VIGNAN'S Institute of Information Technology (Autonomous) VR17

Subject Code: 1000171215

I B.Tech II Semester Regular Examinations, May-2018

Object oriented Programming Through C++

(Common to CSE & IT)

Time:3 Hours **Max.Marks:60**

Answer All Questions

All Questions Carry Equal Marks [5*12=60]

1 A i. Outline Evolution of C++. **[6M]**

Introduction to c++ **[2M]**

C++ is a middle-level programming language and object-oriented programming language and is considered to be an extension of C. BjarneStroustrup at AT&T Bell Laboratories; Murray Hill, New Jersey (USA) developed it in the early eighties of twentieth century.

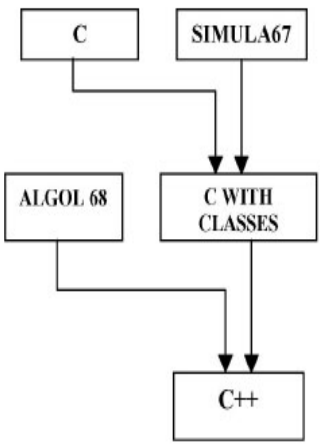


Diagram **[2M]**

Explanation of diagram **[2M]**

Various ideas were derived from SIMULA67 and ALGOL68. Stroustrup called the new language 'C with classes'. However, in 1983, the name was changed to C++. The thought of C++ came from the C increment operator ++. Rick Mascitti coined the term C++ in 1983. Therefore, C++ is an extended version of C. C++ is a superset of C. All the concepts of C are applicable to C++ also.

ii.List the advantages of object oriented programming Language. [6M]

Data Abstraction

Inheritance

Data Encapsulation

Polymorphism

Dynamic Binding

Message Communication

) **Data Abstraction**: - Data Abstraction is that in which A User Can use any of the data and Method from the Class Without knowing about how this is created So in other words we can say that A user can use all the Functions without Knowing about its detail For Example When a User gives Race to Car The Car will be Moved but a User doesn't know how its Engine Will Work.

2**) Inheritance**:- Inheritance is very popular Concept in OOP's This provides the Capability to a user to use the Predefined Code or the code that is not created by the user himself but if he may wants to use that code then he can use that code This is Called Inheritance but Always Remember in Inheritance a user only using the code but he will not be able to change the code that is previously created he can only use that code.

3) **Data Encapsulation** :- Data Encapsulation is also Known as Data Hiding as we know with the inheritance concept of OOP's a user can use any code that is previously created but if a user wants to use that code then it is must thatpreviously code must be Public as the name suggests public means for other peoples but if a code is Private then it will be known as Encapsulate and user will not be able to use that code So With the help of OOP's we can alter or change the code means we can make the Code as Private or public This allows us to make our code either as public or private

4**) Polymorphism** :- Poly Means many and morphism means many function The Concepts Introduces in the form of Many behaviors of an object Like an Operator + is used for Addition of Two Numbers and + is also used for Joining two names The Polymorphism in C++ Introduces in the Form of Functions Overloading and in the Form of Constructor Overloading

5) **Dynamic Binding**:- Binding is used when we call the Code of the Procedure in Binding all the Code that is Linked with the single procedure is Called When a Call is Made to that Procedure Then the Compiler will found the Entire code of the Single Procedure if A Compiler will Fond all the Code of Single Procedure in Compile Time then it is Called as the Early Binding Because Compiler Knows about the code at the time of Compilation but in the Late Binding Compiler will understand all the Code at Run Time or at the Time of the Execution.

6) **Message Communication**: - Message Communication is occurred when an object passes the Call to Method of Class for Execution We Know for executing any method from the class First we have to create the object of class when an object passes References to function of class then In Message Communication First of all we have to Create the Object of the Class the we make Communication between the Object and the Methods of the Class.

