NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CL 217 – Object Oriented Programing Lab

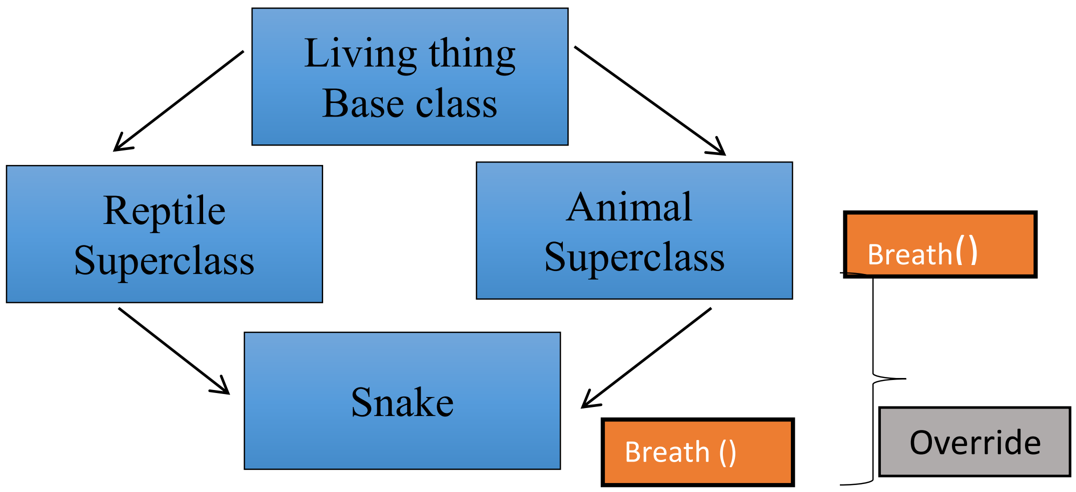
Lab 09 & 10

Outline

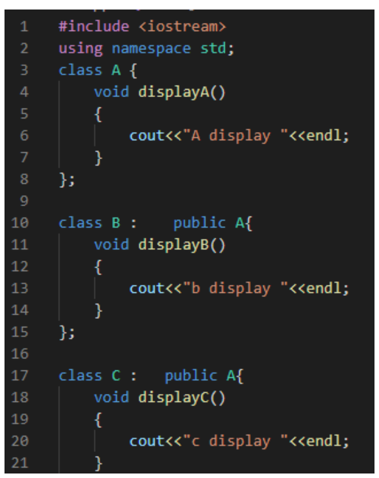
- Diamond Problem
- Runtime Polymorphism
- Friend Function
- Pure virtual Function
- Examples
- Exercise

Multiple Inheritance and its issues

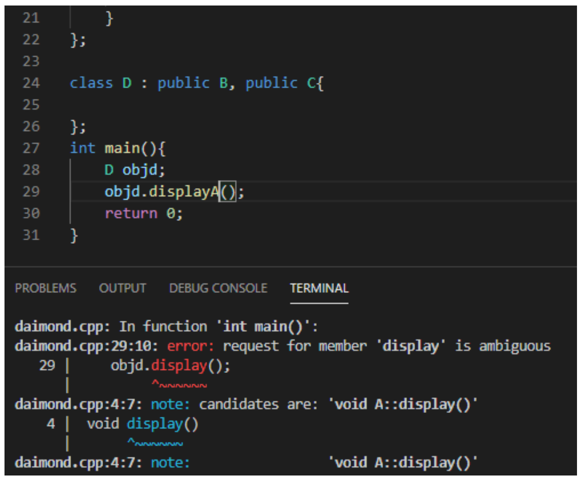
What if?
Reptile class overrides the breathe () method? The **Snake** class would not know which breathe () method to call.”



