***2078 Kartik***

**1)What are the advantages and disadvantages of OOP? What are the features of OOP?**

Ans:

Advantages:

1)It provides greater security due to data abstraction.

2)Using concept of classes and objects, code can be reused.

3)Code is easier to maintain.

4)Inheritance helps reduce the redundancy while creating new classes having some functionality of previous classes.

Disadvantages:

1)File size is usually greater than other types of programming language.

2)They are slower to execute due to larger size

3)Programmers need to be skilled in order to write OOP code.

4)The concept of OOP cannot be used to solve all types of problems.

**Features of OOP:**

1)Data abstraction and Encapsulation –

Data abstraction means hiding the data and providing only the necessary details. The binding of data and methods into a single unit is called encapsulation. With use of access modifiers like public, private and protected , we can control the access of data members and member functions.

2)Polymorphism: -

Polymorphism is the ability to take more than one form. For example, We could not create 2 or more functions with same name in C language while in C++, we can create any number functions having same name.

3)Inheritance: -

The ability to inherit properties of one class to another class such that the latter can access all the data and function members of the initial class is called inheritance. Inheritance helps to reduce redundant code and makes code reusable.

4)Classes and Objects: -

A class is the main block of OOP. It is a user defined data type that holds data members and member functions in a single unit.

While classes may be the blueprint, an object is a physical instance of a class. Unlike classes , objects occupy memory.

**2)What are the properties of Constructors? What are the differences between copy constructor and assignment operator, explain. “A friend function is not a member of any classes but still has access to the members of a class where it is declared as friend”, justify with example.**

Ans:

Properties of constructors: -

1)They must have the same name as the class.

2)Constructors are automatically called when an object is created.

3)Constructors do not have a return type.

4)Constructors can be overloaded.

A copy constructor is used to copy the values of one object into another object while an assignment operator is used to assign values to variables. Copy constructors can assign values to multiple variables at the same time whereas the assignment operator assigns value to only one variable at a time.

Friend function: -

The “friend” keyword allows the function declared using this keyword to access members of the class which is passed as argument to the function.

#include<iostream>

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

**class** **A**{

**private:**

**int** x;

**public:**

**void** **setData**(**int** a){

x=a;

}

**friend** **void** **showdata**(A a);

};

**void** **showdata**(A a){

cout << a.x;

}

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

A a;

a.setData(**5**);

showdata(a);

}

**3)Explain the need of C++ language. Explain the features of C++.**

Ans:

C++ language was developed to enable programmers to write object oriented code. C++ is the better version of C language and has added features. Its overcomes the shortcomings of C language. C++ combines the simplicity of a low-level language with the concept of OOP in a single language which results in a language that is fast and can be used in the real world as well.

**Features of C++ : -**

1)Classes: -

Classes are user-defined data types in C++. They are the blueprint on the basis of which objects are created. They have data members and member functions which are specified as private, public or protected using access modifiers.

2)Constructor & Destructor: -

Constructor is the special type of member function in C++ classes, which are automatically invoked when an object is being created. It is used to initialize data members of a class or assign value to these data members.

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed. Destructor has the same name as their class name preceded by a tiled (~) symbol.

3)Function overloading: -

Function overloading is a special function of C++ which allows us to use the same name for multiple functions by differing the number of arguments or the type of arguments that are to be passed to those functions.

4) Operator overloading: -

Operator overloading allows us to give special meaning to the preexisting operators in C++.Since we cannot use normal operators to operate on objects, operator overloading is required to do such operations.

5) Default Arguments

**4)Define inline functions with example. Explain the use of ‘new’ and ‘delete’ operators for dynamic memory allocation.**

Ans:

Inline function is a powerful concept in C++.When an inline function is called, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Example:-

#include<iostream>

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

**inline** **int** **max**(**int** a, **int** b){

**if**(a>b){

**return** a;

}**else**{

**return** b;

}

}

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

cout << max(**20**,**30**) <<endl;

cout << max(**200**,-**30**);

}

The new operator requests for the memory allocation in heap. If the sufficient memory is available, it initializes the memory to the pointer variable and returns its address.

