**2076 Chaitra**

1. **What are the limitation of POP? Explain features of C++. WAP to multiply 2 complex number using OOP approach. [2+2+4]**

**Answer:**

Limitation of POP are:

* It focus on function rather than data.
* Data security is low.
* Data in POP can be changes unknowingly.
* It is found to have more error while declaring data globally.

Features of OOP are:

Namespace, class, function overloading, operator overloading, inline function, access controller, friend function and class, constructor and destructor, ect.

#include<iostream>

using namespace std;

class complex

{

float real, img;

public:

void getdata();

void display();

complex mul(complex c1);

};

void complex:: getdata()

{

cout<<"enter real part = ";

cin>>real;

cout<<"enter imag part = ";

cin>>img;

}

void complex:: display()

{

cout<<"the sum is "<<real<<"+"<<img<<"j";

}

complex complex:: mul(complex c1)

{

complex ob;

ob.real=(real\*c1.real);

ob.img= (real\*c1.img +img \*c1.real);

return ob;

}

int main()

{

complex c1,c2,c3;

cout<<"enter data for 1st: "<<endl;

c1.getdata();

cout<<"enter data for 2nd: "<<endl;

c2.getdata();

c3=c1.mul(c2);

c3.display();

return 0;

}

1. **Can we have more than one constructor in a class? If yes, explain the need for such situation. WAP designing a class called midpoint to find midpoint between two points by returning object from member function using this pointer. [3+5]**

**Answer:**

Yes, we can have more than one constructor in a class. The main need of multiple constructor is to initialize the data which are send by the programmer. It is used in constructor overloading also. Such as:

#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;

}

void disp()

{

cout<< area<< endl;

}

};

int main()

{

construct o;

construct o2( 10, 20);

o.disp();

o2.disp();

return 1;

}

2nd ans is:

#include<iostream>

using namespace std;

class midpoint

{

private:

float x1,y1,x2,y2,m,n;

public:

void getparam()

{

cout<<"Enter first two points \n";

cin>>this->x1>>this->y1;

cout<<"Enter next two points \n";

cin>>this->x2>>this->y2;

}

void calculate()

{

this->m=(x1+x2)/2;

this->n=(y1+y2)/2;

cout<<"The midpoint between two points is "<<this->m<<","<<this->n<<endl;

}

};

int main()

{

midpoint A;

A.getparam();

A.calculate();

return(0);

}

1. **Why is namespace required? Explain how namespace is created and used in program with suitable example. How is reference variable used for pass by reference? explain[1+4+3]**

**Answer:**

Namespace is used in C++ for the logical grouping of variable, classes and functions.

It is also a container for variable, function, classes and other identifier that avoid conflicts found in different scopes.

Syntax:

Namespace namespace\_name

{

//….

}

#include<iostream>

using namespace std;

namespace num

{

int n=1;

}

int main()

{

int n=3;

cout<<"in main n= "<<n<<endl;

cout<<"n in namepace ="<<num::n;

return 0;

}

#include<iostream>

#include<conio.h>

using namespace std;

void swap(int &a, int &b);

int main()

{

int a,b;

cout<<"Enter value for a \n";

cin>>a;

cout<<"Enter value for b \n";

cin>>b;

cout<<"Before swapping \n";

cout<<"a= "<<a<<endl;

cout<<"b= "<<b<<endl;

swap(a,b);

cout<<"After swapping \n";

cout<<"a= "<<a<<endl;

cout<<"b= "<<b<<endl;

getch();

return(0);

}

void swap(int &a,int &b)

{

int temp;

temp=a;

a=b;

b=temp;

}

1. **Explain how the use of default argument supports the function overloading with suitable example. Define inline function with its merits and demerits. [4+4]**

**Answer**:

We can provide default values for function parameters. If a function with default arguments is called without passing arguments, then the default parameters are used. And if arguments are passed while calling the function, the default arguments are ignored.

Example:

#include<iostream>

#include<conio.h>

using namespace std;

void TM(int m1=40,int m2=40,int m3=40);

int main()

{

TM();

TM(55);

TM(66,77);

TM(66,77,88);

getch();

return(0);

}

void TM(int m1,int m2,int m3)

{

int total;

total=m1+m2+m3;

cout<<"Total marks is "<<total<<endl;

}

Advantages of inline functions:

* It increase the performance of program.
* It doesn’t required function calling overhead.
* It saves overhead of function of result from called function to calling function.
* No jumping is need to the different function.

Disadvantages of inline function:

Inline function make the program to take more memory because the statement that define the inline function are replaced at each point where the function us called.