Diamond problem



```
#include <iostream>
using namespace std;
class A {
    void displayA()
    {
        cout<<"A display "<<endl;
    }
};

class B :    public A{
    void displayB()
    {
        cout<<"b display "<<endl;
    }
};

class C :   public A{
    void displayC()
    {
        cout<<"c display "<<endl;
    }
```



```
    }
};

class D : public B, public C{

};
int main(){
    D objd;
    objd.displayA();
    return 0;
}
```

```
daimond.cpp: In function 'int main()':
daimond.cpp:29:10: error: request for member 'display' is ambiguous
   29 |     objd.display();
      |          ^~~~~~~
daimond.cpp:4:7: note: candidates are: 'void A::display()'
    4 |  void display()
      |       ^~~~~~~
daimond.cpp:4:7: note:                 'void A::display()'
```

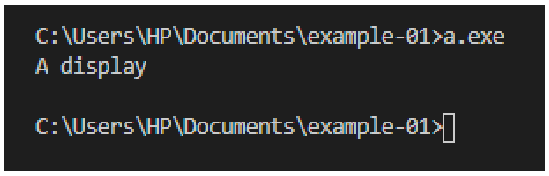
How to Remove Diamond Problem?

- Using virtual keyword-virtual inheritance
  Class b: virtual public A {}

Solution

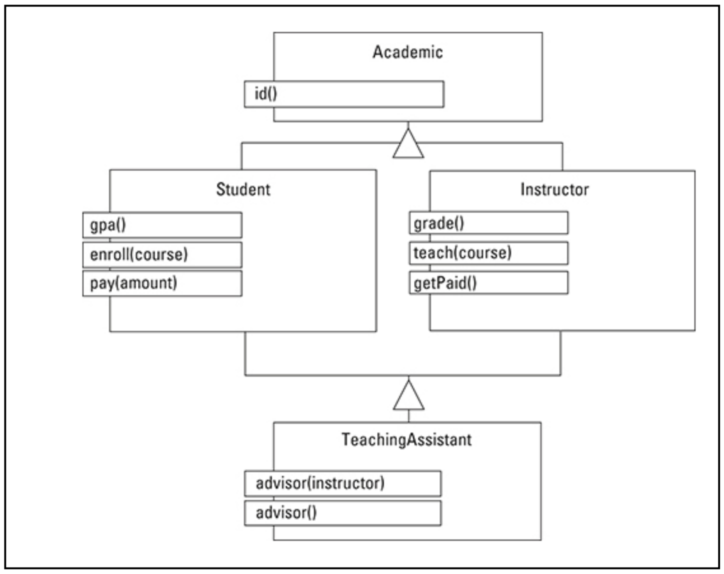
```
#include <iostream>
using namespace std;
class A {
   public:
   void displayA()
   {
      cout<<"A display "<<endl;
   }
};

class B :  virtual  public A{
   void displayB()
   {
      cout<<"b display "<<endl;
   }};
```

```
class C : virtual  public A{
   void displayC()
   {
      cout<<"c display "<<endl;
   }
};

class D : public B, public C{

};
int main(){
   D objd;
   objd.displayA();
   return 0;
}
```



```
C:\Users\HP\Documents\example-01>a.exe
A display

C:\Users\HP\Documents\example-01>
```

Program as Example

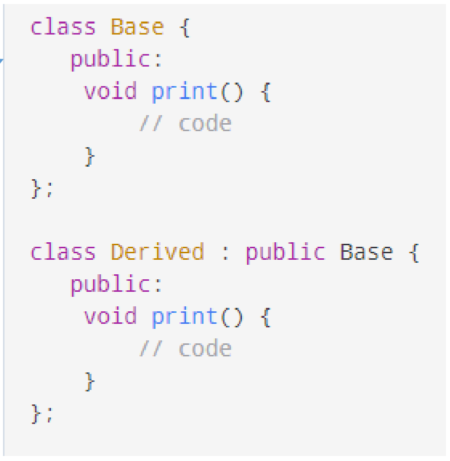


Program as Solution

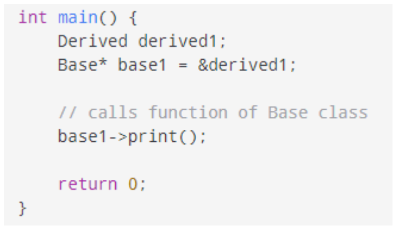
```
class Academic
{
  public:
    int id(int){
         cout<<"calling from teaching Assistant
"<<endl;
        return 0;
    }
};
class Student : virtual public Academic
{
  public:
    double gpa();
    bool  enroll( );
    bool  pay(double dollars);
};
class Instructor : virtual  public Academic
{
  public:
    double grade();
    bool  teach();
    bool  getPaid(double dollars);
};
class TeachingAssistant:public Student, publ
ic Instructor
{
  public:
    Instructor advisor(Instructor&);
    Instructor advisor();
};
int main (){
    TeachingAssistant TA;
    cout << "TA's student id is " << TA.id(2);
// no error now
    return 0;
}
```

