Tutorial 2 CS203 (CSE)

Module 3

Topic: Operator overloading, inheritance, virtual functions and polymorphism

MCQ

Q. (1)

Descriptive Questions

Q. (1)

Topic: Overloading unary operators, overloading binary operators, rules for overloading operators, type conversions

MCQ

Q. (1)

Descriptive Questions

Q. (1) What is Operator Overloading?
Ans: We can easily apply operators on primitive data types (like integers, characters, floating point numbers) but not on user-defined data types like Complex numbers, fractions, etc. We can also make operators work for user-defined data types by explicitly defining the operation performed by a given operator on a given data type, known as Operator overloading.
Q. (2) Explain overloading unary operator?
Ans: Unary operators are those which operate on a single variable. Overloading unary operator means extending the operator's original functionality to operate upon object of the class. The declaration of an overloaded unary operator function precedes the word operator.
Here is the example showing the overloading of unary operators for a class Rectangle.

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

class Rectangle {
  public:

  int length;
  int width;

  Rectangle(int len, int wid) {
    length = len;
    width = wid;
  }

  void area() {
    cout << length * width << endl;
  }

  //overloading ++ operator
  Rectangle operator++(int) {
    length++;
    width++;
  }

  //overloading -- operator
  Rectangle operator--(int) {
    length--;
    width--;
  }

};

int main() {

  Rectangle r(3, 2);
  r.area();

  r++;

  r.area();
  r--;
  r1.area();

  return 0;
}
```

Q. (3) Difference between Operator Functions and Normal Functions
Ans: Operator functions and normal functions share many similarities. The only difference between the two is that the operator symbol is written in operator functions at the place of the function name in normal functions.
Let us understand this through an example -
// Normal Function -

```
ReturnType myFunction (arguments){
    // Do something
    ................
    ................

    // Return something
    ................
}
```

```
// Operator Function -

ReturnType operator + (arguments){
    // Do something
    ................
    ................

    // Return something
    ................
}

// Note that '+' can be replaced by other
// valid operators also like '*', '-', etc.
```

Q. (4) Explain function overloading?

Ans: Function overloading is the process of using the same name for two or more functions. The secret to overloading is that each redefinition of the function must use either different types of parameters or a different number of parameters. It is only through these differences that the compiler knows which function to call in any situation. Consider following program which overloads MyFunction() by using different types of parameters:

```
#include <iostream>
Using namespace std;
int MyFunction(int i);
double MyFunction(double d);
int main(){
        cout <<MyFunction(10)<<”n”; //calls MyFunction(int i);
        cout <<MyFunction(5.4)<<”n”; //calls MyFunction(double d);
        return 0;
}
int MyFunction(int i){
        return i*i;
}
double MyFunction(double d){
        return d*d;
}
```

Now the following program overloads MyFunction using different no of parameters

```
#include <iostream>
Using namespace std;
int MyFunction(int i);
int MyFunction(int i, int j);
int main(){
        cout <<MyFunction(10)<<”n”; //calls MyFunction(int i);
        cout <<MyFunction(1, 2)<<”n”; //calls MyFunction(int i, int j);
        return 0;
}
int MyFunction(int i){
        return i;
}
double MyFunction(int i, int j){
```

```
return i*j;
}
```

Q. (5) Explain the difference between overloaded functions and overridden functions?
Ans: Overloading is a static or compile-time binding and overriding is dynamic or run-time binding.
Redefining a function in a derived class is called function overriding
A derived class can override a base-class member function by supplying a new version of that function with the same signature (if the signature were different, this would be function overloading rather than function overriding).

Q. (6) What is type conversion and its types?
Ans: Type conversion is the method of converting one data type to another. There are two types of Type Conversions in C++:

- Implicit type conversion: - The Implicit Type Conversion is that type conversion that is done automatically by the compiler. It does not require any effort from the programmer. The C++ compiler has a set of predefined rules. Based on these rules, the compiler automatically converts one data type to another. Therefore, implicit type conversion is also known as automatic type conversion.
  For example:

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

int main() {
  int int_var;
  float float_var = 20.5;

  int_var = float_var;
  // trying to store the value of float_var in int_var

  cout << "The value of int_var is: " << int_var << endl;
  cout << "The value of float_var is: " << float_var << endl;

  return 0;
}
```

  Output:

```
The value of (50 + 'a') is: 147
The value of float_var is: 220.5
```

- Explicit type conversion: - Explicit Type Conversions are those conversions that are done by the programmer manually. In other words, explicit conversion allows the programmer to typecast (change) the data type of a variable to another type. Hence, it is also called typecasting. Generally, we use the explicit type conversion if we do not want to follow the implicit type conversion rules.
  For example:

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

int main() {
  char char_var = 'a';
  int int_var;
  // Explicitly converting a character variable to integer variable
  int_var = (int) char_var; // Using cast notation

  cout << "The value of char_var is: " << char_var << endl;
  cout << "The value of int_var is: " << int_var << endl;

  return 0;
}
```

Output:

```
The value of char_var is: a
The value of int_var is: 97
```

Q. (7) Why some operators cannot be overloaded in c++?
Ans:
In C++ we are able to overload several operators like +, -, [], -> etc. But we can’t overload any operators inside it. Several of the operators can’t be overloaded. These operators are as below.
? “.” Member access or dot operator
? “? : ” Ternary or conditional operator
? “::” Scope resolution operator
? “.*” Pointer to member operator
? “sizeof” The object size operator
? “typeid” Object type operator
These operators can’t be overloaded because in case we overload them it is going to make serious programming issues. For instance, the sizeof operator returns the dimensions of the item or maybe datatype as being an operand. This is evaluated through the compiler. It can’t be evaluated throughout the runtime. So we can’t overload it

Q. (8) What are the Different Approaches to Operator Overloading in C++?
Ans: We can perform operator overloading by implementing any of the following types of functions:
Member Function
Non-Member Function
Friend Function
The operator overloading function may be a member function when a Left operand is an object of the Class.
When the Left operand is different, the Operator overloading function should be a non-member function.
We can make the operator overloading function a friend function if it needs to access the private and protected class members.

Q. (9) How are ‘=’and ‘&’ operators overloaded in c++?
Ans: The = and & C++ operators are overloaded by default. For example, we can copy the objects of the same Class directly using the = operator.

HOTS

Q. (10) A student is struggling to add two complex numbers using c++. Help him with the code using the concept of operator overloading.
Ans:

```
#include <iostream>
```

```
using namespace std;

class Complex
{
   private:
     float real;
     float imag;
   public:
      Complex(): real(0), imag(0){ }
      void input()
      {
          cout << "Enter real and imaginary parts respectively: ";
          cin >> real;
          cin >> imag;
      }

      // Operator overloading
      Complex operator + (Complex c2)
      {
          Complex temp;
          temp.real = real + c2.real;
          temp.imag = imag + c2.imag;

          return temp;
      }

      void output()
      {
          if(imag < 0)
              cout << "Output Complex number: "<< real << imag << "i";
          else
              cout << "Output Complex number: " << real << "+" << imag << "i";
      }
};

int main()
{
   Complex c1, c2, result;

   cout<<"Enter first complex number:\n";
   c1.input();

   cout<<"Enter second complex number:\n";
   c2.input();
   result = c1 + c2;
   result.output();

   return 0;
}
```
Output::

```
Enter first complex number:
Enter real and imaginary parts respectively: 2
8
Enter second complex number:
Enter real and imaginary parts respectively: 4
3
Output Complex number: 6+11i
```

Topic: Derived classes, single inheritance, multilevel inheritance, multiple inheritance, hierarchical inheritance, hybrid inheritance, virtual base classes, abstract classes, nesting of classes

MCQ

Q. (1) Which among the following best describes the Inheritance?

(a) Copying the code already written

(b) Using the code already written once

(c) Using already defined functions in programming language

(d) Using the data and functions into derived segment

Ans: (d)

Q. (2) How many basic types of inheritance are provided as OOP feature?

(a) 4

(b) 3

(c) 2

(d) 1

Ans: (a)

Q. (3) Which among the following best defines single level inheritance?

(a) A class inheriting a derived class

(b) A class inheriting a base class

(c) A class inheriting a nested class

(d) A class which gets inherited by 2 classes

Ans: (b)

Q. (4) Which among the following is correct for multiple inheritance?

(a) class student{public: int marks;}s; class stream{int total;}; class topper: public student, public stream{ };

(b) class student{int marks;}; class stream{ }; class topper: public student{ };

(c) class student{int marks;}; class stream:public student{ };

(d) class student{ }; class stream{ }; class topper{ };

Ans: (a)

Q. (5) Which programming language doesn't support multiple inheritance?

(a) C++ and Java

(b) C and C++

(c) Java and Small Talk

(d) Java

Ans: (d)

Q. (6) Which among the following is correct for a hierarchical inheritance?

(a) Two base classes can be used to be derived into one single class

(b) Two or more classes can be derived into one class

(c) One base class can be derived into other two derived classes or more

(d) One base class can be derived into only 2 classes

Ans: (c)

Q. (7) Which is the correct syntax of inheritance?

(a) class derived_classname : base_classname{ /*define class body*/ };

(b) class base_classname : derived_classname{ /*define class body*/ };

(c) class derived_classname : access base_classname{ /*define class body*/ };

(d) class base_classname :access derived_classname{ /*define class body*/ };

Ans: (c)

Q. (8) Which type of inheritance leads to diamond problem?

(a) Single level

(b) Multi-level

(c) Multiple

(d) Hierarchical

Ans: (c)

Q. (9) Which access type data gets derived as private member in derived class?

(a) Private

(b) Public

(c) Protected

(d) Protected and Private

Ans: (a)

Q. (10) If a base class is inherited in protected access mode, then which among the following is true?

(a) Public and Protected members of base class become protected members of derived class

(b) Only protected members become protected members of derived class

(c) Private, Protected and Public all members of base, become private of derived class

(d) Only private members of base, become private of derived class

Ans: (a)

Q. (11) Members which are not intended to be inherited are declared as …………..

(a) Public members

(b) Protected members

(c) Private members

(d) Private or Protected members

Ans: (c)

Q. (12) While inheriting a class, if no access mode is specified, then which among the following is true? (In C++)

(a) It gets inherited publicly by default

(b) It gets inherited protected by default

(c) It gets inherited privately by default

(d) It is not possible

Ans: (c)

Q. (13) If a derived class object is created, which constructor is called first?

(a) Base class constructor

(b) Derived class constructor

(c) Depends on how we call the object

(d) Not possible

Ans: (a)

Q. (14) The private members of the base class are visible in derived class but are not accessible directly.

(a) True

(b) False

Ans: (a)

Q. (15) How can you make the private members inheritable?

(a) By making their visibility mode as public only

(b) By making their visibility mode as protected only

(c) By making their visibility mode as private in derived class

(d) It can be done both by making the visibility mode public or protected

Ans: (d)

Descriptive Questions

Q. (1) What is base class and derived class?
Ans: A base class is a class in Object-Oriented Programming language, from which other classes are derived. The class which inherits the base class has all members of a base class as well as can also have some additional properties. The Base class members and member functions are inherited to Object of the derived class. A base class is also called parent class or superclass.
Derived Class: A class that is created from an existing class. The derived class inherits all members and member functions of a base class. The derived class can have more functionality with respect to the Base class and can easily access the Base class. A Derived class is also called a child class or subclass.
Syntax for creating Derive Class:

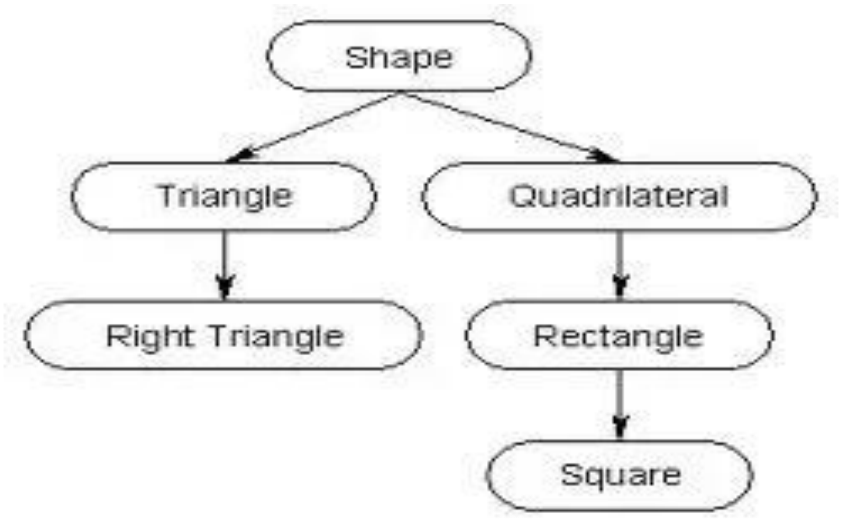
```
class BaseClass{
  // members....
  // member function
}
```

