Term Project: BMP Plotter (20 points)

Deadline

2024/12/23 (Wed) 23:59

Note

1. Embrace encapsulation. Use access specifiers wisely. If a member should not be accessible from the outside of the class, mark it as protected or private. Poor encapsulation will cause a point deduction.
2. Handle wisely. We will run memory leak detection on your code. Memory leak will lead to point deduction.

Introduction

BMP is a simple raster graphics image file format designed to store bitmap digital images independently of a display device, originally and primarily on WSL operating systems. In this final project, you are going to implement a BMP plotter for drawing simple geometric shapes.

Overview

Classes

The following table describes the objective for each class.

Class	Objective
Point	Point (x, y), the atomic plotting element on a Bitmap.
RGB	Color (r, g, b) for every object (point, circle, rectangle, or triangle) on a Bitmap.
Bitmap	The canvas for plotting.
Base	An abstract class for geometric objects (circle, rectangle, or triangle) on a Bitmap.
Circle	The class for circles.
Rect	The class for rectangles.
Tri	The class for triangles.
Oval	The class for ovals.
Diamond	The class for diamonds.
Handle	The handle class for Base.
Desktop	The final API for the BMP Plotter.

The following are all the given header files.

- bitmapbase.h
- base.h
- desktop.h

Grading policy

Objective	Score
Correctly implement `point.h` and `rgb.h`	1
Correct inheritance	1
Correctly implement `drawSolidTriangle` function	1
Correctly implement `drawSolidRect` function	1
Correctly implement `drawSolidCircle` function	1
Correctly implement `drawGradientOval` function	1
Correctly implement `drawSolidDiamond` function	1

Objective	Score
Correctly implement `Base* copy()` in `base.h`	1
Class `Handle` constructor	1
Class `Handle` copy constructor	1
Class `Handle` arithmetic operations (`+` , `*` , `+=` , `*=`)	2
Class `Handle` drawing operations (`setColor` , `draw`)	2
Class `Handle` shallow copier (overloading `=`)	1
Handle copy on write	2
Compilable	1
No memory leak	2

Files to submit

Put all and only the following 12 files in a folder named after your student ID (**lower** case), e.g., `b13901999`. Then, **zip this folder** (again, named after your student ID) and upload it to the designated location in NTU COOL.

1. point.h
2. rgb.h
3. base.h (**DO NOT MODIFY**)
4. bitmapbase.h (**DO NOT MODIFY**)
5. bitmap.h
6. tri.h
7. rect.h
8. circle.h
9. oval.h
10. diamond.h
11. handle.h
12. desktop.h (**DO NOT MODIFY**)

Note: **DO NOT SUBMIT** the main.cpp. We will run our own main.cpp, which includes `desktop.h`. See the example in the Handle section.

```
b13901999.zip    --- unzip -->    b13901999
                                  ├── point.h
                                  ├── rgb.h
                                  ├── base.h
                                  ├── bitmapbase.h
                                  ├── bitmap.h
                                 (└── other additional files)
```

Point

Implement class `Point` in `point.h`. A `Point` object is isomorphic to $(x, y) \in \mathbb{Z} \times \mathbb{Z}$ that supports:

- Translation (`+=`)
- Translation (`+`)

where addition is applied elementwise, that is, $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$. The following code snippet is an example of how we can use this header file.

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

```
int main() {
    Point p1(1, 2), p2(3, 4);
    // Overload <<
    cout << "p1=" << p1 << " p2=" << p2 << endl;
    Point p3(p1);
    cout << "p3=" << p3 << endl;
    p3 = Point(5, 6);
    cout << "p3=" << p3 << endl;
    // Overload +
    cout << "p1+p2=" << p1 + p2 << endl;
    // Overload +=
    p1 += p3;
    cout << "p1=" << p1 << endl;
    return 0;
}
```

Expected Output

```
p1=(1, 2) p2=(3, 4)
p3=(1, 2)
p3=(5, 6)
p1+p2=(4, 6)
p1=(6, 8)
```

RGB Color

Implement class `RGB` in `rgb.h`. This header file supports BMP coloring. A `RGB` object is isomorphic to (r, g, b) where each element is a nonnegative integer in $\{0, 1, ..., 255\}$. The following code snippet is an example of how we can use this header file. Do not worry about that color values might overflow. We have handled this for you in the `setPixel` functions in `bitmapbase.h`.