Virtual Functions

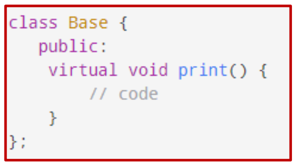
- A virtual function a member function which is declared within base class and is re-defined (Overridden) by derived class.
- Basically, a virtual function is used in the base class in order to ensure that the function is **overridden**. This especially applies to cases where a pointer of base class points to an object of a derived class.



```
class Base {
   public:
    void print() {
        // code
    }
};

class Derived : public Base {
   public:
    void print() {
        // code
    }
};
```



```
int main() {
    Derived derived1;
    Base* base1 = &derived1;

    // calls function of Base class
    base1->print();

    return 0;
}
```

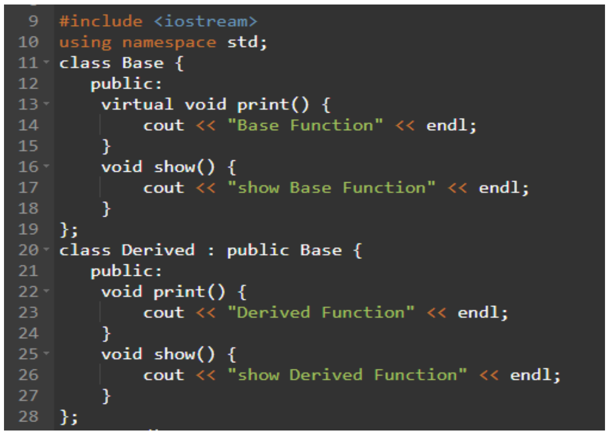


```
class Base {
   public:
    virtual void print() {
        // code
    }
};
```

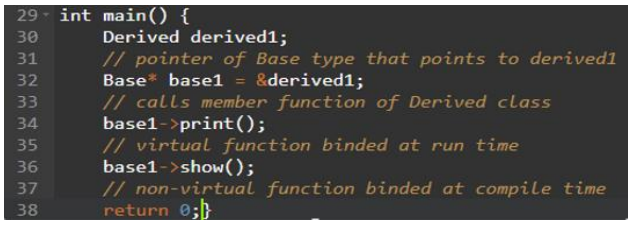
Rules for Virtual Functions

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a **virtual** keyword in base class.
- The resolving of function call is done at Run-time.

Compile-time(early binding) VS run-time(late binding) behavior of Virtual Functions



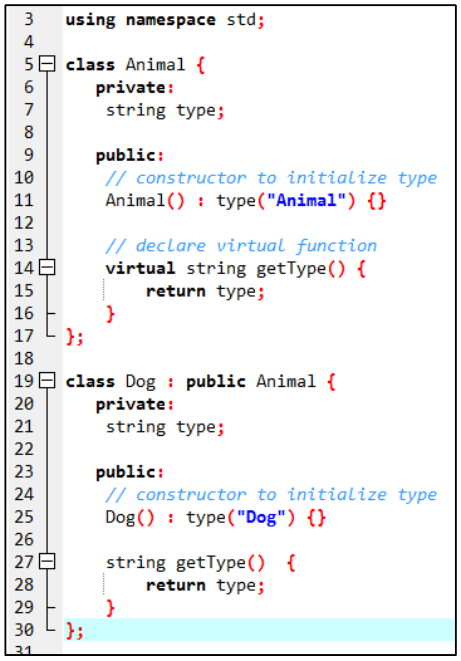
```cpp
#include <iostream>
using namespace std;
class Base {
   public:
    virtual void print() {
        cout << "Base Function" << endl;
    }
    void show() {
        cout << "show Base Function" << endl;
    }
};
class Derived : public Base {
   public:
    void print() {
        cout << "Derived Function" << endl;
    }
    void show() {
        cout << "show Derived Function" << endl;
    }
};
```



