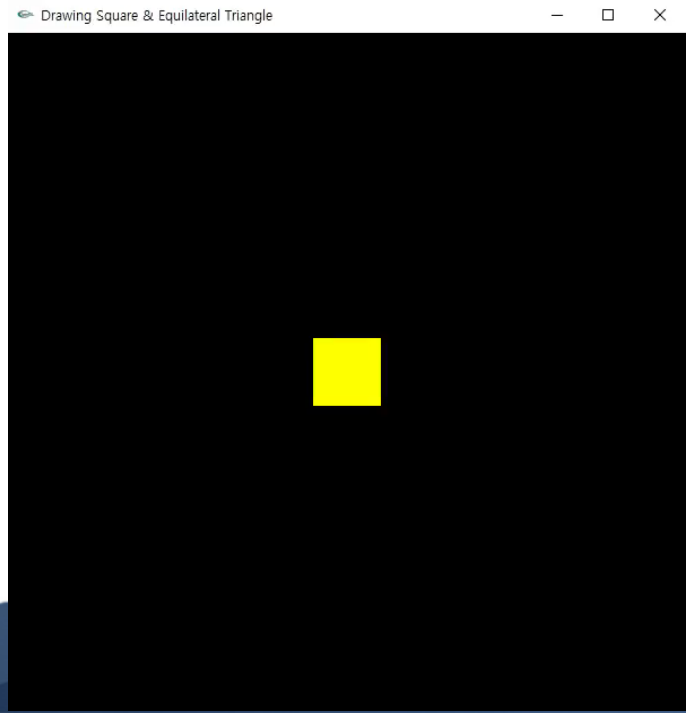
LAB I
Week 10

Seoul National University
Graphics & Media Lab
HyeonSeung Shin

Today's Mission

- Move the square with key input (up, down, right, left)
  - The square should keep moving in that direction until
    - It hits the boundary, or
    - You type a different key
  - The square may move too fast on your computer.
    - Control the FPS so that you can see the movement at a normal speed.

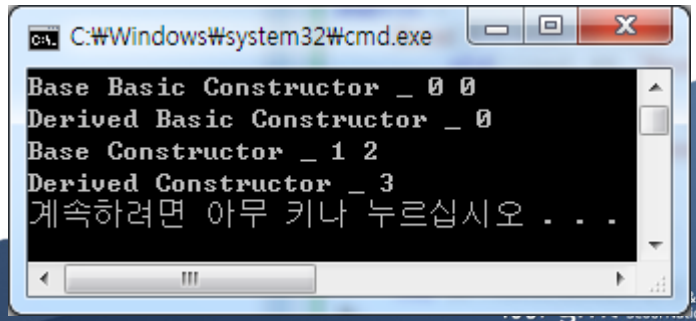


- C++ features you will use
  - Multiple inheritance
  - Pure virtual function
  - Destructor

How Calls to Base Class Constructor are Made?

- If there isn't any explicit call in the constructor initializer, a call to the no-argument constructor of the base class is automatically made.
- An explicit call with arguments can be made in the constructor initializer.

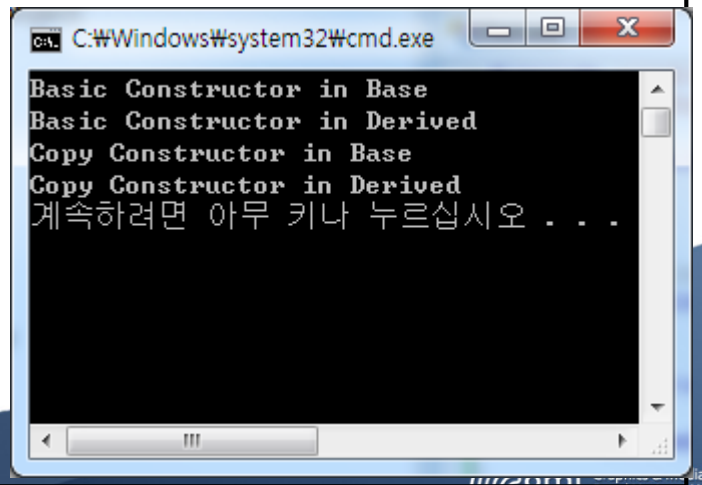
```
class Base {
public :
   Base() : public_base_var(0), private_base_var(0) {
      std::cout << "Base Basic Constructor _ " << public_base_var << " " << private_base_var << std::endl;
   }
   Base(int a, int b) : public_base_var(a), private_base_var(b) {
      std::cout << "Base Constructor _ " << public_base_var << " " << private_base_var << std::endl;
   }

   int public_base_var;
private :
   int private_base_var;
};

class Derived : public Base {
public :
   Derived() : public_derived_var(0) {
      std::cout << "Derived Basic Constructor _ " << public_derived_var << std::endl;
   }
   Derived(int a, int b, int c) : Base(a,b), public_derived_var(c) {
      std::cout << "Derived Constructor _ " << public_derived_var << std::endl;
   }

   int public_derived_var;
};

void main() {
   Derived derived_0;
   Derived derived_1(1,2,3);
}
```



