**2075 Chaitra**

**Object Oriented Programming**

1. What are the main features of Object Oriented Programming? Would you consider it better than structured programming? If you do, what makes it better? Write down its advantages and disadvantages.

***Soln:***

* The followings are the main features of Object Oriented Programming . They are,
  + Emphasis is on data rather than procedures.
  + Programs are divided into objects.
  + Functions and data are tied together in a single unit.
  + Data can be hidden to prevent from accidental modification.
  + Objects may communicate with each other through functions.
  + It supports reusability of code. Also supports inheritance, polymorphism, data abstraction and encapsulation.
  + Functions and data are tied together in a single unit.

**Yes**, It is better than structured programming. Because of its features such as, Inheritance , Polymorphism, Constructor , Class, Encapsulation etc.

**Advantages of OOP**

1) Redundant code is eliminated by various techniques like inheritance and templates,

2) Through data, hiding, programmer can build secure programs.

3) Division of program into objects makes software development easy Existing classes can server as library class for further Code reusability is much easier.

4) Message passing techniques makes communication easier parameter information among various objects.

5) Upgrading and maintenance software is easily

6) Models real world system perfectly manageable

7) Software complexity can be managed easily.

8) Data can be hidden from outside world using, encapsulation, & data hiding.

9c User can create/define new data type or user -defined type by making class.

**Disadvantages OOP**

1) Compiler and runtime overhead is high because object oriented program requires more time during compilation.

2) For dynamic and runtime support it requires More resource and processing time.

3) The message passing between many objects of complex application may be difficult to trace and debug.

4) Benefits projects only in long run while managing large software

5) Requires the mastery in software engineering and programming methodology.

1. What do you mean by constructor and destructor? Explain necessity of copy constructor with example. Also explain order of invocation of constructor and destructor with example.

***Soln:***

A **constructor** is a special member function which is called automatically during the object creation. So the constructor can be used for the necessary initialization of the data members. The constructor is identified as a function whose name is same as its class.

Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.

Constructor is invoked whenever the object is created and destructor is invoked whenever the object goes out of scope( function ends, program ends ).

**Example:**

//program to show the invocation order of constructor and desturctor

#include<iostream>  
using **namespace** std;  
int count=**0**;  
class **example**

{  
 public:  
 example()

{  
 count++;  
 cout<<"Number of object created"<<count<<endl;  
 }  
~example()

{  
 cout<<"The number of object destroyed"<<count<<endl;  
 count--;  
 }  
};  
int **main**()

{

{  
 cout<<"Object created in main:"<<endl;   
 example el,e2,e3,e4;  
 {  
 cout<<"Object created in block1"<<endl;  
 example e5;  
 }  
 {  
 cout<<"Object created in block 2"<<endl;  
 example e6;  
 }  
 }  
cout<<"Return in main()"<<endl;  
return **0**;  
  
}

1. What type of language if C++? Explain its features.

***Soln:***

* C++ is object hybrid oriented programming language.

C++ is a highly portable language and is often the language of choice for multi-device, multi-platform app development. C++ is an object-oriented programming language and includes classes, inheritance, polymorphism, data abstraction and encapsulation. C++ is a powerful, efficient and fast language. It finds a wide range of applications from GUI applications to 3D graphics for games to real-time mathematical simulations.

**Features of C++**

**i)** ***Object***: Objects are the basic run-time entities in an object-oriented system. They may also represent user-defined data such as vectors, time, and lists. Programming problem is analyzed in terms of objects and the nature of communication between them. Object has its own characteristics and behavior. The characteristics of an object is represented by data while its behavior(methods) is represented by function. In fact, objects are variables of the type class.

**ii***)****Class:*** A class is a collection of objects of similar type. For example, mango, apple, and orange are members of the class fruit. Once a class is created, we can create any number of objects belonging to that class.

**iii) *Abstraction***: Abstraction is a design principle. It is the process of removing characteristics from something in order to reduce it to a set of essential characteristics. Through the process of abstraction, a programmer hides all but the relevant data about a class in order to reduce complexity and increase reusability. Abstraction is a basic representation of a concept.

**iv) *Inheritance***: Inheritance is the process by which objects of one class acquire the properties of objects of another class. In OOP, the concept of inheritance provides the idea of reusability. This means we can add additional features to an existing class without modifying it.

**v) *Polymorphism***: Polymorphism means the ability to take more than one form. The function overloading and operator overloading show the concept of compile time polymorphism while run time polymorphism is achieved by using virtual function.

**vi) *Encapsulation***: The wrapping up of data and functions into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

4) What is function overloading? How is pass by reference done in C++. Explain with suitable example.

