DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY AND RESEARCH

Academic Year : 2018-19

**Introduction to Object Oriented Concepts and Design & Principle Of Object-Oriented Programming**

**1. Define to Object Oriented Programming and Explain feature of Object Oriented Programming. How it is different than procedure oriented programming.**

OBJECT ORIENTED PROGRAMMING is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on and set of operations that can access that data.

Different feature of Object Oriented Programming are as follows:

1. OBJECT – Objects are the basic run-time entities in an Object-Object Programming.
2. CLASSES – Objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of Class.
3. ENCAPSULATION – Wrapping up of Data and Functions into a single unit (called class) is known as Encapsulation.
4. DATA ABSTRACTION – It refers to the act of representing essential features without including the background details or explanations.
5. INHERITANCE – It is the process by which objects of one class acquire the properties of objects of another class. Thus less we don’t need to write the code again and again for different classes.
6. POLYMORPHISM – It means ability to take more than one form. It allows us to have more than one function with the same name in a program. It also allows overloading of operators so that an operation can exhibit different behaviors in different instances.

Object Oriented Programming is different than Procedure Oriented Programming in following ways:

* OOP is divided into OBJECTS, while POP is divided into FUNCTIONS.
* OOP follows BOTTOM UP approach, while POP follows TOP DOWN approach.
* OOP has ACCESS SPECIFIERS, while POP doesn’t.
* OOP allows DATA-HIDING, where as POP doesn’t.
* OOP provides ENCAPSULATION, but POP doesn’t.
* OOP allows INHERITENCE, while POP doesn’t.
* OOP supports POLYMORPHISM, but POP doesn’t.

**2. Explain all the major characteristics of Object Oriented Programming with Real World Applications.**

Major Characteristics of Object Oriented Programming are:

1. OBJECT – Object is a the basic unit of Object Oriented Programming. Objects are identified by its unique name. An object represents a particular instance of a class. There can be more than one instance of an object. Each instance of an object can hold its own relevant data. An object is collection of data members and associated member functions also known as methods.
2. CLASSES – Classes are data types based on which objects are created. Objects with similar properties and methods are grouped together to form a class. Thus a Class represents a set of individual objects.

*EXAMPLE –Architecture will have the Blueprints for a House.* Here Blueprint is the Class…The House is the Object. The People living in the house are data stored in Object’s Properties.

1. ENCAPSULATION – It means which binds the data and the code writing operations and methods in single unit (class).

*EXAMPLE – A car is having multiple parts like steering, wheels, engine, etc. which binds together to form a single unit calles Car.* So here, multiple parts of cars encapsulation itself together to form a single object Car.

1. DATA ABSTRACTION – It means to only show important data and details, rest of them are kept hidden.

*EXAMPLE – In Cell phones only Buttons like Home Button, Back Button, etc. are shown. Rest of its small parts is kept hidden.*

1. INHERITENCE – Deriving a new class from existing class, is called Inheritance. Derived class is getting all the features from existing class also incorporating some new features to the sub class.

*EXAMPLE – We inherit genes from our parents and incorporate some of their features in us.*

1. POLYMORPHISM – It means ability to take more than one form that an operation can exhibit different behavior at different instance depend upon the data passed in the operation.

*EXAMPLE – We behave differently in front of our elders and friends*. A single person is behaving differently at different time.

**Introduction to C++**

**3. Write down importance of using Namespace. How we create and use user defined namespace? Explain nesting user defined namespace concepts with small example.**

Namespace allows us to group named entities that otherwise would have global scope into narrower scopes, giving them namespace scope. This allows organizing the elements of programs into different logical scopes referred as names.

User Defined Namespace is created with keyword **namespace** followed by any name of namespace.

*SYNTAX -* ***namespace*** *namespace\_name*

*{*

*\\ statements*

*}*

We can use it as follows:

**namespace** odd

{

Intprint()

{ return 3; }

}

Int main ()

{

cout<<odd::print();

return 0;

}

A namespace can be nested within another namespace for Example:

**namespace** NS1

{

**namespace** NS2

{

Int m = 100;

}

}

To access the varialble**m**, we must qualify the variable with both the enclosing namespace names.

cout<<NS1 :: NS2 :: m;

or alternatively, we can also write

using**namespace**NS1;

cout<< NS2 :: m;

**4. What is the Extraction and Insertion Operator? Write a short note on structure of C++ Program.**

The operator >> is known as Extraction Operator. It is used to take the input from the user. Input operator is used with standard input object **cin.**

*EXAMPLE - int number;*

***cin>>number****;*

The Operator << is known as Insertion operator. It is used to display the output to user. Output operator is used with the standard output object **cout**.

