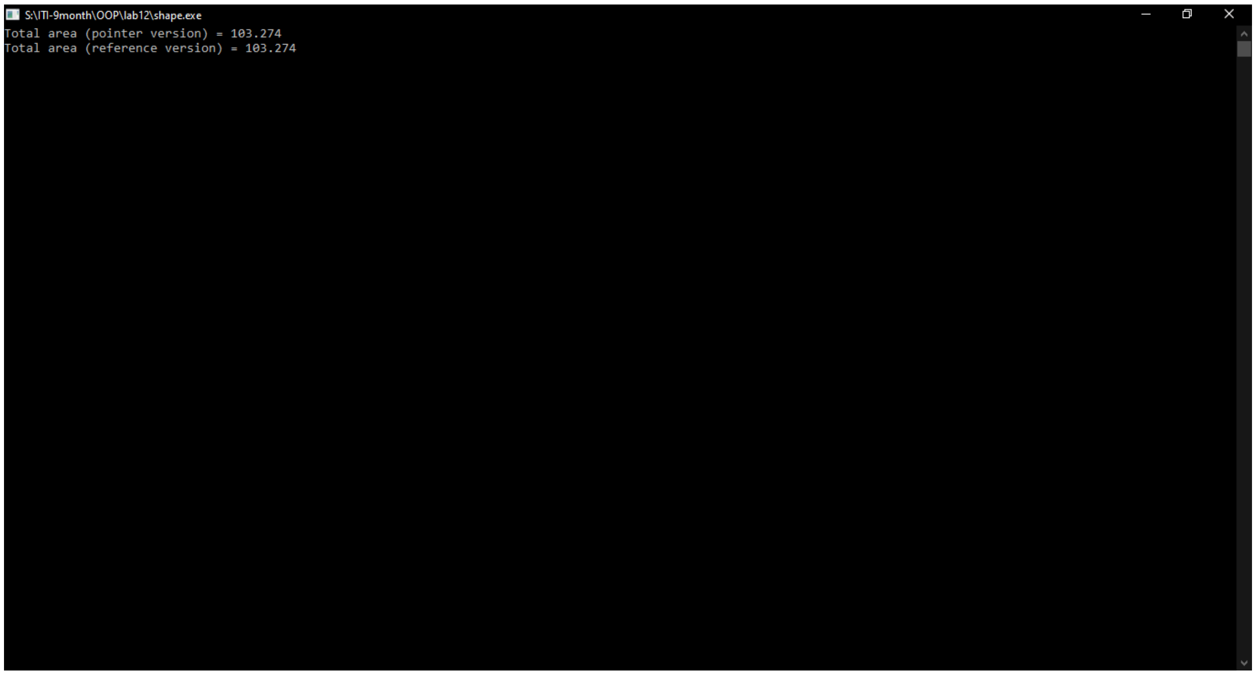
1.

```cpp
#include <iostream>
#include <cmath>
#include <functional>
using namespace std;


// Abstract Base Class
class Shape {
public:
    virtual ~Shape() = default;
    virtual double calcArea() const = 0;    // pure virtual
};

class Circle : public Shape {
private:
    double radius;

public:
    Circle() : radius(0) {}
    Circle(double r) : radius(r) {}
    ~Circle() {}

    void setRadius(double r) { radius = r; }
    double getRadius() const { return radius; }

    double calcArea() const override {
        return M_PI * radius * radius;
    }
};

class Triangle : public Shape {
private:
    double base;
    double height;

public:
    Triangle() : base(0), height(0) {}
    Triangle(double b, double h) : base(b), height(h) {}
    ~Triangle() {}

    void setBase(double b) { base = b; }
    void setHeight(double h) { height = h; }
    double getBase() const { return base; }
    double getHeight() const { return height; }

    double calcArea() const override {
        return 0.5 * base * height;
    }
};

class Rectangle : public Shape {
protected:
    double width;
    double height;

public:
    Rectangle() : width(0), height(0) {}
    Rectangle(double w, double h) : width(w), height(h) {}
    ~Rectangle() {}

    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }

    double getWidth() const { return width; }
    double getHeight() const { return height; }

    double calcArea() const override {