```
class DerivedClass : public BaseClass{
  // members....
  // member function
}
```

Difference between Base Class and Derived Class:

S.No.	BASE CLASS	DERIVED CLASS
1	A class from which properties are inherited.	A class from which is inherited from the base class.
2	It is also known as parent class or superclass.	It is also known as child class subclass.
3	It cannot inherit properties and methods of Derived Class.	It can inherit properties and methods of Base Class.
4	Syntax: Class base_classname{ … }.	Syntax: Class derived_classname : access_mode base_class_name { … }.

Q. (2) What is inheritance?
Ans: Inheritance in C++ is one of the best feature of OOPS. Inheritance is the method by which the features of a existing class can be used with new class. It enhances the concept of re-usability of code as the code once written can be reused and other features can also be added. It is also possible that a class can't be inherited, for this purpose we have to change



the access specifier of the parent class.
A typical example of inheritance in OOP inheritance allows us to define a class that inherits all the methods and attributes from another class. The class that inherits from another class is called a derived class or child class. The class from which we are inheriting is called parent class or base class.
Question 3. How to implement inheritance?
Solution:
For creating a sub-class that is inherited from the base class we have to follow the below syntax.

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
      //body
}
```

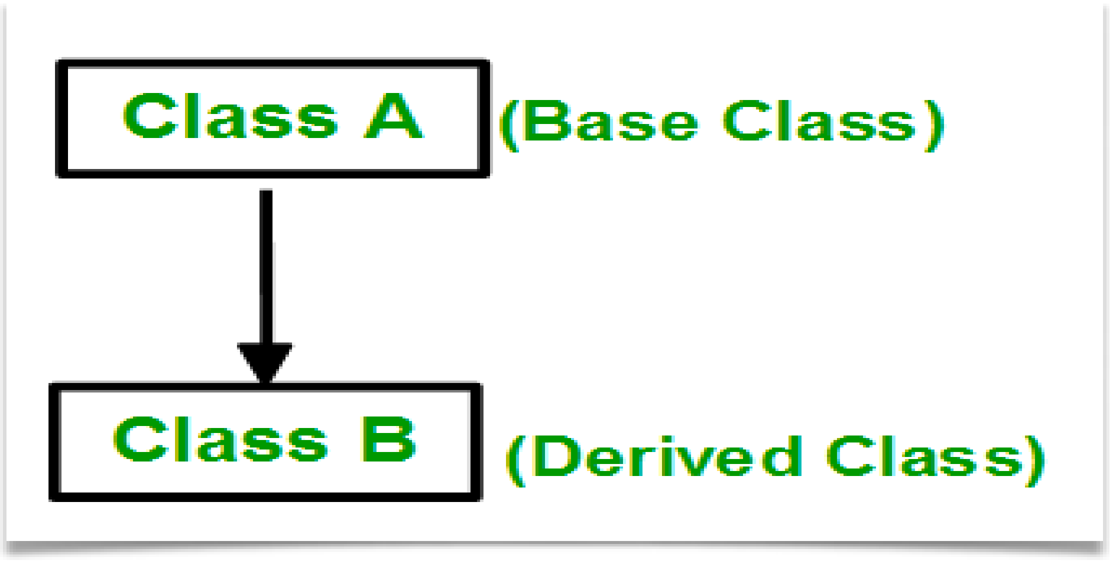
The syntax of inheritance in C++ is very simple. You just create a class as usual but before the opening of braces of the body of class just put a colon and name of the base class with the access specifier.
Here access specifier can be public, private or protected, the derived class is newly created class and base class is already existing class.
class — keyword to create a new class
derived_class_name — name of the new class, which will inherit the base class
access-specifier — either of private, public or protected. If neither is specified, PRIVATE is taken as default
base-class-name — name of the base class
Note: A derived class doesn’t inherit access to private data members. However, it does inherit a full parent object, which contains any private members which that class declares.
Question 4:What are different types of inheritance ?
Solution:
There are five types of inheritance

- Single inheritance
- Multilevel inheritance
- Multiple inheritance
- Hierarchical inheritance
- Hybrid inheritance

Q. (5) Describe briefly the single inheritance?
Ans:

1. Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.

in the given example, Class A is the parent class and Class B is the child class since Class B inherits the features and behavior of the parent class A.



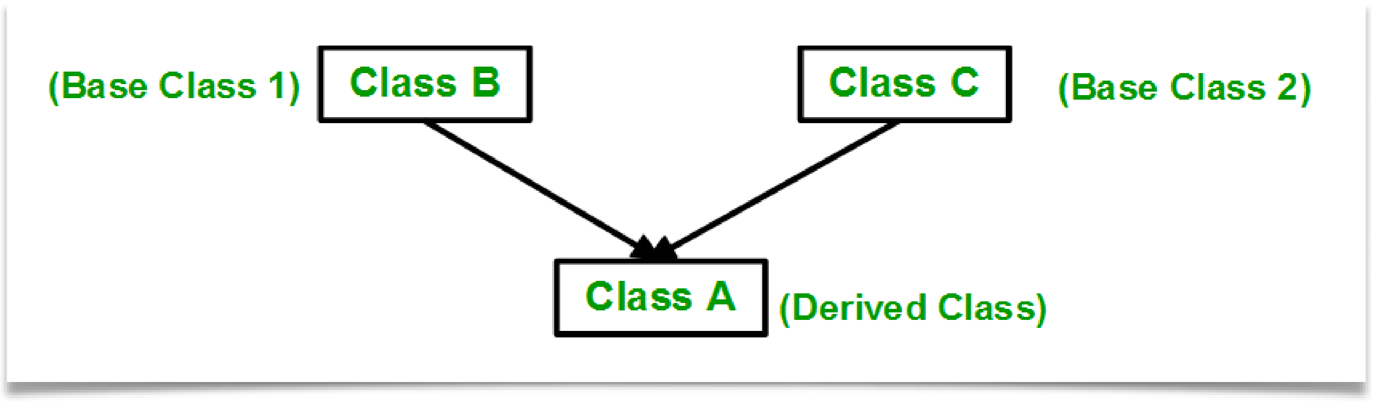
Syntax:
class
subclass_name :
access_mode
base_class
{
 // body of subclass
};

OR

class A
{
... .

```
};

class B: public A
{
... .. ...
};
```

Q. (6) What is multiple inheritance?
Ans: Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one subclass is inherited from more than one base class.



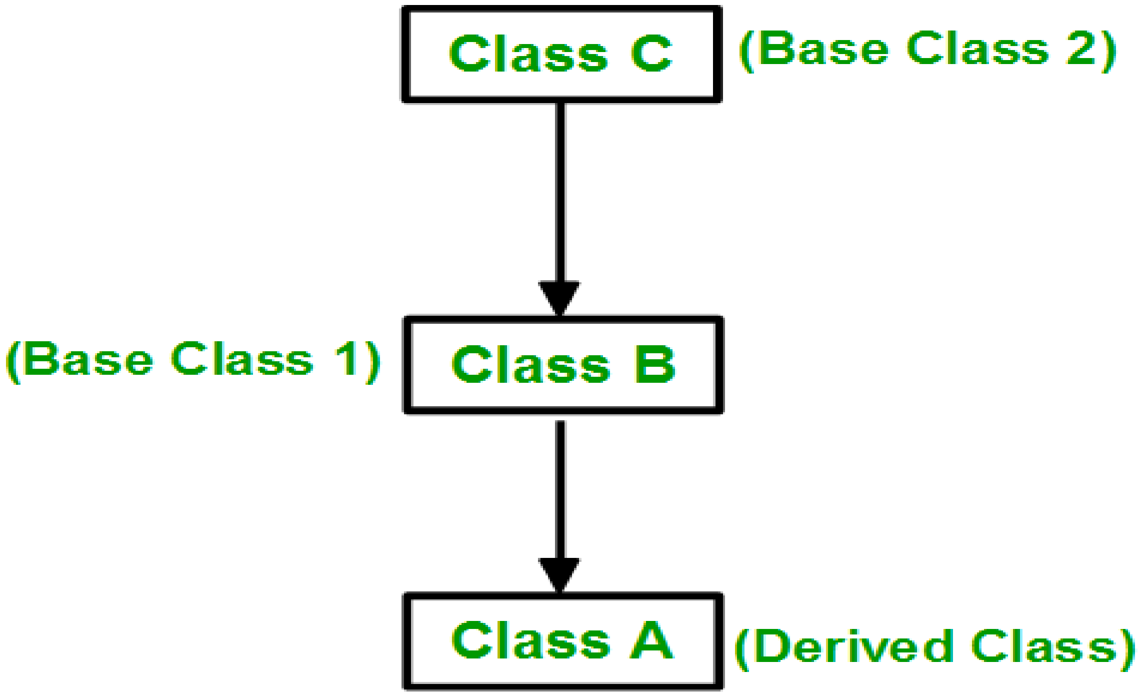
In the given example, class c inherits the properties and behaviour of class B and class A at the same level. So, here A and Class B both are the parent classes for Class C.

Syntax:
```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
  // body of subclass
};

class B
{
... .. ...
};
class C
{
... .. ...
};
class A: public B, public C
{
... ... ...
};
```

Here, the number of base classes will be separated by a comma (‘, ‘) and the access mode for every base class must be specified.

Q. (7) Discuss the multilevel inheritance. Also write the syntax

Ans: Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.



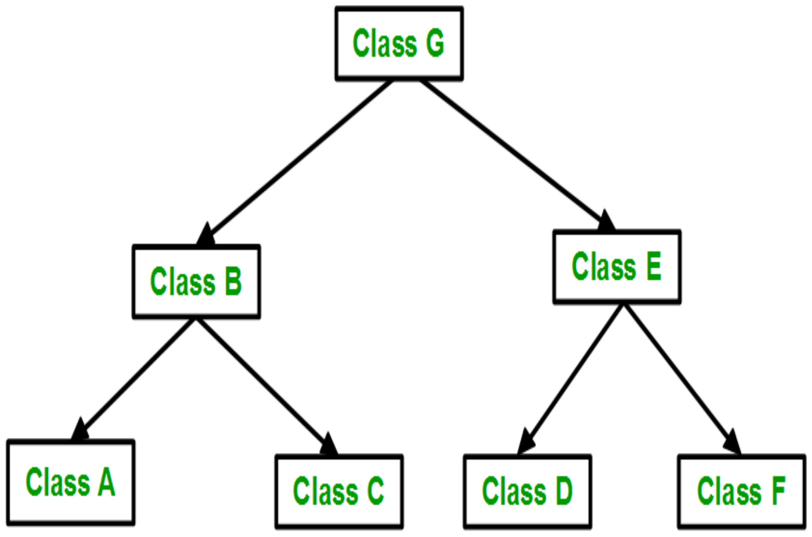
In the given example, class c inherits the properties and behavior of class B and class B inherits the properties and behavior of class B. So, here A is the parent class of B and class B is the parent class of C. So, here class C implicitly inherits the properties and behavior of class A along with Class B i.e there is a multilevel of inheritance.

