2071 Shrawan

**1)What is object oriented programming? What are the drawbacks of procedure oriented programming? List down the features of C++.Write a program with a class to represent distance with feet and inches members. The class should have member function to read and display the data members and member function to add and subtract two distances**.

OOP is a method of implementation in which programs are organized as co-operative collection of objects each of which represents an instance of same class.

The drawbacks of POP are:

-It focuses on function rather than data.

-In a large program it is difficult to identify belongings of global data.

-It is difficult to hide information’s from unauthorized user.

-The use of global data is error prone ie.it may be an obstacle for code maintenance and enhancements.

The features of C++ are:

-Function overloading

-Constructor and destructor

-Inline function

-Polymorphism

-Template

-Friend function and classes

#include<iostream>

**using** **namespace** std;

**class** **dis**

{

**int** feet,inches;

**public:**

dis()

{

feet=**0**;

inches=**0**;

}

dis(**int** ft,**int** in)

{

feet=ft;

inches=in;

}

**void** add(dis d1,dis d2)

{

feet=d1.feet+d2.feet;

inches=d1.inches+d2.inches;

**if** (inches>=**12**)

{

inches=inches-**12**;

feet=feet+**1**;

}

}

**void** sub(dis d1,dis d2)

{

**if**(d1.feet>d2.feet)

{

feet=d1.feet-d2.feet;

inches=d1.inches-d2.inches;

}

**else**

{

feet=d2.feet-d1.feet;

inches=d2.inches-d1.inches;

}

**if** (inches<**0**)

{

inches=inches+**12**;

feet=feet-**1**;

}

}

**void** display()

{

cout<<"distance is"<<feet<<":"<<inches<<endl;

}

};

**int** **main**()

{

dis d1(**5**,**9**),d2(**4**,**11**),d3;

d3.add(d1,d2);

d3.display();

d3.sub(d1,d2);

d3.display();

**return** **0**;

}

**2)What do you mean by namespace and what is its use? Explain about returning a variable from a function by reference with an example. Explain about function overloading with an example.**

The namespace mechanism is used for the logical grouping of variables, classes and functions in c++.It is a container for variables, functions, classes and other identifiers that avoids conflicts residing in different scopes.

In this type of function call address of variable or argument is passed to the function as argument instead of actual value of the variable. So the variables passed as argument during the function call are changed by the called function.

Eg: #include<iostream>

**using** **namespace** std;

**int** &max(**int** &x,**int** &y)

{

**if**(x>y)

{

**return** x;

}

**else**

{

**return** y;

}

}

**int** main()

{

**int** a,b;

cout<<"enter two temperatures"<<endl;

cin>>a>>b;

max(a,b)=**100**;

cout<<a<<endl;

cout<<b<<endl;

}

Some functions conceptually perform the same task on objects of different types and numbers. When the same name is used for different operation it is called function overloading. When an overloaded function is called the function with matching argument is envoked.

Eg: #include<iostream>

**using** **namespace** std;

**int** **vol**(**int** l)

{

**return**(l\*l\*l);

}

**float** **vol**(**int** r,**int** h)

{

**return**(**3.1415**\*r\*h);

}

**int** **vol**(**int** a,**int** b,**int** c)

{

**return**(a\*b\*c);

}

**int** **main**()

{

**int** l,a,b,c;

**float** r,h;

cout<<"enter length"<<endl;

cin>>l;

cout<<"volume of cube"<<vol(l);

cout<<"enter radius and height"<<endl;

cin>>r>>h;

cout<<"volume of cylinder"<<(vol(r,h));

cout<<"enter length ,breadth and height"<<endl;

cin>>a>>b>>c;

cout<<"volume of rectangle"<<vol(a,b,c);

**return** **0**;

}

**3)How do you dynamically allocate objects and object array in C++? Explain about constant member function and constant object with an example. Write a meaningful program to illustrate the use of copy constructor and destructor.**

Object and object array in c++ can be sllocated dynamically using following syntax:

class\_name \*object\_ptr;

object\_ptr=new class\_name (allocates memory for single object)

object\_ptr=new class\_name[size] (allocates memory for array of object)

The type of function that guarantees not to change objects value or data members value are called constant function. It can be declared inside or outside its class. They are useful for constant member objects as they guarantee not to change objects value. Constant objects can call only constant functions. They are declared by placing const after parameter list and before function body.

