4

Ans: (c)

Q. (7) What is the role of destructors in Classes?

- (a) To modify the data whenever required
- (b) To destroy an object when the lifetime of an object ends
- (c) To initialize the data members of an object when it is created
- (d) To call private functions from the outer world

Ans: (b)

Q.8). What is syntax of defining a destructor of class A?

- (a) A(){ }
- (b) ~A(){ }
- (c) A::A(){ }
- (d) ~A(){ };

Ans: (b)

Q. (9) When destructors are called?

- (a) When a program ends
- (b) When a function ends
- (c) When a delete operator is used
- (d) All of the mentioned

Ans: (d)

Q. (10) What happens when a class with parameterized constructors and having no default constructor is used in a program and we create an object that needs a zero-argument constructor?

- (a) Compile-time error.
- (b) Preprocessing error.
- (c) Runtime error.
- (d) Runtime exception.

Ans: (a)

Descriptive Questions

Q. (1) What is a constructor?

Ans: Class constructors in C++ are special member functions of a class and it initializes the object of a class. It is called by the compiler (automatically) whenever we create new objects of that class. The name of the constructor must be the same as the name of the class and it does not return anything.

One point is important that constructor has a secret argument and this argument is “this pointer” (Address of the object for which it is being called).

Q. (2) How to use the constructor in C++?

Ans: A constructor is a special kind of function. It creates an object in memory, and its job is to initialize that object before it's used. And here are some things that you need to know about constructors:

1. Constructors are functions, but they have the name of the class.
2. The arguments passed to a constructor are called arguments of initializers.
3. The arguments passed to a constructor must be constants or previously declared members of the same class or inherited members from parent classes (in other words, they can't change).
4. Constructors cannot return values which means they cannot use the return statement. Instead, they yield an object using the "this" keyword.

Q. (3) What is the use of constructor?

Ans: A constructor is a method that has the same name as a class. And the use of a constructor is to initialize the object when it is created using a new keyword.

When an object is created, the variables are initialized chunks of memory and base values if there are any.

Q. (4) What is the use of destructor?

Ans: A destructor is a method that has the same name as a class preceding a ~ symbol. The use of a destructor is to deallocate the memory chunks one the code goes out of the scope of the object or deleted using the delete keyword.

When the object is deleted the destructor is called and it deallocated all the memory blocks that were created when an object was created.

Q. (5) What is the order of constructor execution in C++?

Ans: A constructor is invoked when the object of a class is created. The order in which a constructor is invoked is the same as the hierarchy of the inheritance. This means that first the object of a base class is invoked then the objects of the child class are invoked and so on.

Q. (6) What is the order of destructor execution in C++?

Ans: A destructor is invoked in the reverse order as the constructor and is invoked when the object of the class is deleted. The order in which a destructor is invoked is just the opposite of the hierarchy of the inheritance. This means that first the object of child class is destroyed then the objects of the parent class are destroyed and so on.

Q.7) How many types of constructors exist in C++?

Ans: Mainly in c++ there are three types of constructors exist “Default constructor”, “Parameterized constructors” and “Copy constructor”. We create the constructor as per our requirement but if we will not create the constructor then the compiler automatically creates the constructor to initialize the class object.

Q. (8) When are copy constructors called in C++?

Ans: There are some possible situation when copy constructor called in C++,

- When an object of the class is returned by value.
- When an object of the class is passed (to a function) by value as an argument.
- When an object is constructed based on another object of the same class.
- When the compiler generates a temporary object.

Q. (9) Why copy constructor takes the parameter as a reference in C++?

Ans: A copy constructor is called when an object is passed by value. The copy constructor itself is a function. So, if we pass an argument by value in a copy constructor, a call to copy constructor would be made to call copy constructor which becomes a non-terminating chain of calls. Therefore, compiler doesn't allow parameters to be passed by value.

Q. (10) Can one constructor of a class call another constructor of the same class to initialize this object?

Ans:

Onward C++11 Yes, let see an example,

```
#include <iostream>
using namespace std;
```

```

class Test
{
    int a, b;
public:
    Test(int x, int y)
    {
        a= x;
        b =y;
    }
    Test(int y) : Test( 7, y) {}
    void displayXY()
    {
        cout <<"a = "<<a<<endl;
        cout <<"b = "<<b<<endl;
    }
};

int main()
{
    Test obj(27);
    obj.displayXY();
    return 0;
}

```

Output:

```

a = 7
b = 27

```

Q. (11) What is the difference between constructor and destructor?

Ans: There are the following differences between the constructor and destructor in C++.