Syntax:-

```
class C
{
... .. ...
};
class B:public C
{
... .. ...
};
class A: public B
{
... ... ...
};
```

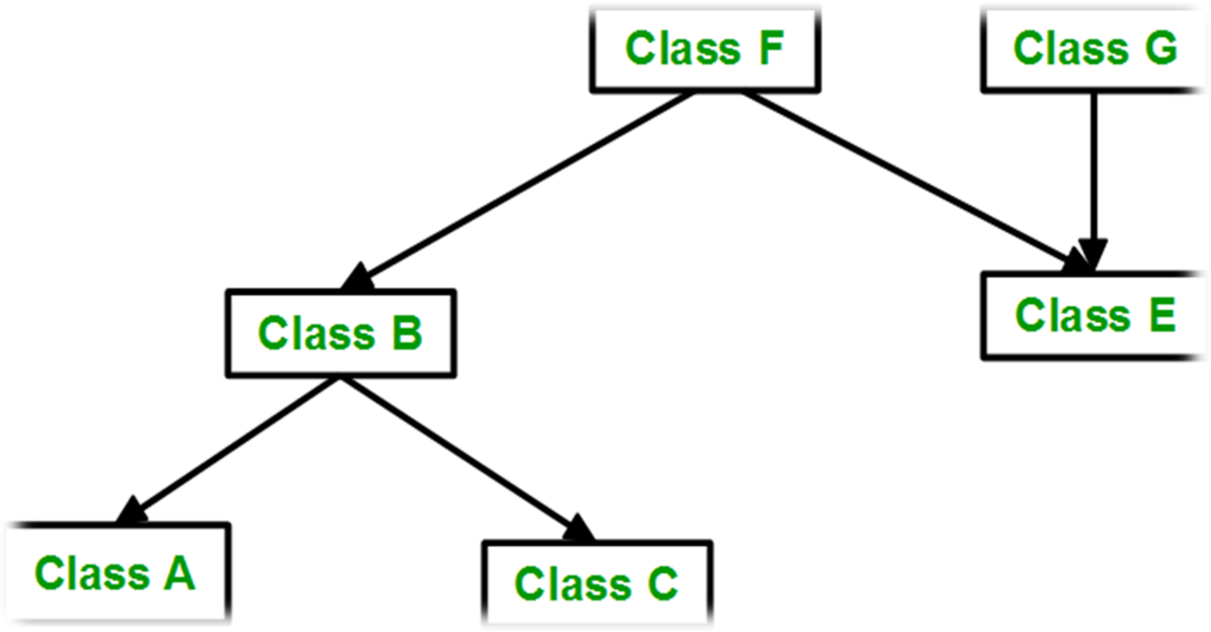
Q. (8) What is hierarchical inheritance?

Ans: Hierarchical Inheritance: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



In the given example, class D inherits the properties and behavior of class C and class B as well as Class A. Both class C and class B inherit the Class A. So, Class A is the parent for Class B and Class C as well as Class D. So it's making it a Multi-path inheritance.

Syntax:-
```
class A
{
   // body of the class A.
}
class B : public A
{
   // body of class B.
}
class C : public A
{
   // body of class C.
}
class D : public A
{
   // body of class
D.
}
```



Q. (9) What is hybrid inheritance?
Ans:
Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.
Below image shows the combination of hierarchical and multiple inheritances:

```
/Base Class
class A
{
 public void fooA()
 {
 //TO DO:
 }
}

//Base Class
class F
{
 public void fooF()
 {
 //TO DO:
 }
}

//Derived Class
class B : A, F
{
 public void fooB()
 {
 //TO DO:
 }
}

//Derived Class
class C : A
{
 public void fooC()
 {
 //TO DO:
 }
}

//Derived Class
class D : C
{
 public void fooD()
 {
 //TO DO:
 }
}

//Derived Class
class E : C
{
 public void fooE()
 {
 //TO DO:
 }
}
```

Q. (10) What is advantages and disadvantages of inheritance?
Ans:
Advantages of Inheritance

- Reduce code redundancy.
- Provides better code reusability's.
- Reduces source code size and improves code readability.
- The code is easy to manage and divided into parent and child classes.
- Supports code extensibility by overriding the base class functionality within child classes. Code usability will enhance the reliability eventually where the base class code will always be tested and debugged against the issues.

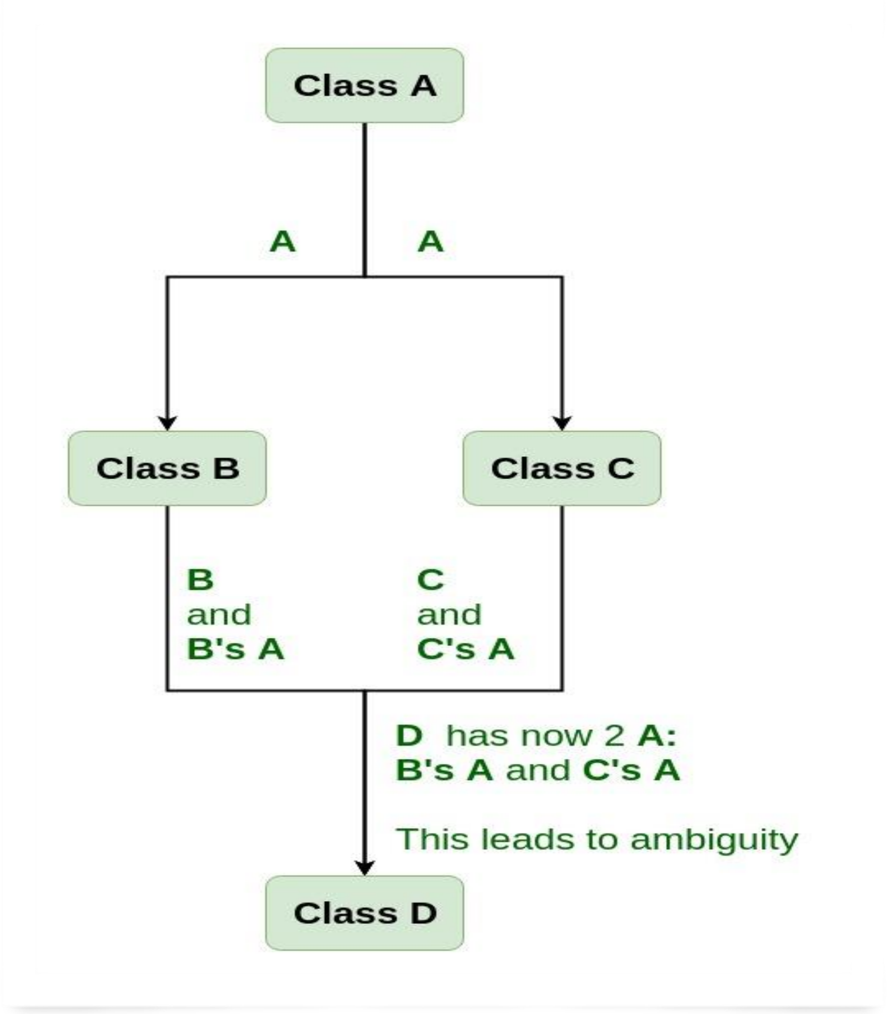
Disadvantages of Inheritance

- In Inheritance base class and child class, both are tightly coupled. Hence If you change the code of the parent class, it will affect all the child classes.
- In a class hierarchy, many data members remain unused and the memory allocated to them is not utilized. Hence it affects the performance of your program if you have not implemented inheritance correctly. Inheritance increases the coupling between the base class and the derived class. any small change in the base class will directly affect all the child classes which are extended to the parent class.

Q. (11) What is virtual base class? What is the need of virtual base class? Virtual base classes are used in virtual inheritance in?

Ans:

An abstract class in C++ has at least one pure virtual function by definition. In other words, a function that has no



definition. The abstract class's descendants must define the pure virtual function; otherwise, the subclass would become an abstract class in its own right.

Abstract classes are used to express broad concepts from which more concrete classes can be derived. An abstract class type object cannot be created. To abstract class types, however, you can use pointers and references. Declare at least one

pure virtual member feature when creating an abstract class. The pure specifier (= 0) syntax is used to declare a virtual function.
Take a look at the example in virtual functions. The aim of the class is to provide general functionality for shape, but objects of type shape are much too general to be useful. Shape is therefore a suitable candidate for an abstract class:
Syntax:

```
C-lass classname //abstract class
{
//data members
public:
//pure virtual function
/* Other members */
};
```

Q. (12) What is nested class?
Ans:
Nested Classes in C++
1. Class inside a class is called nested class.

2. Nested classes are declared inside another enclosing class.

3. A nested class is a member of class and it follows the same access rights that are followed by different members of class.

4. The members of an enclosing class have no other special access to members of nested class. The normal access rules shall be carried out.

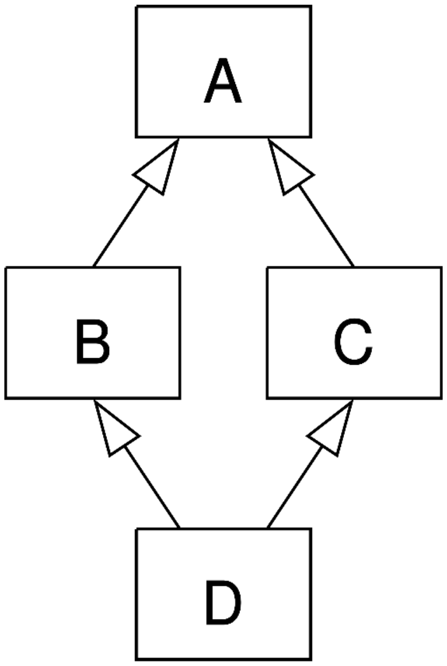
5. If nested class is declared after public access specifiers inside the enclosing class then you must add scope resolution (::) during creating its object inside main function.

High order thinking question

Diamond problem of inheritance

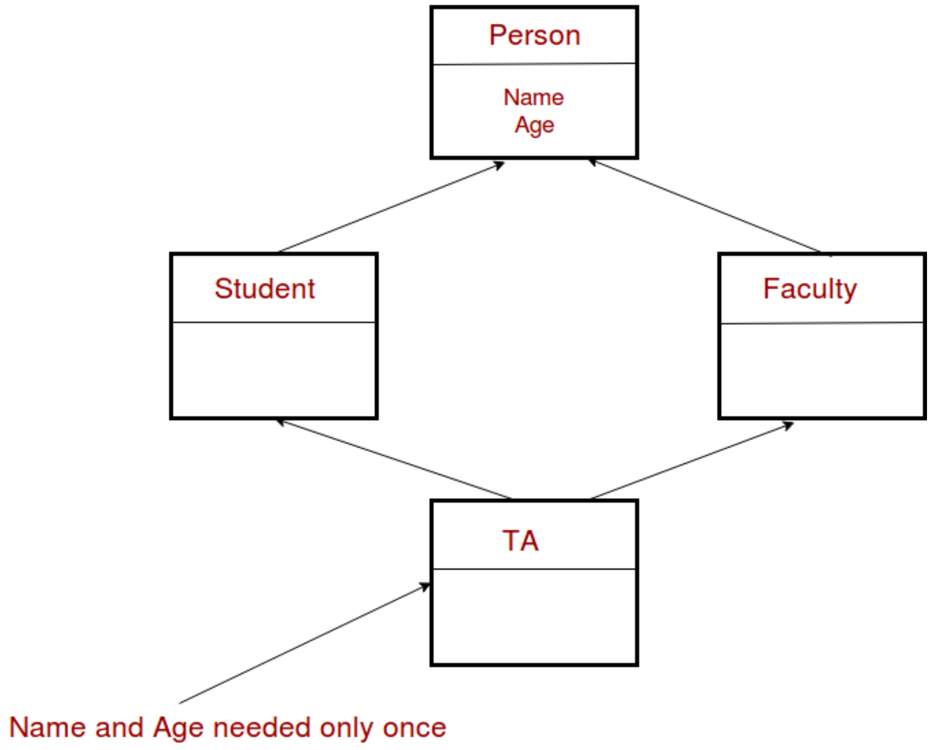
The "diamond problem" (sometimes referred to as the "Deadly Diamond of Death") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C.

If there is a method in A that B and C have overridden, and D does not override it, then which version of the method does D inherit: that of B, or that of C?
It is called the "diamond problem" because of the shape of the class inheritance diagram in this situation. In this case, class A is at the top, both B and C separately beneath it, and D joins the two together at the bottom to form a diamond shape.