        return width * height;
    }
};

class Square : public Rectangle {
public:
    Square() : Rectangle(0, 0) {}
    Square(double side) : Rectangle(side, side) {}

    void setSide(double side) {
        width = height = side;   // always equal
    }

    double getSide() const { return width; }

    // use Rectangle::calcArea()
};

double SumArea(Shape* arr[], int size) {
    double sum = 0;
    for (int i = 0; i < size; i++)
        sum += arr[i]->calcArea();
    return sum;
}

double SumArea(reference_wrapper<Shape> arr[], int size) {
    double sum = 0;
    for (int i = 0; i < size; i++)
        sum += arr[i].get().calcArea();
    return sum;
}

int main() {
    Shape* shapesPtr[4];
    shapesPtr[0] = new Rectangle(5, 10);  // area = 50
    shapesPtr[1] = new Square(4);         // area = 16
    shapesPtr[2] = new Triangle(6, 3);    // area = 9
    shapesPtr[3] = new Circle(3);         // area ≈ 28.274

    cout << "Total area (pointer version) = "
         << SumArea(shapesPtr, 4) << endl;

    for (int i = 0; i < 4; i++)
        delete shapesPtr[i];

    Rectangle r(5, 10);
    Square s(4);
    Triangle t(6, 3);
    Circle c(3);

    reference_wrapper<Shape> shapesRef[] = { r, s, t, c };

    cout << "Total area (reference version) = "
         << SumArea(shapesRef, 4) << endl;

    return 0;
}
```



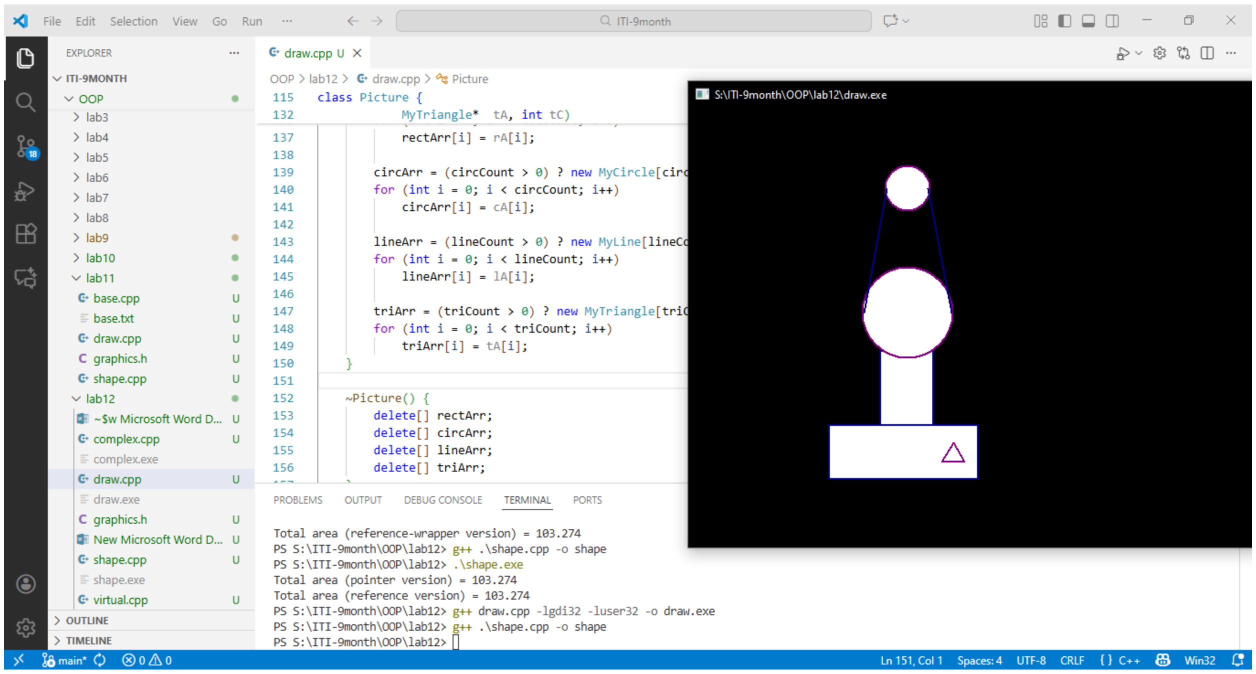
```
Total area (pointer version) = 103.274
Total area (reference version) = 103.274
```

2.

