C++ Inheritance

Dr. Md. Humayun Kabir
CSE Department, BUET

C++ Inheritance

- Inheritance allows one object to inherit member variables and/or member functions from another object.
- Inherited object is called base object
- Inheriting object is called derived object
- Helps programmer to write reusable code
- Helps programmer to write compact code

C++ Inheritance

- Inheritance starts with defining the base class first.

```
class base-class-name {
        .......
};
```

- Derived class is then defined using the base class
- The general form of deriving a class from another class is as follows:

```
class derived-class-name: access base-class-name {
        .........
};
```

- Access can be either **private** or **public** or **protected**
- Default access is **private** for derived class**, public** for derived structure

C++ Inheritance: Base Class

```
#include <iostream>
using namespace std;
class CRectangle {
    int width, length;
    public:
        void set_width (int w) {width=w;}
        void set_length (int l){length=l;}
        int area () {return (width*length);}
};
```

C++ Inheritance: Derived Class

```
class Box: public CRectangle {
    int height;
    public:
        void set_height (int h){height=h;}
        int volume () {return area()*height);}
    /*private members of the base class cannot be accessed by the derived class */
        //int volume () {return width*length*height);}
};
```

C++ Inheritance: Base and Derived Objects

```
int main(){
      CRectangle rect;
      Box box;
      rect.set_width(3);
      rect.set_length(4);
      box.set_width(3);    //inherited
      box.set_length(4); //inherited
      box.set_height(5);
      cout<<“Rectangle area: “<<rect.area()<<endl;
      cout<<“Box base area: “<<box.area()<<endl;     //inherited
      cout<<“Box volume: “<<box.volume()<<endl;
      return 0;
}
```

C++ Inheritance: Derived Class

```
class Box: private CRectangle {
    int height;
    public:
        void set_height (int h){height=h;}
        int volume () {return area()*height);}
    };
```

C++ Inheritance: Base and Derived Objects

```
int main(){
    CRectangle rect;
    Box box;
    rect.set_width(3);
    rect.set_length(4);
    //box.set_width(3);
    //box.set_length(4);
    box.set_height(5);
    cout<<"Rectangle area: "<<rect.area()<<endl;  //it's public to CRectangle object
    /* function area() is private to Box object cannot be accessed outside Box
    object */
    //cout<<"Box base area: "<<box.area()<<endl;
    cout<<"Box volume: "<<box.volume()<<endl; //Will not produce expected result
    return 0;
}
```

C++ Inheritance: Base Class

```
#include <iostream>
using namespace std;
class CRectangle {
    protected:
    int width, length;
    public:
        void set_width (int w) {width=w;}
        void set_length (int l){length=l;}
        int area () {return (width*length);}
};
```

C++ Inheritance: Derived Class

```
class Box: public CRectangle {
    int height;
    public:
        void set_height (int h){height=h;}
    /*protected members of the base class can be accessed by the
    derived class */
         int volume () {return width*length*height);}
};
```

C++ Inheritance: Base and Derived Objects

```
int main(){
    CRectangle rect;
    Box box;
    rect.set_width(3);
    rect.set_length(4);
    box.set_width(3);
    box.set_length(4);
    box.set_height(5);
    cout<<“Rectangle area: “<<rect.area()<<endl;
    cout<<“Box base area: “<<box.area()<<endl;
    cout<<“Box volume: “<<box.volume()<<endl;
    return 0;
}
```

C++ Inheritance with Constructor and Destructor

```
#include <iostream>
using namespace std;
class CRectangle {
      int *width, *length;
      public:
            CRectangle ();
            CRectangle (int,int);
            ~CRectangle ();
            int area () {return (*width * *length);}
};
CRectangle::CRectangle () {
      width = new int;
      length = new int;
      *width = 0;
      *length = 0;
}
```

C++ Inheritance with Constructor and Destructor

```
CRectangle::CRectangle (int a, int b) {
     width = new int;
    length = new int;
    *width = a;
    *length = b;
}

CRectangle::~CRectangle () {
    delete width;
    delete length;
}
```

C++ Inheritance with Constructor and Destructor

```
class Box:public CRectangle{
            int *height;
            public:
              Box();
              Box(int, int, int);
              ~Box();
              int volume(){ return area()*(*height);}
};
Box::Box () {
    height=new int;
    height=0;
}
```

