**SHARWAN 073 old question Solution Set**

1. Explain the characteristics of OOP. Write a program to create class “time” with data members hours, minute and second. Then add two “time” object by taking object as argument and also returning object as argument.

Ans: Object-oriented programming is a computer programming model that organizes software design around data or object, rather than function and logic. The characteristics of OOP are:

* Focus is on data rather than procedure.
* Program are divided into object.
* Function & data are tied together in a single unit.
* Data can be hidden to prevent from accidental modification
* Objects may communicate through functions.
* It support reusability of code. Also supports inheritance, polymorphism, data abstraction and encapsulation.

Program:

#include<iostream>

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

**class** **time**

{

**int** hour,min,sec;

**public:**

time()

{

hour=**0**;

min=**0**;

sec=**0**;

}

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

{

hour=a;

min=b;

sec=c;

}

time add(time t1)

{

time t;

t.hour=hour+t1.hour;

t.min=min+t1.min;

t.sec=sec+t1.sec;

**if**(t.sec>=**60**)

{

t.min++;

t.sec=t.sec-**60**;

}

**if**(t.min>=**60**)

{

t.hour++;

t.min=t.min-**60**;

}

**return** t;

}

**void** display()

{

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

}

};

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

{

time t1(**10**,**12**,**36**),t2(**12**,**54**,**34**),t3;

t3=t1.add(t2);

cout<<"Added time is:";

t3.display();

**return** **0**;

}

1. Why don’t you use an object to call the static Member function, explain with example? Why do you need to use a reference in the argument to the copy constructor? Write a program to calculate the perimeter of Triangle using Default and Parameterized constructor.

Ans: Static function is defined by using the keyword ‘static’ before the member function that is to be declared as static function. It can’t be called by using object because static member function only contains static member only and cannot access non-static member. An example is shown below to demonstrate static member function:

#include<iostream>

#include<conio.h>

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

**class** **test**

{

**int** code;

**static** **int** count;

**public:**

**void** **setcode**()

{

code=++count;

}

**static** **void** **showcount**()

{

cout<<"count:"<<count<<endl;

}

};

**int** test::count=**0**;

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

{

test t1,t2;

t1.setcode();

t2.setcode();

test::showcount();

test t3;

t3.setcode();

test::showcount();

**return** **0**;

}

The type of constructor that initializes the object by copying the value of the object of its own type from the argument is called copy constructor. Since the default behavior of the copy constructor is one by one copy of its member, so we need to use reference in the argument of copy constructor.

Program:

#include<iostream>

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

**class** **perimeter**

{

**private:**

**float** a, b, c, s;

**public:**

perimeter()//Default constructor

{

a=**0**;

b=**0**;

c=**0**;

}

perimeter(**float** x, **float** y, **float** z) //Parameterized constructor

{

a=x;

b=y;

c=z;

}

**void** peri()

{

s=a+b+c;

}

**void** showperi()

{

cout<<"The perimeter is :"<<s;

}

};

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

{

perimeter a(**5**,**6**,**7**);

a.peri();

a.showperi();

**return** **0**;

}

1. When inline function may not work? What do you understand by Default Arguments? Write syntax of Default Arguments. Write a program to display N number of characters by using default arguments for both parameters. Assume that the function takes two argument one character to be printed and another number of character to be printed.

Ans: When the function is larger or if the function body consist of loop or if the function are recursive then inline function may not work.

The mechanism of providing less number of argument than the actual number of parameter is called default argument. In such case, function assign default values to the parameters which doesn’t have a matching argument in the function call.