For mentioning **any four Advantages** [2M]

For **Explanation of any two** Advantages [2*2=4M]

B i. What are the key concepts of Object oriented Programming? [6M]

Objects

classes

Encapsulation

Data Abstraction

Inheritance

Polymorphism

Delegation

Genercity

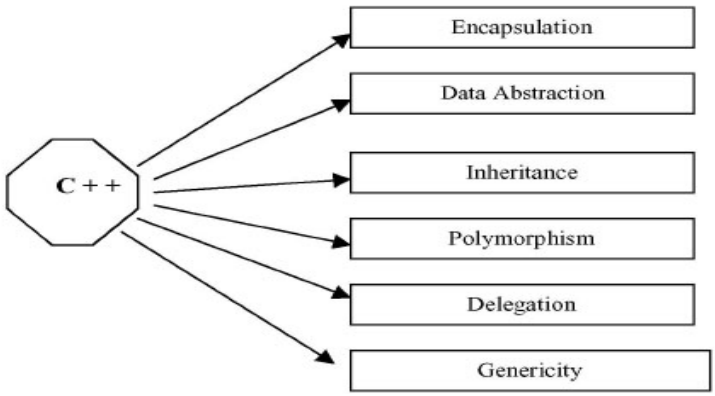


Fig. Features of object-oriented programming

(1) Objects:

Objects are primary run-time entities in an object-oriented programming.

(2) Classes

A class is grouping of objects having identical properties, common behavior, and shared relationship.

A class is the accomplishment of abstract data type. It defines the nature and methods that act on the data structure and abstract data type respectively.

(3) Method

An operation required for an object or entity when coded in a class is called a method.

(4) Data Abstraction

Abstraction directs to the procedure of representing essential features without including the background details.

(5) Encapsulation

The packing of data and functions into a single component is known as encapsulation.

C++ supports the features of encapsulation using classes. The packing of data and functions into a single component is known as encapsulation.

(6) Inheritance

Inheritance is the method by which objects of one class get the properties of objects of another class. Inheritance is the method by which objects of one class get the properties of objects of another class. In object-oriented programming, inheritance provides the thought of reusability.

(7) Polymorphism

Polymorphism allows the same function to act differently in different classes.

(8) Dynamic Binding

Binding means connecting one program to another program that is to be executed in reply to the call.

(9) Message Passing

Object-oriented programming includes objects which communicate with each other. Data is transferred from one object to another.

(10) Reusability

Object-oriented technology allows reusability of the classes by extending them to other classes using inheritance. Object-oriented technology allows reusability of classes by extending them to other classes using inheritance. Once a class is defined, the other programmer can also use it in their programs and add new features to the derived classes**.**

(11) Delegation

In OOP, two classes can be joined either by - inheritance or delegation, which provide reusability of the class.

(12) Genericity

The software components of a program have more than one version depending on the data types of arguments. This feature allows declaration of variables without specifying exact data type. The compiler identifies the data type at run-time. The programmer can create a function that can be used for any type of data. The template feature in C++ allows generic programming.

For Mentioning Key Concepts [2M]

For Explanation of any four concepts [4*1=4M]

ii. Distinguish between C and C++. [6M]

S. No.	C	C++
1	C is a structural or procedural programming language.	C++ is an object oriented programming language.
2	Emphasis is on procedure or steps to solve any problem.	Emphasis is on objects rather than procedure.
3	Functions are the fundamental building blocks.	Objects are the fundamental building blocks.
4	In C, the data is not secured.	Data is hidden and can't be accessed by external functions.
5	C follows top down approach.	C++ follows bottom up approach
6	C uses scanf() and printf() function for standard input and output.	C++ uses cin>> and cout<< for standard input and output.
7	Variables must be defined at the beginning in the function.	Variables can be defined anywhere in the function.
8	In C, namespace feature is absent.	In C++, namespace feature is present.
9	C is a middle level language.	C++ is a high level language.

On Mentioning any Six points [1*6=6M]

2 A i. What is Constructor? List out its Characteristics. [6M]

Constructor Definition [2M]

Constructor is a special member function its name is same as the class name.