```cpp
#include <iostream>
#include <conio.h>
#include "graphics.h"
using namespace std;

class MyPoint {
private:
    int x, y;

public:
    MyPoint() : x(0), y(0) {}
    MyPoint(int x, int y) : x(x), y(y) {}
    MyPoint(const MyPoint& p) : x(p.x), y(p.y) {}
    ~MyPoint() {}

    int getx() const { return x; }
    void setx(int x) { x = x; }

    int gety() const { return y; }
    void sety(int y) { y = y; }

    void show() const {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};

class MyRectangle {
private:
    MyPoint upper, lower;
    int color;

public:
    MyRectangle() : upper(), lower(), color(0) {}

    MyRectangle(int color) : upper(), lower(), color(color) {}

    MyRectangle(int x1, int y1, int x2, int y2, int color)
        : upper(x1, y1), lower(x2, y2), color(color) {}

    ~MyRectangle() {}

    void draw() {
        setcolor(color);
        rectangle(upper.getx(), upper.gety(),
                  lower.getx(), lower.gety());
    }
};

class MyCircle {
private:
    MyPoint center;
    int radius, color;

public:
    MyCircle() : center(), radius(0), color(0) {}

    MyCircle(int color) : center(), radius(0), color(color) {}

    MyCircle(int x, int y, int radius, int color)
        : center(x, y), radius(radius), color(color) {}

    ~MyCircle() {}

    void draw() {
        setcolor(color);
        circle(center.getx(), center.gety(), radius);
    }
};

class MyLine {
private:
    MyPoint p1, p2;
    int color;

public:
    MyLine() : p1(), p2(), color(0) {}

    MyLine(int color) : p1(), p2(), color(color) {}

    MyLine(int x1, int y1, int x2, int y2, int color)
        : p1(x1, y1), p2(x2, y2), color(color) {}

    ~MyLine() {}

    void draw() {
        setcolor(color);
        line(p1.getx(), p1.gety(),
             p2.getx(), p2.gety());
    }
};

class MyTriangle {
private:
    MyPoint p1, p2, p3;
    int color;

public:
    MyTriangle() : p1(), p2(), p3(), color(0) {}

    MyTriangle(int color) : p1(), p2(), p3(), color(color) {}

    MyTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int color)
        : p1(x1, y1), p2(x2, y2), p3(x3, y3), color(color) {}

    ~MyTriangle() {}

    void draw() {
        setcolor(color);
        triangle(p1.getx(), p1.gety(),
                 p2.getx(), p2.gety(),
                 p3.getx(), p3.gety());
    }
};

class Picture {
private:
    MyRectangle* rectArr;
    MyCircle*    circArr;
    MyLine*      lineArr;
    MyTriangle*  triArr;