Defining Copy Constructor of the Derived Class

- You can copy the base part of the object by making a call `Base(derived)` to the base-class copy constructor.
  - Although the call is made with the derived class object argument, in this case, only the Base class portion is used.
- If such a call is not explicitly made, a call to the base-class constructor (no-parameter version) is automatically made. Note that the call is made to the constructor not the copy constructor.

```
class Base {
public :
   Base() { cout << “Basic Constructor in Base" << endl; }
   Base(const Base& base) { cout << "Copy Constructor in Base" << endl; a = base.a; }

protected :
   int a;
};

class Derived : public Base {
public :
   Derived() { size = 10; ptr = new int[size]; cout << “Basic Constructor in Derived" << endl; }

   // Correct definition of copy constructor in derived class
   Derived(const Derived& derived) : Base(derived) {
      cout << "Copy Constructor in Derived" << endl;

      size = derived.size;
      ptr = new int[size];
      for(size_t i=0;i<size;++i)
         ptr[i] = derived.ptr[i];
   }

   size_t     size;
   int *      ptr;
};

void main() {
   Derived derived_0;
   Derived derived_1(derived_0);
}
```



Defining Assignment Operator of the Derived Class

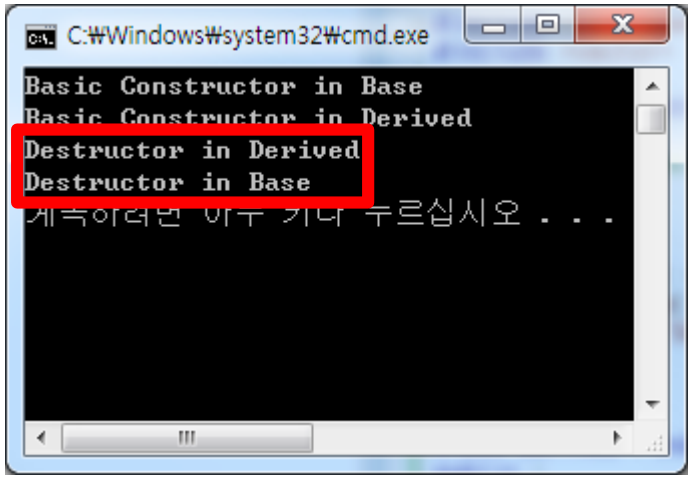
- Similarly to the case of the copy constructor, you can assign the base part of the object by making a call `Base::operator=(rhs);` to the base-class assignment operator.
- If such a call is not explicitly made, assignment of the base part does not occur.
- Therefore, in the implementation of the assignment, you should not forget to make the base part assignment explicitly.

```
Derived& Derived::operator=(const Derived& rhs) {
   if(this != &rhs) {
      Base::operator=(rhs);   // assignment of the base part

      if(ptr != NULL) delete[] ptr;

      size = rhs.size;
      ptr = new int[size];
      for(size_t i=0;i<size;++i)
          ptr[i] = rhs.ptr[i];
   }
}
```

Virtual Destructor

- To ensure that the proper destruction is done in the previous example, the destructor must be defined virtual.