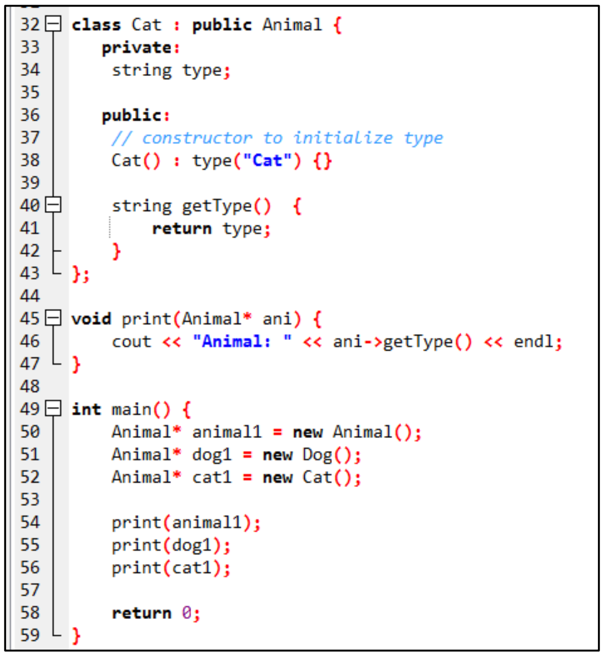
```cpp
int main() {
    Derived derived1;
    // pointer of Base type that points to derived1
    Base* base1 = &derived1;
    // calls member function of Derived class
    base1->print();
    // virtual function binded at run time
    base1->show();
    // non-virtual function binded at compile time
    return 0;}
```

print derived class
show base class

Virtual Functions

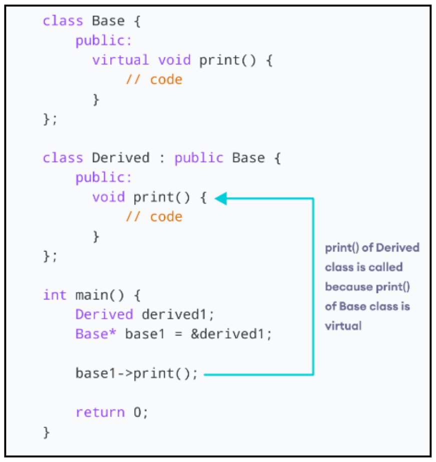


```cpp
using namespace std;

class Animal {
   private:
    string type;

   public:
    // constructor to initialize type
    Animal() : type("Animal") {}

    // declare virtual function
    virtual string getType() {
        return type;
    }
};

class Dog : public Animal {
   private:
    string type;

   public:
    // constructor to initialize type
    Dog() : type("Dog") {}

    string getType()  {
        return type;
    }
};
```



```cpp
class Cat : public Animal {
   private:
    string type;

   public:
    // constructor to initialize type
    Cat() : type("Cat") {}

    string getType()  {
        return type;
    }
};

void print(Animal* ani) {
    cout << "Animal: " << ani->getType() << endl;
}

int main() {
    Animal* animal1 = new Animal();
    Animal* dog1 = new Dog();
    Animal* cat1 = new Cat();

    print(animal1);
    print(dog1);
    print(cat1);

    return 0;
}
```

Run-Time Polymorphism


print() of Derived class is called because print() of Base class is virtual

```cpp
class Base {
    public:
        virtual void print() {
            // code
        }
};

class Derived : public Base {
    public:
        void print() {
            // code
        }
};

int main() {
    Derived derived1;
    Base* base1 = &derived1;

    base1->print();

    return 0;
}
```

