Example -10 : Single Inheritance

File: **basicEmpInfo.h**

```
#ifndef BASICEMPINFO_H
#define BASICEMPINFO_H
#include <iostream>
using namespace std;

class basicEmpInfo {
public:
   basicEmpInfo();
   basicEmpInfo(const basicEmpInfo& orig);
   virtual ~basicEmpInfo();
   void getBasicEmpInfo(void);
private:
protected:
   char    name[30];
   int     empId;
   char    gender;
};
#endif /* BASICEMPINFO_H */
```

- **basicEmpInfo** is a base class to represent employee object.
- **deptInfo** class is derived from basicEmpInfo
- **loanInfo** class is also derived from basicEmpInfo
- In derived classes we have not override any method of base class

- Similarly **vehicle** is a base class
- **autoVehicle** is derived from vehicle class

- We will create objects of derive classes in stack or heap and access the methods of those objects

File: loanInfo.h

```
#ifndef LOANINFO_H
#define LOANINFO_H

#include "basicEmpInfo.h"
#include <iostream>
using namespace std;

class loanInfo: public basicEmpInfo {
public:
   loanInfo();
   loanInfo(const loanInfo& orig);
   virtual ~loanInfo();
   void getLoanInfo(void);
   void printLoanInfo(void);
private:
protected:
   char    loanDetails[30];
   int     loanAmount;
};
#endif /* LOANINFO_H */
```

File: **deptInfo.h**

```
#ifndef DEPTINFO_H
#define DEPTINFO_H
#include "basicEmpInfo.h"
#include <iostream>
using namespace std;

class deptInfo : public basicEmpInfo {
public:
   deptInfo();
   deptInfo(const deptInfo& orig);
   virtual ~deptInfo();
   void getDeptInfo(void);
   void printDeptInfo(void);
private:
protected:
   char    deptName[30];
   char    assignedWork[30];
   int     time2complete;
};
#endif /* DEPTINFO_H */
```

File : **basicEmpInfo.cpp**

```
#include "basicEmpInfo.h"

basicEmpInfo::basicEmpInfo() {   }
basicEmpInfo::basicEmpInfo(
        const basicEmpInfo& orig) {  }
basicEmpInfo::~basicEmpInfo() {    }

void basicEmpInfo::getBasicEmpInfo(void)
{
   cout << "Enter Name: ";
   cin.getline(name,30);
   cout << "Enter Emp. Id: ";   cin  >> empId;
   cout << "Enter Gender: ";    cin  >> gender;
}
```

File : **loanInfo.cpp**

```
#include "loanInfo.h"

loanInfo::loanInfo() {   }
loanInfo::loanInfo(const loanInfo& orig) {    }
loanInfo::~loanInfo() {    }
void loanInfo::getLoanInfo(void)
{
   cout << "Enter Loan Details: ";
   cin.ignore(1);
   cin.getline(loanDetails,30);
   cout << "Enter loan amount: ";
   cin  >> loanAmount;
}
void loanInfo::printLoanInfo(void)
{
   cout << "Loan Information...:"    << endl;
   cout << "Loan Details: "         << loanDetails
      << endl; //accessing protected data
   cout << "Loan Amount : "         <<
        loanAmount   << endl;
}
```

File : **deptInfo.cpp**

```
#include "deptInfo.h"

deptInfo::deptInfo() { }
deptInfo::deptInfo(const deptInfo& orig) {  }
deptInfo::~deptInfo() {  }

void deptInfo::getDeptInfo(void)
{
   getBasicEmpInfo();     //to get basic info of an
                                employee
   cout << "Enter Department Name: ";     cin.ignore(1);
   cin.getline(deptName,30);
   cout << "Enter assigned work: ";       fflush(stdin);
   cin.getline(assignedWork,30);
   cout << "Enter time in hours to complete work: ";
          cin >> time2complete;
}

void deptInfo::printDeptInfo(void)
{
   cout << "Employee's Information is: "   << endl;
   cout << "Basic Information...:"         << endl;
   cout << "Name: "       << this->name   << endl;
                  //accessing protected data
   cout << "Employee ID: " << this->empId  << endl;
                 //accessing protected data
   cout << "Gender: "     << this->gender << endl <<
        endl;      //accessing protected data

   cout << "Department Information...:"    << endl;
   cout << "Department Name: "   << deptName
          << endl;         //accessing protected data
   cout << "Assigned Work: "             << assignedWork
          << endl;        //accessing protected data
   cout << "Time to complete work: "      <<
         time2complete<< endl;

}
```