Syntax:   
return\_type function\_name(default\_values)

{  
//body of function  
}

Program:

#include<iostream>

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

**void** **def**(**char** name[**20**]="Program",**int** n=**4**)

{

**char** \*p=name;

**int** num=n;

cout<<"String to be printed is:";

**for**(**int** i=**0**;i<n;i++)

{

cout<<p[i];

}

}

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

{

def();

**return** **0**;

}

1. Explain the syntax of operator overloading. Create a class named City that will have two member variables CityName(chart[20]) and DistFromKtm(float). Add member functions to set and retrieve the Cityname and DiarFromKtm separately. Add operator overloading to find the distance between these cities (just find the difference of DistFromKtm) and sum of distance of those cities from Kathmandu. In the main function, initialize three city objects. Set first and second city to be Pokhara and Dhangadi. Display the sum of DistFromKtm of Pokhara and Dhangadi and distance between Pokhara and Dhangadi.

Ans: The syntax of operator overloading is:

return\_type operator operator\_symbol( argument\_list)  
{  
//function body  
}

Program:

#include<iostream>

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

**class** **city**

{

**char** cityname[**30**];

**float** DistFromktm;

**public:**

**void** **ctnm**()

{

cout<<"Enter city name;"<<endl;

cin>>cityname;

}

**void** **disfrmktm**()

{

cout<<"Enter distance from ktm to"<<cityname<<":"<<endl;

cin>>DistFromktm;

}

city **operator** +(city p)

{

city dist;

dist.DistFromktm=p.DistFromktm+DistFromktm;

**return** dist;

}

city **operator** -(city p)

{

city dist;

dist.DistFromktm=p.DistFromktm-DistFromktm;

**if**(dist.DistFromktm<**0**)

{

dist.DistFromktm=-dist.DistFromktm;

}

**return** dist;

}

};

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

{

city pokhara,dhangadi,kathmandu;

pokhara.ctnm();

pokhara.disfrmktm();

dhangadi.ctnm();

dhangadi.disfrmktm();

kathmandu=pokhara+dhangadi;

cout<<"The sum of distance between these cities:";

kathmandu.displaydist();

cout<<"The distance between these cities:";

kathmandu=pokhara-dhangadi;

kathmandu.displaydist();

**return** **0**;

}

1. What do you mean by function overriding and how can we access every overridden function from the derived class object? Explain with example. Write a program to show the execution order of constructor and destructor in multilevel inheritance. Show your program output.

Ans: The process of creating members in the derived class with the same name as that of the visible members of the base class is called overriding .It is called overriding because the new name overrides the old name inherited from base class. After overridden, when the members are accessed with the overridden n names in derived class or through the object of derived class the derived class members are accessed.

To access every overridden function from the derived class, there are two ways:

1. From member function of derived class
2. From object of derived class

It can be illustrated from example below:

#include<iostream>

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

**class** **base**

{

**protected:**

**int** num;

**public:**

**void** **readdata**()

{

cout<<"Enter number in base:";

cin>>num;

}

**void** **showdata**()

{

cout<<"Number in base ="<<num<<endl;

}

};

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

{

**private:**

**int** num;

**public:**

**void** **readdata**()

{

base::readdata(); // cout<<"Enter number in base through derived:";

//cin>>base::num;

cout<<"Enter number in derived:";

cin>>num;

}

**void** **showdata**()

{

base::showdata();// cout<<"Number in base through derived:";

//cout<<base::num;

cout<<"Number in derived ="<<num<<endl;

}

};

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

{

derived d1;

d1.readdata();

d1.showdata();

**return** **0**;

}

In multilevel inheritance, firstly base constructor is invoked and then first derived class constructor. In similar way constructor are invoked till last derived class and then destructor is invoked. But destructor is invoked in opposite way that is last derived class destructor is invoked first and then the preceding one making base class destructor invoked at last. It can be illustrated from the example below:

#include<iostream>

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

**class** **base**

{

**public:**

base()

{

cout<<"Base Constructor invoked."<<endl;

}

~base()

{

cout<<"Base destructor invoked."<<endl;

}

};

**class** **derived1**:**public** base

{

**public:**

derived1()

{

cout<<"Derived 1 Constructor invoked."<<endl;

}

~derived1()

{

cout<<"Derived 1 destructor invoked."<<endl;

}

};

**class** **deriveda**:**public** base

{

**public:**

deriveda()

{

cout<<"Derived 1 Constructor invoked."<<endl;

}

~deriveda()

{

cout<<"Derived 1 destructor invoked."<<endl;

}

};

**class** **derivedb**:**public** deriveda

{

**public:**

derivedb()

{

cout<<"Derived 2 Constructor invoked."<<endl;

}

~derivedb()

{

cout<<"Derived 2 destructor invoked."<<endl;

}

};

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

{

derivedb d1;

**return** **0**;

}

And its output will be as:

Base Constructor invoked.