*EXAMPLE – cin>> number;*

***cout<< number****;*

STRUCTURE OF C++ PROGRAM

|  |
| --- |
| Include Files |
| Class Definitions |
| Member Function Definition |
| Main Function Program |

A C++ Program includes various built-in functions which are stored in standard C++ Library. To use all that functions and keywords we need to include all that header files into the program. They are defined with preprocessor *#include*and then the name of header file between *<* and *>*.

Then different classes are defined in class definition section. These Classes contains their own data members and functions.

Then comes the section of member function deinition. Many Times Functions are already defined inside the class, but we can just declare that function inside the class and then function must be defined outside the class, in global section using scope resolution operator. (::)

Then comes the Main function which is the main section of the program, so called starter function, from where the execution of program starts. All the function calling and the other statements that we want to execute are written inside the main function. By Defaut the Main function is of type int, so we need to return an integer value at the end of Main function.

**Tokens and Expression & Control Structure**

**5. Explain C++ Data types and operators in C++ in detail. If we use boolean data type in code then which line must be needed if we want to write “true/false” instead of “1/0” in output screen?**

Data types define the type of data a variable can hold, for example an integer variable can hold integer data, a chracter type variable can hold character data etc. Therefore we can say that datatypes are used to tell the variables the type of data it can store. Every data type requires different amount of memory.

Data Types in C++ are catagorised into three groups:

* BUILT-IN :*int, char, float, double, boolean, void, etc.*
* USER DEFINED : *structure, union, enum*
* DERIVED : *array, function, pointer*

Operators are the foundation of any programming language. Thus the functionality of C++ programming language is incomplete without the use of operators. We can define Operators as symbols that tells computer to perform specific mathematical or logical operations . C++ is rich in built-in operators and provide the following types of Operators –

* ARITHMETIC OPERATORS – These operators are used to perform arithmetic/mathematical operatios on operands. Eg: *+,-,\*,/, etc*
* RELATIONAL OPERATORS – They are used for comparrision of values of two operands. Eg: *<,>,=,!=*
* LOGICAL OPERATORS –They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. Eg: *AND,OR,NOT, etc.*
* BITWISE OPERATORS - They are used to perform bit-level operations on the operands. Eg: *~,^,&*
* ASSIGNMENT OPERATOR – They are used to assign value to a variable. Eg: *=,+=, etc.*
* Other Misc Operators are SIZEOF OPERATOR, CONDITIONAL OPERATOR, etc.

########################################################################

**6. What is Scope Resolution Operator? What are the applications of it?**

Scope Resolution Operator (::) is used to get hidden names due to variable scopes so that you can still use them. The scope resolution operator can be used as both uanry and binary. With the use of it Global Version of Variable can also be accessed from within the inner block.

Applications of Scope Resolution Operator are as follows:

1. To access a global variable when there is a local variable with same name:

#include<iostream>

using namespace std;

intnum;

int main ()

{

intnum = 10;

cout<< "Value of Global num is"<< :: num;

cout<< "Value of local num is" << num;

return 0;

}

1. To Define a function outside a class

#include<iostream>

using namespace std;

class A

{

pulic :

void fun();

};

void A ::fun();

{

cout<<"fun() called";

}

int main()

{

A a;

a.fun();

return 0;

}

1. To access a class's static variable

#include<iostream>

using namespace std;

classtodo

{

staticint a;

public:

staticint b;

voidfunc(int a)

{ cout<<"Variable Of Static a is "<<todo :: a;

cout<<"Variable Of Local a is"<< a }

};

inttodo::a=1;

inttodo::b=2;

int main()

{

todoobj;

int a=3;

obj.func(a);

cout<<"Todo :: b = "<<todo :: b;

return 0;

}

1. In case of multiple inheritance

#include<iostream>

using namespace std;

class A

{

protected :

int x;

public:

A() { x = 10; }

};

class B

{

protected:

int x;

public:

B() { x = 20; }

};

class C : public A, public B

{

public:

void fun()

{

cout<< "A's x is "<< A::x;

cout<< "B's x is "<< B::x;

}

};

int main()

{

C c;

c.fun();

return 0;

}

**7. Explain How to allocate and de-allocate memory dynamically in C++.**

Since C uses malloc() and calloc() function to allocate memory dynamically at run time and uses free()function to free dynamically allocated memory. C++ supports these functions and also has two operators **new** and **delete** that perform the task of allocating and freeing the memory in a better way.

ALLOCATING BLOCK OF MEMORY USING **NEW**

New operator is also used to allocate a block (an array) of memory of data-type.