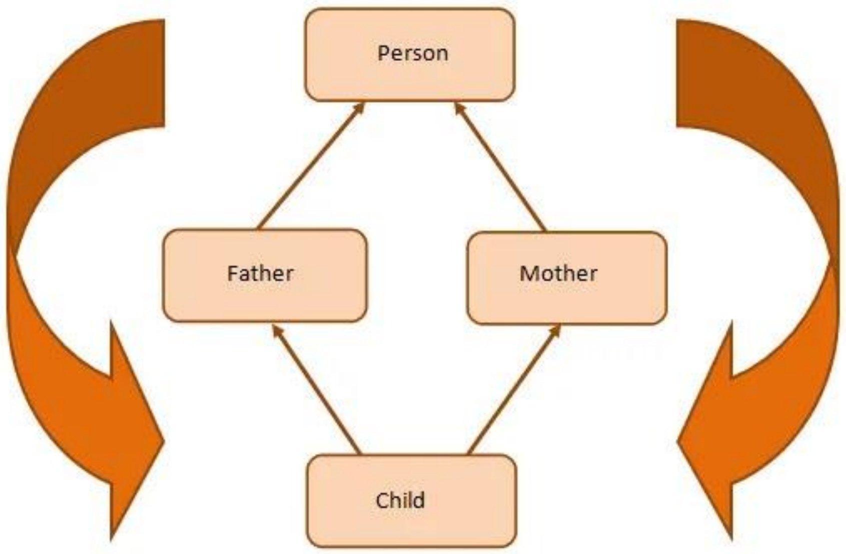


Explanation 02:

The diamond problem The diamond problem occurs when two superclasses of a class have a common base class. For example, in the following diagram, the TA class gets two copies of all attributes of Person class, this causes ambiguities.



Explanation 3: The Diamond Problem occurs when a child class inherits from two parent classes who both share a common grandparent class. This is illustrated in the diagram below:



we have a class Child inheriting from classes Father and Mother. These two classes, in turn, inherit the class Person because both Father and Mother are Person.
As shown in the figure, class Child inherits the traits of class Person twice—once from Father and again from Mother. This gives rise to ambiguity since the compiler fails to understand which way to go.
Solution of the problem:
The solution to the diamond problem is to use the virtual keyword. We make the two parent classes (who inherit from the same grandparent class) into virtual classes in order to avoid two copies of the grandparent class in the child class.

Topic: Pointers, pointer to objects, this pointer, pointer to derived classes, virtual functions, pure virtual functions

MCQ

Q. (1) Which is the pointer which denotes the object calling the member function?

(a) Variable pointer

(b) This pointer

(c) Null pointer

(d) Zero pointer

Ans: (b)

Explanation: The pointer which denotes the object calling the member function is known as this pointer. The this pointer is usually used when there are members in the function with same name as those of the class members.

Q. (2) This pointer is accessible……….

(a) Within all the member functions of the class

(b) Only within functions returning void

(c) Only within non-static functions

(d) Within the member functions with zero arguments

Ans: (c)

Explanation: This pointer is available only within the non-static member functions of a class. If the member function is static, it will be common to all the objects and hence a single object can't refer to those functions independently.

Q. (3) The result of sizeof() function ….

(a) Includes space reserved for this pointer

(b) Includes space taken up by the address pointer by this pointer

(c) Doesn't include the space taken by this pointer

(d) Doesn't include space for any data member

Ans: (c)

Explanation: The space taken by this pointer is not reflected in by the sizeof() operator. This is because object's this pointer is not part of object itself. This is a cross verification for the concept stating that this pointer doesn't take any space in the object

Q. (4) Whenever non-static member functions are called …….

(a) Address of the object is passed implicitly as an argument

(b) Address of the object is passed explicitly as an argument

(c) Address is specified globally so that the address is not used again

(d) Address is specified as return type of the function

Ans: (a)

Explanation: The address is passed implicitly as an argument to the function. This doesn't have to be passed explicitly. The address is passed, of the object which is calling the non-static member function.

Q. (5) Which is the correct interpretation of the member function call from an object, object.function(parameter);

(a) object.function(&this, parameter)

(b) object(&function,parameter)

(c) function(&object,&parameter)

(d) function(&object,parameter)

Ans: (d)

Explanation: The function name is specified first and then the parameter lists. The parameter list is included with the object name along with & symbol. This denotes that the address of the object is being passed as an argument.

Q. (6) Which among the following is/are type(s) of this pointer?

(a) const

(b) volatile

(c) const or volatile

(d) int

Ans: (c)

Explanation: This pointer can be declared const or volatile. This depends on need of program and type of code. This is just an additional feature.

Q. (7) Which syntax doesn't execute/is false when executed?

(a) if(&object != this)
(b) if(&function !=object)
(c) this.if(!this)
(d) this.function(!this)

Ans: (a)

Explanation: The condition becomes false when executed and hence doesn't executes. This is the case where this pointer can guard itself from the self-reference. Here if the address of the object doesn't match with this pointer that means the object doesn't refer itself.

Q. (8) Earlier implementations of C++ …….

(a) Never allowed assignment to this pointer
(b) Allowed no assignment to this pointer
(c) Allowed assignments to this pointer
(d) Never allowed assignment to any pointer

Ans: (c)

Explanation: The earlier, most initial versions of C++ used to allow assignments to this pointer. That used to allow modifications of this pointer. Later that feature got disabled.

Descriptive Questions

Q. (1) What are some real-world applications of pointers in programming with examples? How do you think about pointers in C/C++? What are some practical applications of using pointers?

Ans: Pointer is the address of a variable. It is very powerful feature in C and C++. Have you ever tried to use objective-C? You cannot learn objective-C without learning pointers in C.

Imagine a pointer is an address of a building in a street like Manhattan (where each house is identical). When you forward your pointer (increment it), you will land up in the next identical building. Similarly, when you retreat the pointer.

Size of building (length of building) is determined by the type of pointer. Off-course, C has facility to change the size of building.

I think this metaphor will help.

There are many practical uses. In C, you cannot write swap function without pointers. Also, programs of dynamic linked list cannot be written without pointers.

If you plan to write apps for iOS and Mac OS X, I think you will not be able to do anything without pointers because in objective-C, pointers are almost everywhere.

And this pointer lets you to visualize an object with different perspective (for example, you can visualize a structure as an array of characters or integers using pointers).

Q. (2) Explain pointer to derived classes?

Ans: A single pointer variable can be made to point to objects belonging to different classes.

For example:

```
B *ptr //pointer to class B type variable
B b; //base object
D d; // derived object
ptr = &b; // ptr points to object b
```

In the above example B is base class and D isa derived class from B, then a pointer declared as a pointer to B and point to the object b.

We can make ptr to point to the object d as follow ptr = &d;

We can access those members of derived class which are inherited from base class by base class pointer.

But we cannot access original member of derived class which are not inherited by base class pointer.

We can access original member of derived class which are not inherited by using pointer of derived class.

Example:

```
#include <iostream>
using namespace std;
class base{
 public:
 int b;
 void show(){
 cout<<"\nThe value of b"<<b;
 }
 };
 class derived:public base
 {
 public:
 int d;
 void show()
 {
 cout<<"\nThe value of b="<<b
  <<"\nThe value of d="<<d;

}
};
```

```
int main()
{
 base B;
 derived D;
 base *bptr;
 bptr=&B;
 cout<<"\nBase class pointer assign address of base class object";
 bptr->b=100;
 bptr->show();
 bptr=&D;
 bptr->b=200;
 cout<<"\nBase class pointer assign address of derived class object";
 bptr->show();
 derived *dptr;
 dptr=&D;
 cout<<"\nDerived class pointer assign address of derived class object";
 dptr->d=300;
 dptr->show();
 return 0;
}
```

Q. (3) Explain Virtual Function? And also explain rules for Virtual function?

Ans: A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

They are mainly used to achieve Runtime polymorphism

Functions are declared with a virtual keyword in base class.

The resolving of function call is done at runtime.

Rules for Virtual Functions

Virtual functions cannot be static.

A virtual function can be a friend function of another class.

Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.

The prototype of virtual functions should be the same in the base as well as derived class.

They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.

A class may have virtual destructor but it cannot have a virtual constructor.

Q. (4) Write a program to illustrate how pointers to a base class is used for both base and derived class.

Ans:

```
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
using namespace std;
class Person{
 public:
 void Print(){
 cout << "print" << endl;
}
private:
int size;
};
class Teacher :
 public Person{
 public: void PrintT(){
cout << "teacher" << endl;
}
 private:
 int age;
};
void print(Person* P) {
P->Print();
}
int main(){
Person P;
Teacher T;
print(&T);
}
```

Q. (5) (Pointers to objects) Define a class Item that is used to store and display the information regarding the item number and its price. Write a program to store and display the details of one item by using both normal object and pointer to object separately. Display appropriate message wherever necessary.

Ans:

```
#include<iostream>
using namespace std;
class Item{
public:
int num,p;
double price;
void getDetails(){
cout<<"Enter the number of the items: ";
cin>>num;
cout<<"Enter the price of the item: ";
cin>>p; price=num*p;
}
void Display () {
cout<<"The price of the item is: "<<price;
}
};
int main () {
Item m;
m.getDetails();
m.Display();
}
```

Module 4

Topic: Console I/O operations, working with files and templates – C++ streams and stream classes

MCQ

Q. (1) Select the namespace on which the stream classes are defined?

(a) System.IO
(b) System.Input
(c) System.Output
(d) All of the mentioned

Ans: (a)

Clarification: The core stream classes are defined within the System.IO namespace. To use these classes, you will usually include the following statement near the top of your program: using System.IO;

Q. (2) Choose the class on which all stream classes are defined?

(a) System.IO.stream

(b) Sytem.Input.stream

(c) System.Output.stream

(d) All of the mentioned

Ans: (a)

Explanation: The core stream class is System.IO.Stream. Stream represents a byte stream and is a base class for all other stream classes. It is also abstract, which means that you cannot instantiate a Stream object. Stream defines a set of standard stream operations.

Q. (3) Choose the stream class method which is used to close the connection?

(a) close()

(b) static close()

(c) void close()

(d) none of the mentioned

Ans: (c)

Explanation: void close() closes the stream.

Clarification: void close() close.

Q. (4) The method used to write a single byte to an output stream?

(a) void WriteByte(byte value)

(b) int Write(byte[] buffer ,int offset ,int count)

(c) write()

(d) none of the mentioned

Ans: (a)

Explanation: Writes a single byte to an output stream.

Q. (5) How many streams are automatically created when executing a program?

(a) 1

(b) 2

(c) 3

(d) 4

Ans: (c)

Explanation: There are 3 streams that are automatically created when executing a program. They are stdin, stdout and stderr.

Descriptive Questions

Q. (1) Why getline() is preferred over get() for inputting multiline strings ?
Ans: Since getline () functions discards the delimiter rather than leaving it in the input stream until the next input operation. This is not in the caseof get() function.
Which of the predefined stream objects are automatically opened when the program execution starts ?

Answer-: cin, cout, cerr and clog.

Q. (2) List out C++ Stream Classes.
Ans: ios (General I/O stream class)

- Contains basic facilities that are used by all other input and output classes.
- Also contains a pointer to a buffer object.
- Declares constants and functions that are necessary for handling formatted input and output functions.

istream (Input stream)

- Inherits the properties of ios.
- Declares input functions such as get(), getline() and read()
- Contains overloaded extraction operator >>

ostream (output stream)

- Inherits the properties of ios.
- Declares output functions such as put() and write()
- Contains overloaded insertion operator <<

iostream (I/O stream)

- Inherits the properties of ios, istream and ostream through multiple inheritance and
- thus contains all the input and output functions.

streambuf

- Provides an interface to physical devices through buffers.
- Acts as a base for filebuf class used ios files.

Q. (3) List out and explain Unformatted I/O Operations.
Ans:

- C++ language provides a set of standard built-in functions which will do the work of reading and displaying data or information on the I/O devices during program execution.
- Such I/O functions establish an interactive communication between the program and user.