- The constructor is invoked automatically whenever an object of its associated class is created.
- It is called constructor because it constructs the values of the data member of the class

Example:

```
float height;   // variable declaration
height=5.5;     // assigning value to variable
```

Characteristics of Constructors:

1. They should be declared in the public section.
2. They are invoked automatically when the objects are created.
3. They do not have return (data type) type not even void and there for they cannot return any values.
4. They cannot be inherited; the derived class can call the base class constructor.
5. They make implicit calls to the operator new and delete when memory allocation is required.
6. Constructor is of following types
   Default constructor

   Copy constructor

   Parameterised constructor

Any Four Characteristics [4M]

ii. What is friend Function? List its properties and advantages. [6M]

Definition of Friend Function [2M]

In object-oriented programming, a **friend function**, that is a "friend" of a given class, is a function that is given the same access as methods to private and protected data.

Syntax of Friend Function in C++ [1M]

```
Class classname
{
Friend returntypefunction_name;
}
```

Properties of Friend Function:

- Keyword “Friend Should is used”
- Cannot be invoked through object
- Friend Function is a normal Function
- It can be defined anywhere in the program
- It can be declared with any access specifier.

Any 2 Properties carries [1*2=2M]

Advantages: [1M]

To access the private,protected, data memebrs of a class friend function is used.

Example Program **[1M]**

B i. What is Nested Class? What are the differences between class and Nested Class? [6M]

Nested Class Definition [2M]

When a class is defined in another class, it is known as nesting of classes. In nested class the scope of inner class is restricted by outer class.

Explanation of Differences between Nested class and Normal Class with example Program [4M]

A program to display some message using nested class**.**

```
#include<iostream.h>                    //for including header files -1M
#include<conio.h>
class one
{
public:
class two  //nested class
{
public:
void display()
{
cout<< "Wonderful language C++\n";
}
};
};
int main()
{
clrscr();
one::two x;
x.display();
return 0;
}
```

For nested classes declaration -**1M**

Main function -**1M**

OUTPUT

Wonderful language C++

Explanation: In the above example, one class is nested in another; that is class two is nested in class one. By using scope resolution operator, the inner class member function is accessed and "Wonderful language C++" is displayed in the above program. – **[1M]**

ii.Illustrate the concept of Constructor overloading with example Program.[6M]

Constructor Overloading definition [1M]

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading and is quite similar to function overloading.

Example Program [4m]

```
// Constructor overloading
#include <iostream>
using namespace std;

class construct
{

public:
float area;

   // Constructor with no parameters
construct()
   {
area = 0;
   }

   // Constructor with two parameters
construct(int a, int b)
   {
area = a * b;
   }

voiddisp()
   {
cout<< area<<endl;
   }
};

int main()
{
   // Constructor Overloading
   // with two different constructors
   // of class name
construct o;
construct o2( 10, 20);
```

Including header files -**1M**

Declaration of default constructor &parameterized constructor-**1M**

Declaration of main function and passing arguments to main-**2M**

```
o.disp();
o2.disp();
return 1;
}
```
Output:

0
200

Explanation of Program [1M]

3 A i. Explain Operator Overloading with suitable Program. [8M]

Operator Overloading Definition [1M]

An operator is a symbol that indicates an operation. It is used to perform operation with constants and variables. Without an operator, programmer cannot built an expression.

Rules for operator overloading

1) Only built-in operators can be overloaded. New operators can not be created.
   2) Arity of the operators cannot be changed.
   3) Precedence and associativity of the operators cannot be changed.
   4) Overloaded operators cannot have default arguments except the function call operator () which can have default arguments.
   5) Operators cannot be overloaded for built in types only. At least one operand must be used defined type.
   6) Assignment (=), subscript ([]), function call ("()"), and member selection (->) operators must be defined as member functions
   7) Except the operators specified in point 6, all other operators can be either member functions or a non member functions.
   8) Some operators like (assignment)=, (address)& and comma (,) are by default overloaded.