*pointer-variable =* ***new*** *data-type[size];*

where size(a variable)specifies the number of elements in an array.

*For Eg:int \*p =* ***new****int[10];*

Dynamically allocates memory for 10 integers continuouslyof type int and returns pointer to the first element of the sequence, which is assigned to p(a pointer). p[0] refers to first element, p[1] refers to second element and so on.....

DE-ALLOCATING MEMORY USING **DELETE**

Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided **delete**operator by C++ language.

***delete****pointer-variable;*

Here, pointer-variable is the pointer that points to the data object created by **new.**

To free dynamically allocated array pointer by pointer-variale, use following form of delete:

***delete[]*** *pointer-variable;*

**Functions in C++**

**8. When do we need to use default arguments in a function? Discuss with small program.**

Since, when we create any function we pass the argument in that particular function definition as well as in main function when we call the function. But many times we forget to pass all the argument5s and we does not get the expected output. So at that time default arguments are needed. When we define the function, we give a value to the variable in brackets so that when we forget to pass the values in brackets, compiler will automatically intake the default arguments given by users.

But, there are some rules for default arguments.

* If we want to give default value to an parameter which is at the end the we have to give default values to all the parameters after that parameters.
* Default Arguments must be at least in the list of arguments.

*EXAMPLE - voidmemfun(int a=3, int b=1)*

*{*

*//statements*

*}*

*int main ()*

*{*

*memfun(5);*

*return 0;*

*}*

Here, we can see that we have passed the value of a as 10. We haven't passed the value of b, so in the end we will not get error because we had given default argument, so compiler will automatically take b = 1 as its value and output will be shown.

**9. Explain different use of ‘const’ keyword in C++**

The **const** keyword specifies that a variable's value is constant and tells compiler to prevent the programmer to change in that variable throughout the program.

In C++, we can use the **const** keyword instead of pre-processor directive *#define* to define constant values. Values defined with const are subjected to type checking and can be used i place of constant expressions.

*EXAMPLE1-****const****int x = 10;*

This will declare an integer variable with the constant value of 10.

The const keyword can also be used with the pointer declaration.

*EXAMPLE2 - char\** ***const*** *p = bull;*

A pointer to a variable declared as const can be assigned only to pointer that is also declared as const.

*EXAMPLE3 -* ***const****char\*bull = "constant";*

We can also use pointers to constant data as function parameters to prevent the function from modifying a parameter passed through a pointer.

*EXAMPLE4 - int Class::getSection()* ***const***

*{*

*return Section;*

*}*

Here, 'return Section;' will not modify anything.

**Classes And Objects**

**10. What is Friend Function? What is the difference between Friend Function of a Class and a member function of a class?**

**Friend**Function can access private and protected members of other class in which it is declared as friend. It is sometime useful to allow a particular class to access private members of other class. For Example LinkedList Function can be allowed to access private members of Node:

*class Node*

*{*

*private:*

*int key;*

***friend****void LinkedList();*

*};*

*voidLinkedList()*

*{*

*key = 7;*

*}*

*int main()*

*{*

*int lock;*

*lock = LinkedList();*

*cout<<"Value of key is "<<lock;*

*returm 0;*

*}*

Main Difference between **Friend** Function and **Member** Function is that Friend Function helps to access private and protected data while Normal member Function is not allowed to access private and protected data of different class. Other Differences are -

* A member Function can access all the data members directly in compared to Friend Function.
* When we call a member function we need an object of that class and dot operator. But if global function is declared as a friend function the we have to wrote function name only at the time of calling the function.

**11. When do we declare a member of a class ‘static’? Why? Discuss static member function with example.**

We declare a member of class as **static**when we want to create a variable which should be common to all the object of that particular cars. That means only one copy of static member is created irrespective of number of objects.

*EXAMPLE - class exp*

*{*

*void demo()*

*{*

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

*cout<< count <<"";*

*count ++ ;*

*}*

*};*

*intexp :: count =0;*

*int main()*

*{*

*for(int i=0;i<5;i++)*

*{*

*demo();*

*}*

*reutrn 0;*

*}*

All the static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using scope resolution operator (::) to identify which class it belongs to.

We are allowed to invoke a static member function using the object and the ‘.’ Operator, but it is recommended to invoke the static member using the class name and the scope resolution operator. Static Member Function also does not depend on object of class. Static Member Function are allowed to access inly the static data member or other static member functions, they cannot access the non-static data members or member functions of the class.

**12. What is the difference between C sturcture, C++ sturcture and Class? How class can be specify?**

Since we know that C++ Structure and C++ Class are almost same, the only difference between them is that in Class the default visibility if PRIVATE and in Structure it is PUBLIC.