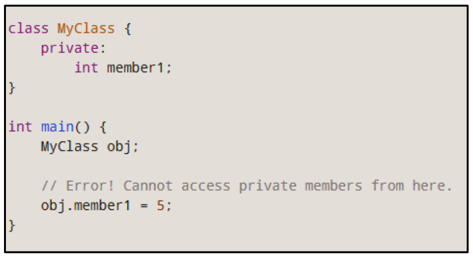
What if-we need to call both function of classes

```
#include <iostream>
using namespace std;
class base{
    public:
            virtual void show(){
                    cout<<"base  is
calling"<<endl;
            }
};
class child: public base{
    public:
            void show(){
                    cout<<"child is
caling"<<endl;
            }
```

```
};
int main(){
    base bobj;
    child cobj;

    base *bptr;

    bptr= &cobj;
    bptr->show();

    bptr = &bobj;
    bptr->show();

    return 0;
}
```

Friend Function

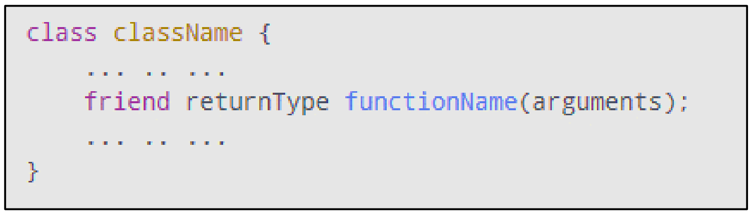
- Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class.
- Similarly, protected members can only be accessed by derived classes and are inaccessible from outside. For example,



```
class MyClass {
    private:
        int member1;
}

int main() {
    MyClass obj;

    // Error! Cannot access private members from here.
    obj.member1 = 5;
}
```

Friend Function

- There is a feature in C++ called friend functions that break this rule and allow us to access member functions from outside the class.
- A friend function can access the private and protected data of a class. We declare a friend function using the friend keyword inside the body of the class.



```
class className {
    ... .. ...
    friend returnType functionName(arguments);
    ... .. ...
}
```

Working of friend Function-example

```
#include <iostream>
using namespace std;
class Distance {
   private:
```

```
    int meter;

    // friend function
    friend int addFive(Distance);
```

```
  public:
     Distance() : meter(0) {}

};
// friend function definition
int addFive(Distance d) {
   //accessing private members from the
friend function
```

```
    d.meter += 5;
    return d.meter;
}
int main() {
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

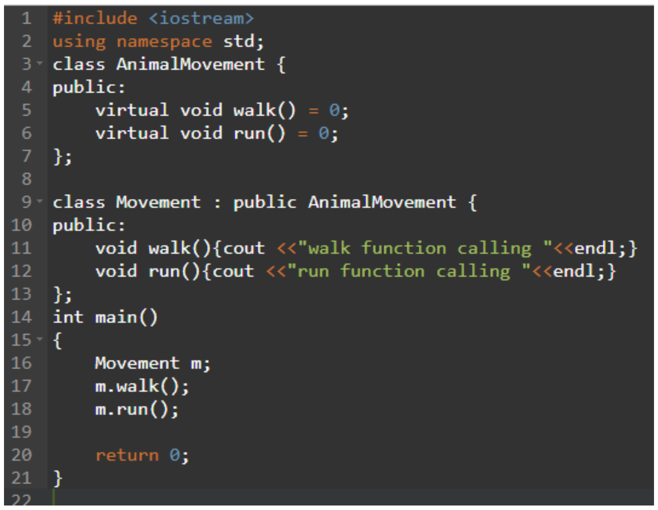
Working of friend Class-example

```
#include <iostream>
using namespace std;
class XYZ {
private:
   char ch='A';
   int num = 11;
public:
friend class ABC;
};
class ABC {
public:
```

```
   void disp(XYZ obj){
     cout<<obj.ch<<endl;
     cout<<obj.num<<endl;
   }
};
int main() {
   ABC obj;
   XYZ obj2;
   obj.disp(obj2);
   return 0;
}
```

Pure Virtual Function

- Pure virtual functions are used to make the class abstract, so that it can't be instantiated, but a child class can override the pure virtual methods to form a concrete class.
- A pure virtual function (or abstract 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. This is a good way to define an interface in C++.