File : vehicle.h

```
#ifndef VEHICLE_H
#define VEHICLE_H
```

File : autovehicle.h

```
#ifndef AUTOVEHICLE_H
#define AUTOVEHICLE_H
```

```
#include <iostream>
using namespace std;

class vehicle {
public:
   vehicle();
   vehicle(int input_wheels, float
input_weight);
   vehicle(const vehicle& orig);
   virtual ~vehicle();
   int   get_wheels(void);
   float get_weight(void);
   float wheel_load (void);
private:
protected:
   int  wheels;
   int  weight;
};

#endif /* VEHICLE_H */
```

```
#include "vehicle.h"
class autoVehicle: public vehicle {
public:
   autoVehicle();
   autoVehicle(int input_wheels, float  input_weight,
                  int input_sensors);
   autoVehicle(const autoVehicle& orig);
   virtual ~autoVehicle();
   int   get_sensors(void);
private:
   int  sensors;
};

#endif /* AUTOVEHICLE_H */
```

File : **vehicle.h**

```
#include "vehicle.h"

vehicle::vehicle() { }
vehicle::vehicle(int input_wheels, float
input_weight)
{
   wheels = input_wheels;
   weight = input_weight;{ }
}
vehicle::vehicle(const vehicle& orig) {  }
vehicle::~vehicle() {   }

// get the number of wheels of this vehicle
int  vehicle::get_wheels()  {   return wheels;   }

// return the weight of this vehicle
float vehicle::get_weight() {   return  weight;
}
// return the load on each wheel
float vehicle::wheel_load()  {
          return  (weight/wheels);   }
```

File : **autoVehicle.h**

```
#include "autoVehicle.h"

autoVehicle::autoVehicle() {    }
autoVehicle::autoVehicle(const autoVehicle& orig) {  }

autoVehicle::~autoVehicle() {    }

//   Calling a particular constructor of base class
autoVehicle::autoVehicle(int input_wheels,
          float  input_weight, int input_sensors)
               : vehicle(input_wheels, input_weight)
{
   sensors = input_sensors;
}

int  autoVehicle::get_sensors(void){  return sensors;  }
```

Calling a specific constructor of base class

mainSingleInheritance.cpp

```
#include "deptInfo.h"
#include "loanInfo.h"
#include "vehicle.h"
#include "autoVehicle.h"

int mainSingleInheritance()
{
   //  Example - Inheritance
   //  Accept Employee and Department Information
   deptInfo objD;                        objD.getDeptInfo();

   //Accept loan information
   loanInfo objL;                       objL.getLoanInfo();

   //  Print all informationobjL
   objD.printDeptInfo();      cout << endl << endl ;
   objL.printLoanInfo();

   //  Vehicle Class Example Inheritance
   //  Four objects initiated, two in heap and two in stack

   vehicle *car = new vehicle(4,3000.0);
   vehicle motorcycle = vehicle(2,900.0);   //  When using non-default constructor
   vehicle truck = vehicle(20,30000.0);
   vehicle *sedan_car = new vehicle(4,3000.0);

   cout << endl << endl ;
   // display the data
   cout<<"The car has " << car->get_wheels() << " tires.\n";
   cout<<"Truck has load " << truck.wheel_load() <<" kg per tire.\n";
   cout<<"Motorcycle weight is  " << motorcycle.get_weight() <<" kg.\n";
   cout<<"Weight of sedan car is " << sedan_car->get_weight() <<" kg, and has "
                        << sedan_car->get_wheels() << " tires.\n";

   autoVehicle *smartCar = new autoVehicle(4,3000.0,10);
   cout << endl << endl ;
   cout<<"The smart car has " << smartCar->get_wheels() << " tires and has "
                     << smartCar->get_sensors() << " sensors. \n";
   delete car;
   delete sedan_car;
   delete smartCar;
}
```