Any four rules [2M]

Syntax[1M]

Syntax:
```
Return type operator operator symbol (parameters)
{
  Statement1;
  statement2;
}
```

Example program-[4M](Note: any operator overloading can be explained)


```
class Test
{
   private:
      int count;

   public:
       Test(): count(5){}

       void operator ++()
       {
          count = count+1;
       }
       void Display() { cout<<"Count: "<<count; }
};

int main()
{
    Test t;
    // this calls "function void operator ++()" function
    ++t;
    t.Display();
    return 0;
}
```

For logic -2M
For main function-1M

ii.What is inheritance? List different types of inheritance available. [4M]

Inheritance Definition [1M]

Acquiring the properties of base class by derived class is called inheritance

Types of Inheritance :**(for mentioning types of inheritance -1M)**

Single Inheritance

Multiple Inheritances

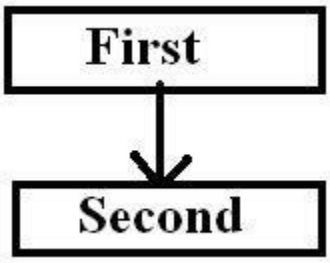
Hierarchical Inheritance

Hybrid Inheritance

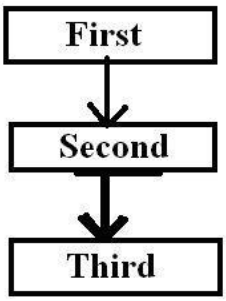
Multilevel Inheritance

Any Two Inheritance Explanation with its diagram [2M]

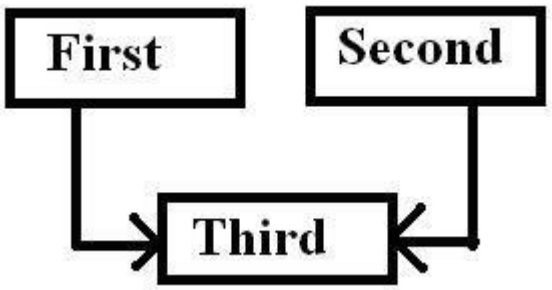
1) Single Inheritance there is only one Super Class and Only one Sub Class Means they have one to one Communication between them



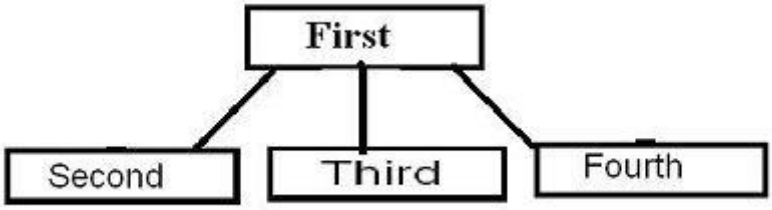
2) Multilevel Inheritance a Derived class can also inherited by another class Means in this Whena Derived Class again will be inherited by another Class then it creates a Multiple Levels.



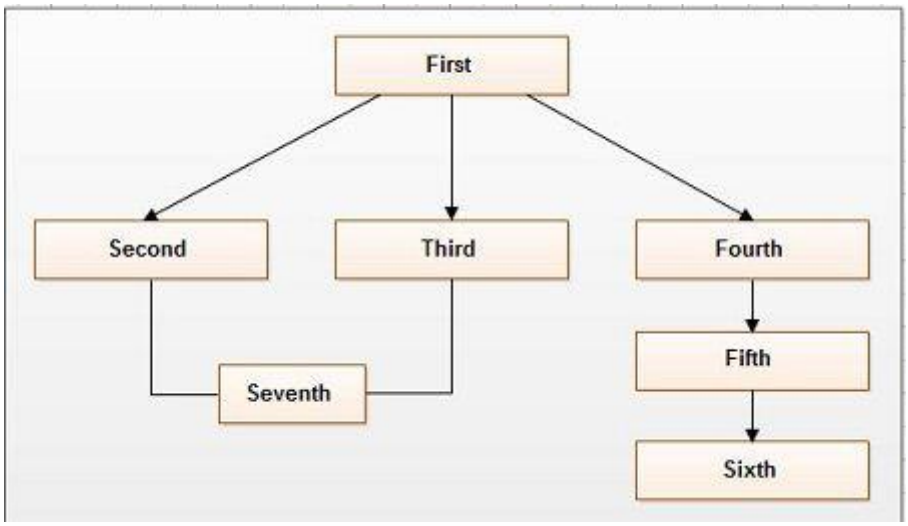
3) Multiple Inheritances is that in which a Class inherits the features from two Base Classes When a Derived Class takes Features from two Base Classes.