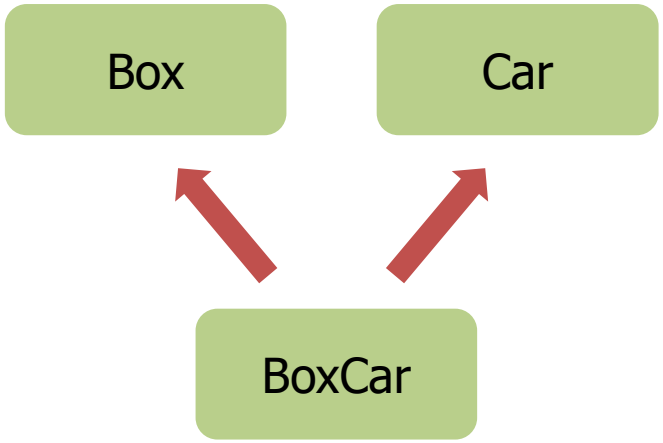
```
class Base {
public :
   Base() { cout << "Basic Constructor in Base" << endl; }
   virtual ~Base() { cout << "Destructor in Base" << endl; }
};

class Derived : public Base {
public :
   Derived() {
      size = 10; ptr = new int[size];
      cout << "Basic Constructor in Derived" << endl;
   }
   ~Derived() {
      delete[] ptr;
      cout << "Destructor in Derived" << endl;
   }

   size_t  size;
   int *   ptr;
};

void main() {
   Base * p = new Derived();
   delete p;
}
```



The destructors of both Derived and Base classes are called, in that order.
Therefore ptr of Derived class is now successfully deleted!

An Example of Multiple Inheritance

```
class Box { … };

class Car { … };

class BoxCar : public Box, public Car { … };
```



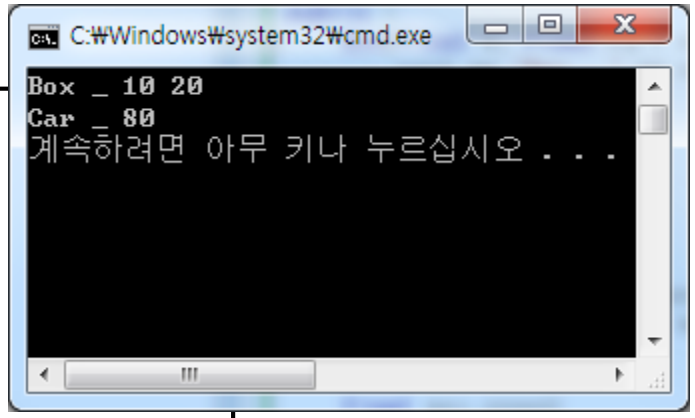
Construction of Multiply Derived Classes

- As is the case for inheriting from a single base class, a constructor of multiply derived class may pass values to its base class constructors in the constructor initializer.
- The base class constructors are invoked in the order in which they appear in the class derivation list.

```
class Box {
public :
   Box(float w, float h) : width(w), height(h) {
      cout << "Box _ " << width << " " << height << endl;
   }
   float width, height;
};
class Car {
public :
   Car(float msp) : max_speed(msp) {
      cout << "Car _ " << max_speed << endl;
   }
   float max_speed;
};
class BoxCar : public Box, public Car {
public :
   BoxCar(float w, float h, float msp) : Car(msp), Box(w,h) {}
};

void main() {
   BoxCar boxcar(10,20,80);
}
```

Constructor is invoked in this order

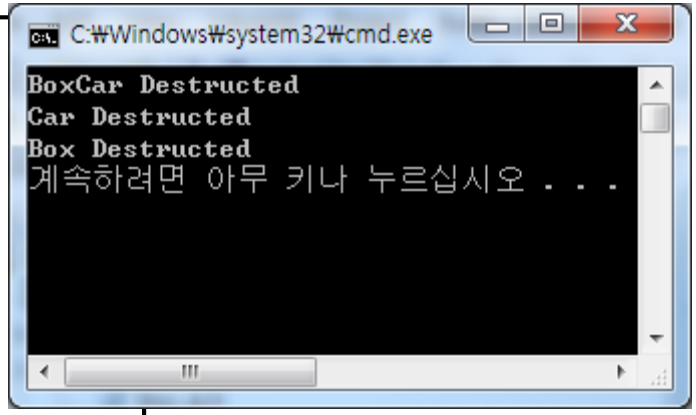
Can pass values in the constructor initializer