So the Difference between C Structure and C++ Structure are as follows:

* C Structure can’t contain functions, means only data members are allowed, but C++ Structure and Class both can have functions & data members.
* **struct** keyword is necessary in C to create structure type variable, but it is redundant & not necessary in C++.
* Structure in C can’t have static members, but C++ structure can have **static** members.
* We can’t directly initialize structure data member in C, but in C++ we can do that.
* Structures in C can’t have constructors and destructors but C++ Structures can have both to initialize and delete the object.
* Structures in C do not give facility of data hiding. But in C++, we have that feature with the help of private, public and protected access specifiers. They define scope of any data member.

Class can be specified in this way –

A class is defined in C++ using keyword **class** followed by the name of class. The body of class is defined inside the curly brackets and terminated by semicolon at the end.

***class****Class\_Name*

*{*

*Acces\_specifiers : //* can be private, public or protected

*Data\_Members ; //* Variable to be used

*Member\_Functions() { } //* Methods to access data members

}; // Class ends with a semicolon

**13. Describe the concept of private member function of a class.**

As we know in C++ we can use Private Member Function in Class but cannot call it directly into the main function because it is declared in Private section of class. So, in order to access that private member function we need to Declare and Define a function in Public section which will call that private member function and then we can indirectly access private member function through calling public member function in main function using object of that class.

*EXAMPLE - class X*

*{*

*private:*

*Int p;*

*Void func1()*

*{*

*P=1;*

*}*

*public:*

*void func2()*

*{*

*func1();*

*cout<< ”p : ” << p*

*}*

*}:*

*int main ( )*

*{*

*X obj;*

*obj.func2( ) ;*

*}*

*OUTPUT – 1*

Here to access *funct1( )* we had call private member function *funct1( )* in public member function *func2( )*. So when we call *func2( )* using object of *class X*, it will call private member function *funct1( )*. So in this way we have indirectly called private member function using public member function, because object *obj* cannot directly call private member function *func1( )*.

**Constructor And Destructor**

**14. What is the purpose of copy constructor? Name any two situations in which a copy constructor executes.**

The **Copy Constructor** is a constructor which creates an object by initializing it with an object of same class, which has been created previously. The copy constructor is used to –

* Initialized one object from another of the same type
* Copy an object to pass it as argument to a function.
* Copy an object to return it from a function.

*EXAMPLE - class Point*

*{*

*private:*

*int x,y;*

*public :*

*point (int x1, int y1) { x = x1; y = y1 ;}*

*point (const Point &p2) {x=p2.x ; y = p2.y ;}*

*intgetX ( ) { return x; }*

*intgetY ( ) { return y; }*

*};*

*Int main ( )*

*{*

*Point p1(10,15);*

*Point p2 = p1;*

*cout<< ” p1.x = ”<< p1.getX() << “ , p1.y = ” << p1.getY ( ) ;*

*cout<< ” p2.x = ”<< p2.getX() << “ , p2.y = “ << p2.getY ( ) ;*

*return 0;*

*}*

Two situations in which Copy Constructor can execute are –

1. **When an existing object is assigned an object of its own class**

*MyClassa,b;*

*A = new MyClass( )*

*B = a ;*

Here, when we write *B = A,* the copy constructor is called and the value of object A is assigned to object B

1. **If a function receives as arguments, passed by value, an object of a class**

*void foo( MyClass a1 ) ;*

*foo ( a ) ; //*  copy constructor invoked

**15. What is Virtual Destructor? Discuss with small Examples. Can I overload any Destructor?**

If the destructor in the base class is not made virtual, then an object that might have been declared of type base class and instance of child class would simply call the base class destructor without calling the derived class destructor.

Hence by making the destructor in the base class virtual, we ensure that the derived class destructor gets called before the base class destructor.

*EXAMPLE –*

*class a*

*{*

*public :*

*a( )*

*{*

*cout<< “ Base Constructor ” ;*

*}*

*~ a ( )*

*{*

*cout<< “ Base Constructor ”;*

*}*

*};*

*class b : public a*

*{*

*public :*

*b ( )*

*{*

*cout<< “ Derived Constructor ” ;*

*}*

*~ b ( )*

*{*

*cout<< “ Derived Destructor ” ;*

*}*

*};*

*Int main ( )*

*{*

*A\*obj = new b ;*

*deleteobj;*

*return 0;*

*}*

A Destructor can never be overloaded. An overloaded destructor would mean that the destructor has taken arguments. Since a destructor does not take arguments, it can never be overloaded. Only one destructor per class should be there and it must have void parameter.