    int rectCount, circCount, lineCount, triCount;

public:
    Picture()
        : rectArr(nullptr), circArr(nullptr), lineArr(nullptr), triArr(nullptr),
          rectCount(0), circCount(0), lineCount(0), triCount(0) {}

    Picture(MyRectangle* rA, int rC,
            MyCircle*    cA, int cC,
            MyLine*      lA, int lC,
            MyTriangle*  tA, int tC)
        : rectCount(rC), circCount(cC), lineCount(lC), triCount(tC)
```

```
{
    rectArr = (rectCount > 0) ? new MyRectangle[rectCount] : nullptr;
    for (int i = 0; i < rectCount; i++)
        rectArr[i] = rA[i];

    circArr = (circCount > 0) ? new MyCircle[circCount] : nullptr;
    for (int i = 0; i < circCount; i++)
        circArr[i] = cA[i];

    lineArr = (lineCount > 0) ? new MyLine[lineCount] : nullptr;
    for (int i = 0; i < lineCount; i++)
        lineArr[i] = lA[i];

    triArr = (triCount > 0) ? new MyTriangle[triCount] : nullptr;
    for (int i = 0; i < triCount; i++)
        triArr[i] = tA[i];
}

~Picture() {
    delete[] rectArr;
    delete[] circArr;
    delete[] lineArr;
    delete[] triArr;
}

void paint() const {
    for (int i = 0; i < rectCount; i++)
        rectArr[i].draw();

    for (int i = 0; i < circCount; i++)
        circArr[i].draw();

    for (int i = 0; i < lineCount; i++)
        lineArr[i].draw();

    for (int i = 0; i < triCount; i++)
        triArr[i].draw();
}

void setRectangles(MyRectangle* rA, int count) {
    delete[] rectArr;
    rectCount = count;
    rectArr = (count > 0) ? new MyRectangle[count] : nullptr;
    for (int i = 0; i < count; i++)
        rectArr[i] = rA[i];
```

```cpp
    }

    void setCircles(MyCircle* cA, int count) {
        delete[] circArr;
        circCount = count;
        circArr = (count > 0) ? new MyCircle[count] : nullptr;
        for (int i = 0; i < count; i++)
            circArr[i] = cA[i];
    }

    void setLines(MyLine* lA, int count) {
        delete[] lineArr;
        lineCount = count;
        lineArr = (count > 0) ? new MyLine[count] : nullptr;
        for (int i = 0; i < count; i++)
            lineArr[i] = lA[i];
    }

    void setTriangles(MyTriangle* tA, int count) {
        delete[] triArr;
        triCount = count;
        triArr = (count > 0) ? new MyTriangle[count] : nullptr;
        for (int i = 0; i < count; i++)
            triArr[i] = tA[i];
    }
};

int main() {
    initgraph();

    MyRectangle rectArr[2] = {
        MyRectangle(162, 347, 326, 407, 1),