Derived **1** Constructor invoked.

Derived **2** Constructor invoked.

Derived **2** destructor invoked.

Derived **1** destructor invoked.

Base destructor invoked.

1. What are the different ios class functions and flags that are used for formatted I/O operation? Write a program to read and write the information of 10 students in a file. Also modify the students information according to the given roll number.

Ans: The different ios class functions for formatted I/O are:

* width()
* fill()
* precision()
* setf()

The flags that are used for formatted I/O are:

* ios::left
* ios::right
* ios::interval
* ios::dec
* ios::hex
* ios::oct

Program:

#include <iostream>

#include<fstream>

**const** **char**\* datafile="final.dat";

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

**class** **student**

{

**private:**

**char** name[**25**];

**int** age,roll;

**public:**

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

{

cout<<"Enter name:";

cin>>name;

cout<<"Enter age:";

cin>>age;

cout<<"Roll";

cin>>roll;

cout<<endl;

}

**void** **show\_rec**()

{

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

cout<<"Age:"<<age<<endl;

cout<<"roll"<<roll<<endl;

cout<<endl;

}

**void** **write2file**()

{

ofstream outfile(datafile,ios::binary|ios::app);

read\_data();

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

}

**void** **edit**()

{

**int** n;

fstream iofile(datafile,ios::in|ios::binary);

**if**(!iofile)

{

cout<<"file not found";

**return**;

}

iofile.seekg(**0**,ios::end);

**int** count=iofile.tellg()/**sizeof**(\***this**);

cout<<"**\n**There are"<<count<<"records";

cout<<"\Enter record number to edit :";

cin>>n;

iofile.seekg((n-**1**)\***sizeof**(\***this**));

iofile.read(**reinterpret\_cast**<**char** \*>(**this**),**sizeof**(\***this**));

show\_rec();

iofile.close();

iofile.open(datafile,ios::out|ios::in|ios::binary);

iofile.seekp((n-**1**)\***sizeof**(\***this**));

cout<<"ENter data to modify:";

read\_data();

iofile.write(**reinterpret\_cast**<**char** \*>(**this**),**sizeof**(\***this**));

}

};

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

{

student stu;

**int** choice;

cout<<"STUDENT RECORD SYSTEM";

**while**(true)

{

cout<<"**\n**Select your option:";

cout<<"**\n**t1-->Write record";

cout<<"**\n**t2-->Edit record";

cout<<"**\n**t2-->Exit";

cout<<"**\n**Enter your choice:";

cin>>choice;

**switch**(choice)

{

**case** **1**:

stu.write2file();

**break**;

**case** **2**:

stu.edit();

**break**;

**case** **3**:

exit(**0**);

**break**;

**default:**

cout<<"**\n**Choice not available";

exit(**0**);

}

}

**return** **0**;

}

1. What do you mean by class template and function template? Write down the syntax of Class template and Function template. Write a program to read your Data of Birth and display it. Your program should throw multiple exception for day, month and other values not in range using exception class and each exception is handled by separate handler.

Ans: Templates support generic programming, allowing development of reusable software components with function and classes, supporting different data types in a single framework. The template declared for function are called function template and those for class are called class template.