Example -12 : Multiple Inheritance (Diamond Problem)

File : **Animal.h**

```
#ifndef ANIMAL_H
#define ANIMAL_H

#include <iostream>
using namespace std;

class Animal {
public:
   Animal();
   Animal(const Animal& orig);
   virtual ~Animal();
   int getAge();
   void setAge(int input_age);
   int getWeight();
   void setWeight(int input_weight );
   void walk();
   virtual void  look();
private:
   int age;
   int weight;
};
```

File : **Lion.h**

```
#ifndef LION_H
#define LION_H

#include "Animal.h"        #include <iostream>
using namespace std;

class Lion : virtual public Animal  {
public:
   Lion();
   Lion(const Lion& orig);
   virtual ~Lion();
   void look();   // If we want to override a method
                  of base class we need to declare in .h file
};

#endif /* LION_H */
```

File : **Tiger.h**

```
#ifndef TIGER_H
#define TIGER_H

#include "Animal.h"          #include <iostream>
using namespace std;

class Tiger : virtual public Animal{
public:
   Tiger();
   Tiger(const Tiger& orig);
   virtual ~Tiger();
   void look();    // If we want to override a method
                   of base class we need to declare in .h file
};
#endif /* TIGER_H */
```

```
#ifndef LIGER_H
#define LIGER_H

#include <iostream>
#include "Lion.h"        #include "Tiger.h"
using namespace std;

class Liger : public Tiger , public Lion {
```

```
public:
   Liger();
   Liger(const Liger& orig);
   virtual ~Liger();
   void look();    // If we want to override a method
           of base class we need to declare in .h file
};
#endif /* LIGER_H */
```

File : **Animal.cpp**

```
#include "Animal.h"

Animal::Animal() {  cout << "Animal constructor "<<endl;  }
Animal::Animal(const Animal& orig) {  }
Animal::~Animal() { }

int Animal::getAge()    { return age;  }
void Animal::setAge(int input_age)  { age = input_age;  }
int Animal::getWeight()  {  return weight;  } ;
void Animal::setWeight(int input_weight)   {  weight = input_weight;  }
void Animal::walk()  {  cout <<"animal walks"<<endl;   }
void Animal::look()  {  cout <<"looks like an animal" << endl; }
```

File : **Lion.cpp**

```
#include "Lion.h"

Lion::Lion() { cout <<"constructor of Lion" <<endl; }
Lion::Lion(const Lion& orig) {  }
Lion::~Lion() { }
void Lion::look()  {  cout <<"looks like a king of jungle "  << endl;  }
```

Overridden method

File : **Tiger.cpp**

```
#include "Tiger.h"

Tiger::Tiger() {  cout <<"constructor of Tiger " <<endl; }
Tiger::Tiger(const Tiger& orig) {}
Tiger::~Tiger() { }
void Tiger::look()   {  cout << "looks like a big cat " <<endl;  }
```

File : **Liger.h**

```
#include "Liger.h"

Liger::Liger() {  cout <<"constructor of Liger" <<endl;   }
Liger::Liger(const Liger& orig) {  }
```

```
Liger::~Liger() {  }
void Liger::look()   {  cout << "looks like half Lion- half Tiger"  << endl; }
```

File : **mainMultiInheritance.cpp**

```
#include "Lion.h"          #include "Tiger.h"
#include "Animal.h"        #include "Liger.h"

//  If we do not use virtual base class we will get following error
//  mainMultiInheritance.cpp:9:10: error: request for member 'walk' is ambiguous

int mainMultiInheritance()
{
   Liger anil;
   anil.setAge(10);
   anil.setWeight(1500);

   cout << "Liger Anil's Age is " << anil.getAge()  << " and weight is " << anil.getWeight() << endl;
   anil.walk();   anil.look();

   //  As the look method is overridden we can call specific base class method using following syntax
   anil.Tiger::look();
   anil.Lion::look();

   return 0;
}
```