```
#include <iostream>
using namespace std;
class AnimalMovement {
public:
    virtual void walk() = 0;
    virtual void run() = 0;
};

class Movement : public AnimalMovement {
public:
    void walk(){cout <<"walk function calling "<<endl;}
    void run(){cout <<"run function calling "<<endl;}
};
int main()
{
    Movement m;
    m.walk();
    m.run();

    return 0;
}
```

Example

```
#include<iostream>
using namespace std;
class Animal{
public:
  //Pure Virtual Function
  virtual void sound() = 0;
  //Normal member Function
  void sleeping() {
    cout<<"Sleeping";
  }
};
class Dog: public Animal{
public:
  void sound() {
    cout<<"Woof"<<endl;
  }
};
int main(){
  Dog obj;
  obj.sound();
  obj.sleeping();
  return 0;
}
```

Abstract Class

- A class with at least one pure virtual function or abstract function is called abstract class
- Abstract class is used in situation, when we have partial set of implementation of methods in a class.
    - Abstract classes provide an generic base class that can be used by concrete classes to provide an interface and/or implementation.

```
class Shape {
  public:
    virtual int Area() = 0;
// Pure virtual function is declared as follows.
    // Function to set width.
    void setWidth(int w) {
      width = w;
    }
    // Function to set height.
    void setHeight(int h) {
      height = h;
    }

  protected:
    int width;
    int height;
};
// A rectangle is a shape; it inherits shape.
class Rectangle: public Shape {
  public:
    // The implementation for Area is specific
to a rectangle.
    int Area() {
      return (width * height);
    }
};
// A triangle is a shape too; it inherits shape.
class Triangle: public Shape {
  public:
    // Triangle uses the same Area function
but implements it to
    // return the area of a triangle.
    int Area() {
      return (width * height)/2;
    }
};
int main() {
 Rectangle R;
 Triangle T;
 R.setWidth(5);
 R.setHeight(10);
 T.setWidth(20);
 T.setHeight(8);
 cout << "The area of the rectangle is: " <<
R.Area() << endl;
 cout << "The area of the triangle is: " <<
T.Area() << endl;
}
```

Concepts

> ***1) A class is abstract if it has at least one pure virtual function.***
> ***2) We can have pointers and references of abstract class type.***
> ***3) If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.***
> ***4) An abstract class can have constructors.***

Concrete Class

- An abstract class is meant to be used as the base class from which other classes are derived.
- The derived class is expected to provide implementations for the methods that are not implemented in the base class.
- A derived class that implements all the missing functionality is called a concrete class
- A concrete class is an ordinary class which has no purely virtual functions and hence can be instantiated.

Program as Example

```
class Abstract {
private:
string info;
public:
virtual void printContent() = 0;
};
 class Concrete {
private:
string info;
public:
Concrete(string s) : info(s) { }
void printContent() {
cout << "Concrete Object Information\n" <<
info << endl;
}
```

```
};
int main()
{
/*
  * Abstract a;
  * Error : Abstract Instance Creation Failed
  */
string s;

s = "Object Creation Date : 23:26 PM 15 Dec
2013";
Concrete c(s);
c. printContent();
}
```

Example

- ***Write a program to create a class name with A that use the constructors of the abstract class to find the sum of two numbers and display the result with this string "the sum is 9". Base class constructor display the message "hello I am the constructor of base class"***

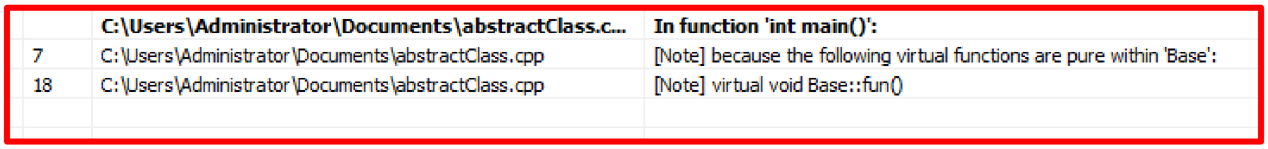
```
#include <iostream>
using namespace std;
class A {
public:
    int a, b, c;
    virtual void test() = 0;
    A(int a, int b)
    { cout << "Hello I am the constructor" <<
endl; } };
class B : public A {
    public: B(int i, int j) : A(a, b)
```

```
    { a = i; b = j; c = a + b; }
    void test()
    { cout << "The sum is = " << c << endl;
} };
int main(void)
{
B obj(4, 5);
obj.test();
return 0;
}
```

We can have pointers and references of abstract class type.