**Function Overloading:**

C++ enables several functions of the same name to be defined, as long as these functions have different sets of parameters. This capability is called function overloading. An overloaded functions appears to perform different activities depending on the kind of data sent to it. When an overloaded function is called, the C++ compiler selects the proper function by examining the number, types and order of the arguments in the call. Function overloading is commonly used to create several functions of the same name that performs similar tasks, but on different data types.

**Pass by Reference:**

C++ supports one more type of variable called reference variable, in addition to the value variable and pointer variables of C. Value variables are used to hold some numeric values; pointer variables are used to hold the address of some other value variables. Reference variable behaves similar to both, a value variable and a pointer variable. In program code, it is used similar to that of a value variable, but has an action of a pointer variable. Thus, a reference variable provides an alias (alternative name) of the variable that is previously defined. In C, the corresponding parameter in the calling function must be declared as a pointer type. In C++, the corresponding parameter can be declared as any reference type, not just a pointer type. In pointer variable the returning from the function will be the value while in reference variable in case of returning the value it will return the variable.

**Example:**

#include<iostream>

using **namespace** std;

int **main**()

{

int x=**5**;

int &y=x;

cout<<"X="<<x<<"andY="<<<<endl; y++;

cout<<"X="<<x<<"andY="<<<<endl;

return**0**;

}

5) Write the syntax of operator overloading. Create a class called time that has separate int member data for hours, minutes, and seconds. Once constructor should initialize this data to zero (0) , and another should initialize it to fixed values. A member function should display it in 10:45:30 format. The final member function should add two objects of type time passed as arguments using operator overloading.

***Soln:***

**Syntax of operator overloading:**

class class\_name

{........

public:

friend return\_type **operator** operator\_symbol ([arg,[arg]]);

..........

};  
return\_type **operator** **operator\_symbol**([arg,[arg]])

{

//body of function

}

*// program to create class called time that has separate int member data for hours, minutes, and seconds. One constructor initializing this data to zero(0), and another should initialize it to fixed values. A member function should display it in 10:45:30 format. The final member functions should add two objects of type time passed as arguments using operator overloading*

#include<iostream>

#include<conio.h>

using **namespace** std;

class **time**

{

int hr, min, sec;

public:

time()

{

hr=**0**;

min=**0**;

sec=**0**;  
 }  
   
 time( **int** h, **int** m, **int** s)

{  
 hr=h;

min=m;

sec=s;

}

void display()

{

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

}

void add(time a, time b)

{  
 hr=hr+a.hr+b.hr;

min=min+a.min+b.min;

sec=sec+a.sec+b.sec;

if(sec>**60**)

{

sec=sec-**60**;

min++;

}

if(min>=**60**)

{  
 min=min-**60**;

hr++;  
 }

}  
};

int **main**()

{

time t1, t2(**12**,**34**,**56**),t3(**10**,**35**,**14**);

cout<<"Before adding "<<endl;

t1.display();

t2.display();

t3.display();

cout<<"After adding"<<endl;

t1.add(t2,t3);

t1.display();

}

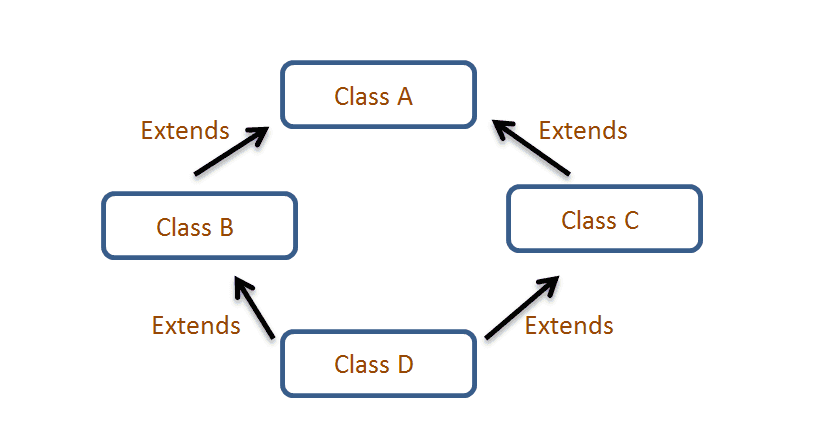
6) How the function over-riding differ from function overloading? When do we face ambiguity problem in multiple inheritance? Explain.