Example -13 : Polymorphism

File : **Employee.h**

```
#include <iostream>
#include <string>
using namespace std;

class Employee {
 public:
   Employee();
   Employee(const Employee& orig);
   Employee(string theName, float thePayRate);
   virtual ~Employee();
      //  Without virtual destructor warning “Class
         Employee has virtual functions but non-
         virtual destructor" appears
   string getName() const;
```

File : **Manager.h**

```
#include "Employee.h"

class Manager : public Employee {
public:
   Manager();
   Manager(const Manager& orig);
   Manager(string theName, float thePayRate,
                  bool isSalaried);
   virtual ~Manager();
   bool getSalaried() const;
   // The pay() method is overridden.
      If the manager is salaried, payRate is the
      fixed rate for the pay period; otherwise,
       it represents an hourly rate, just like
```

```
   float  getPayRate() const;
   virtual float  pay(float hoursWorked) const;
   //    Calling  virtual  method  pay  within  this
method
   void printPay(const Employee &empl,
                 float hoursWorked) const;
 protected:
    string name;
    float payRate;
};
```

```
       a regular employee.
   float pay(float hoursWorked) const;
  protected:
     bool salaried;
private:

};
```

File : **Employee.cpp**

```
#include "Employee.h"

Employee::Employee() {  }
Employee::Employee(const Employee& orig) {   }
Employee::Employee(string theName,
                   float thePayRate)  {
   name = theName;    payRate = thePayRate;
}
Employee::~Employee() {   }
string Employee::getName() const    {
       return name;
}
float Employee::getPayRate() const  {
       return payRate;
}
float Employee::pay(float hoursWorked) const
{
     return hoursWorked * payRate;
}

//  If the pay() method is declared virtual, the
   function can be written much simpler way
   Here calling object reference is send as
   parameter and virtual functions behave
   polymorphically with base class pointers!

void Employee::printPay(const Employee &empl,
                float hoursWorked) const
{
       cout << "Pay: " << empl.pay(hoursWorked)
       << endl;
}
```

File : **Manager.cpp**

```
#include "Manager.h"

Manager::Manager() {  }
Manager::Manager(const Manager& orig) {   }
Manager::Manager(string theName,
        float thePayRate,  bool isSalaried)
        : Employee(theName, thePayRate)   {
   salaried = isSalaried;
}
Manager::~Manager() { }
bool Manager::getSalaried() const  {
      return salaried;
}
float Manager::pay(float hoursWorked) const
{
     if (salaried)    return payRate;
     else     return Employee::pay(hoursWorked);
}
```

mainPolyEmployee.cpp

```
#include "Manager.h"

int mainPolyEmployee()
{
   string mgrType;    float payment;        Employee *emplP;   // Base class pointer
   // Print out name and pay (based on 40 hours work).
   Employee empl("Kamal Podder", 25.0);
   emplP = &empl;     // Point to an Employee

   cout << "Name : " << emplP->getName() << " Rate per hr : " << emplP->getPayRate() <<endl;
   payment = emplP->pay(40.0);     cout << " Pay for 40 hr : " << payment << endl;

   Manager mgr("Subhasis Majumder", 50000, true);
   emplP = &mgr;     // Point to the Manager

   //  As getSalaried() method is not defined in base class we will not able to call it in this way
   //  We have to use the object directly
   //    if (emplP->getSalaried() == true)    mgrType = "Salaried -50000";
   //    else                          mgrType = "Hourly basis";

   if (mgr.getSalaried() == true)    mgrType = "Salaried -50000";
   else                         mgrType = "Hourly basis";

   cout << "\nName : " << emplP->getName() << " ( " << mgrType << " ) \n";
   payment = emplP->pay(40.0);       cout << " Pay for 40 hr : " << payment << endl;

   cout << "\n\n    1. Calling virtual method pay using printPay method of Employee\n";

   emplP->printPay(empl, 40.0);
   emplP->printPay(mgr, 40.0);

   cout << "\n\n    2. Calling virtual method pay using printPay method of Employee passing "
              << " base class pointer as parameter \n";
   emplP = &empl;     emplP->printPay(*emplP, 40.0);
   emplP = &mgr;      emplP->printPay(*emplP, 40.0);

}
```