**16. What is Dynamic initialization of object? Why it is required? Illustrate with example in C++.**

Dynamic Initialization of object refers to initializing the object at run time i.e. the initial value of an object is to be provided during run time. Dynamic initialization can be achieved using constructors and passing values to the constructors. Thus type of initialization is required to initialize the class variables during run time.

Dynamic Initialization off object is required as:

* It utilizes memory efficiently.
* Various initialization formats can be provided using overloaded constructor.
* It has the flexibility of using different formats of data at run time considering the situation.

*EXAMPLE –*

*class A*

*{*

*int a,b;*

*public:*

*A(int p, int q)*

*{*

*a=p;*

*b=q;*

*}*

*void put()*

*{*

*cout<<”a:”<<a<<” b:”<<b;*

*}*

*};*

*int main()*

*{*

*int c,d;*

*c=10;*

*d=20;*

*A obj(c,d);*

*put();*

*return 0;*

*}*

Here we can see that we passed the value in the main function in the constructor. So the object will be initialized at the run time.

**Operator Overloading**

**17. When should a class overload the assignment operator and define the copy constructor?**

An assignment operator is called when an object is already initialised and then again assigned a new value from another existing object.

*EX:*

*#include<iostream>*

*Using namespace std;*

*Class simple*

*{*

*Private:*

*Char\* data;*

*Public:*

*Simple() { data=Null;}*

*Simple(char\*S);*

*{*

*Strcpy(data,S);*

*}*

*Char getdata()*

*{*

*Return data;*

*}*

*Void setdata(char\*S)*

*{*

*if(data!=NULL)*

*Strcpy(data,S);*

*}*

*};*

*int main()*

*{*

*Simple A(“Mickey Mouse”);*

*Simple B;*

*B=A;*

*Simple(A);*

*cout<<A.getdata()<<endl;*

*cout<<B.getdata()<<endl;*

*cout<<C.getdata()<<endl;*

*A.setdata (“Donald Duck”);*

*cout<<A.getdata()<<endl;*

*cout<<B.getdata()<<endl;*

*cout<<C.getdata()<<endl;*

*return 0;*

*}*

The copy assignment operator often just called the “assignment operator” is a special case of assignment operator where the source(right hand side)and destination(left hand side)are of the same class type.

Copy constructor:

Copy constructor is used to create a copy of an already existing object of a calsstype.It is usually of the from X(X&) where X is the class name.

*Syntax:*

*Classname(classname&object name)*

*Example:*

*#include<iostream>*

*Using namespace std;*

*Class Sample;*

*{*

*Private:*

*Intx,y;*

*Public:*

*Sample(int x , int y)*

*{*

*X=x;*

*Y=y;*

*}*

*Sample(Sample &sam)*

*{*

*X=sam.x;*

*Y=sam.y;*

*}*

*void display()*

*{*

*Cout<<x<<””<<y<<endl;*

*}*

*};*

*main()*

*{*

*Samplea1(10,15);*

*Sample a2=01;*

*cout<<”Normal constructor:”;*

*a1.display();*

*cout<<”Copy constructor:”;*

*a2.display();*

*}*

**18. What is overloading in OOP? What is function overloading and operator overloading in C++? Give suitable example.**

C++ allows you to specify more than one definition for a **function** name or an **operator** in the same scope, which is called **overloading**. When you call an overloaded **function** or **operator**, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

**FUNCTION OVERLOADING -** You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

*EXAMPLE -*

*#include <iostream>*

*using namespace std;*

*void print(int i)*

*{*

*cout << " Here is int " << i << endl;*

*}*

*void print(double  f)*

*{*

*cout << " Here is float " << f << endl;*

*}*

*void print(char const \*c)*

*{*

*cout << " Here is char\* " << c << endl;*

*}*

*int main()*

*{*

*print(10);*

*print(10.10);*

*print("ten");*

*return 0;*

*}*

**OPERATOR OVERLOADING -** In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

*EXAMPLE* **–**

*#include <iostream>*

*using namespace std;*

*class Distance*

*{*

*public:*

*int feet, inch;*

*Distance(int f, int i)*

*{*

*this->feet = f;*

*this->inch = i;*

*}*

*void operator - ( )*

*{*

*feet--;*

*inch--;*

*cout << "\nFeet & Inches(Decrement): " << feet << "'" << inch;*

*}*

*};*

*int main()*

*{*

*Distance d1(8, 9);*

*-d1;*

*return 0;*

*}*

**19. What is a conversion function? How is it created. Explain with example.**.

You can define a member function of a class, called a **conversion function**, that converts from the type of its class to another specified type.