Syntax: -

Pointer\_variable = new datatype;

The delete operator is used to deallocate the memory. User has privilege to deallocate the created pointer variable by this delete operator.

Syntax: -

delete pointer\_variable;

**5) Why do we use operator overloading in C++? List the operators that cannot be overloaded. Write a program that converts object of Celsius type to object of Fahrenheit type.**

Ans:

Operator overloading is used in C++ to specify additional meaning to an operator. It is usually used to perform operations related to objects.

The operators that cannot be overloaded are: -

1) :: -Scope resolution operator

2) ?: -ternary operator.

3) . - member selector

4) Sizeof operator

5) \* -member pointer selector

6) typeid operator

#include<iostream>

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

**class** **fahr**{

**float** temp;

**public:**

fahr(){

temp = **0.0**;

}

fahr (**float** t){

temp = t;

}

**void** display(){

cout << "Temp in Fahrenheit = " << temp << endl;

}

};

**class** **celcius**{

**float** tempr;

**public:**

celcius(){

tempr = **0.0**;

}

celcius(**float** t){

tempr = t;

}

**void** display(){

cout << "Temp in celcius = " << tempr << endl;

}

**operator** fahr(){

**float** t;

t = tempr\***9**/**5** + **32**;

**return** **fahr**(t);

}

};

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

fahr f;

celcius c(**32**);

f = c;

c.display();

f.display();

**return** **0**;

}

**6)List the rules of operator overloading. Write a program to add two time objects using operator overloading.**

Rules of operator overloading: -

1)Only preexisting operators can be overloaded.

2)Overloaded operator must have one user-defined-data-type operand.

3)Overloaded operators follow the syntax rules of the original operator. They can’t be overridden.

4)The precedence and associativity of the operators remains same even after overloading.

5)Friend function cannot be used to overload “=”,”( )”,”[ ]” and “->” operators.

6)The operator function for overloaded operators cannot have default arguments.

7)We can’t change the number of operands an operator takes.

8)The basic meaning of operator cannot be changed.

#include<iostream>

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

**class** **Time**{

**int** hr,min,sec;

**public:**

Time(){

min = **0**;

hr = **0**;

sec = **0**;

}

Time(**int** a, **int** b, **int** c){

hr = a;

min = b;

sec = c;

}

**void** display(){

cout << "Time is " << hr <<":" << min << ":" << sec << endl;

}

Time **operator** +(Time t1){

Time temp;

temp.hr = hr + t1.hr;

temp.min = min + t1.min;

temp.sec = sec + t1.sec;

**if**(temp.sec > **60**){

temp.sec = temp.sec % **60**;

temp.min = temp.min + **1**;

}

**if**(temp.min > **60**){

temp.min = temp.min % **60**;

temp.hr = temp.hr + **1**;

}

**return** temp;

}

};

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

Time t1(**1**,**35**,**40**),t2(**1**,**40**,**22**), t3;

t3 = t1 + t2;

t3.display();

**return** **0**;

}

**7)Explain virtual function with example. What do you mean by Run Time type Information? Exlpain.**

Ans:

A virtual function is a member function which is declared within a base class and is overridden by a derived class.

Example: -

#include<iostream>

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

**class** **base** {

**public:**

**virtual** **void** print()

{

cout << "print base class**\n**";

}

**void** show()

{

cout << "show base class**\n**";

}

};

**class** **derived** : **public** base {

**public:**

**void** print()

{

cout << "print derived class**\n**";

}

**void** show()

{

cout << "show derived class**\n**";

}

};

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

{

base \*bptr;

derived d;

bptr = &d;

bptr->print();

bptr->show();

**return** **0**;

}

In C++, Run-time type information is a mechanism that exposes information about an object’s data type at runtime and is available only for the classes which have at least one virtual function. It allows the type of an object to be determined during program execution.