Function	Syntax	Use
cout	cout<<" "<<" ";	To display character, string and number on output device.
cin	cin>> var1>>var2;	To read character, string and number from input device.
get(char*)	char ch;	To read character including blank space, tab and newline character from input device.
	cin.get(ch);	It will assign input character to its argument.

get(void)	char ch;
	ch=cin.get();
		To read character including blank space, tab and newline character from input device.
		
		It will returns input character.

put()	char ch;
	cout.put(ch);
		To display single character on output device.
		
		If we use a number as an argument to the function put(), then it will convert it into character.

getline()	char name[20];
	int size=10;
	cin.getline(name,size);
		It is used to reads a whole line of text that ends with a newline character or size -1 character.
		First argument represents the name of string and second argument indicates the number of character to be read.

write()	char name[20];
	int size=10; cout.write(name,size);
		It is used to display whole line of text on output device.
		
		First argument represents the name of string and second argument indicates the number of character to be display.

Topic: unformatted I/O operations, formatted console I/O operations, managing output with manipulators. Classes for file stream operations, opening/closing of file, file pointers and their manipulation, error handling during file operation

MCQ

Q. (1) By default, all the files in C++ are opened in ……..mode.

(a) Text

(b) Binary

(c) ISCII

(d) VTC

Ans: (a)

Explanation: By default, all the files in C++ are opened in text mode. They read the file as normal text.

Q. (2) Which of the following operators cannot be overloaded

(a) . (Member Access or Dot operator)

(b) ?: (Ternary or Conditional Operator)

(c) :: (Scope Resolution Operator)

(d) .* (Pointer-to-member Operator)

(e) All of the above

Ans: (e)

Q. (3) The default constructor of ios class is

(a) public

(b) private

(c) protected

(d) any of them

Ans: (b)

Q. (4) The ……..operator must have at least one operand of user defined type.

(a) New

(b) Overloaded

(c) Existing

(d) Binary

Ans: (b)

Q. (5) Select a function which is used to a write a string to a file ____

(a) pits()

(b) putc()

(c) fputc()

(d) fgets()

Ans: (c)

Descriptive Questions

Q. (1) What is Set precision()?
Ans: setprecision() is a function in Manipulators in C++:
It is an output manipulator that controls the number of digits to display after the decimal for a floating point integer.
Syntax:

```
    setprecision (int p)
```

Example:

```
        float A = 1.34255;

        cout <<fixed<< setprecision(3) << A << endl;
```

Q. (2) What are Filling or Padding Characters in C++? Give examples.
Ans: It Behaves as if member fill were called with c as argument on the stream on which it is inserted as a manipulator (it can be inserted on output streams).
This manipulator is declared in header <iomanip>.
Example

```
#include <iostream>
#include <iomanip> //for setfill and setw

int main ()
{
  std::cout << std::setfill ('x') << std::setw (10);
  std::cout << 9 << std::endl;
  getchar();
  return 0;
}
OUTPUT: xxxxxxxxx9
```

Q. (3) What is the difference between put() and get()?
Ans: A GET request get() fetches data from a server / database.
A PUT request put() updates a record, or multiple records that already exist on a server / database.

Q. (4) A file contains a list of telephone numbers in the following form:
Arvind 7258031
Sachin 7259197

The names contain only one word the names and telephone numbers are separated by white spaces Write program to read a file and display its contents in two columns.
Ans:
CODE:

```
print("Name\t|\tPhone no. ")

file = open("Path Walla.txt", "r")
lst = file.readlines()

for i in lst :
    data = i.split()
    print( data[0] ,end = "\t" )
    print("|" , end = "\t")
    print ( data[1] )

file.close()
```

INPUT:
```
Kundan 7258031
Aditya 7259197
Ram 256445
Rohit 799645
Virat 6566363
Rahul 534553
```

OUTPUT:
```
Name      |    Phone no.
Kundan |    7258031
Aditya    |    7259197
Ram        |    256445
Rohit      |    799645
Virat     |    6566363
Rahul  |    534553
```

Q. (5) Two files named “Source1” and “Source2” contain sorted list of integers. Write a program that reads the contents of both the files and stores the merged list in sorted form in a new file named „Target‟.

Ans: CODE:

```
#include<stdio.h>
#include<stdlib.h>
int main()

{
    // open two files to be merged
   FILE *fp1=fopen(“source1.txt”,”r”);
   FILE *fp2=fopen(“source2.txt”,”r”);

//open file to store the result
FILE *fp3=fopen(“Target.txt”,”W”);
char c;
```

```
if(fp1==NULL||fp2==NULL||fp3==NULL)
{
      puts("could not open files");
      exit(0);
}
//copy contents of first file to Target.txt
while((c==fget(fp1))!==EOF)
{
       fputc(c,fp3);
//copy contents of second file to Target.txt
while((c==fgetc(fp2))!=EOF)
{
     fputc(c,fp3);
     printf("Merged Source1.txt and Source2.txt into Target.txt");

     fclose(fp1);
     fclose(fp2);
     fclose(fp3);

     return 0;

}
```

Topic: command line arguments. Class templates, class template with multiple parameters, function templates

MCQ

Q. (1) How the template class is different from the normal class?

(a) Template class generate objects of classes based on the template type

(b) Template class saves system memory

(c) Template class helps in making genetic classes

(d) All of the mentioned

Ans: (d)

Explanation: Size of the object of template class varies depending on the type of template parameter passed to the class. Due to which each object occupies different memories on system hence saving extra memories. Template class also helps in making generic classes.

Q. (2) What is the difference between normal function and template function?

(a) The normal function works with any data types whereas template function works with specific types only

(b) Template function works with any data types whereas normal function works with specific types only

(c) Unlike a normal function, the template function accepts a single parameter

(d) Unlike the template function, the normal function accepts more than one parameter

Ans: (b)

Explanation: As a template feature allows you to write generic programs. therefore, a template function works with any type of data whereas normal function works with the specific types mentioned while writing a program. Both normal and template function accepts any number of parameters.

Q. (3) 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 are separated by space in the terminal

(d) All of the mentioned

Ans: (d)

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

Q. (4) What does this template function indicates?

```
==================
template<class T>
T func (T a)
{
        cout<<a;
}
==================
```

(a) A function taking a single generic parameter and returning a generic type

(b) A function taking a single generic parameter and returning nothing

(c) A function taking single int parameter and returning a generic type

(d) A function taking a single generic parameter and returning a specific non-void type

Ans: (a)

Explanation: As the return type of function is template T, therefore, the function is returning a general type. Now as the function is taking a template T as its argument which is a general type, therefore, it is accepting a single general type argument.

Q. (5) Which of the following gives the name of the program if the second parameter to the main function is char **argv?

(a) argv[3]

(b) argv[1]

(c) argv[0]

(d) argv [2]

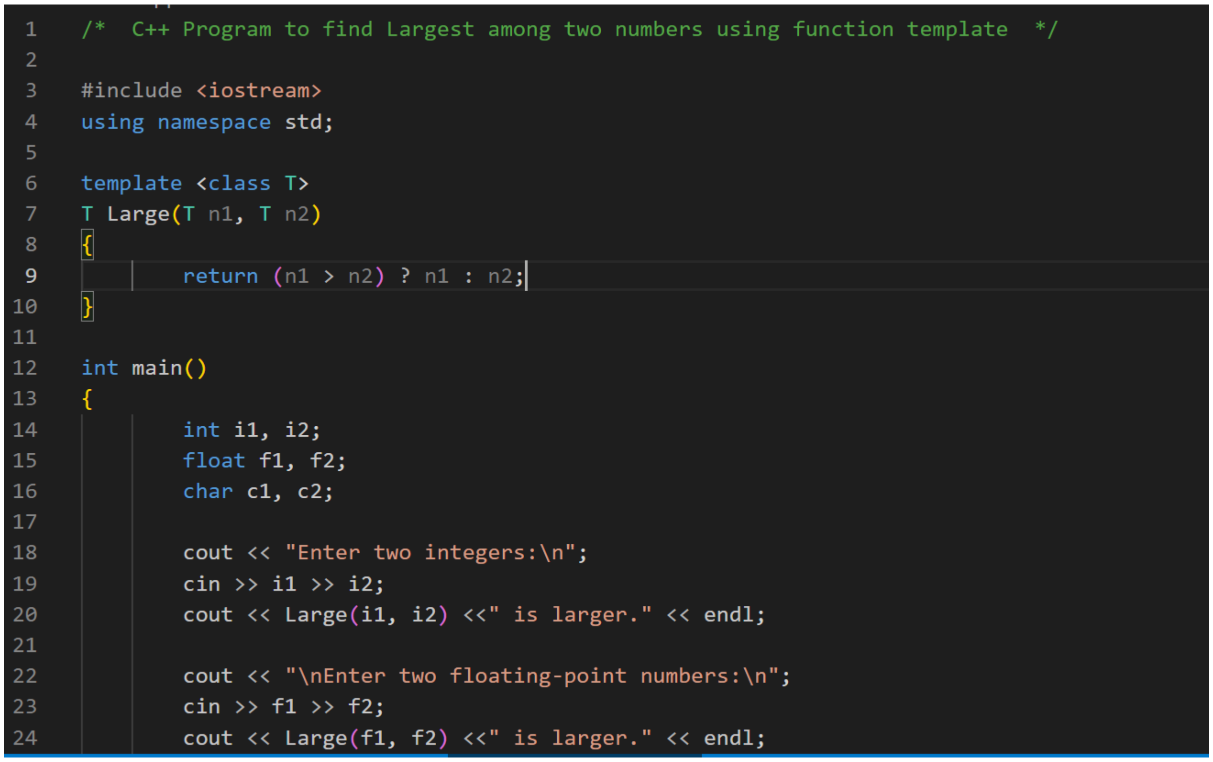
Ans: (c)

Explanation: The first string in the list of command line arguments represents the name of the program which can be accessed by using argv [0].

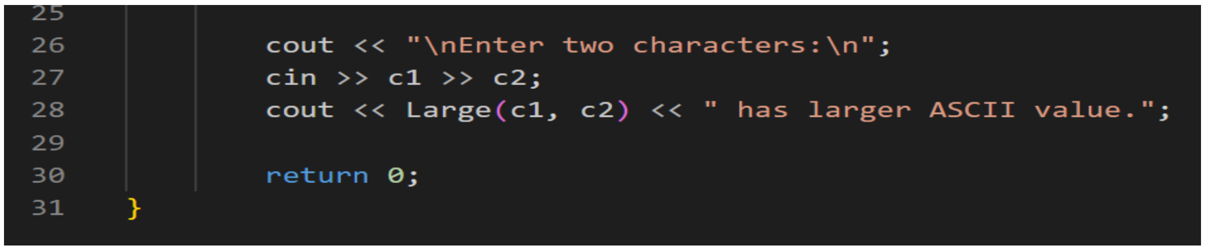
Descriptive Questions

Q. (1) Write c++ program to find largest among two numbers using function template.

Ans:

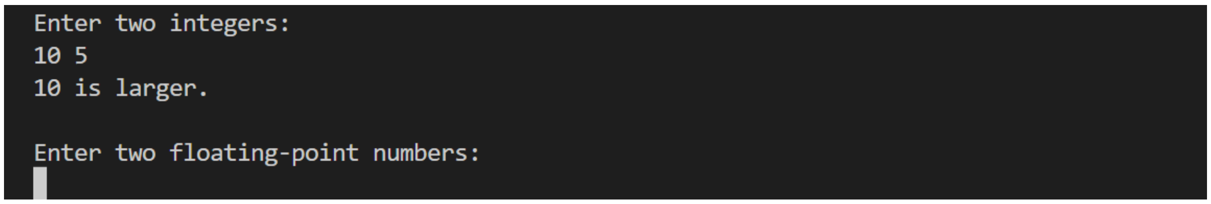


```cpp
/*  C++ Program to find Largest among two numbers using function template  */

#include <iostream>
using namespace std;

template <class T>
T Large(T n1, T n2)
{
        return (n1 > n2) ? n1 : n2;
}

int main()
{
        int i1, i2;
        float f1, f2;
        char c1, c2;

        cout << "Enter two integers:\n";
        cin >> i1 >> i2;
        cout << Large(i1, i2) <<" is larger." << endl;

        cout << "\nEnter two floating-point numbers:\n";
        cin >> f1 >> f2;
        cout << Large(f1, f2) <<" is larger." << endl;
```



```cpp

        cout << "\nEnter two characters:\n";
        cin >> c1 >> c2;
        cout << Large(c1, c2) << " has larger ASCII value.";

        return 0;
}
```