If we use too many inline function then size of binary execution file will be large because of same duplicate codes.

1. **Define operator overloading. What are the rules of operator overloading? How do you overload unary operator? Explain with example [1+2+5]**

**Answer:**

Operator overloading is the feature in c++ in which the operator is overloaded to provide the special meaning to the user-defined data type.

Rules of operator overloading are:

* Only existing operator can be overloaded
* New operator cannot be created
* It should follow the syntax rule of the original operator , they cannot be overridden
* Basic meaning of the operator cannot be changed
* Friend function cannot overload the operators like =, [], (),->

Unary operator overload the single argument. It helps in increment or decrement and other by the number that we want. One example of unary operator overloading is:

#include<iostream>

using namespace std;

class complex

{

int real,img;

public :

complex()

{

real=0;

img=0;

}

complex(int i,int r)

{

real=r;

img=i;

}

complex operator++();

void display()

{

cout<<"real part="<<real<<endl;

cout<<"imag part= "<<img<<endl;

}

};

complex complex :: operator++()

{

complex t;

t.real=++real;

t.img=++img;

return t;

}

int main()

{

complex c1(1,2),c2;

cout<<"before overloading: "<<endl;

c1.display();

c2=++c1;

cout<<"after overloading: "<<endl;

c2.display();

c1.display();

return 0;

}

1. **What are different form of inheritance? Give an example of for each. WAP which contain base class that ask the user to enter a complex number and make a derived class that add a complex number of its own with the base. Finally make third class that is friend of derived and calculate the difference of base complex number and its own complex number.[3+5]**

**Answer:**

Different form of inheritance are:

Single, multiple, multiple, hierarchical, hybrid, multipath

Examples  
single inheritance:

#include<iostream>

using namespace std;

class Base

{

protected:

int x;

public:

void getnum()

{

cout<<"Enter any integer number"<<endl;

cin>>x;

}

};

class Child : public Base

{

public:

void displaynum()

{

cout<<"The entered integer is "<<x<<endl;

}

};

int main()

{

Child d;

d.getnum();

d.displaynum();

getch();

return (0);

}

**Multilevel inheritance:**

#include<iostream>

#include<conio.h>

using namespace std;

class Base

{

protected:

float x,y;

public:

void getvalue()

{

cout<<"Enter two numbers"<<endl;

cin>>x>>y;

}

};

class Intermediate : public Base

{

protected:

float sum;

public:

void calculatesum()

{

sum=x+y;

}

void displaysum()

{

cout<<"The sum is "<<sum<<endl;

}

};

class Child : public Intermediate

{

protected:

float ave;

public:

void calculateaverage()

{

ave=sum/2;

}

void displayaverage()

{

cout<<"The average is "<<ave<<endl;

}

};

int main()

{

Child d;

d.getvalue();

d.calculatesum();

d.displaysum();

d.calculateaverage();

d.displayaverage();

getch();

return(0);

}

**Hierarchical inheritance :**

#include<iostream>

#include<conio.h>

using namespace std;

class Base

{

protected:

int x,y;

public:

void getdata()

{

cout<<"Enter two numbers"<<endl;

cin>>x>>y;

}

};

class Child1 : public Base

{

protected:

int sum;

public:

void calculatesum()

{

sum=x+y;

}

void displaysum()

{

cout<<"The sum is "<<sum<<endl;

}

};

class Child2 : public Base

{

protected:

int mul;

public:

void calculatemultiple()

{

mul=x\*y;

}

void displaymultiple()

{

cout<<"The multiple is "<<mul<<endl;

}

};

class Child3 : public Base

{

protected:

int diff;

public:

void calculatedifference()

{

diff=x-y;

}

void displaydifference()

{

cout<<"The difference is "<<diff<<endl;

}

};

int main()

{

Child1 A;//for sum

Child2 B;//for multiple

Child3 C;//for difference

A.getdata();

A.calculatesum();

A.displaysum();

B.getdata();

B.calculatemultiple();

B.displaymultiple();

C.getdata();

C.calculatedifference();

C.displaydifference();

getch();

return(0);

}

**Hybrid inheritance :**

#include <iostream>

using namespace std;

class A{

public:

int x;};

class B : public A {

public:

B() //constructor to initialize x in base class A

{ x = 10;}

};

class C {

public:

int y;

C() //constructor to initialize y

{ y = 4; }

};

class D : public B, public C //D is derived from class B and class C {

public:

void sum()

{ cout << "Sum= " << x + y; }

};

int main() {

D obj1; //object of derived class D

obj1.sum();

return 0;}

**multiple inheritance**

#include<iostream>

using namespace std;

class A

{

public:

  A()  { cout << "A's constructor called" << endl; }

};

class B

{

public:

  B()  { cout << "B's constructor called" << endl; }

};

class C: public B, public A  // Note the order

{

public:

  C()  { cout << "C's constructor called" << endl; }

};

int main()

{

    C c;

    return 0;

}

**Ans of last part:**

#include<iostream>

using namespace std;

class complex

{

public:

int real, img ;

void getdata()

{

cout<<"enter real and img part in base"<<endl;

cin>>real>>img;

}

};

class complex1 :public complex

{

public :

int i,r,ig,rl;

void get()

{

cout<<"enter img and real part in complex1"<<endl;

cin>>r>>i;

}

void add()

{

ig=i+img;

rl=r+real;

cout<<"sum is : "<<rl<<"+"<<ig<<"j"<<endl;

}

friend class complex2;

};

class complex2

{

public:

int r1,i1;

void getdata2()

{

cout<<"enter img and real part in complex2"<<endl;

cin>>r1>>i1;

}

void diff(complex a)

{

int x,y;

x= a.real - r1;

y= a.img - i1;

cout<<"difference is :"<<x<<"+"<<y<<"j"<<endl;

}

};

int main()

{

complex1 c1;

complex2 c2;

c1.getdata();

c1.get();

c1.add();

c2.getdata2();

c2.diff(c1);

return 0;

}

1. **Define virtual function with suitable example. Explain how dynamic\_cast and typeid operator are used to achieve RTTI.[5+3]**

**Answer:**

A virtual function is a member function in the base class that we expect to redefine in derived classes. Basically, a virtual function is used in the base class in order to ensure that the function is overridden. This especially applies to cases where a pointer of base class points to an object of a derived class.

Example:

#include <iostream>

using namespace std;

class Base {

public:

virtual void print() {

cout << "Base Function" << endl;

}

};

class Derived : public Base {

public:

void print() {

cout << "Derived Function" << endl;

}

};

int main() {

Derived d1;

// pointer of Base type that points to d1

Base\* b1 = &d1;

// calls member function of Derived class

b1->print();

return 0;

}

RTTI is provided through two operators: The typeid operator, which returns the actual type of the object referred to by a pointer (or a reference). The dynamic\_cast operator, which safely converts from a pointer (or reference) to a base type to a pointer (or reference) to a derived type.

Dynamic\_cast operator

Syntax:

Dynamic\_cast<target\_type>expressiom;

Typeid() operator

Syntax:

Typeid(expression)

1. **Write short note on file access pointer and their manipulators. Write a program to make simple library management system of collage. Your program should store and retrieve the information (book name, book ID, number of book and purchase date).[3+5]**

**Answer:**

A pointer is used to handle and keep track of the files being accessed. A file pointer stores the current position of a read or write within a file. All operations within the file are made with reference to the pointer. The data type of this pointer is defined in stdio. h and is named FILE.

Manipulators are special functions that can be included in the I/O statement to alter the format parameters of a stream. Manipulators are operators that are used to format the data display. To access manipulators, the file iomanip. h should be included in the program. Every file maintains two pointers called get\_pointer (in input mode file) and put\_pointer (in output mode file), which tells the current position where reading or writing will take place with the use of opening modes and their respective manipulators.

#include<iostream>

#include<conio.h>

#include<fstream>

using namespace std;

class College

{

private:

char Bname[30],Date[30];

int Bid,Bn;

public:

void getdata()

{

cout<<"Enter the book name "<<endl;

cin>>Bname;

cout<<"Enter the book id "<<endl;

cin>>Bid;

cout<<"Enter the number of books purchased "<<endl;

cin>>Bn;

cout<<"Enter the purchased date (YY:MM:DD) "<<endl;

cin>>Date;

}

void showdata()

{

cout<<"Book Name : "<<Bname<<endl;

cout<<"Book ID : "<<Bid<<endl;

cout<<"Number of Book Purchased : "<<Bn<<endl;

cout<<"Purchased Date (YY:MM:DD): "<<Date<<endl;

}

};

void writetofile()

{

College C;

ofstream outfile("Library.dat",ios::binary|ios::app);

cout<<"Writing Data to File"<<endl;

C.getdata();

outfile.write(reinterpret\_cast < char \* > (&C),sizeof(C) );

}

void readfromfile()

{

College C;

cout<<"Reading Data From File"<<endl;

ifstream infile("Library.dat",ios::binary|ios::app);

while(!infile.eof())

{

if(infile.read(reinterpret\_cast<char \* >(&C),sizeof(C))>0)