C++ Inheritance with Constructor and Destructor

```
Box::Box (int w, int l, int h):CRectangle(w,l) {
    height=new int;
    height=h;
}
Box::~Box () {
    delete height;
}
```

C++ Inheritance: Base and Derived Objects

```
int main(){
    CRectangle rect(3,4);
    Box box (3,4,5);
    cout<<“Rectangle area: “<<rect.area()<<endl;
    cout<<“Box base area: “<<box.area()<<endl;
    cout<<“Box volume: “<<box.volume()<<endl;
    return 0;
}
```

C++ Multiple Inheritance

- A derived class can inherit more than one base class
- It can happen in two ways:
  - A derived class can be used as the base class for another derived class: creates a multilevel class hierarchy
  - A derived class can directly inherit more than one base class

C++ Multiple Inheritance: Multilevel class hierarchy

```
class Point{
    double x;
    double y;
    public:
          Point(double x, y){this->x=x; this->y->y;}
         void get_xy(double &x, double &y){x=this->x, y=this->y;}
}
```

C++ Inheritance: : Multilevel class hierarchy

```
class Circle: public Point{
    protected:
        double rad;
    public:
        Circle(double x, double y, double r);
        double area(){return 3.14*rad*rad;}
}
Circle::Circle(double x, double y, double r):Point(x,y){
    rad=r;
}
```

C++ Inheritance: : Multilevel class hierarchy

```
class Cylinder: public Circle{
    double height;
    public:
        Cylinder(double x, double y, double r, double h);
        double volume(){return 3.14*rad*rad*height;}
}
Cylinder::Cylinder(double x, double y, double r, double h):Circle(x,y,r){
    height=h;
}
```

C++ Multiple Direct Inheritance

```
class Point{
    double x;
    double y;
    public:
        Point(double x, y){this->x=x; this->y->y;}
        void get_xy(double &x, double &y){x=this->x, y=this->y;}
}
```

C++ Multiple Direct Inheritance

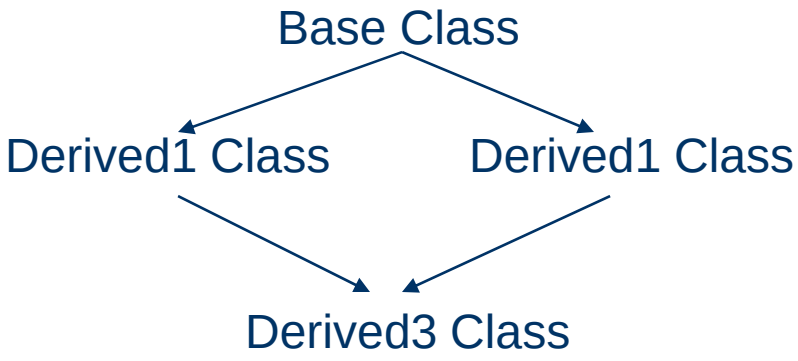
```
class Circle{
        protected:
                double rad;
        public:
            Circle(double r) {rad=r;}
            double area(){return 3.14*rad*rad;}
}
```

C++ Multiple Direct Inheritance

```
class Cylinder: public Point, public Circle{
        double height;
        public:
            Cylinder(double x, double y, double r, double h);
            double volume(){return 3.14*rad*rad*height;}
}
Cylinder::Cylinder(double x, double y, double r, double h):Point (x,y),Circle(r)
{
        height=h;
}
```

C++ Multiple Inheritance: Virtual Base Class

- **Virtual Base Class prevents a derived class to inherit more than one copy of the base class**
- **This may happen when a derived class directly inherits two base classes and these base classes are also derived from another common base class.**



C++ Multiple Inheritance: Virtual Base Class

```
#include <iostream>
using namespace std;
class Base{
    protected:
        int x;
    public:
        Base (int x) {this->x=x;}
        int getx() {return x;}
};
```