        MyRectangle(218, 266, 277, 348, 1)
    };

    MyCircle circArr[2] = {
        MyCircle(248, 224, 100, 5),
        MyCircle(248, 88, 50, 5)
    };

    MyLine lineArr[2] = {
        MyLine(225, 88, 200, 224, 1),
        MyLine(270, 88, 296, 224, 1)
    };
    MyTriangle triArr[1] = {
        MyTriangle(285, 387, 310, 387, 298, 366, 5)
    };

    Picture pic(rectArr, 2,
                circArr, 2,
                lineArr, 2,
                triArr, 1);

    // Draw everything
    pic.paint();

    return 0;
}
```



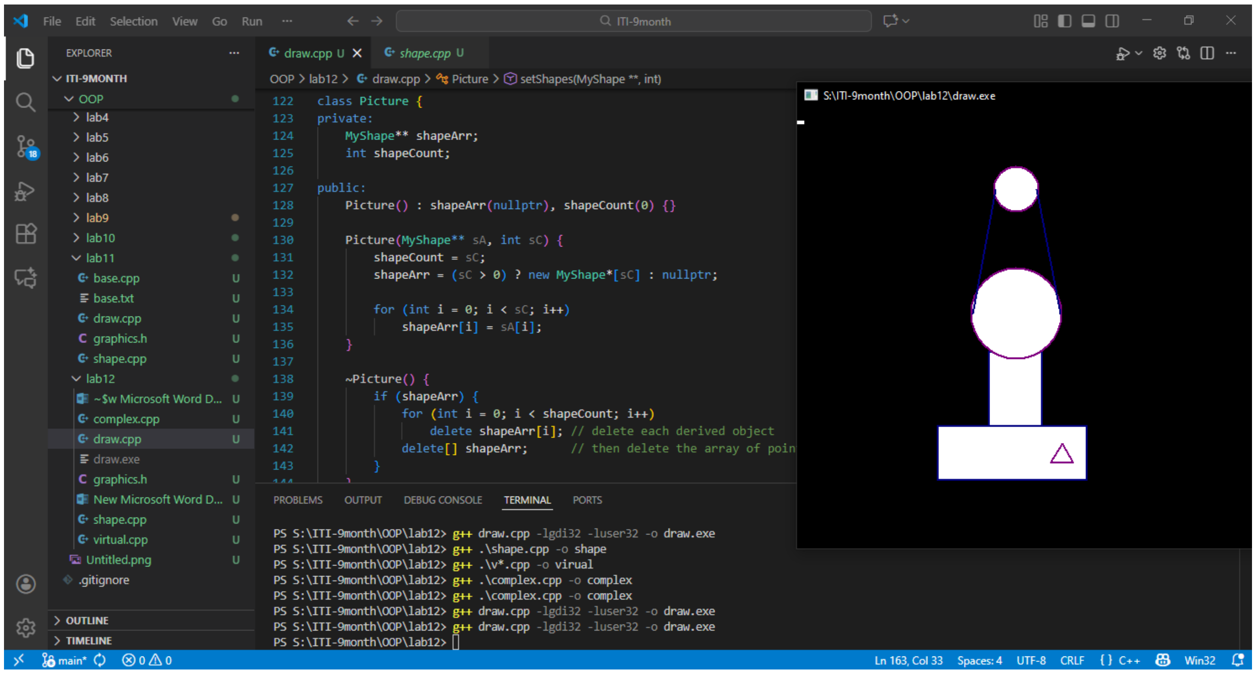
```cpp
class Picture {
            MyTriangle*  tA, int tC)
            rectArr[i] = rA[i];

        circArr = (circCount > 0) ? new MyCircle[circ
        for (int i = 0; i < circCount; i++)
            circArr[i] = cA[i];

        lineArr = (lineCount > 0) ? new MyLine[lineCo
        for (int i = 0; i < lineCount; i++)
            lineArr[i] = lA[i];

        triArr = (triCount > 0) ? new MyTriangle[tri
        for (int i = 0; i < triCount; i++)
            triArr[i] = tA[i];
    }

    ~Picture() {
        delete[] rectArr;
        delete[] circArr;
        delete[] lineArr;
        delete[] triArr;
```

```
Total area (reference-wrapper version) = 103.274
PS S:\ITI-9month\OOP\lab12> g++ .\shape.cpp -o shape
PS S:\ITI-9month\OOP\lab12> .\shape.exe
Total area (pointer version) = 103.274
Total area (reference version) = 103.274
PS S:\ITI-9month\OOP\lab12> g++ draw.cpp -lgdi32 -luser32 -o draw.exe
PS S:\ITI-9month\OOP\lab12> g++ .\shape.cpp -o shape
PS S:\ITI-9month\OOP\lab12>
```

3.
```
#include <iostream>
#include <conio.h>
#include "graphics.h"
using namespace std;

// Abstract Base Class
class MyShape {
public:
    virtual ~MyShape() = default;
    virtual void draw() const = 0; // Pure virtual function
};

class MyPoint {
private:
    int x, y;

public:
    MyPoint() : x(0), y(0) {}
    MyPoint(int x, int y) : x(x), y(y) {}
    MyPoint(const MyPoint& p) : x(p.x), y(p.y) {}
    ~MyPoint() {}

    int getx() const { return x; }
    void setx(int x) { x = x; }

    int gety() const { return y; }
    void sety(int y) { y = y; }

    void show() const {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};

class MyRectangle : public MyShape {
private:
    MyPoint upper, lower;
    int color;

public:
    MyRectangle() : upper(), lower(), color(0) {}

    MyRectangle(int color) : upper(), lower(), color(color) {}

    MyRectangle(int x1, int y1, int x2, int y2, int color)