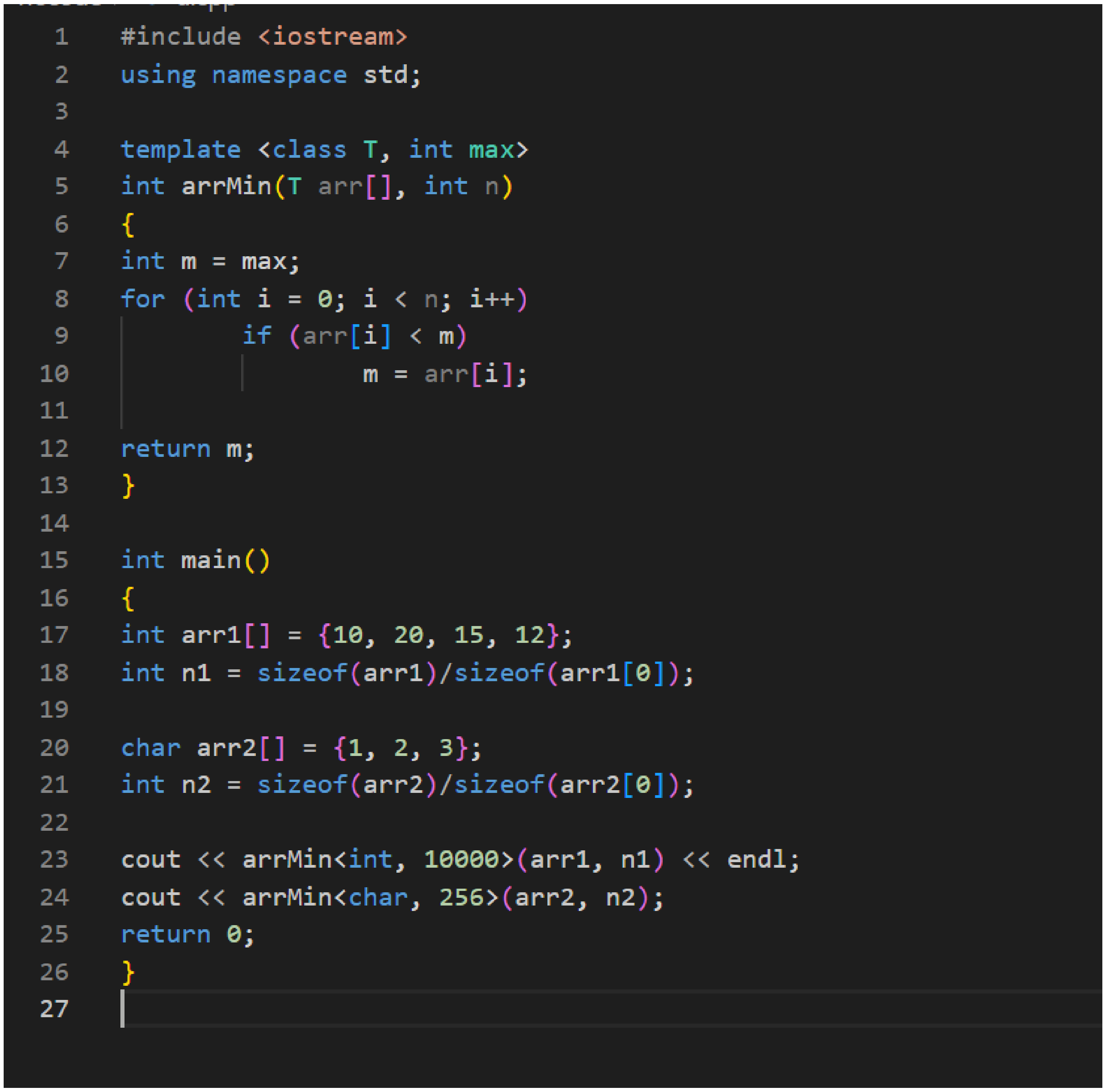
<u>Output</u>



```
Enter two integers:
10 5
10 is larger.

Enter two floating-point numbers:
```

Q. (2) Give an example of Non type template.

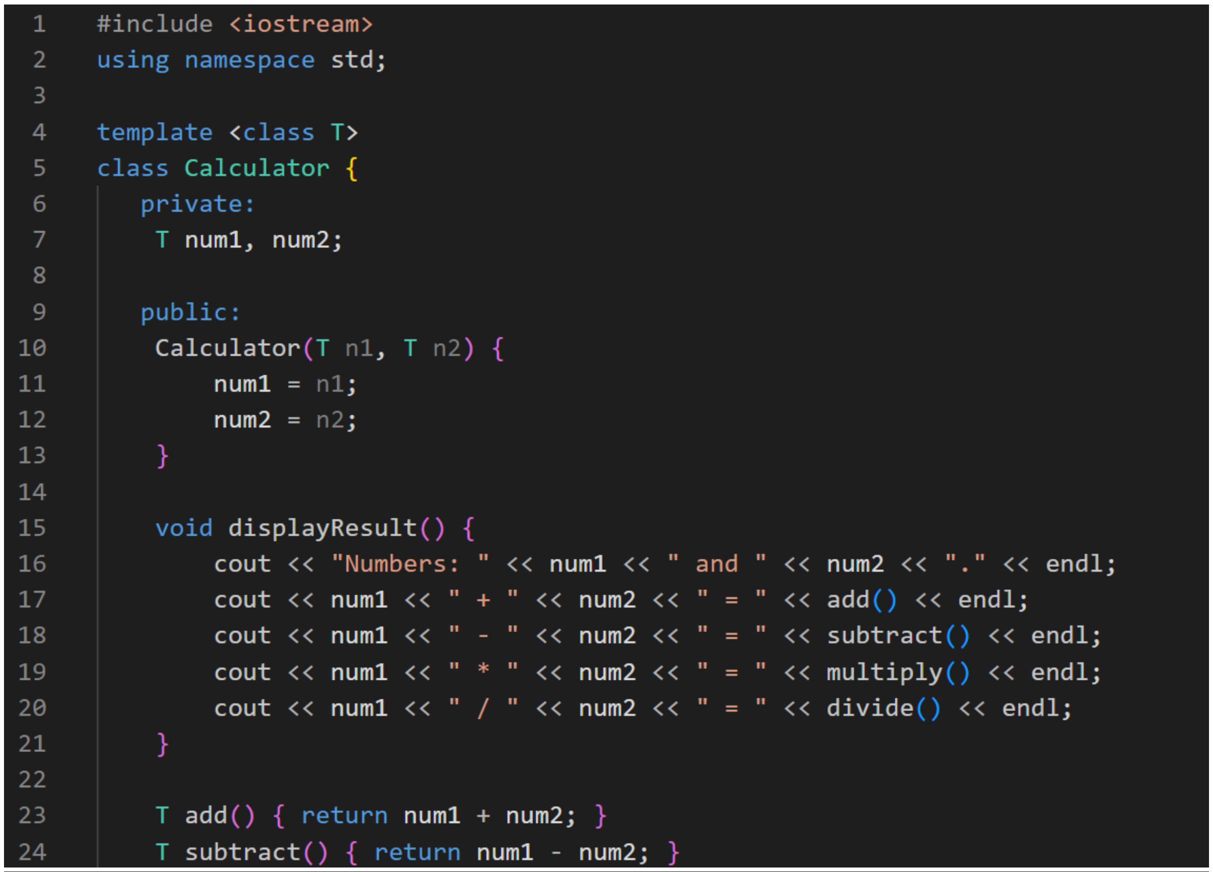
Ans:



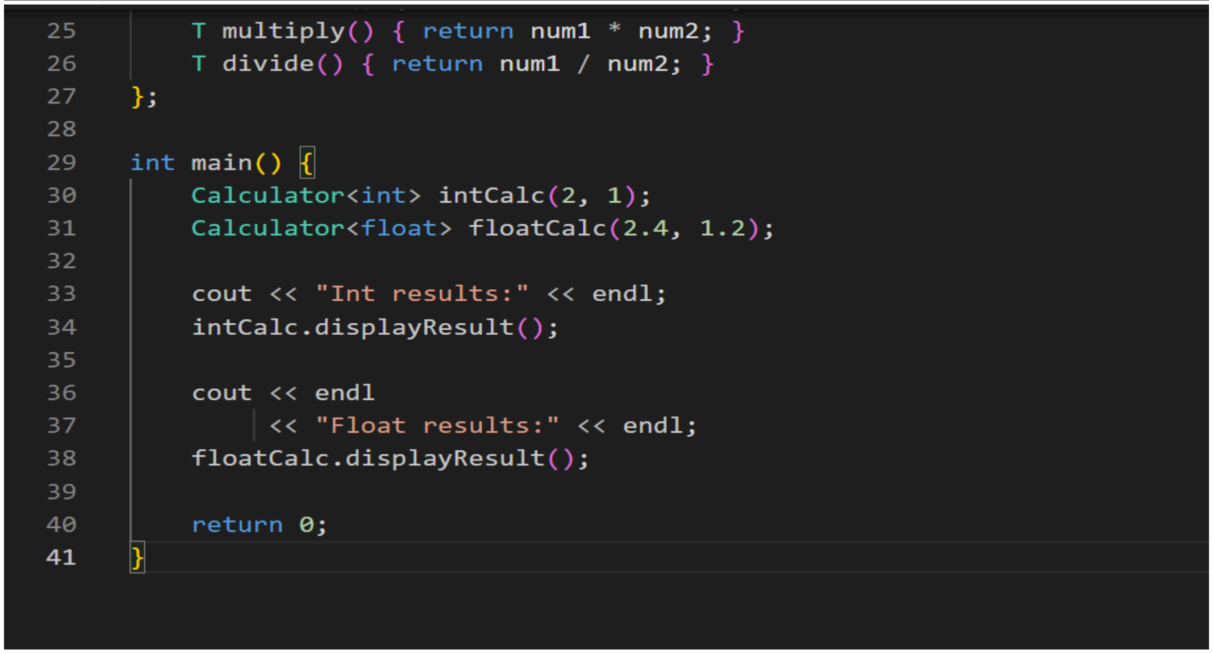
```cpp
#include <iostream>
using namespace std;

template <class T, int max>
int arrMin(T arr[], int n)
{
int m = max;
for (int i = 0; i < n; i++)
        if (arr[i] < m)
                m = arr[i];

return m;
}

int main()
{
int arr1[] = {10, 20, 15, 12};
int n1 = sizeof(arr1)/sizeof(arr1[0]);

char arr2[] = {1, 2, 3};
int n2 = sizeof(arr2)/sizeof(arr2[0]);

cout << arrMin<int, 10000>(arr1, n1) << endl;
cout << arrMin<char, 256>(arr2, n2);
return 0;
}
```

Q. (3) Make a calculator using template.

Ans:

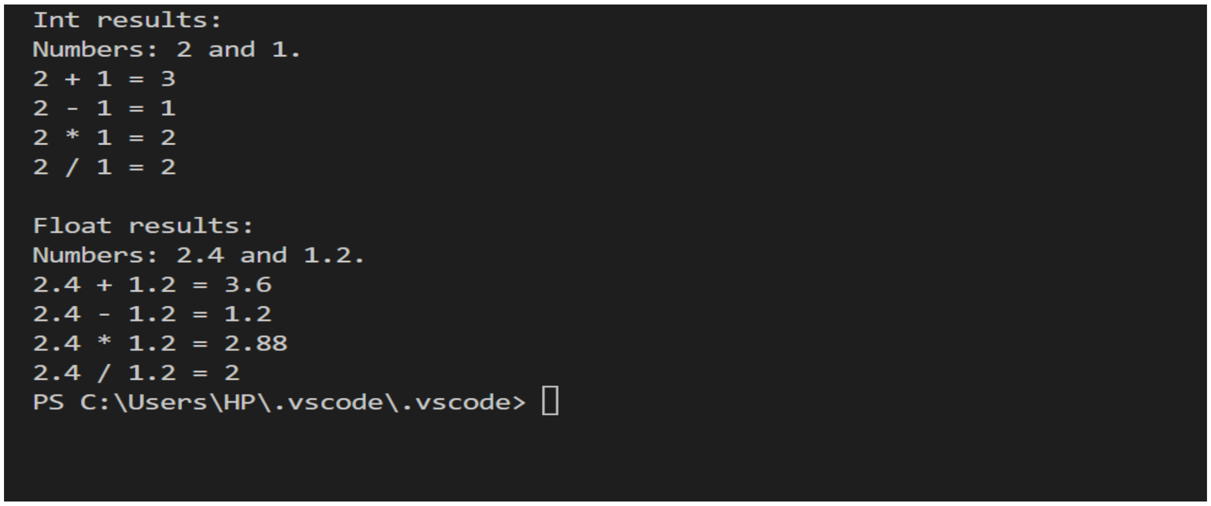


```cpp
#include <iostream>
using namespace std;

template <class T>
class Calculator {
   private:
    T num1, num2;

   public:
    Calculator(T n1, T n2) {
        num1 = n1;
        num2 = n2;
    }

    void displayResult() {
        cout << "Numbers: " << num1 << " and " << num2 << "." << endl;
        cout << num1 << " + " << num2 << " = " << add() << endl;
        cout << num1 << " - " << num2 << " = " << subtract() << endl;
        cout << num1 << " * " << num2 << " = " << multiply() << endl;
        cout << num1 << " / " << num2 << " = " << divide() << endl;
    }

    T add() { return num1 + num2; }
    T subtract() { return num1 - num2; }
```



```cpp
    T multiply() { return num1 * num2; }
    T divide() { return num1 / num2; }
};

int main() {
    Calculator<int> intCalc(2, 1);
    Calculator<float> floatCalc(2.4, 1.2);

    cout << "Int results:" << endl;
    intCalc.displayResult();

    cout << endl
         << "Float results:" << endl;
    floatCalc.displayResult();

    return 0;
}
```

Output:-



```
Int results:
Numbers: 2 and 1.
2 + 1 = 3
2 - 1 = 1
2 * 1 = 2
2 / 1 = 2

Float results:
Numbers: 2.4 and 1.2.
2.4 + 1.2 = 3.6
2.4 - 1.2 = 1.2
2.4 * 1.2 = 2.88
2.4 / 1.2 = 2
PS C:\Users\HP\.vscode\.vscode> 
```

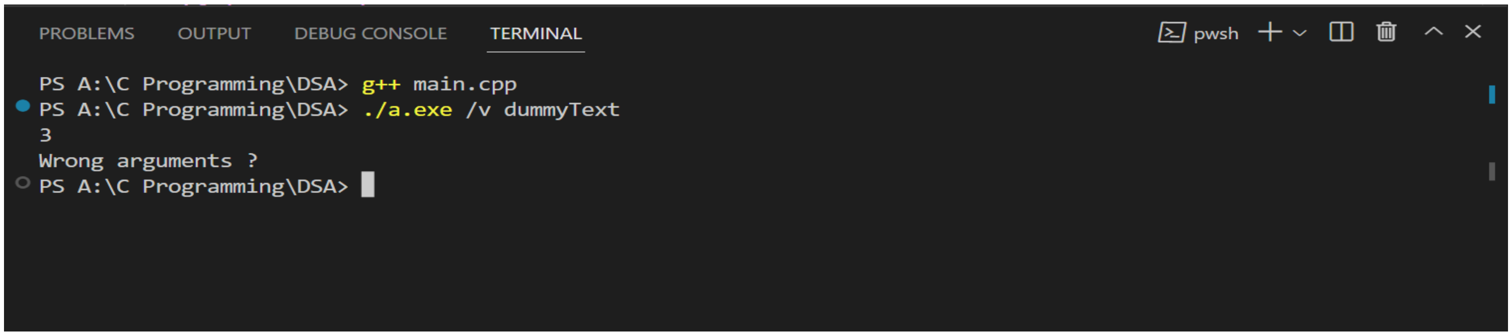
Q. (4) Give a real-life example of Templates.

Ans: When we go to pizza shop, we order pizza, there we can order default pizza or we can modify it. Like we have ordered mushroom pizza so we can modify it by replacing Mushroom with paneer or anything else so we can modify that pizza according to our taste, this is real life example of template.

Q. (5) Why the code given below is not giving desired result?

Ans:

```
#include <bits/stdc++.h>
#define ll long long int
using namespace std;
int main(int argc, char * argv[])
{
   cout << argc << endl;
   if (argc == 3)
   {
      if (argv[1] == "/v"){cout << (argv[2]) <<endl;}
      else if (string(argv[1]) == "/a") {cout << argv[2] << endl;}
      else {cout << "Wrong arguments ?" << endl;}

   }
   else cout << "Give 3 arguments only" << endl;
return 0;
}
```



```
PS A:\C Programming\DSA> g++ main.cpp
PS A:\C Programming\DSA> ./a.exe /v dummyText
3
Wrong arguments ?
PS A:\C Programming\DSA> 
```

Q. (6) In terminal_g++ main.cpp./a.exe /v dummyText, How to solve it?

Ans:

Before Getting to the problem, let us understand the nature of `char *argv[]`.

`char *argv[]` is the array of pointers to char which means that basically each index has a base address of a character array stored at it. And therefore, first we must dereference it.

After dereferencing the first index of argv (*(argv[1])) we will get access of ‘/’.

However, to get the access of ‘v’ we must get access to address next to argv[1].

We will do it by *(argv[1]+1) .

But this is a very tedious job. Therefore, a short method for doing the same is:

Give `argv[1]` as the argument to the string function as given in the code given below.

```
#include <bits/stdc++.h>
using namespace std;
int main(int argc, char * argv[])
{
    cout << argc << endl;
    if (argc == 3)
    {
        if (*(argv[1]) == '/' and *(argv[1]+1) == 'v') {cout << (string(argv[2])) <<endl;}
        else if (string(argv[1]) == "/a") {cout << argv[2] << endl;}
        else {cout << "Wrong arguments ?" << endl;}
    }
    else cout << "Give 3 arguments only" << endl;
return 0;
}
```

Topic: overloading template functions, member function templates, non-type template arguments

MCQ

Q. (1)

Descriptive Questions

Q. (1)

Module 5

Topic: Exception handling and Standard template library – Basics of exception handling

MCQ

Q. (1) What should be put in a try block?
1. Statements that might cause exceptions
2. Statements that should be skipped in case of an exception

(a) Only 1
(b) Only 2

(c) Both 1 and 2
(d) None

Ans: (c) Both 1 and 2
Explanation:
The statements which may cause problems are put in try block. Also, the statements which should not be executed after a problem occurred, are put in try block. Note that once an exception is caught, the control goes to the next line after the catch block.

Q. (2) Which of the following is true about exception handling in C++?

1. There is a standard exception class like Exception class in Java.
2. All exceptions are unchecked in C++, i.e., compiler doesn't check if the exceptions are caught or not.
3. In C++, a function can specify the list of exceptions that it can throw using comma separated list like following.

void fun(int a, char b) throw (Exception1, Exception2, ..)

(a) 1 and 3
(b) 1, 2 and 3
(c) 1 and 2
(d) 2 and 3

Ans: (b) 1, 2 and 3

Q. (3) In nested try-catch block, if the inner catch block gets executed, then …………

(a) Program stops immediately
(b) Outer catch block also executes
(c) Compiler jumps to the outer catch block and executes remaining statements of the main() function
(d) Compiler executes remaining statements of outer try-catch block and then the main() function

Ans: (d) Compiler executes remaining statements of outer try-catch block and then the main() function
Explanation:
The inner catch block will be executed then remaining part of the outer try block will be executed and then the main bock will be executed.

Q. (4) By default, what a program does when it detects an exception?

(a) Continue running
(b) Calls other functions of the program
(c) Results in the termination of the program
(d) Removes the exception and tells the programmer about an exception

Ans: (c) Results in the termination of the program
Explanation:
By default, whenever a program detects an exception the program crashes as it does not know how to handle it hence results in the termination of the program.

Q. (5) Irrespective of exception occurrence, catch handler will always get executed.

(a) True
(b) False

Ans: (b) False
Q. (6) From which STL we can insert/remove data from anywhere?

(a) Vector
(b) Deque
(c) Stack
(d) List

Ans: (d)

Q. (7) What is the correct way to initialize vector in C++?

(a) std::vector<integer> vecOfInts;
(b) std::vector int <vecOfInts>
(c) std::vector(int) vecOfInts;
(d) std::vector<int> vecOfInts;

Ans: (d)

Q. (8) What is true about his statement in C++?

```
std::vector<int> vecInts(5);
```

(a) Initialize a Vector with 5 int & all default value is 0
(b) Initialize a Vector with 5 int
(c) Initialize a Vector with 5 int & all default values will be garbage
(d) Initialize a Vector with 5 int & All default value with be -1

Ans: (a)

Q. (9) Which data structure is used by Map?

(a) AVL Tree
(b) Binary Tree
(c) Balance Binary Tree
(d) None of these

Ans: (c)

Q. (10) From which STL we can insert/remove data from anywhere?

(a) Vector
(b) Deque
(c) Stack
(d) List

Ans: (d)

Q. (11)

Output of following program

```
#include<iostream>
using namespace std;
 class Base {};
class Derived: public Base {};
int main()
{
   Derived d;
   try {
       throw d;
   }
   catch(Base b) {
        cout<<"Caught Base Exception";
   }
   catch(Derived d) {
        cout<<"Caught Derived Exception";
   }
   return 0;
}
```

(a) Caught Derived Exception
(b) Caught Base Exception
(c) Compiler Error

Ans: (b) Caught Base Exception

Q. (12)

Output of following program

```
#include <iostream>
using namespace std;
 int main()
{
    try
    {
       throw 'a';
    }
    catch (int param)
    {
        cout << "int exceptionn";
    }
    catch (...)
    {
        cout << "default exceptionn";
    }
    cout << "After Exception";
    return 0;
}
```

(a) default exception
    After Exception
(b) int exception
    After Exception
(c) int exception
(d) default exception

Ans: (a) default exception
              After Exception

Q. (13)

Output of following program

```
#include <iostream>
using namespace std;
 class Test {
public:
   Test() { cout << "Constructing an object of Test " << endl; }
  ~Test() { cout << "Destructing an object of Test "  << endl; }
};

int main() {
  try {
    Test t1;
    throw 10;
  } catch(int i) {
    cout << "Caught " << i << endl;
  }
}
```

(a) Caught 10
(b) Constructing an object of Test
    Caught 10
(c) Constructing an object of Test

Destructing an object of Test
Caught 10

(d) Compiler Errror

Ans: (c) Constructing an object of Test
Destructing an object of Test
Caught 10

Q. (14)Will below program run without any compilation errors?
Assume that all header files are included.

```
int main()
{

std::list<std::string> listOfStr;
listOfStr.push_back("1");
listOfStr.push_back("2");
listOfStr.push_back("3");
listOfStr.push_back("4");

// Initialize a vector with std::list
std::vector<std::string> vecOfStr(listOfStr.begin(),listOfStr.end());

for(std::string str : vecOfStr)
   std::cout<<str<<std::endl;

return 0;
}
```