C++ Multiple Inheritance: Virtual Base Class

```
class Derived1: public Base{
    protected:
        int y;
    public:
        Derived1 (int x, int y);
        int gety() {return y;}
};
Derived1::Derived1(int x, int y): Base (x){
     this->y=y;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
class Derived2: public Base{
    protected:
        int z;
    public:
        Derived2 (int x, int z);
        int getz() {return z;}
};
Derived2::Derived2 (int x, int z): Base(x) {
     this->z=z;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
class Derived3: public Derived1, public Derived2{
    protected:
        int l;
    public:
        Derived3 (int x, int y, int z, int l);
        int getl() {return l;}
};
Derived3::Derived3 (int x, int y, int z, int l): Derived1(x,y),
    Derived2(x,z) {
     this->l=l;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
int main(){
    Derived3 d3(1,2,3,4);
    //cout<<"x: "<<d3.getx()<<endl; //ambiguous
    cout<<"y: "<<d3.gety()<<endl;
    cout<<"z: "<<d3.getz()<<endl;
    cout<<"l: "<<d3.getl()<<endl;

    return 0;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
#include <iostream>
using namespace std;
class Base{
    protected:
        int x;
    public:
        Base(){x=0;}
        Base (int x) {this->x=x;}
        void setx(int x){this->x=x;}
        int getx() {return x;}
};
```

C++ Multiple Inheritance: Virtual Base Class

```
class Derived1: virtual public Base{
    protected:
        int y;
    public:
        Derived1 (int y){this->y=y;}
        Derived1 (int x, int y);
        int gety() {return y;}
};
Derived1::Derived1(int x, int y): Base (x){
     this->y=y;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
class Derived2: virtual public Base{
    protected:
        int z;
    public:
        Derived2 (int z){this->z=z;}
        Derived2 (int x, int z);
        int getz() {return z;}
};
Derived2::Derived2 (int x, int z): Base(x) {
     this->z=z;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
class Derived3: public Derived1, public Derived2{
	protected:
		int l;
	public:
		Derived3 (int x, int y, int z, int l);
		int getl() {return l;}
};
/*
Derived3::Derived3 (int x, int y, int z, int l): Derived1(x,y), Derived2(x,z) {this->l=l;}
*/
Derived3::Derived3 (int x, int y, int z, int l): Derived1(y), Derived2(z) {
	//this->x=x;
	this->l=l;
}
```

C++ Multiple Inheritance: Virtual Base Class

```
int main(){
    Derived3 d3(1,2,3,4);

    cout<<"x: "<<d3.getx()<<endl; // print x: 0, which has been set by Base default constructor
    cout<<"y: "<<d3.gety()<<endl;
    cout<<"z: "<<d3.getz()<<endl;
    cout<<"l: "<<d3.getl()<<endl;

    d3.setx(5);
    cout<<"x: "<<d3.getx()<<endl; // print x: 5, which has been set by setx() function
    cout<<"y: "<<d3.gety()<<endl;
    cout<<"z: "<<d3.getz()<<endl;
    cout<<"l: "<<d3.getl()<<endl;
    return 0;
}
```

C++ Inheritance: Function Overriding

```
#include <iostream>
using namespace std;
class Point{
protected:
    double x;
    double y;
public:
    Point() {x=0.0; y=0.0;}
    Point(double x, double y);
    double area(){return 0;}
};
Point::Point(double x, double y){
    this->x=x; this->y=y;
}
```

C++ Inheritance: Function Overriding

```
class Circle: public Point{
protected:
    double radious;
public:
    Circle(){radious=0.0;}
    Circle(double x, double y, double r);
    double area();
};
Circle::Circle(double x, double y, double r) : Point(x,y) {
    radious=r;
}
double Circle::area(){
    return 3.14*radious*radious;
}
```

C++ Inheritance: Function Overriding

```
class Cylinder:public Circle{
    double height;
public:
    Cylinder(){height=0.0;}
    Cylinder(double x, double y, double r, double h);
    double area();
};
Cylinder::Cylinder(double x, double y, double r, double h): Circle(x,y,r){
    height=h;
}
double Cylinder::area(){
    return 3.14*radious*radious*height;
}
```

C++ Inheritance: Function Overriding

```
int main(){

    Point p(1.0, 1.0);
    Circle c(1.0, 1.0, 3.0);
    Cylinder cl(1.0, 1.0, 3.0, 2.0);

    cout<<"The area of the point is: "<<p.area()<<endl;
    cout<<"The area of the circle is: "<<c.area()<<endl;
    cout<<"The area of the cylinder is: "<<cl.area()<<endl;

    return 0;
}
```