```

```cpp
        : upper(x1, y1), lower(x2, y2), color(color) {}

    ~MyRectangle() {}

    void draw() const override {
        setcolor(color);
        rectangle(upper.getx(), upper.gety(),
                  lower.getx(), lower.gety());
    }
};

class MyCircle : public MyShape {
private:
    MyPoint center;
    int radius, color;

public:
    MyCircle() : center(), radius(0), color(0) {}

    MyCircle(int color) : center(), radius(0), color(color) {}

    MyCircle(int x, int y, int radius, int color)
        : center(x, y), radius(radius), color(color) {}

    ~MyCircle() {}

    void draw() const override {
        setcolor(color);
        circle(center.getx(), center.gety(), radius);
    }
};

class MyLine : public MyShape {
private:
    MyPoint p1, p2;
    int color;

public:
    MyLine() : p1(), p2(), color(0) {}

    MyLine(int color) : p1(), p2(), color(color) {}

    MyLine(int x1, int y1, int x2, int y2, int color)
        : p1(x1, y1), p2(x2, y2), color(color) {}
    ~MyLine() {}

    void draw() const override {
        setcolor(color);
        line(p1.getx(), p1.gety(),
             p2.getx(), p2.gety());
    }
};

class MyTriangle : public MyShape {
private:
    MyPoint p1, p2, p3;
    int color;

public:
    MyTriangle() : p1(), p2(), p3(), color(0) {}

    MyTriangle(int color) : p1(), p2(), p3(), color(color) {}

    MyTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int color)
        : p1(x1, y1), p2(x2, y2), p3(x3, y3), color(color) {}

    ~MyTriangle() {}

    void draw() const override {
        setcolor(color);
        triangle(p1.getx(), p1.gety(),
                 p2.getx(), p2.gety(),
                 p3.getx(), p3.gety());
    }
};

class Picture {
private:
    MyShape** shapeArr;
    int shapeCount;

public:
    Picture() : shapeArr(nullptr), shapeCount(0) {}

    Picture(MyShape** sA, int sC) {
        shapeCount = sC;
        shapeArr = (sC > 0) ? new MyShape*[sC] : nullptr;

        for (int i = 0; i < sC; i++)
```

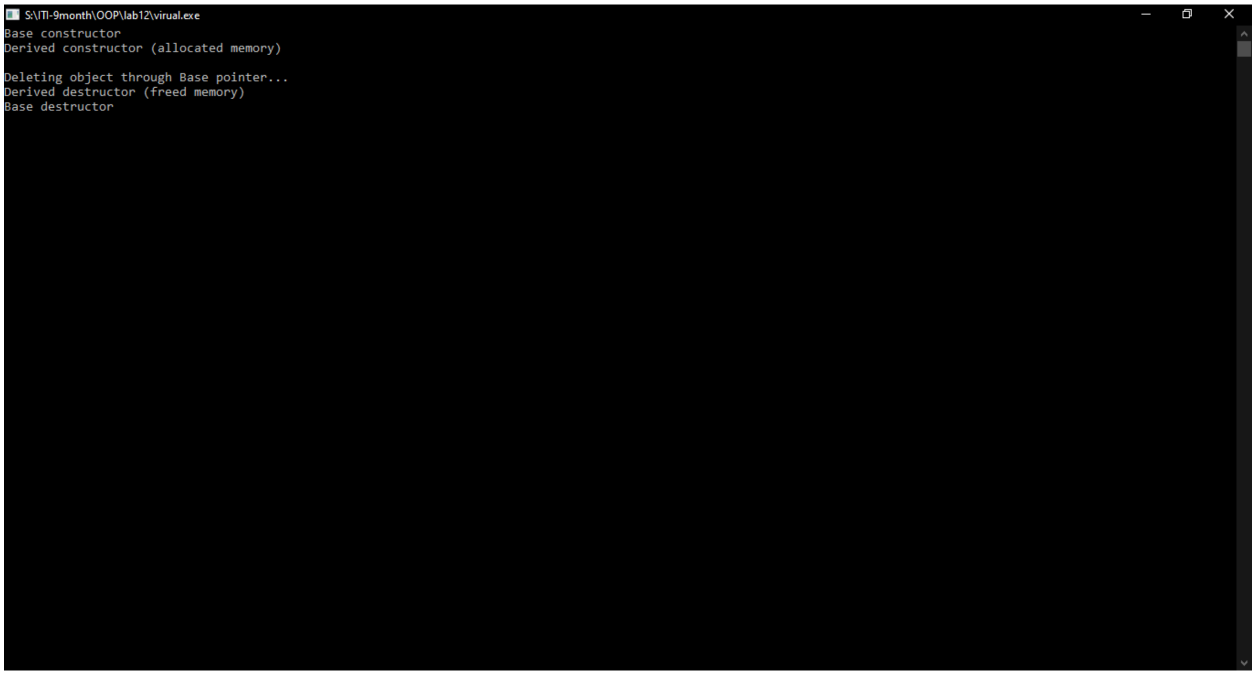
```
            shapeArr[i] = sA[i];
    }

    ~Picture() {
        if (shapeArr) {
            for (int i = 0; i < shapeCount; i++)
                delete shapeArr[i]; // delete each derived object
            delete[] shapeArr;      // then delete the array of pointers
        }
    }

    void paint() const {
        for (int i = 0; i < shapeCount; i++)
            shapeArr[i]->draw();
    }

    void setShapes(MyShape** sA, int count) {
