**OPERATOR OVERLOADING**

* **Operator overloading** is a feature that allows operators to not only work with built in data type but also with user-defined data types like class objects. Like function overloading, it is a form of compile-time polymorphism.

**Operators that cannot be overloaded**  :- scope resolution

(::), Member selection operator(.), ternary operator(?:)

* **IMPORTANT POINTS ABOUT OPERATOR OVERLOADING:**

1. Operator overloading cannot change the operation performed by the operator.’ + ’ operator when overloaded will always perform addition ,not any other operation like subtraction,multiplication etc
2. Assignment operator ‘=’ and address operator ‘&’ need not to be overloaded because they are already overloaded in c++ library.
3. Only existing operators can be overloaded. New operators like \*\*, <>, |& cannot be created.
4. On operators like &&,|| to perform short-circuit evaluation like ‘=&&’ ‘=||’ they need to be overloaded separately.
5. Overloaded operators cannot have default arguments.
6. Overloaded operator must have access to atleast one object of the class in which it is being overloaded.

* While using friend function in operator overloading if you want to get changes done as you are passing to the function then you should use ‘call by reference’ else if you want return the object by taking a temporary object inside the function then you should use ‘call by value’.
* **Pointers to derived class:**

A single base classs pointer can point to objects of base classs as well as to the objects of derived class but reverse of this is not true. // for rest look at the program

* Pointer to a class at a particular level can point to that class as well as to the classes that are below that level in class hierarchy.
* **Run time polymorphism:**
* **Virtual Function** is a function in base class, which is overrided in the derived class, and which tells the compiler to perform **Late Binding(run time polymorphism)** on this function.
* Virtual functions are only for base class so that we can override the functions in base class with same name as in derived class.
* If we don’t use virtual functions for function with same name then whatever be the type of address of object base class pointer is holding ,always the base class version of function will be called,

So in order to call the desired version of function we use virtual function and this is called run time polymorphism.

* Virtual Keyword is used to make a member function of the base class Virtual.
* Function with same name and different implementation is Polymorphism.
* **Binding** means the process used for converting variables and function into machine language addresses.
* Run-time polymorphism also known as **late binding** or **dynamic polymorphism** allows polymorphism(allows objects of different classes to respond to same message but in different form) to take place at run-time rather then at compile time..i.e binding is postponed till runtime. Therefore, **appropriate member function to be called is decided at run-time.** **such function are called virtual functions and corresponding classes are called polymorphic classes.**
* Function overloading and operator overloading are examples of static polymorphism where compiler has all the information about the type and number of operands and can easily invoke the appropriate function. With virtual functions compiler doesn’t knw which function to call in advance so,this decision is postponed till execution of program(run time).
* **Runtime or dynamic polymorphism is only achieved through a pointer to the base class.**
* **RULES FOR VIRTUAL FUNCTIONS:** virtual functions

1. Must be declared in public section
2. Cannot be static
3. Cant be declared as a friend function of another class
4. Are always defined in base class
5. Prototype of virtual functions must be same in base as well as derived class because if fun has same name in base and derived class and has different prototype then it becomes function overloading not function overriding(we are concerned with function overriding).
6. Not mandatory for derived class to redefine virtual functions. If

Derived class has given a specific definition for virtual functions then derived class version will be called otherwise base class version of function is called.

1. A class may have virtual destructor but cannot have a virtual constructor.

|  |  |
| --- | --- |
| **FUNCTION OVERLOADING** | **FUNCTION OVERRIDING** |
| 1.providing multiple definitions of the function by changing signature i.e changing number of parameters, change datatype of parameters ,return type doesn’t play anyrole  2.can be done in parent and its child class  3.comes under compile time polymorphism  4. examples:  >>Void area(int l);  Void area(float l);  >>void area(int a);  Void area(int a, int b); | 1.redefinition of parent class function in its child class with same signature i.e return type and parameters must be same  2.can only be done in and its child class  3.comes under run time polymorphism  4. examples:  Class a{  Public: void display(){cout<<”hello”};  }  Class b:public a{  Public: void display(){cout<<”bye”};  } |

* **PURE VIRTUAL FUNCTION :** since concept of virtual function is to perform polymorphism…I.e. accessing derived class version of function so it becomes insignificant to give definition of a function in base class. We should only declare a virtual function in base class and by making it pure virtual, compiler gets to know that only function declaration is there otherwise it looks for definition outside the class.