Eg:

int second ( )const

{

return s;

}

#include<iostream>

**using** **namespace** std;

**class** **Add**

{

**int** x,y,z;

**public:**

Add()

{

}

Add(**int** a,**int** b)

{

x=a;

y=b;

}

Add(Add &p)

{

x=p.x;

y=p.y;

}

**void** calculate()

{

z=x+y;

}

**void** display()

{

cout<<z<<endl;

}

~Add()

{

cout<<"destructor is called"<<endl;

}

};

**int** **main**()

{

Add a(**5**,**6**);

Add b(a);

b.calculate();

b.display();

}

**4)List the operators that cannot be overloaded in C++.Explain about explicit constructor with an example. Write a program having a class to represent money. The class should have two integer members to represent rupees and paisa. Overload + and – operators for adding and subtracting the objects. Then overload >, <, ==and != operators for comparing the objects.**

The operators that cannot be overloaded in C++ are:

-Scope resolution operator (: :)

- Membership operator (.)

- Size of operator (size of)

- Conditional operator(? :)

- Pointer to member operator(.\*)

When a constructor is prefixed with a keyword explicit it is known as explicit constructor. The main use of explicit constructor .The main use of explicit constructor is to prevent compiler from implicit type conversion when constructor is used as a conversion routine .

Eg:

#include<iostream>

**using** **namespace** std;

**class** **dis**

{

**private:**

**float** feet;

**float** inch;

**public:**

dis()

{

feet=**0.00**;

inch=**0.00**;

}

**explicit** dis(**float** m)

{

**float** feetf=**3.2808**\*m;

feet=**int**(feetf);

inch=**12**\*(feetf-feet);

}

**void** showdata()

{

cout<<"the dis is"<<endl;

cout<<feet<<"feet"<<inch<<"inch";

}

};

**int** **main**()

{

dis d;

**float** metres;

cout<<"enter dis in metres";

cin>>metres;

d=(dis)metres;

d.showdata();

**return** **0**;

}

-Program

#include<iostream>

**using** **namespace** std;

**class** **money**

{

**private:**

**int** r,p;

**public:**

money()

{

r=**0**;

p=**0**;

}

money (**int** rs ,**int** pa)

{

r=rs;

p=pa;

}

**friend** money **operator** +(money,money);

**friend** money **operator** -(money,money);

**friend** **int** **operator** >(money,money);

**friend** **int** **operator** <(money,money);

**friend** **int** **operator** ==(money,money);

**friend** **int** **operator** !=(money,money);

**void** **display**()

{

cout<<r<<"rupees"<<p<<"paisa"<<endl;

}

};

money **operator** +(money m1,money m2)

{

money temp;

temp.r=m1.r+m2.r;

temp.p=m1.p+m2.p;

**if**(temp.p>=**100**)

{

temp.p=temp.p-**100**;

temp.r=temp.r+**1**;

}

**return** temp;

}

money **operator** -(money m1,money m2)

{

money temp;

**if**(m1.r>m2.r)

{

temp.r=m1.r-m2.r;

temp.p=m1.p-m2.p;

}

**else**

{

temp.r=m2.r-m1.r;

temp.p=m2.p-m1.p;

}

**if**(temp.p<**0**)

{

temp.p=temp.p+**100**;

temp.r=temp.r-**1**;

}

**return** temp;

}

**int** **operator** >(money m1,money m2)

{

**if**(m1.r>m2.r)

{

**return** **1**;

}

**else**

{

**return** **0**;

}

}

**int** **operator** <(money m1,money m2)

{

**if**(m1.r<m2.r)

{

**return** **1**;

}

**else**

{

**return** **0**;

}

}

**int** **operator** ==(money m1,money m2)

{

**if**(m1.r==m2.r)

{

**return** **1**;

}

**else**

{

**return** **0**;

}

}

**int** **operator** !=(money m1,money m2)

{

**if**(m1.r!=m2.r)

{

**return** **1**;

}

**else**

{

**return** **0**;

}

}

**int** main()

{

money m1(**25**,**80**),m2(**30**,**45**),m3,m4;

m3=m1+m2;

m3.display();

m4=m1-m2;

m4.display();