Destruction of Multiply Derived Classes

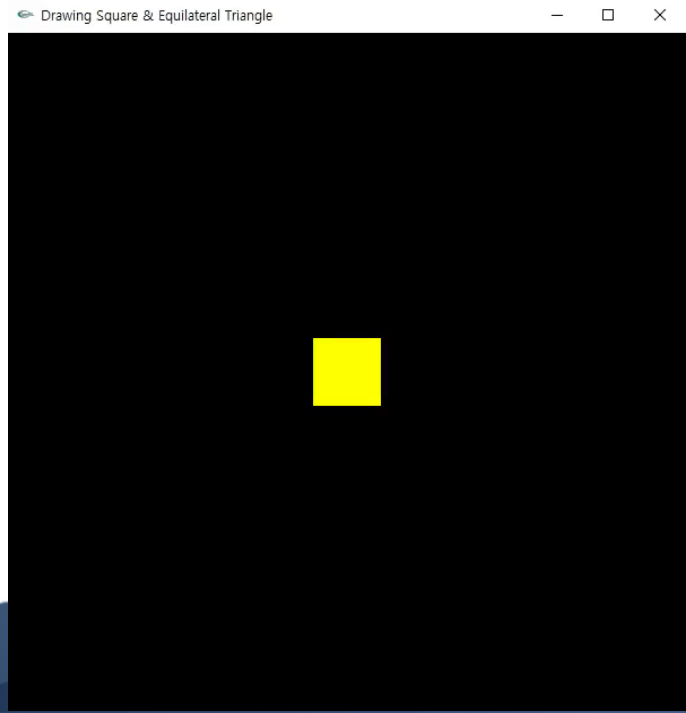
- Destructors are invoked in the reverse order from the order the constructors are invoked.

```
class Box {
public :
   Box() {}
   ~Box() { cout << "Box Destructed" << endl; }
};

class Car {
public :
   Car() {}
   ~Car() { cout << "Car Destructed" << endl; }
};

class BoxCar : public Box, public Car {
public :
   BoxCar() {}
   ~BoxCar() { cout << "BoxCar Destructed" << endl; }
};

void main() {
   BoxCar boxcar;
}
```



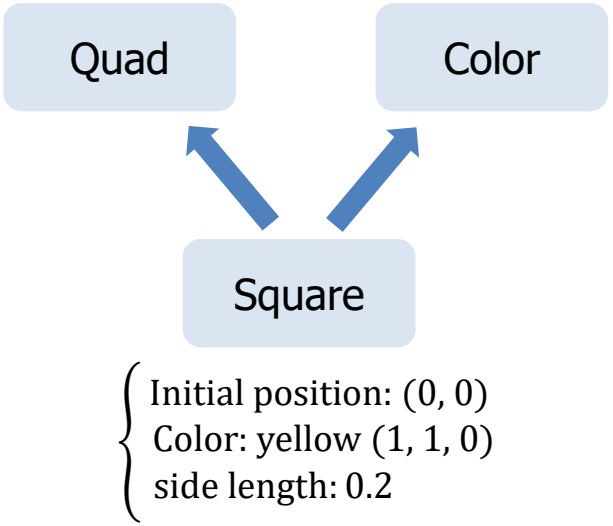
Today's Mission

- Move the square using key input (up, down, right, left)
  - The square should keep moving in that direction until
    - It hits the boundary, or
    - You press a different key
  - The square may move too fast on your computer.
    - Control the FPS so that you can see the movement at a normal speed.



- C++ features you will use
  - multiple inheritance
  - Pure virtual function
  - Destructor

To Do

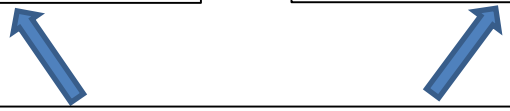


- Define classes (Quad, Color, and Square)
  - Multiple inheritance
  - Pure virtual function draw() of Quad
  - Destructor of each class
- Control FPS if you need
- Move the square using key input (up, down, right, left)
  - The square should move toward the inputted direction.
  - It should stop when reaching the wall.