**>> To declare virtual function:** virtual void show () = 0;

>> =0 informs only declaration is provided in class without definition.

* When class has a pure virtual function, it is the duty of the derived class to define that function. If the derived class also doesn’t define that function, then derived class must make that function as a pure virtual function.
* A class that has pure virtual function is called **Abstract class.**
* A Abstract class doesn’t have any object;
* Class that are derived from base class and provide implementation of pure virtual functions are called **concrete classes.**
* **ABSTRACT BASE CLASSES:**

1. They have atleast one pure virtual function.
2. Classes inheriting abstract class either must define all pure virtual functions or themselves become abstract class.
3. There are no object of abstract class
4. Pointers of refrence to abstract classes can be created
5. An abstract class can have other data member and member function in addition to the pure virtual functions

* **Advantages of Abstract class:**

1. Provides common and standardized interface for derived classes to operate in similar manner

2. Enables new application to be easily added to pre-defined system.

* **Working of virtual functions (VTABLE AND VPTR)**

|  |
| --- |
| Class A  {  Public:  Void f1(){}  Virtual void f2(){}  Virtual void f3(){}  Virtual void f4(){}  };  Class B : public A  {  Public:  void f1(){}  void f2(){}  void f4(int x){}  } |

|  |  |  |  |
| --- | --- | --- | --- |
| F2() | F3() | F4() |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| F2 | F3 | F4 |  |  |

* If a class contains a virtual function then **compiler itself secretly creates two things:**

1. If object of that class is created then a data member i.e a virtual pointer **(vptr)** is inserted in that class. For each new object created of that class a new virtual pointer which actually is contained in object of the class is inserted as a member of that class.
2. Irrespective of object is created or not a static array of function pointer called **vtable** where each cell contains the address of each virtual function contained in that class. Vtable only conatins address of virtual functions
3. IMP: if we have created virtual function in base class and it is being overrided in child class then we don’t need virtual keyword in child class, functions are automatically considered as virtual functions.
4. Each class containing virtual functions has their own vtable. So A and B have their own vtable.
5. Virtual functions are binded at run-time whereas non-virtual functions are binded at compile time.
6. One thing to note is that since fun f2() is overrided in class B i.e contains the new version of f2() then vtable of B will contain addess of f2() of B class version

>> B class publicly inherits class A, B class doesn’t overrides f3() but is inherited by class B so vtable of B will contain address of parent version of f3()

>>address of f4() in vtable of B will alsoe be of parent class since it is not overrided in B.

>>EB: early binding , LB: late binding

1. During run-time appropriate virtual functions are called by fetching the address of virtual function from vtable
2. Vtable is shared among all the objects of same class
3. **For pure virtual function, NULL pointer is inserted in the vtable of abstract class**
4. Vptr is inherited to all the derived class

Example-1:

|  |
| --- |
| Class A \*p,ob1;  P=&ob1;  Ob1  p  vptr        Contains address of vtable of A    p->f1(); // EB..A  p->f2();// LB..B  p->f3(); //LB..A  p->f4(); //LB..A  p->f4(5); // EB..illegal because class A doesn’t have any such function |

Example-2:

|  |
| --- |
| Class A \*p,  Class B ob2;  P=&ob2;  Ob2  p  vptr        Contains address of vtable of B    p->f1(); // EB..A  p->f2();// LB..B  p->f3(); //LB..A  p->f4(); //LB..A  p->f4(5); // EB..illegal because it is EB and pointer is of A type..to make it work pointer should be of B type |

* **In early binding i.e static binding, type of pointer is considered** like f1() is non-virtual function, so EB of f1() will be done, pointer is of type A so A version of f1() is called. Also, see that in second example we when try calling

p->f4(5) it is illegal because early binding is done, so in early binding pointer type is considered, here pointer type is A but A class doesn’t have any function like f4(5)..so it is illegal.

* In late binding, address of function that pointer is pointing to is considered. Here in example-2 see that pointer is of A type but object is of B type so

>> For fun f2 (), B version of f2 is called since it is overriding f2 of parent class

>>for fun f3 (), parent version is called since it is not being overrided in child class.

>> For fun f4 () also, parent version is called since it is not being overrided in child class.

* Why virtual constructor cannot be created?