C++ Inheritance: Run-time Polymorphism

```
int main(){
    Point p(1.0, 1.0), *pt;
    Circle c(1.0, 1.0, 3.0);
    Cylinder cl(1.0, 1.0, 3.0, 2.0);
    pt=&p;
    cout<<"The area of the pointer object pointed by point pointer is: "<<pt->area()<<endl;
    pt=&c;
    cout<<"The area of the circle object pointed by point pointer is: "<<pt->area()<<endl;
    pt=&cl;
    cout<<"The area of the cylinder object pointed by point pointer is: "<<pt->area()<<endl;
    return 0;
}
```

Run-time Polymorphism by Virtual Function

```
#include <iostream>
using namespace std;
class Point{
protected:
    double x;
    double y;
public:
    Point() {x=0.0; y=0.0;}
    Point(double x, double y);
    virtual double area(){return 0;}
};
Point::Point(double x, double y){
    this->x=x; this->y=y;
}
```

Run-time Polymorphism by Virtual Function

```
class Circle: public Point{
protected:
    double radious;
public:
    Circle(){radious=0.0;}
    Circle(double x, double y, double r);
    double area();
};
Circle::Circle(double x, double y, double r) : Point(x,y) {
    radious=r;
}
double Circle::area(){
    return 3.14*radious*radious;
}
```

Run-time Polymorphism by Virtual Function

```
class Cylinder:public Circle{
    double height;
public:
    Cylinder(){height=0.0;}
    Cylinder(double x, double y, double r, double h);
    double area();
};
Cylinder::Cylinder(double x, double y, double r, double h): Circle(x,y,r){
    height=h;
}
double Cylinder::area(){
    return 3.14*radious*radious*height;
}
```

Run-time Polymorphism by Virtual Function

```
int main(){
    Point p(1.0, 1.0), *pt;
    Circle c(1.0, 1.0, 3.0);
    Cylinder cl(1.0, 1.0, 3.0, 2.0);
    pt=&p;
    cout<<"The area of the pointer object pointed by point pointer is: "<<pt->area()<<endl;
    pt=&c;
    cout<<"The area of the circle object pointed by point pointer is: "<<pt->area()<<endl;
    pt=&cl;
    cout<<"The area of the cylinder object pointed by point pointer is: "<<pt->area()<<endl;
    return 0;
}
```

Run-time Polymorphism by Pure Virtual Function

```
#include <iostream>
using namespace std;
class Point{
protected:
    double x;
    double y;
public:
    Point() {x=0.0; y=0.0;}
    Point(double x, double y);
    virtual double area()=0;
};
Point::Point(double x, double y){
    this->x=x; this->y=y;
}
```

Run-time Polymorphism by Pure Virtual Function

```
class Circle: public Point{
protected:
    double radious;
public:
    Circle(){radious=0.0;}
    Circle(double x, double y, double r);
    double area();
};
Circle::Circle(double x, double y, double r) : Point(x,y) {
    radious=r;
}
double Circle::area(){
    return 3.14*radious*radious;
}
```

Run-time Polymorphism by Pure Virtual Function

```
class Cylinder:public Circle{
    double height;
public:
    Cylinder(){height=0.0;}
    Cylinder(double x, double y, double r, double h);
    double area();
};
Cylinder::Cylinder(double x, double y, double r, double h): Circle(x,y,r){
    height=h;
}
double Cylinder::area(){
    return 3.14*radious*radious*height;
}
```

Run-time Polymorphism by Pure Virtual Function

```
int main(){
    //Point p(1.0, 1.0), *pt;
    Point  *pt;
    Circle c(1.0, 1.0, 3.0);
    Cylinder cl(1.0, 1.0, 3.0, 2.0);
    pt=&c;
    cout<<"The area of the circle object pointed by point pointer is: "<<pt->area()<<endl;
    pt=&cl;
    cout<<"The area of the cylinder object pointed by point pointer is: "<<pt->area()<<endl;
    return 0;
}
```

cout<<“Thank You”<<endl;

cout<<“**Have a Good Day**”<<endl;