For example, the following program fails with the error “cannot dynamic\_cast `b’ (of type `class B\*’) to type `class D\*’ (source type is not polymorphic) ” because there is no virtual function in the base class B.

#include <iostream>

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

**class** **B** {};

**class** **D** : **public** B {};

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

{

B\* b = **new** D;

D\* d = **dynamic\_cast**<D\*>(b);

**if** (d != NULL)

cout << "works";

**else**

cout << "cannot cast B\* to D\*";

getchar();

**return** **0**;

}

Adding a virtual function to the base class B makes it work.

#include <iostream>

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

**class** **B** {

**virtual** **void** fun() {}

};

**class** **D** : **public** B {

};

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

{

B\* b = **new** D;

D\* d = **dynamic\_cast**<D\*>(b);

**if** (d != NULL)

cout << "works";

**else**

cout << "cannot cast B\* to D\*";

getchar();

**return** **0**;

}

**8)Why use file handling? Write a program in a file of student to add record, list the record, search by roll and delete record.**

Ans:

File Handling is used for store a data permanently in computer. Using file handling we can store our data in Secondary memory (Hard drive). Without using file handling, our data is stored in primary memory which is immediately deleted after we exit the program.

#include<iostream>

#include<fstream>

#include<conio.h>

//Code by Biswas Khanal

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

**int** recordnum=**0**;

**class** **Student**{

**char** name[**20**];

**int** roll\_no;

**public:**

**void** **read\_data**(){

cout << "Enter name: " ;

cin >> name;

cout << "Enter roll no: ";

cin >> roll\_no;

}

**void** **show\_data**(){

cout << "Name: " << name << endl;

cout << "Roll-no: " << roll\_no << endl;

}

**int** **getroll**(){

**return** roll\_no;

}

};

**void** **write**(){

cout<<"WRITING DATA TO FILE: **\n**";

Student st[**100**];

**int** i =**0**;

**char** choice = 'Y';

ofstream fbout("student.bin", ios::binary|ios::out);

**if**(!fbout){

cout<<"Error in opening file!";

}

**else**{

**while**(choice =='Y')

{ cout<<"**\n**Record Number: "<<i+**1**<<endl;

st[i].read\_data();

fbout.write((**char**\*)&st[i], **sizeof**(st[i]));

cout<<"**\n**Add more records? Y/N**\n**";

choice = toupper(getch());

i++;

recordnum++;

}

fbout.close();

}

}

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

cout<<"**\n**Reading DATA from FILE: **\n**";

Student st[**100**];

**int** i =**0**;

**char** choice = 'Y';

ifstream fbin("student.bin", ios::binary|ios::in);

**if**(!fbin){

cout<<"Error in opening file!";

}

**else**{

**while**(fbin.read((**char**\*)&st[i], **sizeof**(st[i])))

{ cout<<"**\n**Record Number: "<<i+**1**<<endl;

st[i].show\_data();

i++;

}

fbin.close();

}

}

**void** **search**(){

cout<<"**\n**searching DATA from FILE: **\n**";

Student st[**100**];

**int** i =**0**;

**int** r;

cout<<"Enter roll number to search: ";

cin>>r;

ifstream fbsearch("student.bin", ios::binary|ios::in);

**if**(!fbsearch){

cout<<"Error in opening file!";

}

**else**{

**while**(fbsearch.read((**char**\*)&st[i], **sizeof**(st[i])))

{

**if**(st[i].getroll()==r){

cout<<"**\n**Record Number: "<<i+**1**<<endl;

st[i].show\_data();

**break**;

}

i++;

}

**if**(i==recordnum){

cout<<"Record with matching roll not found!";

}

fbsearch.close();

}

}

**void** **deletion**(){

cout<<"**\n**deleting DATA from FILE: **\n**";

Student st[**100**];

**int** i =**0**;

**int** r;

cout<<"Enter roll number to delete: ";

cin>>r;

ifstream fbin("student.bin", ios::binary|ios::in);

ofstream fbout("temp.bin", ios::binary|ios::out);