***Soln:***

* The function over-riding differ form function overloading in the following ways;

|  |  |
| --- | --- |
| **Function Overloading** | **Function Overriding** |
| Function overloading can be used in normal functions as well as in classes (for eg: constructor overloading is a classic example where you would vary the number/type of arguments for different initialisations) | Function overriding is applicable exclusively to an inherited class (or in other words a subclass) |
| Function overloading is resolved at compile time | Function overriding is resolved at run time. |
| Overloaded functions are in same scope | Overridden functions are in different scopes |
| Overloaded functions have different function signatures | Overridden functions have same function signatures |
| Example:  #include <iostream> using **namespace** std; int **calc**(**int**); float **calc**(**float**); int **calc**(**int**, **float**); int **main**() { int a = **10**; float b = **11.2**; calc(a); calc(b); calc(a,b); } int **calc**(**int** a) { int tot=a\*a; cout << " Area of Square A is: " << tot<<endl; } float **calc**(**float** b) { float tot = b\*b; cout << " Area of Square B is: " << tot <<endl; } int **calc**(**int** a, **float** b) { int tot = a\*b; cout << " Area of Square C is: " << tot << endl; } | Example:  #include <iostream> using **namespace** std; class **First** { public: virtual **void** Calc( **int** a , **float** b) { int tot= a\*b; cout << "Square of First class is: "<< tot <<endl; } void Other() { cout<<"Other function in first class"<<endl; } }; class **Second** : **public** First { public: void Calc(**int** a ,**float** b) { float tot = a+b; cout << "Addition of second class is: " <<tot<<endl; } }; int **main**() { int a= **5**; float b=**2.5**; Second s; First &f1 = s; f1.Calc(a,b); s.Other(); } |

If we derive a new class by inheriting features frow two classes derived from the same base class then same feature from the first base is inherited to the finally derived class from two paths (from it’s two parents). → This causes ambiguity in accessing first base class members.



In child class have two copies of base class. There are two duplicate copies of int↑ of base class. One copy through parents and another copy from Parent A. Causes ambiguity in accessing first base class members. This problem is called as Diamond problem.

**Example:**

//This program contains an error and will not compile.

#include<iostream>  
  
class **base**

{

public:

int i;

};

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

{ **public**:

int j;

};

class **derived2**: **public** base

{

public: **int** k;

};

class **derived3**: **public** derived1, **public** derived2

{

public: **int** sum;

};

int **main**()

{

derived3 ob;

ob.i= **10**; // this is ambiguous, which ???

ob.j = **20**;

ob.k = **30**;

ob.sum =ob.i + ob.j +ob.k; // ambiguous here.

cout<<ob.i<<endl;

cout<<ob.j<<" "<<ob.k" ";

cout<<ob.sum;

}

Removing ambiguity

1) Applying Scope Resolution Operator

. 2) Using Virtual base class.

7) What is pure virtual function? Discuss the role of virtual functions in C++ to cause dynamic polymorphism. Show with example how it is different from the compile time polymorphism.

***Soln:***

A **pure virtual function** is a virtual function with no body. Every concrete derived class must override all base-class pure virtual functions and provide concrete implementations of those functions.

The main use of virtual function is **to achieve Runtime Polymorphism**. Runtime polymorphism can be achieved only through a pointer (or reference) of base class type. Also, a base class pointer can point to the objects of base class as well as to the objects of derived class.

**Example**:

( **Run time polymorphism** can be achieve using Pure virtual function)

#include<iostream>

using **namespace** std;

class **B**

{

**public**:

**virtual** **void** s()

{

cout<<" In Base **\n**";

}

};

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

**public**:

**void** s() {

cout<<"In Derived **\n**";

}

};

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

D d; // An object of class D

B \*b= &d; // A pointer of type B\* pointing to d

b->s(); // prints "D::s() called"

**return 0**;

}

And **Complie time polymorphism** can be achieved by method of overloading

**Example**: //program to achieve the complie time polymorphism using the

method overlaoding

#include<iostream>

#include<conio.h>

using **namespace** std;

class **Addition**

{

public:

void sum(**int** a, **int** b)

{

cout<<"a+b:"<<a+b;

}

void sum(**int** a , **int** b, **int** c)

{

cout<<"a+b+c:"<<a+b+c;

}

};

int **main**()

{

Addition obj;

obj.sum(**10**,**20**);

cout<<endl;

obj.sum(**11**,**20**,**30**);

return **0**;

}

8) What are different file access pointer? Write a program to store and retrieve the information of Client (Client\_ID, Account\_ID, name, address and age) in Bank Management system. Also calculate the total number of clients in a bank.