```
using namespace std;
class Base
{
public:
   virtual void fun() = 0;
   int getX() {cout << "get() called"; }
};
// This class inherits from Base and
implements fun()
class Derived: public Base
{
   int y;
```

```
public:
   void fun() { cout << "fun() called"; }
};
int main(void)
{
     Base obj;
     obj.getX();// genrate compile time error
     Base* b;
     Derived d;
     b= &d;
      d.fun();
   return 0;}
```



```
    C:\Users\Administrator\Documents\abstractClass.c...   In function 'int main()':
7   C:\Users\Administrator\Documents\abstractClass.cpp    [Note] because the following virtual functions are pure within 'Base':
18  C:\Users\Administrator\Documents\abstractClass.cpp    [Note] virtual void Base::fun()
```

Interface vs Abstract Classes

Abstract classes:

These are base classes where you have to derive from them and then implement the pure virtual functions.

Interfaces:

These are 'empty' classes where all functions are pure virtual and hence you have to derive and then implement all of the functions.

Activity

1. Write a program to calculate bonus of the employees. The class master derives the information from both admin and account classes which intern derives information from class person. Create base and all derived classes having same member functions called getdata, display data and bonus. Create a base class pointer that capable of accessing data of any class and calculates bonus of the specified employee. (Hint: Use virtual functions)

2. Write a program that has an abstract base class named Quad. This class should have four member data variables (floats) representing side lengths and a pure virtual function Area. It should also have a method for setting the data variables. Derive a class Rectangle from Quad and override the Area method so that it returns the area of the Rectangle. Write a main function that creates a Rectangle and sets the side lengths. Also write a top-level function that will take a parameter of type Quad and return the value of the appropriate Area function.

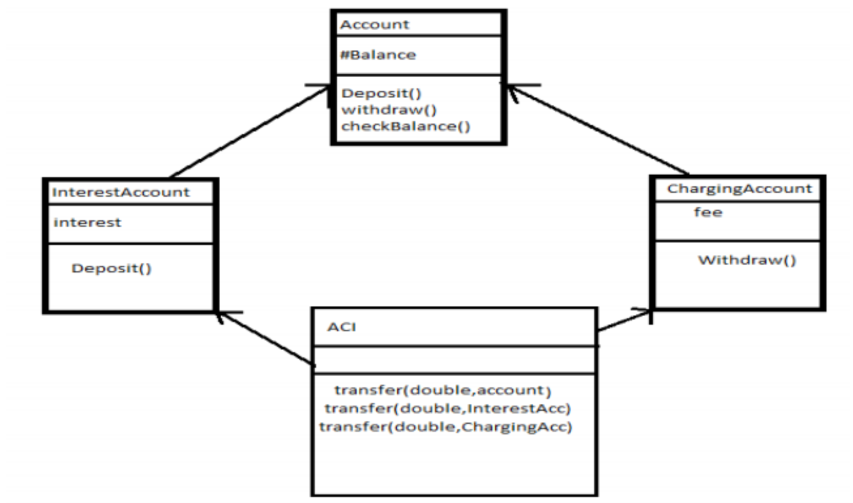
3. The interestaccount class adds interest for every deposit,assume a default of 30%.

    1. The charging account class charges a default fee of $3 for every withdrawal.

    2. Transfer method of aci class takes two parameters,amount to be transfer and object of class in which we have to transfer that amount.

3. Make parameterized constructor, and default constructor to take user input for all data members.

4. Make a driver program to test all functionalities.



4. Design and implement a program that shows the relationship between person, student and professor. Your person class must contain two pure virtual functions named getData() of type void and isOutstanding() of type bool and as well getName() and putName() that will read and print the person name. Class student must consist of function name getData (), which reads the GPA of specific person and isOutstanding() function which returns true if the person GPA is greater than 3.5 else should return false. Class professor should take the respective persons publications in getData() and will return true in Outstanding() if publications are greater than 100 else will return false . Your main function should ask the user either you want to insert the data in professor or student until and unless user so no to add more data.