        // Delete existing objects and array
        if (shapeArr) {
            for (int i = 0; i < shapeCount; i++)
                delete shapeArr[i];
            delete[] shapeArr;
        }

        // Allocate new array and copy pointers
        shapeCount = count;
        shapeArr = (count > 0) ? new MyShape*[count] : nullptr;
        for (int i = 0; i < count; i++)
            shapeArr[i] = sA[i];
    }
};

int main() {
    initgraph();

    // Create dynamic polymorphic list of shapes
    MyShape* shapes[7];

    shapes[0] = new MyRectangle(162, 347, 326, 407, 1);
    shapes[1] = new MyRectangle(218, 266, 277, 348, 1);

    shapes[2] = new MyCircle(248, 224, 100, 5);
    shapes[3] = new MyCircle(248, 88, 50, 5);

    shapes[4] = new MyLine(225, 88, 200, 224, 1);
    shapes[5] = new MyLine(270, 88, 296, 224, 1);

    shapes[6] = new MyTriangle(285, 387, 310, 387, 298, 366, 5);

    // Build the picture
    Picture pic;
    pic.setShapes(shapes, 7);

    // Draw
    pic.paint();

    // Clean up dynamic memory
    for (int i = 0; i < 7; i++)
        delete shapes[i];

    return 0;
}
```


S:\ITI-9month\OOP\lab12\draw.exe
OOP > lab12 > draw.cpp > Picture > setShapes(MyShape **, int)

```cpp
class Picture {
private:
    MyShape** shapeArr;
    int shapeCount;

public:
    Picture() : shapeArr(nullptr), shapeCount(0) {}

    Picture(MyShape** sA, int sC) {
        shapeCount = sC;
        shapeArr = (sC > 0) ? new MyShape*[sC] : nullptr;

        for (int i = 0; i < sC; i++)
            shapeArr[i] = sA[i];
    }

    ~Picture() {
        if (shapeArr) {
            for (int i = 0; i < shapeCount; i++)
                delete shapeArr[i]; // delete each derived object
            delete[] shapeArr;      // then delete the array of poin
        }
```

```
PS S:\ITI-9month\OOP\lab12> g++ draw.cpp -lgdi32 -luser32 -o draw.exe
PS S:\ITI-9month\OOP\lab12> g++ .\shape.cpp -o shape
PS S:\ITI-9month\OOP\lab12> g++ .\v*.cpp -o virual
PS S:\ITI-9month\OOP\lab12> g++ .\complex.cpp -o complex
PS S:\ITI-9month\OOP\lab12> g++ .\complex.cpp -o complex
PS S:\ITI-9month\OOP\lab12> g++ draw.cpp -lgdi32 -luser32 -o draw.exe
PS S:\ITI-9month\OOP\lab12> g++ draw.cpp -lgdi32 -luser32 -o draw.exe
PS S:\ITI-9month\OOP\lab12> 
```

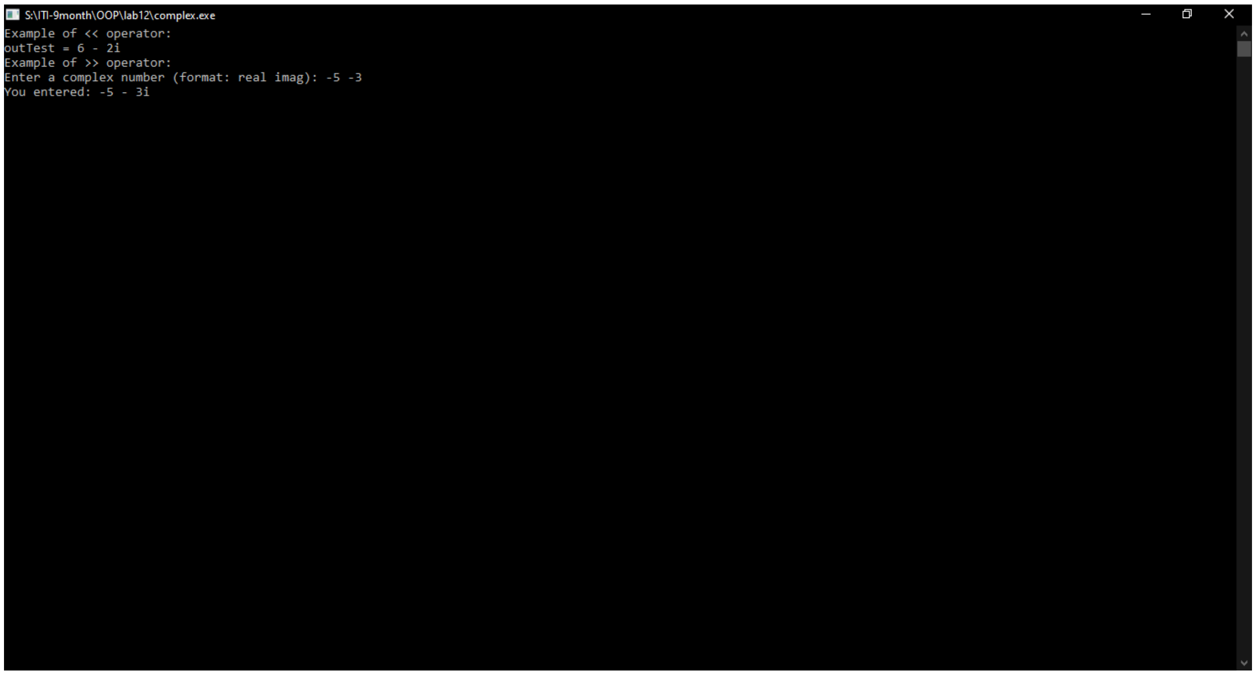
4.
```cpp
#include <iostream>
using namespace std;

class Base {
public:
    Base() { cout << "Base constructor\n"; }

    // virtual destructor