{

C.showdata();

}

}

}

void readonerec() //reading specific data from the file

{

College C;

int n; //n is declarec to specifically choose the file number

ifstream infile("Library.dat",ios::binary);

cout<<"Enter record Number : ";

cin>>n;

infile.seekg((n-1)\*sizeof(C)); //searching the data in the file

infile.read(reinterpret\_cast<char\*>(&C),sizeof(C)); //reading the data from file

C.showdata(); //displaying the readed data from file

}

int main()

{

int Choice;

cout<<"LIBRARY RECORD SYSTEM "<<endl;

cout<<"SELECT ONE OPTION BELOW "<<endl;

cout<<"1 to write record to file"<<endl;

cout<<"2 to read from file"<<endl;

cout<<"3 to read one record"<<endl;

cout<<"Any other Key to Exit"<<endl;

while(true)

{

cout<<"Enter Your Choice : "<<endl;

cin>>Choice;

switch(Choice)

{

case 1:

writetofile();

break;

case 2:

readfromfile();

break;

case 3:

readonerec();

break;

default:

cout<<"Choice Not Available";

exit(0);

}

}

getch();

return(0);

}

1. **Briefly explain the importance of function template and class template with suitable example. WAP to create a derive class which is a template form of base class which is also template with additional template parameter in the derive class than that of the parameter in base class.[4+4]**

**Answer:**

A template is a simple yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types..

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

Templates simplify the creation of documents. Templates can ease our workload and make us feel less stressed, and, at the same time, they increase efficiency. Templates increase the attention of the audience.

**Example of class template:**

#include <iostream>

using namespace std;

// Class template

template <class T>

class Number

{

T num;

public:

Number(T n) : num(n) {} // constructor

T getNum()

{

return num;

}

};

int main()

{

Number<int> n (7);

Number<double> nd (7.7);

cout << "int Number = " << n.getNum() << endl;

cout << "double Number = " << nd.getNum() << endl;

return 0;

}

**Example of Function template:**

#include <iostream>

using namespace std;

template <typename T>

T add(T num1, T num2) {

return (num1 + num2);

}

int main()

{

int result1;

double result2;

// calling with int parameters

result1 = add(2, 3);

cout << "2 + 3 = " << result1 << endl;

// calling with double parameters

result2 = add(2.2, 3.3);

cout << "2.2 + 3.3 = " << result2 << endl;

return 0;

}

**Answer for 2nd part**

#include<iostream>

using namespace std;

template<class t>

class base

{

t data;

public:

base()

{

}

base(t a)

{

data=a;

}

void display()

{

cout<<"data :"<<data<<endl;

}

};

class derived1 : public base<int>

{

public :

derived1()

{

}

derived1(int a) : base<int>(a)

{

}

};

template<class t>

class derived2 : public base<int>

{

t data;

public :

derived2(int a, t b): base<int> (a)

{

data= b;

}

void display()

{

cout<<"in base"<<endl;

base<int>::display();

cout<<"in derived data: "<<data;

}

};

int main()

{

derived1 ob(5);

ob.display();

derived2<float> obb(10,12.5);

obb.display();

return 0;

}

1. **What is advantage of having exception handling in the program? How are multiple exceptions handled? Explain about catching all exception in exception handling mechanism. [2+3+3]**

**Answer:**

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program

Advantages of having exception handling in the program are:

* Separating Error Handling Code from "Regular" Code.
* Propagating Errors Up the Call Stack.
* Grouping Error Types and Error Differentiation.

Multiple catch blocks are used when we have to catch a specific type of exception out of many possible type of exceptions i.e. an exception of type char or int or short or long etc.

Example:

#include<iostream.h>

#include<conio.h>

void test(int x)

{

try

{

if (x > 0)

throw x;

else

throw 'x';

}

catch (int x)

{

cout << "Catch a integer and that integer is:" << x;

}

catch (char x)

{

cout << "Catch a character and that character is:" << x;

}

}

void main()

cout << "Testing multiple catches\n:";

test(10);

test(0);

getch();

}

We can catch all the expression on the program by using this keyword (…) in the place where we take or indicate the argument that we want to catch. (…) this keyword allow us to catch any data type argument.

**Example:**

#include<iostream>

using namespace std;

void test(int x)

{

try

{

if(x==1)

throw x;

else

if(x==0)

throw 'x';

else

if(x== -1)

throw 1.0;

}

catch(...)

{

cout<<"caught a exception you want"<<endl;

}

}

int main()

{

cout<<"testing multiple catches"<<endl;

test(1);

test(0);

test(-1);

return 0;

}