Constructor	Destructor
Constructor helps to initialize the object of a class.	Whereas destructor is used to destroy the instances.
The constructor's name is the same as the class name.	The destructor name is the same as the class name but preceded by a tiled (~) operator.
A constructor can either accept the arguments or not.	While it can't have any argument.
A constructor is called when the instance or object of the class is created.	It is called while the object of the class is freed or deleted.
A constructor is used to allocate the memory to an instance or object.	While it is used to deallocate the memory of an object of a class.
A constructor can be overloaded.	While it can't be overloaded.
There is a concept of copy constructor which is used to initialize an object from another object.	While here, there is no copy destructor concept.

Q. (12) Can a copy constructor accept an object of the same class as a parameter, in place of reference of the object? If No, why not possible?

Ans: No. It is specified in the definition of the copy constructor itself. It should generate an error if a programmer specifies a copy constructor with a first argument that is an object and not a reference.

HOTS QUESTIONS

Q. (1) All the banks operating in India are controlled by RBI. RBI has set a well-defined guideline (e.g. minimum interest rate, minimum balance allowed, maximum withdrawal limit etc) which all banks must follow. For example, suppose RBI has set minimum interest rate applicable to a saving bank account to be 4% annually; however, banks are free to use 4% interest rate or to set any rates above it.

Write a program to implement bank functionality in the above scenario. Note: Create few classes namely Customer, Account, RBI (Base Class) and few derived classes (SBI, ICICI, PNB etc). Assume and implement required member variables and functions in each class.

Ans) `#include <iostream>`
`using namespace std;`

```

// class BB hold the baisc guidelines for bank rules
class BB
{
public:
float minimumInterestRate = 4;
double mnimumBalanceAllowed = 500;
double maximumWithdrawl = 50000;
};
int main()
{
BB obj;
float rate;
double withdrawl;
double balance;
cout << "Enter interest rate \n";
cin >> rate;
if (obj.minimumInterestRate <= rate)
{
cout << "Eligible under BB rules\n";
} else
{
cout << "Not elligible under BB rules\n";
}
cout << "Enter withdrawl \n";
cin >> withdrawl;
if (obj.maximumWithdrawl >= withdrawl)
{
cout << "Eligible under BB rules\n";
}
else
{
cout << "Not elligible under BB rules\n";
}
cout << "Enter minimum balance\n";
cin >> balance;
if (obj.mnimumBalanceAllowed <= balance)
{
cout << "Eligible under BB rules\n";
}
else
{
cout << "Not elligible under BB rules\n";
}
cin.get();
cin.get();
return 0;
}

```

Q. (2) Give a real life examples of constructors?

Ans) Constructor is a method that is called when instance of an object is created.

- They have the same name as a class.
-
- In our example default constructor is to be set with values such as eye color, skin color and mouse color.
- After calling the default constructor Ganesh statue will look like the following:



```
//constructor
public Ganesh_Statue() {
    Console.WriteLine("**** Dafult Constructor in base class starts from here ****");

    Eye_Color = "Black";

    Skin_Color = "Redish_White";

    mouse_color = "Black";

    Console.WriteLine("Default values of Eye color: " + Eye_Color);
    Console.WriteLine("Default values of skin color: " + Skin_Color);
    Console.WriteLine("Default values of mouse color: " + mouse_color);

    Console.WriteLine("_____");
    Console.WriteLine();
}
```

Parameterized Constructor

A constructor with at least one parameter is called parameterized constructor.

In our example suppose a wealthy businessman came to this family & gave some parameter like he wants to add his own jewellery to statue like chain & ring.

```
// parameterized constructor
```

```
public Ganesh_Statue(string x, string y)
{
    chain = x;
    ring = y;

    Console.WriteLine("_____");

    Console.WriteLine("**** Parameterized Constructor of base class starts from here ****");

    Console.WriteLine("chain color: " + x + " ring color: " + y);

    Console.WriteLine("_____");

    Console.WriteLine();

}
```