4) Hierarchical Inheritance is that in which a Base Class has Many Sub Classes or When a Base Class is used or inherited by many Sub Classes.



5) Hybrid Inheritance: - This is a Mixture of two or More Inheritance and in this Inheritance a Code May Contains two or Three types of inheritance in Single Code.



B i. List the differences between class and Abstract Class? [6M]

Definition of Class [1M]

The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be

accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Describing Abstract class [1M]

Abstract class is used in situation, when we have partial set of implementation of methods in a class. For example, consider a class have four methods. Out of four methods, we have an implementation of two methods and we need derived class to implement other two methods. In these kind of situations, we should use abstract class.

A virtual function will become pure virtual function when you append "=0" at the end of declaration of virtual function.

A class with at least one pure virtual function or abstract function is called abstract class.

Pure virtual function is also known as abstract function.

Explanation of Class and Abstract Class with example program [3M]

Example:

```
#include<iostream.h>
    #include<conio.h>

classBaseClass      //Abstract class
    {

public:
virtual void Display1()=0;    //Pure virtual function or abstract function
virtual void Display2()=0;    //Pure virtual function or abstract function

void Display3()
        {
cout<<"\n\tThis is Display3() method of Base Class";
        }

    };

classDerivedClass : public BaseClass
```

```
    {

public:
void Display1()
        {
cout<<"\n\tThis is Display1() method of Derived Class";
        }

void Display2()
        {
cout<<"\n\tThis is Display2() method of Derived Class";
        }

    };

void main()
    {

DerivedClass D;

D.Display1();       // This will invoke Display1() method of Derived Class
D.Display2();       // This will invoke Display2() method of Derived Class
D.Display3();       // This will invoke Display3() method of Base Class

    }
```

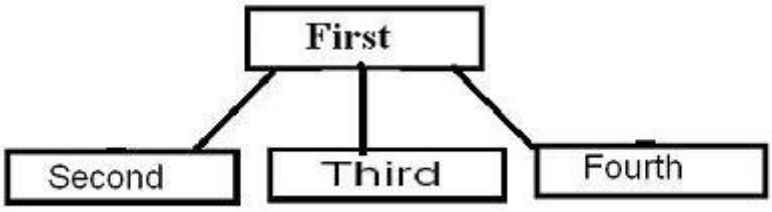
Output :

```
        This is Display1() method of Derived Class
        This is Display2() method of Derived Class
        This is Display3() method of Base Class
```

Ii.Demonstrate the Concept of Hierarchical Inheritance with suitable Program. [6M]

Hierachical Inheritance Diagram [**2m]**

Hierarchical Inheritance is that in which a Base Class has Many Sub Classes or When a Base Class is used or inherited by many Sub Classes.



Example Program **[4M]**

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

Including header files-1M

```
// base class
class Vehicle
{
public:
        Vehicle()
        {
        cout << "This is a Vehicle" << endl;
        }
};
```

```
// first sub class
class Car: public Vehicle
{

};

// second sub class
class Bus: public Vehicle
{

};

// main function
int main()
{
        // creating object of sub class will
        // invoke the constructor of base class
        Car obj1;
        Bus obj2;
        return 0;
}
```

Logic of the program -2M

Main function - 1M

4 A i. Differentiate Early Binding and Late Binding. [6M]

Any 3 Differences [2*3=6M]

	Early binding	Late binding
1	This binding is performed at compile time and also called as compile time binding	this is done at run time so it is called dynamic binding
2	This type of binding is performed in following cases 1.function overloading 2.operator overloading 3.constructor overloading	This type of binding is performed on virtual functions
3	If p is a pointer invoking the function then the decision about which function to call is taken based on the data type of p and is immaterial of data of target type of object	If p is a pointer invoking the function, the decision about which function to call is taken based on the type of target object to which p points to
4	Used to implement compile time polymorphism	Used to implement run time polymorphism

ii. Differentiate Virtual and Pure Virtual Functions. [6M]

Pure virtual function	virtual function
Pure Virtual Function is declared as Ex : virtual return_typefunction_name(function arguments) = 0;	Virtual Function is declared with keyword '**virtual**' at the declaration. Ex : virtual return_typefunction_name(function arguments);

Pure-virtual functions **need not** be implemented in the base class, but they **must** be implemented in the derived classes.
 	