File : **Polygon.h**	File : **Rectangle.h**
#include <iostream> using namespace std;	#include "Polygon.h"

```
class Polygon {
public:
   Polygon();
   Polygon(const Polygon& orig);
   void set_values (int a, int b);
   virtual int area (void) =0;   // Indicates Pure
virtual function by appending =0
          // instead of specifying an implementation
             for the function.
   virtual ~Polygon();
   void printarea (char typ);
protected:
   int width, height;
};
```

```
class Rectangle : public Polygon  {
public:
   Rectangle();
   Rectangle(const Rectangle& orig);
   virtual ~Rectangle();
   int area (void);
};
```

File : **Triangle.h**

```
#include "Polygon.h"
class Triangle : public Polygon {
public:
   Triangle();
   Triangle(const Triangle& orig);
   virtual ~Triangle();
   int area (void);
};
```

File : **Polygon.cpp**

```
Polygon::Polygon() {  }
Polygon::Polygon(const Polygon& orig) {   }

Polygon::~Polygon() {  }
void Polygon::set_values (int a, int b) {  width=a;  height=b; }
void Polygon::printarea (char typ)  {
   if (typ == 'T' )   cout << "Area of Traingle : " << this->area() << endl;
   else               cout << "Area of Rectangle : " << this->area() << endl;
}
```

File : **Rectangle.cpp**

```
#include "Rectangle.h"

Rectangle::Rectangle() {   }

Rectangle::Rectangle(const Rectangle& orig) {  }

Rectangle::~Rectangle() {  }
int Rectangle::area (void)    {
   return (width * height);
}
```

File : **Triangle.cpp**

```
#include "Triangle.h"

Triangle::Triangle() {  }

Triangle::Triangle(const Triangle& orig) {  }

Triangle::~Triangle() {  }
int Triangle::area (void) {
   return (width * height / 2);
}
```

File : **mainPolyPolygon.cpp**

```
#include "Triangle.h"
#include "Rectangle.h"

int mainPolyPolygon()  {
   //  Objects are declared being of type pointer to CPolygon but the objects
   //  dynamically allocated have been declared having the derived class type directly.

     Polygon * ppoly1 = new Rectangle;
     Polygon * ppoly2 = new Triangle;
     ppoly1->set_values (4,5);
     ppoly2->set_values (4,5);
     ppoly1->printarea('R');
     ppoly2->printarea('T');

     delete ppoly1;
     delete ppoly2;
     return 0;
}
```

Example -14 : Operator overloading

- **Overloading postfix and prefix ++ operator using a Time object**
- **Overloading + operator to add two boxes.**

File : **Time.h**

```
#include <iostream>
using namespace std;

class Time {
public:
   Time();
   Time(const Time& orig);
   Time(int h, int m);
   virtual ~Time();
   void displayTime();
   Time operator++ ();       // Overloaded prefix ++   operator
   Time operator++( int );   // Overloaded postfix  ++ operator
private:
   int hours;       int minutes;         // 0 to 59
};
```

- We will overload ++ operator to increment minutes by one.
- Postfix ++ operator signature uses int as argument to indicate it is postfix.
- Postfix operation will return the original object and then increment.

File : **Time.cpp**

```
#include "Time.h"

Time::Time()    { hours = 0; minutes = 0;  }
//    Copy constructor implementation
Time::Time(const Time& orig)     { this->hours = orig.hours;  this->minutes = orig.minutes;   }
Time::Time(int h, int m)  {  hours = h;   minutes = m;   }
Time::~Time() {   }

// Method to display time
void Time::displayTime()    {    cout << "Hour : " << hours << " Minute :" << minutes <<endl; }

// Overloaded prefix ++ operator
Time Time::operator++ ()  {
   ++minutes;   // increment this object
   if(minutes >= 60)   {  ++hours;    minutes -= 60;  }
   return Time(hours, minutes);
}
// Overloaded postfix ++ operator
Time Time::operator++( int )  {
   Time T(hours, minutes);    // Save the orignal value
   ++minutes;       // Increment this object
   if(minutes >= 60)   {  ++hours;    minutes -= 60;  }
   // Return old original value
   return T;
}
```