Class Diagram

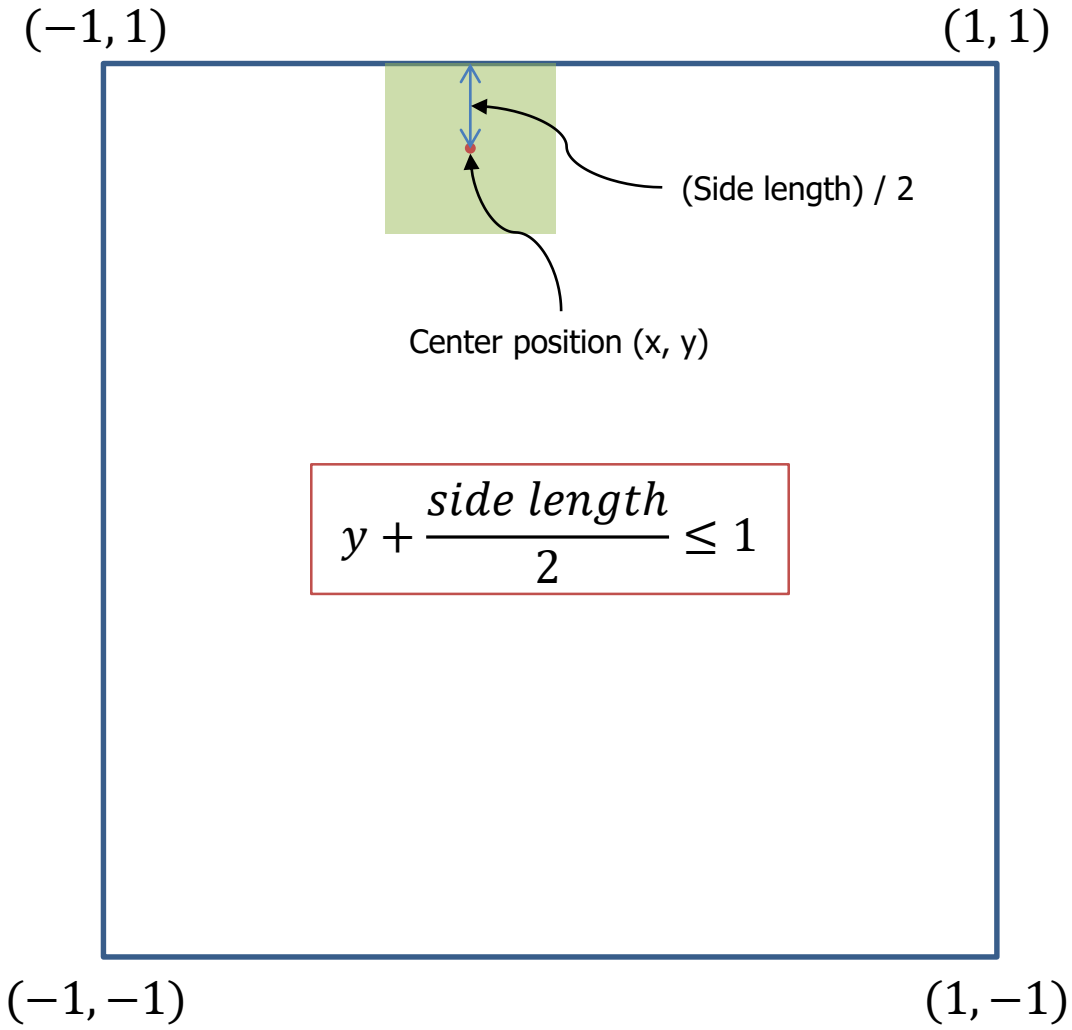
```
class Quad {
public:
        Quad(float x, float y);
        ~Quad();

        void setPos(float x, float y);
        virtual void draw() const = 0;
        float getX() const;
        float getY() const;

protected:
        float* center_pos;
};
```

```
class Color {
public:
        Color(float r, float b, float g);
        ~Color();

        void setColor(float r, float g, float b);

protected:
        float* color;
};
```



```
class Square : public Quad, public Color {
public:
        Square(float x, float y, float r, float g, float b, float sl);
        ~Square();

        virtual void draw() const;
        float getSideLength() const;

private:
        float* side_length; // single scalar value
};
```

Screen boundary check



FPS Control

- FPS
  - Frames Per Second
- In real time graphic
  - 30 or 60 fps
  - In this lab, use 30 fps.
  - frame duration = 1/30 sec ≈ 33ms
- What to do between frames?
  - Update object
    - Only once for each frame duration
  - Draw object
    - For every iteration of the idle() loop

```
#include <time.h>

using namespace std;

clock_t start = clock();
clock_t end;

void idle() {
        end = clock();

        if (end - start > 1000 / 30) {
                (translate square)    // update object

                start = end;
        }

        glutPostRedisplay();          // draw object
}
```

30 fps (33ms)

- We will measure the time passed from the start of the frame using the clock() function
  - clock() returns current time in millisecond
  - You have to include <time.h>