Example:-
class Base {
// ...
virtual void f() = 0;
// ...
	Virtual functions **must** be implemented in the base class, but they **need not** be implemented in their derived classes. Derived classes will automatically inherit the base class implementations of all virtual functions, but can override those functions by providing their own implementations.
	
	Example:-
	Derived d;
	Base&rb = d;
	// if Base::f() is virtual and Derived overrides it,
	Derived::f() will be called
	rb.f();

pure virtual function is a kind of virtual functions with a specific syntax:
class B {
public:
virtual void f() =0; // =0 means pure virtual
};
If a class has at least one pure virtual function, it will be abstract class, so instance creation is impossible.B::f() says that you should
	virtual function is a primary tool for polymorphic bevaviour.

implement f() in derived classes:
class D : public B {
void f() { /* ... */ }
};
If you do not implement f() in D, the D is abstract class by default, because it inherits all the pure virtual functions of class B.	

Any 3 Differences.Each Carries 1 Mark[3*2=6M]

B i. Demonstrate Dynamic Polymorphism using program. [6M]

Dynamic Polymorphism is achieved through Virtual Function

Usage of Virtual Keyword **[2M]**

```
#include<iostream.h>
 #include<conio.h>
class first
{
int b;
public:
first()
{
b=10;
}
virtual void display()
 {
cout<<"\n b=" <<b;
}
};
class second : public first
{
int d;
public:
second()
{
d=20;
}
void display()
{
cout<<"\n d="<<d;
}
};
int main()
 {
```

```
clrscr();
first f,*p;
second s;
p=&f;
p->display();
p=&s;
p->display();
return 0;
 }
```

OUTPUT
b = 10
d = 20

Explanation of Concept through program **[4M]**

ii.Explain Pointer to Base Class and Derived Class. [6M]

Pointer Definition **[2M]**

Sample Program using Inheritance **[2M]**

Explanation and Creation of Base Pointer **[2M]**

```
#include<iostream.h>
#include<conio.h>
class A
{
 public :
 int b;
void display()
{
 cout<<“b = ” <<b <<“\n”;
  }
};
class B : public A
{
public :
int d;
void display()
{
cout<<“b= ” <<b <<“\n” <<“ d=”<<d <<“\n”;
}
};
 main()
{
clrscr();
```

```
A *cp;
 A base;
cp=&base;
cp->b=100;
// cp->d=200; Not Accessible
 cout<<"\n cp points to the base object \n";
 cp->display();
B b;
cout<<"\n cp points to the derived class \n";
 cp=&b;
cp->b=150;
// cp->d=300 // Not accessible
cp->display();
return 0;
}
```

OUTPUT

cp points to the base object

b = 100

cp points to the derived class

b = 150

5 A i. What is Class Template? Explain. [4M]

Definition of Template **[1M]**

Template is a technique that allows using a single function or class to work with different data types. Using template we can create a single function that can process any type of data i.e., the formal arguments of template functions are of template (generic) type.

Syntax of Class Template **[1M]**

To declare a class of template type, following **syntax** is used.

```
template class <T>
   classname_of_class
   {
     // class data member and function
   }
```

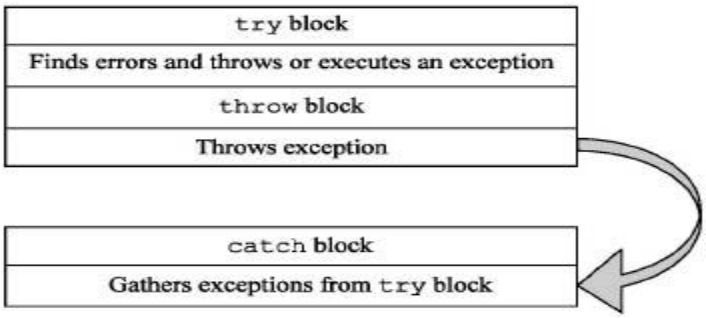
Explanation of Template concept [**2M]**

ii. Explain in detail about Exception Handling. [8M]

Definition of Exception Handling **[2M]**

An exception is an abnormal termination of a program, which is executed in a program at run-time or it may be called at runtime when an error occurs. The exception contains warning messages like invalid argument, insufficient memory, division by zero, and so on.