*EXAMPLE -* You can define a member function of a class, called a conversion function, that converts from the type of its class to another specified type

*class Y*

*{*

*int b;*

*public:*

*operator int();*

*};*

*Y::operator int()*

*{*

*return b;*

*}*

*void f(Y obj)*

*{*

*int i = int(obj);*

*int j = (int)obj;*

*int k = i + obj;*

*}*

All three statements in function f(Y) use the conversion function Y::operator int(). Conversion functions have no arguments, and the return type is implicitly the conversion type. Conversion functions can be inherited. You can have virtual conversion functions but not static ones.

**Inheritance**

**20. Explain the difference between overriding and overloading a member function of a base class in a derived class.**

|  |  |
| --- | --- |
| Overloading | Overriding |
| Provides functionality to reduce method name for different arguments. | Provides functionality to override a behaviour which the class have inherited from parent class. |
| Occurs within a single class may also occur in child class. | Occurs in two classes that have child-parent or is a relationship. |
| Must have different argument list. | Must have the same argument list. |
| May have different return types. | Must have the same or variant return type. |
| May have different access specifier. | Must not have a more restrictive access modifier but may have less restrictive access specifier. |
| May throw different exceptions. | Must not throw new or broader checked eception but may throw no error vheckrd exceptions or any unchecked exception. |

Overloading doesn’t work for derived class in C++ programming language. There is no overload resolution between Base and Derived.

*EXAMPLE –*

*#include <iostream>*

*using namespace std;*

*class Base*

*{*

*public:*

*int f(int i)*

*{*

*cout << "f(int): ";*

*return i+3;*

*}*

*};*

*class Derived : public Base*

*{*

*public:*

*double f(double d)*

*{*

*cout << "f(double): ";*

*return d+3.3;*

*}*

*};*

*int main()*

*{*

*Derived\* dp = new Derived;*

*cout << dp->f(3) << '\n';*

*cout << dp->f(3.3) << '\n';*

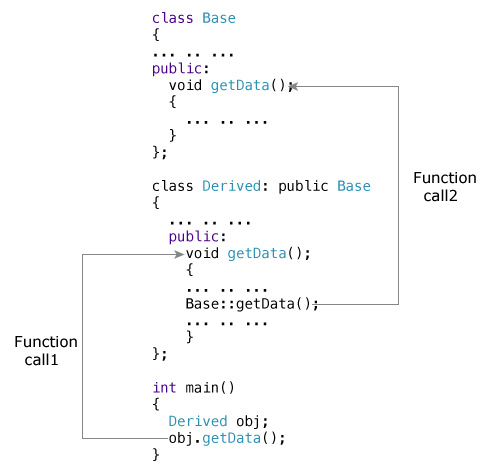
*delete dp;*

*return 0;*

*}*

If you create an object of the derived class and call the member function which exists in both classes (base and derived), the member function of the derived class is invoked and the function of the base class is ignored. This feature in C++ is known as function overriding.

Inheritance allows software developers to derive a new class from the existing class. The derived class inherits features of the base class

To access the overridden function of the base class from the derived class, scope resolution operator :: is used. For example,

**21. State hoe in private, public and protected inheritance ,the member of the base class are inherited by a derived class.**

**PUBLIC MODE:**

If we derive a sub class from a public base class then the public member of the base class will become public in derive class and protected members of the base class will become protected in derived class.

**PROTECTED MODE:**

If we derive a sub class from a protected base class then both public member and protected member of the base class will become protected in derived class.

**PRIVATE MODE:**

If the derive a subclass from a private base class then both public member and protected members of the base class will become private in derived class.

The private members in the base calss cannot directly accessed in the derived class while protected members can be directly accessed.

*Example:*

*Class A*

*{*

*Public:*

*int x;*

*Protected:*

*int y;*

*Private:*

*int z;*

*};*

*class B:public A*

*{*

*//x is public*

*//y is protected*

*//z is not accessible from B*

*};*

*class c:protected A*

*{*

*//x is protected*

*//y is protected*

*//z is not accessible from C*

*};*

*class D:private A // ‘private’ is default for classes*

*{*

*//x is private*

*//y is private*

*//z is not accessible from D*

*};*

**22. What is containership? How does it differ from inheritance?**

Containership also referred to as composition allows a class to contain an object of a different class as a member data.