**if**(!fbin||!fbout){

cout<<"Error in opening file!";

}

**else**{

**while**(fbin.read((**char**\*)&st[i], **sizeof**(st[i])))

{

**if**(st[i].getroll()!=r){

fbout.write((**char**\*)&st[i], **sizeof**(st[i]));

}

**else**{

cout<<"**\n**Record found!";

st[i].show\_data();

}

i++;

}

fbin.close();

fbout.close();

remove("student.bin");

rename("temp.bin", "student.bin");

cout<<"Deletion Complete!";

}

}

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

**char** choice;

**while**(true){

cout <<"**\n\n**1)Write record to file" << endl;

cout <<"2)Read record from file" << endl;

cout <<"3)Search record" << endl;

cout <<"4)Delete record" << endl;

cout <<"Or, Any other key to exit" << endl;

cout << "Enter choice";

choice=getch();

system("cls");

**switch**(choice){

**case** '1':

write();

**break**;

**case** '2':

read();

**break**;

**case** '3':

search();

**break**;

**case** '4':

deletion();

**break**;

**default:**

exit(**0**);

}

}

**return** **0**;

}

**9)Define Class Templates with example. Write a program to demonstrate example of function overloading with function template and normal function.**

Ans:

A class that operates on any type of data is called class template.

Syntax:

Template<**class** **template\_type**, ….>

**class** **class\_name** {

**private:**

//data member of template or non template type

**Public:**

//function members with template type argument and return type

};

Example:-

#include<iostream>

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

**template**<**class** **T1**, **class** **T2**>

**class** **Test**{

**private:**

T1 a;

T2 b;

**public:**

Test(){};

Test(T1 n1, T2 n2){

a = n1;

b = n2;

}

**void** display(){

cout << "DATA:" << a << " And " << b <<endl;

}

};

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

Test<**int**,**double**> v(**5**,**2.0**);

v.display();

**return** **0**;

program to demonstrate example of function overloading with function template and normal function: -

#include<iostream>

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

**template** <**class** **T1**, **class** **T2**>

//Overloading using function template

**void** find\_max(T1 c, T2 d){

**if**(c>d){

cout << c <<endl;;

}**else**{

cout << d <<endl;

}

};

//overloading using normal function

**void** **find\_min**(**int** a, **int** b, **int** c){

**if**(a<b && a<c){

cout << a <<endl;

}**else** **if**( b<a && b< c){

cout << b << endl;

}**else**{

cout << c <<endl;

}

}

**void** **find\_min**(**float** x , **int** y){

cout << x<<" ," << y;

}

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

**int** i1 = **2**,i2= **3** , i3=**5**;

find\_max(i1,i2);

find\_min(i1,i2,i3);

find\_min(**2.2** , **3**);

}

**10)How is exceptional handling better than conventional error handling?Explain the exception handling mechanism in C++ with example.**

Ans:

Conventional error handling is inconvenient because every call must be surrounded with an if..else statement to handle error or call the error routine. This increases the size of the program.

Exception handling separates the error handling code from other code making program more readable.

In C++ we use three keyword to perform exception handling:-

1)Throw – when a program experiences an issue, it throws an Exception. The throw keyword assists the program by performing throw.

2)Catch – a program uses an exception handler to catch an exception. It is added to the part of a program where you need to deal with the error.

3)Try – the try block recognizes the code block for which certain exceptions will be enacted. It ought to be followed by one/more catch blocks.

Example:-

#include <iostream>

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

**double** **zeroDivision**(**int** x, **int** y) {

**if** (y == **0**) {

**throw** "Division by Zero!";

}

**return** (x / y);

}

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

**int** a = **11**;

**int** b = **0**;

**double** c = **0**;

try {

c = zeroDivision(a, b);

cout << c << endl;

}

**catch** (**const** **char**\* message) {

cout << message << endl;

}

**return** **0**;

}