**if**(m1>m2)

cout<<"m1 is greater"<<endl;

**else**

cout<<"m2 is greater"<<endl;

**if**(m1<m2)

cout<<"m1 is smaller"<<endl;

**else**

cout<<"m2 is smaller"<<endl;

**if**(m1==m2)

cout<<"m1 and m2 are equal"<<endl;

**else**

cout<<"m1 and m2 are not equal"<<endl;

**if**(m1!=m2)

cout<<"m1 and m2 are not equal"<<endl;

**else**

cout<<"m1 and m2 are equal"<<endl;

}

**5)What do you understand by protected access specifier? Explain about different forms of inheritance. Define a class named course. Derive three classes from this class named: Mathematics, Science and Engineering. Then, derive two classes from Science named: Physics and Chemistry. Define data members and member functions as appropriate. Illustrate the concept of member function overriding and accessing overridden member from the derived class in your program.**

According to protected access specifier

-The private members can’t be inherited and not be available for derived class immediately.

-Public members can be easily accessed from anywhere.

-Protected members are accessible from the member function of same class and from the member function of derived class.

Inheritance are classified into following types:

1. Single inheritance

If a class is derived from only one base class, then that is called single inheritance.

2. Multiple inheritance

If a class is derived from more than one base class then inheritance is called as multiple inheritance. Multiple inheritance allows us to combine the features of several existing classes as starting point for defining new class.

3. Multilevel inheritance

The mechanism of deriving a class from another derived class is called multilevel.

4. Hierarchical inheritance

When from one base class more than one classes are derived that is called hierarchical inheritance.

5. Hybrid inheritance

If we apply more than one type of inheritance to design a problem, then that is known as hybrid inheritance.

#include<iostream>

**using** **namespace** std;

**class** **course**

{

**protected:**

**char** a[**50**];

**int** roll;

**public:**

**void** **getdata**()

{

cout<<"enter name and roll no"<<endl;

cin>>a>>roll;

}

**void** **display**()

{

cout<<"Name"<<a<<endl;

cout<<"Roll no"<<roll<<endl;

}

};

**class** **math**:**public** course

{

**private:**

**int** marks;

**public:**

**void** **getdata**()

{

cout<<"enter marks in maths"<<endl;

cin>>marks;

}

**void** **display**()

{

cout<<"marks:"<<marks;

}

};

**class** **engineering**:**public** course

{

**private:**

**int** marks;

**public:**

**void** **getdata**()

{

cout<<"enter marks in engineering"<<endl;

cin>>marks;

}

**void** **display**()

{

cout<<"marks:"<<marks;

}

};

**class** **science**:**public** course

{

**private:**

**int** marks;

**public:**

**void** **getdata**()

{

cout<<"enter marks in science"<<endl;

cin>>marks;

}

**void** **display**()

{

cout<<"marks:"<<marks;

}

};

**class** **physics**:**public** science

{

**private:**

**int** marks;

**public:**

**void** **getdata**()

{

cout<<"enter marks in physics"<<endl;

cin>>marks;

}

**void** **display**()

{

cout<<"marks in physics:"<<marks;

}

};

**class** **chemistry**:**public** science

{

**private:**

**int** marks;

**public:**

**void** **getdata**()

{

cout<<"enter marks in chemistry"<<endl;

cin>>marks;

}

**void** **display**()

{

cout<<"marks in chemistry:"<<marks;

}

};

**int** **main**()

{

math a;engineering b;science c;physics d; chemistry e;

a.course::getdata();

a.getdata();

b.getdata();

c.getdata();

d.getdata();

e.getdata();

a.course::display();

a.display();

b.display();

c.display();

d.display();

e.display();

}

**6)List any four formatting flags of iOS class with their usage. Explain with an example how a non-parameterized user-defined manipulator can be defined. Write a program for managing a simple library database. The information to be stored in the database are book id, book name, book name, borrower’s id, borrower’s name, issue date and due date. Your program should have features to add a record, display all the records and display a set of records corresponding to a particular borrower’s id or a particular borrower’s name.**

The four formatting flags of IOS class are:

Width( ) - To specify the required field size for displaying an output value.