Ex: class A could contain an object of class b as a member.Here all the public methods defined in b can be executed within the class A. class A becomes the container while class B becomes the container class.This can also be enquired as class A is composed of class B.

|  |  |
| --- | --- |
| Inheritance | Containership |
| Inheritance is the ability for a class to inherit properties and behaviour from a parent class by extending. | Containership is the ability of a class to contain object of different classes as a member data if a class is extended. |
| It inherits all the public and protected properties/behaviour and those behaviour may be overridden by the sub class. | Class is contained in another,the container does not get the ability change or add behaviour to contained. |
| Inheritance represents an “is-a” relationship. | Containership represent a “ has-a” relationship |

**23. What is the ambiguity that arises in multiple inheritance? How it can be overcome. Explain with example.**

The order in which the superclass or parents are given define which property and/or method will be accessible by the conflict causing name; the others will remain hidden.

The subclass must resolve the conflict by providing a property and/or method with the name and by defining how to use the ones from its parents. C++ user resolve the ambiguity with the ancestral class and a scope resolution operator.

The first solution is not very convenient because it introduces implicit consequences, depending on the order in which classes inherit from each other. For the second case, subclasses must explicitly redefine properties which are involved in a naming conflict.

*Example:*

*#include<iostream>*

*Using namespacec std;*

*classLivingThing*

*{*

*protected:*

*void breathe()*

*{*

*cout<<”I am breathing as a living thing”;<<endl;*

*}*

*};*

*class Animal : protected LiningThing*

*{*

*protected:*

*void breath()*

*{*

*cout<<I’m breathing as an animal.”<<endl;*

*}*

*};*

*class Reptile : protected LiningThing*

*{*

*protected:*

*voidGrawl()*

*{*

*cout<<I’m breathing as an reptile.”<<endl;*

*}*

*};*

*class Snake : protected Animal,protected Reptile*

*{*

*protected:*

*void breathe()*

*{*

*cout<<I’m breathing as an snake.”<<endl;*

*}*

*void crawl()*

*{*

*cout<<I’m crawling as an snake.”<<endl;*

*}*

*};*

*main()*

*{*

*Snake s;*

*s.breathe()*

*s.crawl();*

*}*

***Output:***

***I’m breathing as a snake.***

***I’m crawling as a snake.***

**Pointers and Virtual Functions**

**24. Suppose that you have the declaration int \*numptr;. What is the difference between the expressions :**

**\*numptr and &numptr ?**

|  |  |
| --- | --- |
| **&numptr** | **\*numptr** |
| The ampersand & called the address of operator, is a unary operator that returns operator that return the address of its operand. | \*numotr means the content of the memory location to which p point. |

**25. What are the different type of polymorphism? What is the difference between compile time binding and runtime binding? Write down synonyms of the same functionality.**

Polymorphism is a feature of OOPC that allows the object to have differently in different condition.

Two types of polymorphisms:

1. Compile time polymorphism: - This is also known as static or early binding.
2. Runtime polymorphism: - This is also known as dynamic or late binding.

|  |  |
| --- | --- |
| **Compile time binding** | **Runtime binding** |
| The static binding happens at the compile time and early binding. | The dynamic binding happens at the run time binding and late. |
| The function definition and the function call are linked during the compile time. | The function calls are not resolved until runtime so they are bound. |
| It happens when all information needed to call a function is available at the compile time. | It happens when all information for a function call cannot be determined at compile time. |
| It can be achieved using the normal function calls, function overloading and operator overloading. | It can be achieved using the virtual functions. |
| It is flexible since a single function can handle different type of objects at runtime. | This significantly reduce the size of codebase and also makes the source code mode readable. |
| Static binding result in faster. | Dynamic binding result in slower. |

**26. Write down a short note on Abstract class. Is it legal to have an abstract class with all member functions pure virtual?**

Sometimes implementation of all function cannot be provided in a base class because we don’t know the implementation such a class is called abstract class.

A class is abstract if it has at least one pure virtual function.

If we don’t override the pure virtual function in derived class, then derived class also becomes abstract class.

A pure virtual function in c++ is a virtual function for which we don’t have implementation we only declare it.

A pure virtual function is declared by assigning 0 in declaration.

*Syntax:*

*virtualreturn\_typefunction\_name()=0;*

Abstract class cannot be initiated but pointers and references of abstract class type can be created.

Abstract class can have normal function and variables along with a pure virtual function.

Abstract classes are mainly used for up casting so that its derived classes can use its interface.

Classes inheriting on Abstract class must implement all pure virtual functions or else they will become Abstract class.