Usage of Keywords try, catch and throw **[3M]**



THE KEYWORDS try, throw AND catch

Exception handling technique passes control of program from a location of exception in a program to an exception handler routine linked with the try block. An exception handler routine can only be called by throw statement.

1.try:

The try keyword is followed by a series of statements enclosed in curly braces**.**

Syntax of try statement

```
try
{
statement 1;
statement 2;
}
```

2.throw:

The function of throw statement is to send the exception found. The declaration of throw statement is as given below:

Syntax of throw statement

```
throw (excep);
throwexcep;
throw // re-throwing of an exception
```

3.catch:

catch block also contains a series of statements enclosed in curly braces. It also contains an argument of exception type in parenthesis.

Syntax of catch statement:

```
try
{
 Statement 1;
 Statement 2;
}
catch ( argument)
{
statement 3; // Action to be taken
 }
```

Explanation with example program **[3M]**

```
#include <iostream>

using namespace std;


double division(int a, int b) {

   if( b == 0 ) {

      throw "Division by zero condition!";

   }

   return (a/b);

}


int main () {

   int x = 50;

   int y = 0;

   double z = 0;


   try {

      z = division(x, y);

      cout << z << endl;

   } catch (const char* msg) {

     cerr << msg << endl;

   }


   return 0;

}
```

B i. What is Function Template? Explain. [4M]

Function Template Definition [1M]

Function template that works with all data types. After compilation, the compiler cannot guess with which type of data the template function will work. When the template function is called at that moment, from the type of argument passed to the template function, the compiler identifies the data type. Every argument of template type is then replaced with the identified data type and this process is called as instantiating**.**

Syntax [2M]

Syntax:

```
template<class T>
return_data_typefunction_name (parameter of template type)
   {
     statement1;
     statement2;
     statement3;
  }
```

Any Sample Program Explanation [2M]

```
# include <iostream.h>
# include <conio.h>
template<class E>
void exchange(E &a, E &b)
{

  E t=a;
  a=b;
  b=t;
}

int main( )
{
clrscr( );
int x=5,y=8;
cout<<"\n Before exchange "<<"x= "<<x <<" y= "<<y;
exchange (x,y);
cout<<"\n After exchange "<<"x= "<<x <<" y= "<<y;
return 0;
}
```

ii. When do we need multiple Catch blocks for a single try block? Give an Example. [8M]

Definition of Exception Handling [2M]

An exception is an abnormal termination of a program, which is executed in a program at run-time or it may be called at runtime when an error occurs. The exception contains warning messages like invalid argument, insufficient memory, division by zero, and so on.

Explanation about multiple catch statements [3M]

MULTIPLE CATCH STATEMENTS:

We can also define multiple catch blocks, in try blocks. Such program also contain multiple throw statements based on certain conditions. The format of multiple catch statement is given below:

Syntax:

```
try
{
   // try section
}
catch (object1)
{
   // catch section1
}
catch (object2)
   // catch section2
}
.......
.......
catch (type n object)
{
   // catch section-n
}
```

CATCHING MULTIPLE EXCEPTIONS

It is also possible to define single or default catch( ) block from one or more exceptions of different types. In such a situation, a single catch block is used for catch exceptions thrown by multiple throw statements.

```
catch( )
{
  // statements for handling
  // all exceptions
}
```

Example program -3M

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

```
voidnum (int k)
{
try
  {
if (k==0) throw k;
else
if (k>0) throw ‘P’;
else
if (k<0) throw .0;
cout<<"*** try block ***\n";
  }
catch (...)
{
cout<<"\n Caught an exception\n";
}
}
int main( )
{
num(0);
num(5);
num(-1);
return 0;
}
```

******* THE END*******