Precision( ) -To specify the no. of digits to be displayed after a decimal point of a float value

fill( ) -To specify a character that is used to fill the unused portion of a field

setf( ) -To specify format flags that can control the form of output display (such as leftjustification and right-justification)

A non-parameterized user-defined manipulator can be defined in following way:

Syntax:

Ostream & manipulator\_name(ostream &os)

{

//body

return output;

}

7)What are pure virtual function and abstract class? How is dynamic cast used? Write a meaningful program to illustrate overloading of a function template with both a normal function and a function template.

A pure virtual function is a virtual function with no function body.

Syntax:

class test

{

Virtual return-type function\_name( )=0;

};

Object can’t be created from base class containing pure virtual function. Such type of class is known as abstract class.

The dynamic cast helps to change the object of one type to object of another type during run-time and also checks validity of the cast.

Syntax:

dynamic\_cast <target\_type>expression;

-Using normal function

#include<iostream>

#include<cstring>

**using** **namespace** std;

**template**<**class** **T**>

T find\_max(T a,T b)

{

T result;

**if**(a>b)

result=a;

**else**

result=b;

**return** result;

}

**char** \*find\_max(**char**\*a,**char**\*b)

{

**char** \*result;

**if** (strcmp(a,b)>**0**)

result=a;

**else**

result=b;

**return** result;

}

**int** main()

{

**int** i1=**15**,i2=**20**;

cout<<find\_max(i1,i2)<<endl;

**char** str1[]="apple",str2[]="chocolate";

cout<<"greater is:"<<find\_max(str1,str2)<<endl;

**return** **0**;

}

-using function template

#include<iostream>

**using** **namespace** std;

**class** **complex**

{

**private:**

**float** real,imag;

**public:**

complex()

{

real=**0**;

imag=**0**;

}

complex(**float** re,**float** im)

{

real=re;

imag=im;

}

**bool** **operator**>(complex cc);

**void** **display**()

{

cout<<"real:"<<real<<"imag:"<<imag<<endl;

}

};

**bool** complex::**operator**>(complex cc)

{

**float** mag1,mag2;

mag1=real\*real+imag\*imag;

mag2=cc.real\*cc.real+cc.imag\*cc.imag;

**if**(mag1>mag2)

**return** true;

**else**

**return** false;

}

**template**<**class** **T**>

T find\_max(T a,T b)

{

**if**(a>b)

**return** a;

**else**

**return** b;

}

**int** main()

{

**int** n1=**7**,n2=**19**;

cout<<find\_max(n1,n2)<<endl;

complex **c1**(**2.9**,**7.3**),c2(**8.5**,**5.7**);

complex c3=find\_max(c1,c2);

c3.display();

**return** **0**;

}

8)What are class templates? What do you understand by rethrowing an exception and catching all the exceptions? Define a class to represent time. It should have a member function to read time from the user and a member function to display the time. The function to read time must raise an exception if the user enters invalid values for hours, minutes or seconds. The exceptions should be handled outside the member function of the class.

Class template is a class that operates on any type of data.

Syntax:

Template<class template\_type,….>

Class class\_name

{

private:

//data member of template type or non template type.

//…..

public:

//function members with template type argument and return type

};

A handler may decide to rethrow the exception caught without processing it. In such situations, we may simplify invoke throw without any arguments as

throw;

This causes the current exception to be thrown to the next enclosing try/catch sequence and is caught by a catch statement listed after that enclosing try block.

Catching Mechanism: Code for handling exceptions is included in catch blocks. A catch block looks like a function definition and is of the form:

catch(type arg)

{

// statements for

// managing exceptions

}

#include<iostream>

**using** **namespace** std;

**class** **time**

{

**private:**

**int** hour,min,sec;

**public:**

**void** **read**()

{

cout<<"enter hour,min and sec"<<endl;

cin>>hour>>min>>sec;

try{

**if**(sec>**60**)

**throw** sec;

**else** **if**(min>**60**)

**throw** min;

**else** **if**(hour>**24**)

**throw** hour;

}

**catch**(**int** i)

{

cout<<"Invalid data "<<i<<endl;

cout<<"Enter again**\n**";

read();

}

}

**void** **display**()

{

cout<<"time"<<endl;

cout<<hour<<":"<<min<<":"<<sec;

}

};

**int** **main**()

{

time t;

t.read();

t.display();

}