ANS: Virtual functions in C++ are an implementation of run-time polymorphism, and they will do function overriding. Generally the virtual keyword is used in C++ when you need dynamic behavior. It will work only when object exists. Whereas constructors are used to create the objects. Constructors will be called at the time of object creation.

So if you create the constructor as virtual, as per the virtual keyword definition, it should have existing object to use, but constructor is used to to create the object, so this case will never exist. So you should not use the constructor as virtual.

So, if we try to declare virtual constructor compiler throw an Error:

>>constructor are used to create object?

* Simply speaking, a constructor does not create an object. It just *initializes* the *state* of the object. It's the new operator which creates the object. Now, let's understand this in little detail.
* When you create an object using statement like this:
* new MyClass();
* The object is first created by the new operator. Just before a reference to the newly created object is returned as the result, the indicated constructor is processed to initialize the new object.

2nd ans :There's a very basic reason: Constructors are effectively static functions, and in C++ no static function can be virtual.

>> If you have much experience with C++, you know all about the difference between static & member functions. Static functions are associated with the CLASS, not the objects (instances), so they don't see a "this" pointer. Only member functions can be virtual, because the vtable- the hidden table of function pointers that makes 'virtual' work- is really a data member of each object.

>> Now, what is the constructor's job? It is in the name- a "T" constructor initializes T objects as they're allocated. This automatically precludes it being a member function! An object has to EXIST before it has a "this" pointer and thus a vtable. That means that even if the language treated constructors as ordinary functions (it doesn't, for related reasons I won't get into), they'd have to be static member functions.

* Virtual Destructor

|  |
| --- |
| Class A{  Int a;  Public:  A(){Cout<<”A constructor”;}  ~ A(){Cout<<”A destructor”;}  }  Class B:public A{  Int b;  Public:  B(){Cout<<”B constructor”;}  ~ B(){Cout<<”B destructor”;}  }  Int main()  {  A \*p=new B();  Delete p;  } |

>>In main after creating the object of B class address of the object is returned and is stored in pointer p of A type.

>>if we don’t write Delete p, then memory that has been allocated wont be released.

>>if we run above program the output will be:

A constructor

B constructor

A destructor

* Now if I remove line delete p, then only constructor would be called, inorder to call destructor we need to release memory by deleting p .
* Now here,Early binding is done i.e at compile time, so at compile time the type of pointer is considered not the content of the pointer.
* Now ,as soon as u create the object of derived class through new derived() the constructor of first base and then derived class is called ,now see to it that early binding is done, so when we call delete p, only the base class destructor will be called..i.e deallocating the memory of base class object as pointer p is of base type. Now see that when we create the object of derived through new,memory for base class+derived class is allocated as derived is inheriting base class but at the time of deletion only base class memory is being deallocating but not derived class, so to do so we do **LATE BINDING** through **VIRTUAL KEYWORD**.
* We declare base class destructor as virtual. By doing so binding is done at runtime ,so content of pointer is considered rather than type of pointer ,since content of pointer is of derived class type i.e derived class address ,so first destructor of derived class is called and then the destructor of base class is called as derived class inherits base class.
* Virtual ~base(){}..after writing this ,output will be

>>base constructor

Derived constructor

Derived destructor

base destructor

* **Pros of inheritance**

1. Code reusability
2. Reusability in turn enhances code reliability as user know they are using pre-written ,debugged and tested code
3. Inheritance reduces development cost
4. Inheritance helps in avoiding code redundancy and supports code extensibility by permitting base class functions to be overridden by derived class function

* **Cons of inheritance**

1. Inherited function often work slower than specialized codes
2. Program requires more space to store its own code and inherited cost
3. Also, if some code is changed in base class ,it would affect all the derived classes i.e. base and derived class are tightly coupled.

* **CONTAINERSHIP**

It is the ablility of one class to contain the object of one or more class as its data member. For example:-

Class one contains the object of class two then class one is called container class and class two is called contained class. Class one object can call public member function of class two.

**Containership is called composition**. It is used when feature of existing class is required in new class.

|  |  |
| --- | --- |
| Inheritance | Containership |
| 1. Enables a class to inherit data and functions by extending it.  2.derived class may override functionality of base class | 1. enables class to contain object of another class as its data member  2. container class cannot override or alter the functionality of contained class |
|  |  |