**27. What are the difference between reference and pointer?**

|  |  |
| --- | --- |
| **Reference** | **Pointers** |
| A reference & is like an alias to an existing variable | A pointer is a variable that holds a memory address. |
| Reference unlike pointer have to be initialized at the point of definition. | Pointer can be initialized at any time. |
| A reference can refer to only one object during its lifetime i.e. a reference cannot be rebound. | A pointer can point to many different objects during its lifetime. |
| Array of reference cannot be created as each reference in the array cannot be initialized at the time of creation. | Array of pointer can be created. |
| No operator is needed to de reference a reference. | One has to use an explicit operator to de reference a pointer. |
| A valid reference must refer to an object. | Pointer need not refer to any object. |
| Reference cannot be NULL. | A pointer even a constant pointer can have a NULL value. |

**28. What is a virtual function? Write rules for virtual function. Explain with example.**

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

They are mainly used to achieve runtime polymorphism.

Functions are declared with a virtual keyboard in base class.

The resolving of function call is done at runtime.

**Rules for Virtual function:**

They must be declared in public section of class.

Virtual functions cannot be static and also cannot be a friend function of another class.

Virtual functions should be accessed using pointer or reference of base class type achieve run time polymorphism.

The prototype of virtual function should be same in base as well as derived class.

They are always defined in base class and overriding in derived class. It is not mandatory for derived class to override in that case base class version of function is used.

*Example:*

*#include<iostream>*

*using namespace std;*

*class base*

*{*

*public:*

*virtual void print()*

*{*

*cout<<”print base class”;*

*}*

*void show()*

*{*

*cout<<”show base class”;*

*}*

*};*

*class derived: public base*

*{*

*public:*

*void print()*

*{*

*cout<<”print derived class”;*

*}*

*void show()*

*{*

*cout<<”show derived class”;*

*}*

*};*

*main()*

*{*

*base \*bptr;*

*derived d;*

*bptr=&d;*

*bptr->print();*

*bptr->show();*

*}*

**Managing Console I/O Operations**

**29. Explain C++ stream classes. Describe briefly the feature of I/O system supported by c++ with reference to c++ stream class.**

The c++ I/O system contains a heirchy of classes that are used to define various streams to deal with both the console and disk file.

These classes are called stream classes.

These classes are declared in iostream class.

*Input stream: - The source stream that providesdata to the program.*

*Output stream: - The destination stream that receive output from the program.*

The data in the input stream can come from keyboard or any other storage device.

The data in the output stream can go to the screen any other storage device.

The c++ I/O Ios provides the asic support for formatted and unformatted I/O operators.

*istream: provides the facilities for formatted and unformatted input.*

*ostream:provides the facilities for formatted output.*

**30. What do you mean by manipulators? How do we create user defined manipulator ? Discuss with example.**

Manipulators are special functions that can be used for getting formatted output.

Before using the manipulators the file iomanip should be included in the program.

Manipulators are classified in two parts:

1. Manipulator operators
2. Manipulator functions

Designing our own Manipulators:

We can design our own Manipulator for certain special purpose.

*Syntax:*

*ostream&manip\_name(ostream&output)*

*{*

*return output;*

*}*

*Example:*

*ostream&unit(ostream&output)*

*{*

*output<<”inches”;*

*return output;*

*}*

*cout<<36<<unit;*

*it will produce “36 inches”.*

**Working with Files**

**31.Explain different file operations. How many ways are there to open file.**

To read and write from a file requires another standard c++ library called fstream which defines three new data types:

1. **ofstream**

This data type represents the output file stream and is used to create files and to write information to files.

1. **ifstream**

This data type represents the input file stream and is used to read information from files.

1. **fstream**

This data type represents the file stream generally and has the capabilities of both ofstream and ifstream.

**Opening a file:**

A file must be opened before you can read from it or write to it

Either ofstream or ifstream object may be used to open a file for writing.

And ifstream object object is used to open a file for reading purpose only.

Two ways to open a file are:

1. Constructor

Syntax: *ofstreamoutfile(“hello.txt”);*

Here outfile is an object of type ofstream which is associated with the file hello.txt and is used to open it.

1. Using the open() function

Syntax:

*ofstreamobj;*

*obj.open(“hello.txt”);*

The only difference is using a function called open to open a file instead of using a constructor of the ofstream class.

If you want to open two files at the same time use two different objects.

**32. Briefly explain error handling functions of file.**

File error handling functions are used with file stream object.

1. **eof()**

Returns true (non-zero) value of end of file is encountered while reading otherwise return false (zero).

1. **fail()**

Return true when an input or output operation has failed.

1. **bad()**

Returns true if an invalid operation is attempted or any unrecoverable error has occurred.

However, if it is false it may be possible to recover from any other error reported and continue operation.

1. **good()**

Return true if no error has occurred.