File : **Box.h**

```
#include <iostream>
using namespace std;

class Box {
public:
   Box();
   Box(const Box& orig);
   virtual ~Box();
   double getVolume(void);
   void   setLength( double len );
   void   setBreadth( double bre );
   void   setHeight( double hei );

   double getLength() const;  //  It means that the method do not modify member variable
   double getBreadth()const;
   double getHeight() const;
```

- The operator+ method uses const keyword in parameter to indicate the box passed as parameter will not be changed inside the function.
- Inside this method we have to use all getter methods to access the properties of the box object passed as parameter. So we need to use const in all getter methods to indicate that these methods will not change the object. Otherwise there will be a compilation error.

```
   // Overload + operator to add two Box objects.
   Box operator+(const Box& b);
private:
   double length;       // Length of a box
   double breadth;      // Breadth of a box
   double height;       // Height of a box
};
```

File : **Box.cpp**

```
#include "Box.h"
Box::Box() {  }
Box::Box(const Box& orig) {   }

Box::~Box() {  }

double Box::getVolume(void)          { return length * breadth * height; }
void   Box::setLength( double len )   { length = len;  }
void   Box::setBreadth( double bre )  { breadth = bre; }
void   Box::setHeight( double hei )   { height = hei;  }

double Box::getLength()  const  { return length;  }
double Box::getBreadth() const  { return breadth; }
double Box::getHeight()  const  { return height;  }

// Overload + operator to add two Box objects.
// As all the getter methods use const we can use these methods for getting member variable of the
    object b which is defined as const
Box Box::operator+(const Box& b)    {
   Box box;
   box.setLength( this->getLength() + b.getLength());
   box.setBreadth( this->getBreadth() + b.getBreadth());
   box.setHeight( this->getHeight() + b.getHeight());
   return box;
}
```

File: **mainOverloadOperator.cpp**

```
#include "Time.h"
#include "Box.h"

int  mainOverloadOperator() {
   cout <<  "========== Overloaded ++ ============" << endl;
   Time t1(3,59);   Time t2(10, 40);
   cout << "Before Prefix operation : ";   t1.displayTime();
```

```
    ++t1;    //  Prefix addition
    Time temp = t1;    //  Another object temp is created using copy constructor
    cout << "After Prefix addition : ";    temp.displayTime();

    cout << "Before Postfix addition : ";   t2.displayTime();
    Time t3 = t2++;
    cout << "After Postfix addition return value : ";            t3.displayTime();
    cout << "After postfix addition incremented value : ";   t2.displayTime();

    cout <<  "========== Overloaded + ============" << endl;

    // Declare Box1, Box2, Box3  of type Box
    Box Box1;    Box Box2;   Box Box3;

    double volume = 0.0;    // To Store the volume of a box here
    // box 1 specification
    Box1.setLength(6.0);   Box1.setBreadth(7.0);   Box1.setHeight(5.0);

    // box 2 specification
    Box2.setLength(12.0);  Box2.setBreadth(13.0);  Box2.setHeight(10.0);

    // Volume of box 1
    volume = Box1.getVolume();
    cout << "Volume of Box1 ( " << Box1.getLength() << "X" << Box1.getBreadth() << "X" <<
                Box1.getHeight() << " ) : " << volume <<endl;
    // Volume of box 2
    volume = Box2.getVolume();
    cout << "Volume of Box2 ( " << Box2.getLength() << "X" << Box2.getBreadth() << "X" <<
        Box2.getHeight() << " ) : " << volume <<endl;

    // Add two Box objects (+ operator overloading )
    Box3 = Box1 + Box2;
    // volume of box 3
    volume = Box3.getVolume();
    cout << "Volume of Box1 + Box2 ( " << Box3.getLength() << "X" << Box3.getBreadth() << "X" <<
                Box3.getHeight() << " ) : " << volume <<endl;
    return 0;
    //   Operator overloading function can be called either implicitly using the operator, or
         explicitly using the function name:    c = a + b;      c = a.operator+ (b);    Both expressions are
         equivalent.

}
```