Syntax for Function template is:

template<class template\_type,…>  
return\_type func\_name(parameter\_list)  
{  
//body  
}

A class that operates on any type of data is called class template. The syntax for class template is:

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  
};

Program:

#include<iostream>

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

**class** **DAYERR**{};

**class** **MONTHERR**{};

**class** **YEARERR**{};

**class** **Date**{

**private:**

**int** day,month,year;

**public:**

**void** **read\_date**() **throw** (DAYERR,MONTHERR)

{

cout<<"Enter day:";

cin>>day;

**if**(day>**31**)

**throw** DAYERR();

cout<<"Enter month:";

cin>>month;

**if**(month>**12**)

**throw** MONTHERR();

cout<<"Enter year:";

cin>>year;

**if**(year<**0**)

**throw** YEARERR();

}

**void** **show\_date**()

{

cout<<"Date of birth is:"<<day<<"/"<<month<<"/"<<year;

}

};

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

{

Date d1;

try{

cout<<"Enter your date of birth"<<endl;

d1.read\_date();

d1.show\_date();

}

**catch** (DAYERR)

{

cout<<"Exception:day cannot be greater than 31";

}

**catch** (MONTHERR)

{

cout<<"Exception:month cannot be greater than 12";

}

**catch** (YEARERR)

{

cout<<"Exception:day cannot be negative";

}

**return** **0**;

}

1. Explain different manipulators available in C++. Create class student to store name, age and CRN of students. Write a program to write records of N numbers of students into the file. And your program should search complete information of students from file according to CRN entered by user and display it.

Ans: The different manipulators available in C++ are:

1. endl:

It is used for changing line in output section. It is one of the most commonly used manipulators in C++. Its syntax is:   
cout<<endl;

1. setw():

It is use to set width of line for output section. It is a function and it needs <iomanip.h> as a header file. Its syntax is:

cout<<setw(width);

1. setfill():

It is use fill empty space with the character provided in the function name. As it is also manipulator function, it also requires <iomanip.h> as header file. Its syntax is:

cout<<setw(10)<<setfill(‘character\_to\_fill’);

1. setprecision():

It is used to specify number of digit to be displayed after decimal point and use <iomanip.h> as a header file. Its syntax is:

cout<<setprecision(number of digit to be printed after decimal);

Program:

#include <iostream>

#include<fstream>

**const** **char**\* datafile="final.dat";

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

**class** **student**

{

**private:**

**char** name[**25**];

**int** age,crn;

**public:**

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

{

cout<<"Enter name:";

cin>>name;

cout<<"Enter age:";

cin>>age;

cout<<"CRN";

cin>>crn;

cout<<endl;

}

**void** **show\_rec**()

{

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

cout<<"Age:"<<age<<endl;

cout<<"CRN"<<crn<<endl;

cout<<endl;

}

**void** **write2file**()

{

ofstream outfile(datafile,ios::binary|ios::app);

read\_data();

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

}

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

{

**int** n;

ifstream infile(datafile,ios::binary);

**if**(!infile)

{

cout<<"file not found";

**return**;

}

infile.seekg(**0**,ios::end);

**int** count=infile.tellg()/**sizeof**(\***this**);

cout<<"**\n**There are"<<count<<"records";

cout<<"\Enter CRN number :";

cin>>n;

infile.seekg((n-**1**)\***sizeof**(\***this**));

infile.read(**reinterpret\_cast**<**char** \*>(**this**),**sizeof**(\***this**));

show\_rec();

}

};

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

{

student stu;

**int** choice;

cout<<"STUDENT RECORD SYSTEM";

**while**(true)

{

cout<<"**\n**Select your option:";

cout<<"**\n**t1-->Write record";

cout<<"**\n**t2-->Search record";

cout<<"**\n**t2-->Exit";

cout<<"**\n**Enter your choice:";

cin>>choice;

**switch**(choice)

{

**case** **1**:

stu.write2file();

**break**;

**case** **2**:

stu.read();

**break**;

**case** **3**:

exit(**0**);

**break**;

**default:**

cout<<"**\n**Choice not available";

exit(**0**);

}

}

**return** **0**;

}