    virtual ~Base() {
        cout << "Base destructor\n";
    }
};

class Derived : public Base {
private:
    int* data;

public:
    Derived() {
        data = new int[5];  // allocating dynamic resource
        cout << "Derived constructor (allocated memory)\n";
    }

    ~Derived() {
        delete[] data;
        cout << "Derived destructor (freed memory)\n";
    }
};

int main() {
    Base* ptr = new Derived();  // Base pointer to Derived object

    cout << "\nDeleting object through Base pointer...\n";
    delete ptr;  // Calls Derived destructor THEN Base destructor

    return 0;
}
```



```
Base constructor
Derived constructor (allocated memory)

Deleting object through Base pointer...
Derived destructor (freed memory)
Base destructor
```

5.

```cpp
#include <iostream>
#include <cmath>
#include <cstring>

using namespace std;

class Complex {
private:
    double real;
    double imag;

public:
    Complex() : real(0), imag(0) {}

    Complex(double _real, double _imag) : real(_real), imag(_imag) {}

    Complex(double _real) : real(_real), imag(0) {}

    Complex(const Complex &c) : real(c.real), imag(c.imag) {}

    ~Complex() {}

    void setReal(double _real) { real = _real; }
    void setImag(double _imag) { imag = _imag; }

    double getReal() const { return real; }
    double getImag() const { return imag; }

    // ostream<<complex
    friend ostream& operator<<(ostream& out, const Complex& c) {
        if (c.real == 0 && c.imag == 0)
            out << "0";
        else if (c.real == 0)
            out << c.imag << "i";
        else if (c.imag == 0)
            out << c.real;
        else if (c.imag > 0)
            out << c.real << " + " << c.imag << "i";
        else
            out << c.real << " - " << -c.imag << "i";

        return out;
    }
    // istream>>complex
    friend istream& operator>>(istream& in, Complex& c) {
        in >> c.real >> c.imag;
        return in;
    }
};

int main() {
    cout << "Example of << operator: \n";
    Complex outTest(6, -2);
    cout << "outTest = " << outTest << endl;

    cout << "Example of >> operator: \n";
    Complex inTest;
    cout << "Enter a complex number (format: real imag): ";
    cin >> inTest;
    cout << "You entered: " << inTest << endl;

    return 0;
}
```



S:\ITI-9month\OOP\lab12\complex.exe

```
Example of << operator:
outTest = 6 - 2i
Example of >> operator:
Enter a complex number (format: real imag): -5 -3
You entered: -5 - 3i
```