(a) Yes
(b) No

Ans: (a) Yes

Q. (15) What should be the output of below program? Assume that all header files are included.

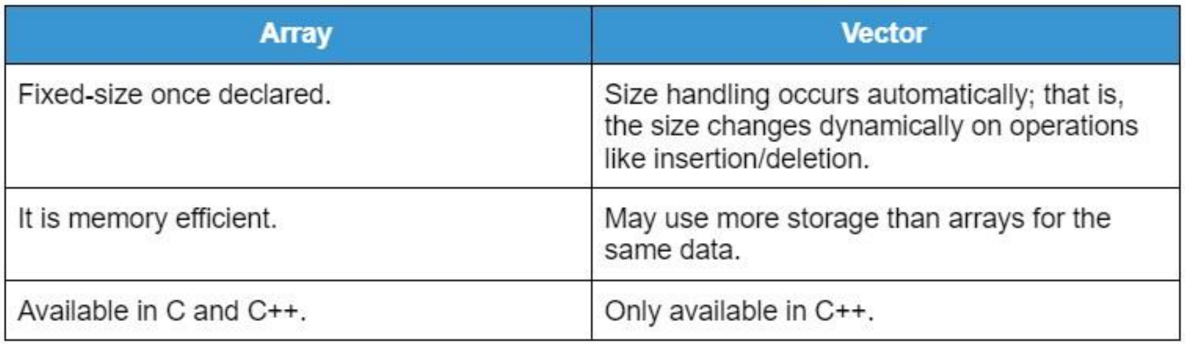
```
int main()
{
std::list<std::string> listOfStr;
listOfStr.push_back("1");
listOfStr.push_back("2");
listOfStr.push_back("3");
listOfStr.push_back("4");

// Initialize a vector with std::list
std::vector<std::string> vecOfStr(listOfStr.begin(), listOfStr.end());

for(std::string str : vecOfStr)
   std::cout<<str;

return 0;
}
```

(a) 1234
(b) 1234
(c) 12340
(d) 01234

Ans: (b) 1234

Descriptive Questions

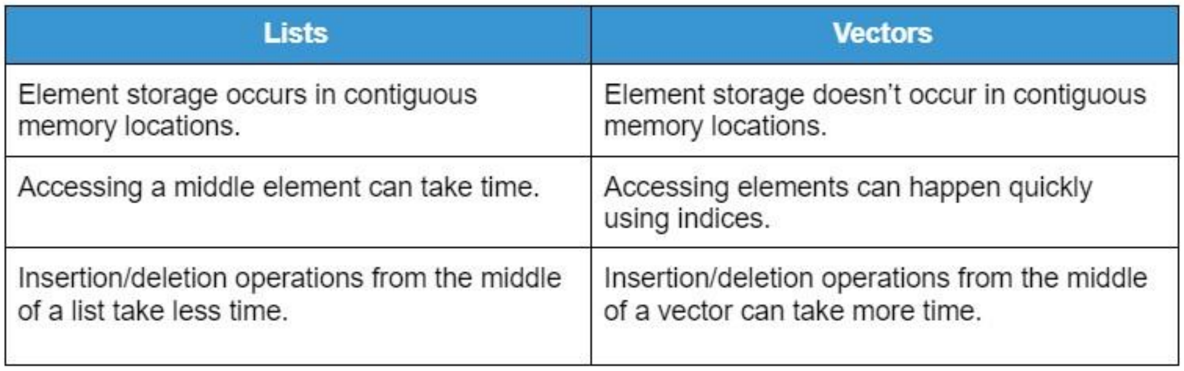
Q. (1) What are some differences between arrays and vectors?
Ans:



Array	Vector
Fixed-size once declared.	Size handling occurs automatically; that is, the size changes dynamically on operations like insertion/deletion.
It is memory efficient.	May use more storage than arrays for the same data.
Available in C and C++.	Only available in C++.

Q. (2) How is storage handled dynamically in a vector?
Ans: Vectors internally use a dynamically allocated array for storage. Since allocating a larger array for each insertion is expensive, vectors don't heavily rely on that. That said, they do use this when necessary. Instead, vectors usually allocate some extra space to accommodate the possibility of insertion of more elements.
Hence the vector's actual capacity is often more than what's required, making it less memory efficient than arrays.
When we need to insert a new element and the vector is full, the vector increases its size to twice the current size.

Q. (3) What are some differences between lists and vectors?
Ans:



Lists	Vectors
Element storage occurs in contiguous memory locations.	Element storage doesn't occur in contiguous memory locations.
Accessing a middle element can take time.	Accessing elements can happen quickly using indices.
Insertion/deletion operations from the middle of a list take less time.	Insertion/deletion operations from the middle of a vector can take more time.

<u>Higher Order Thinking Question</u>

Q. You are provided with marks of **N** students in Physics, Chemistry and Maths.

Perform the following 3 operations:

- Sort the students in Ascending order of their Physics marks.
- Once this is done, sort the students having same marks in Physics in the descending order of their Chemistry marks.
- Once this is also done, sort the students having same marks in Physics and Chemistry in the ascending order of their Maths marks.

Example 1:

Input:

N = 10

phy[] = {4 1 10 4 4 4 1 10 1 10}

chem[] = {5 2 9 6 3 10 2 9 14 10}

math[] = {12 3 6 5 2 10 16 32 10 4}

Output:

```
1 14 10
1 2 3
1 2 16
4 10 10
4 6 5
4 5 12
4 3 2
10 10 4
10 9 6
10 9 32
```

Explanation: Firstly, the Physics marks of students are sorted in ascending order. Those having same marks in Physics have their Chemistry marks in descending order. Those having same marks in both Physics and Chemistry have their Math marks in ascending order.

Solution

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

class Solution{
   public:
   void customSort (int phy[], int chem[], int math[], int N)
   {
      // your code here
   }
};

int main ()
{
   int t; cin >> t;
         while (t--)
         {
                  int n; cin >> n;
                  int phy[n];
                  int chem[n];
                  int math[n];
```

```
                for (int i = 0; i < n; ++i)
                        cin >> phy[i] >> chem[i] >> math[i];
     Solution ob;
                ob.customSort (phy, chem, math, n);
                for (int i = 0; i < n; ++i)
                        cout << phy[i] << " " << chem[i] << " " << math[i] << endl;
        }
}
```

Higher Order Thinking Question

Mishka has got n empty boxes. For every i ($1 \le i \le n$), i-th box is a cube with side length a_i.

Mishka can put a box i into another box j if the following conditions are met:

- i-th box is not put into another box;
- j-th box doesn't contain any other boxes;
- box i is smaller than box j ($a_i < a_j$).

Mishka can put boxes into each other an arbitrary number of times. He wants to minimize the number of *visible* boxes. A box is called *visible* iff it is not put into some another box.

Help Mishka to determine the minimum possible number of *visible* boxes!

Input

The first line contains one integer n ($1 \le n \le 5000$) — the number of boxes Mishka has got.

The second line contains n integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^9$), where a_i is the side length of i-th box.

Output

Print the minimum possible number of *visible* boxes.

Examples

input

1 2 3

output

1

input

4

4 2 4 3

output

2

Note

In the first example it is possible to put box 1 into box 2, and 2 into 3.

In the second example Mishka can put box 2 into box 3, and box 4 into box 1.

Topic: exception handling mechanism, throwing mechanism, catching mechanism, rethrowing exception, specifying exception

MCQ

Q. (1)

Descriptive Questions

Q. (1)

Topic: Components of STL, Containers, Algorithms, Iterators, Application of Container classes, Functions objects.

MCQ

Q. (1) Which function is used to increment the iterator by a particular value?

(a) move()

(b) advance()

(c) Both a and b

(d) None of these

Ans: (b)

Q. (2) A container class is a class whose instances are ………..

(a) Functions

(b) Container

(c) Both a and b

(d) None of these

Ans: (b)

Q. (3) How many sets of requirements are needed in designing a container and what are they?

(a) 3, Container interface requirements, allocator interface requirements and iterator requirements.

(b) 2, iterator requirements and allocator interface requirements.

(c) 1, iterator requirements

(d) None of the above are correct.

Ans: (a)

Q. (4) What are the advantages of function objects than the function call?

(a) A function object can contain state and is a type so can be used as a template.

(b) It cannot contain a state.

(c) A function object can contain state and is a type so cannot be used as a template.

(d) None of the above

Ans: (a)

Q. (5) What are instances of a class with member function operator () when it is defined?

(a) Function objects

(b) Member

(c) Both a and b

(d) None of these

Ans: (a)

Q. (6) How many parameters does a operator() in a function should take?

(a) 3

(b) 1

(c) 4

(d) 2

Ans: (d)

Q. (7) How many types of iterators are there in c++ and what are they?

(a) 1, Random iterators

(b) 3, Input iterators, Output iterators and Random iterators

(c) 5, Input iterators, Output iterators, Forward iterators, Bi-directional iterators and random iterators.

(d) 2, Input iterators and Output iterators

Ans: (c)

Descriptive Questions

Q. (1) What are some of the things A Programmer should be aware of when using the stl? (Many of these may not Make Sense Until You haave actually tried to use the Stl.)

Ans:

- Understand member functions size(), max_size() and resize().
- Understand member functions begin(), end(), rbegin() and rend().
- Understand the fact that a reference to a "range" of values in STL generally means those values from (and including) the first value in the range up to (but not including) the last value in the range. Such a range can extend forward or backward, depending on the kinds of iterators being used.
- Even though it may (sometimes) work, it is better not use a loop like for (i = c.begin(); i < c.end(); i++) ... for processing all the elements of a container c; it is preferable instead to use a loop like this: for (i = c.begin(); i != c.end(); i++) ... This second form is often necessary, in fact, since not all container iterators support operator<.
- RIt is safest to assume that insert() applied to a deque or a vector will invalidate any iterators or references to elements of the deque or vector.
- Know that any insert() member function applied to a container may (or may not) invalidate iterators and references already pointing at elements of that container.

Q. (2) How do you use the stl?

Ans: By including the necessary header files to permit access to the parts of the STL that you need, by declaring objects of the appropriate container, iterator and function types, and then using member functions and/or algorithms, as appropriate, to perform whatever tasks your application requires. It is also generally necessary to ensure that whatever objects you plan to put into your container(s) are objects of classes that have a default constructor, a copy constructor, and an overloaded operator=. In addition, if you plan to sort or compare such container objects, the corresponding classes must provide definitions for operator== and operator<. Finally, since it is often the case that different containers can be used in the same problem situation, the user needs to be able to make an appropriate choice for each occasion, and this choice will usually be based on performance characteristics.

Q. (3) What is the Design Philosophy Of The Stl?

Ans: The STL exemplifies generic programming rather than object-oriented programming, and derives its power and flexibility from the use of templates, rather than inheritance and polymorphism. It also avoids new and delete for memory management in favor of allocators for storage allocation and deal location. The STL also provides performance guarantees, i.e., its specification requires that the containers and algorithms be implemented in such a way that a user can be confident of optimal runtime performance independent of the STL implementation being used.

Q. (4) Why Should A C++ Programmer Be Interested In The Stl?

Ans: Because the STL embodies the concept of reusable software components, and provides off-the-shelf solutions to a wide variety of programming problems. It is also extensible, in the sense that any programmer can write new software (containers and algorithms, for example), that "fit in" to the STL and work with the already-existing parts of the STL, provided the programmer follows the appropriate design guidelines.

Q. (5) What Is Sorted Linked List?

Ans: Linked list means node which is connected each other with a line. It means that each node is connected with another one. Each node of the list holds the reference of the next node.

If we talk about the sorted linked list , it is also a list just like another list. but the difference is only that it hold all the nodes in a sequential manner either in ascending order descending order.

Q. (6) Explain overloading.

Ans: Overloading is a concept used to avoid redundant code where the same method name is used multiple times but with a different set of parameters. The actual method that gets called during runtime is resolved at compile time, thus avoiding runtime errors. Overloading provides code clarity, eliminates complexity, and enhances runtime performance.

Q. (7) What the applications of stl?

Ans:

- A common library that allows containers and algorithm are called it STL and stores and manipulate various types of data. So, it protects us from determining these data structures & algorithms from the scratch.
- Due to the STL, presently we don’t need to determine our sort function every time we make a distinct program or determine the same function twice for the various data types, rather we can simply utilize the generic container and algorithms in STL.
- Also, it helps in saving more time, code, and effort while programming, therefore STL is gradually utilized in competitive programming, and moreover, it is reliable and fast.