***Soln:***

* The C++ I/O system supports four functions for setting a file pointer to any desired position inside the file or to get the current file pointer.

|  |  |  |
| --- | --- | --- |
| **Function** | **Member of the class** | **Action performed** |
| seekg() | Ifstream | Moves get file pointer to a specific location Moves put file pointer to a specific location |
| seekp() | Ofstream | Moves put file pointer to specific location. |
| tellg() | ifstream | Returns the current position of the get pointer |
| tellp() | Ofstream | Return the current position of the put pointer |

**Source code:**

//program to store and retrieve the information of client (Client\_ID, Accoutn\_ID, name, address, and age) in Bank Management System. Also calculate the total number of clients in a bank

#include<iostream>

#include<conio.h>

using **namespace** std;

class **Bank**

{

char Name[**20**];

int Client\_ID;

int Account\_ID;

char Address[**50**];

int Age;

public:

void **input**()

{

cout<<"**\n** Enter the Name";

cin>>Name;

cout<<"**\n** Enter the Client\_ID";

cin>>Client\_ID;

cout<<"**\n** Enter the Account\_ID";

cin>>Account\_ID;

cout<<"**\n** Enter the Address";

cin>>Address;

cout<<"**\n** Enter the Age";

cin>>Age;

}

void **display**()

{

cout<<"**\n** Name:"<<Name;

cout<<"**\n** Client\_ID:"<<Client\_ID;

cout<<"**\n** Account\_ID:"<<Account\_ID;

cout<<"**\n** Address:"<<Address;

cout<<"**\n** Age:"<<Age;

}

void **add**()

{  
 fstream fin;

Bank B;

fin.open("Bank.txt",ios::app|ios::out|ios::binary);

cout<<"**\n** The Bank Record";

B.input();

fin.write((**char**\*)&B,**sizeof**(B));

fin.close();

}

void **displayAll**()

{

fstream fout;

Bank B;

fout.open("Bank.txt",ios::init|ios::binary);

while(four.read((**char**\*)&B,**sizeof**(B)))

{

B.display();

}

fout.close();

}  
};

int **main**()

{

cout<<"Enter teh detail of persona"<<endl;

Bank B1;

B1.add();

B1.displayAll();

return **0**;

}

**Example:**

9) Explain function template? How do you use function template with multiple template types? Give example.

***Soln:***

* A ***function templates*** work in similar manner as function but with one key difference. A single function template can work on different types at once but, different functions are needed to perform identical task on different data types. If you need to perform identical operations on two or more types of data then, you can use function overloading. But better approach would be to use function templates because you can perform this task by writing less code and code is easier to maintain. A generic function that represents several functions performing same task but on different data types is called function template. For example, a function to add two integer and float numbers requires two functions. One function accept integer types and the other accept float types as parameters even though the functionality is the same. Using a function template, a single function can be used to perform both additions. It avoids unnecessary repetition of code for doing same task on various data types.

**Example:**

//Program to use function template with multiple template types  
#include <iostream>  
#include<conio.h>  
  
template <**typename** T>

T max(T x, T y)

{

**return** (x > y) ? x : y;

}  
  
int main()

{

std::cout << max(**1**, **2**) << '\n'; // will instantiate max(int, int)

std::cout << max(**1.5**, **2.5**) << '\n'; // will instantiate max(double, double)

**return** **0**;

}

10) What is exception and what is mechanism of exception handling in C++? Write a program to illustrate the process of handling multiple exceptions.

***Soln:***

* ***Exceptions*** are run time anomalies or unusual conditions that a program may encounter while executing. E.g., division by zero, access to an array outside its bound, running out of memory or disk space, etc.

The phenomenon of providing means to detect and report an “exceptional circumstances” so that appropriate actions can be taken.

Multiple Exceptional Handling :

It is possible that a program segment has more than one condition to throw an exception. In such cases, we can associate more than one catch statement with a try. (much like the conditions in a switch statement). The first handler that yields a match is executed . It is possible that arguments of several catch statements match the type of an exception . In such case, the first handler that matches the exception type is executed.

Example:

//Program to show multiple exception Handling  
#include<iostream>  
#include<conio.h>  
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**;  
 else **if**(x==**2**) **throw** **1**;  
 cout<<"End of try-block**\n**";  
 }  
 catch (**char** c)  
 {  
 cout<<"Caught a character**\n**";  
 }  
 catch(**int** m)  
 {  
 cout<<"Caught a integer**\n**";  
 }  
 catch(**double** d)  
 {  
 cout<<"Caught a double**\n**";  
 }  
 cout<<"End of try-catch system**\n\n**";  
}  
int **main**()  
{  
 cout<<"Testing Multiple Catches**\n**";  
 cout<<"Z==1**\n**";  
 test(**1**);  
 cout<<"X==0**\n**";  
 test(**0**);  
 cout<<"X==-1**\n**";  
 test(-**1**);  
 cout<<"X==2**\n**";  
 test(**2**);  
 return **0**;  
}

Thank You !!!