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

int main() {
    RGB a(155, 165, 175);
    cout << a << endl;
    cout << a + RGB(10, 10, 10) << endl;
    cout << a - RGB(10, 10, 10) << endl;
    RGB c(a);
    cout << c << endl;
    return 0;
}
```

Expected Output

```
(155, 165, 175)
(165, 175, 185)
(145, 155, 165)
(155, 165, 175)
```

BitmapBase and Bitmap

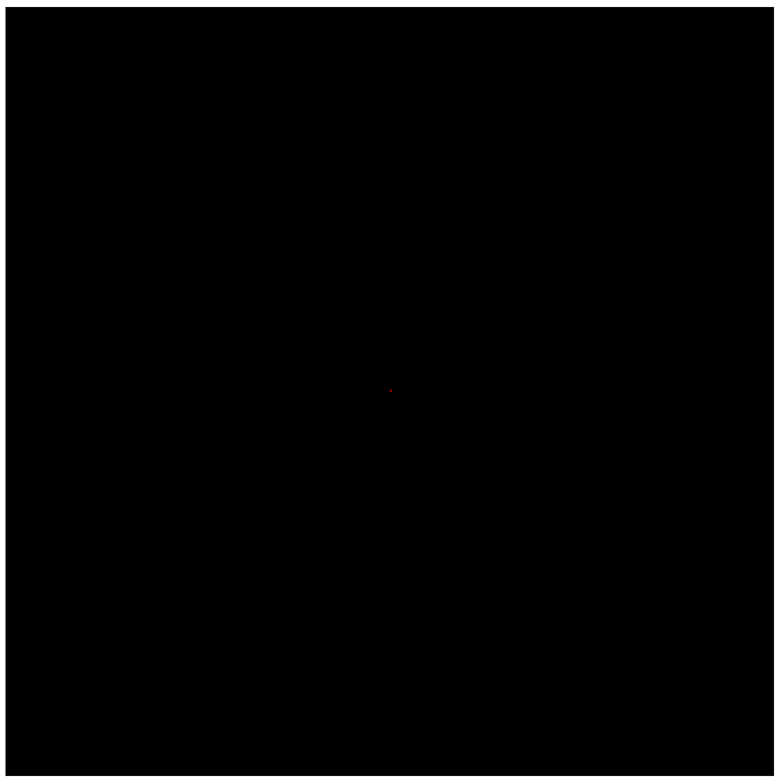
The `bitmapbase.h` header file contains an abstract class `BitmapBase` which accounts for generating BMP files and plotting dots. **Do not modify this file.** The `xSize` and `ySize` of the map is required to be a multiple of 4, and the center of the Bitmap has coordinate $(0, 0)$. Use the `setPixel` functions to plot a dot. Notice there are three methods (-1, 0, 1) for plotting dots. There are five pure virtual functions in `BitmapBase`:

```
virtual void drawSolidTriangle(const Point& ref, int left, int right, int height,
                         const RGB& c, int m) = 0;
virtual void drawSolidRect(const Point& base, int width, int height,
                         const RGB& c, int m) = 0;
virtual void drawSolidCircle(const Point& center, int radius, const RGB& c, int m) = 0;
virtual void drawGradientOval(const Point& center, double radiusX, double radiusY,
                         const RGB& c, const RGB& c2, int m) = 0;
virtual void drawSolidDiamond(const Point& center, double width, double height,
                         const RGB& c, int m) = 0;
```

which accounts for plotting triangles (`tri.h`), rectangles (`rect.h`), circles (`circle.h`), oval(`oval.h`), and diamond(`diamond.h`) respectively. Implement class `Bitmap` in `bitmap.h` that inherits `BitmapBase` publicly. A naïve example would be as follow:

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

int main() {
    Bitmap map(400, 400);
    RGB red(255, 0, 0);
    Point center(0, 0);
    map.setPixel(center, red, 0);
    map.save("foo.bmp");
    map.clear();
    return 0;
}
```

This code snipnet generates a BMP file named `foo.bmp`.



Base Class

Do not modify this file. Class `Base` is an abstract class, which has five derived classes, namely `Circle` (circles), `Rect` (rectangles), `Tri` (triangles), `Oval` (ovals), and `Diamond` (diamonds). It has three protected members:

- `Point ref`: The reference point (anchor) of the derived object
- `RGB color`: The color of the derived object
- `int count`: The number of pointers pointing to the derived object

In addition, there are four virtual and pure virtual functions:

- `virtual ~Base() {}`: Virtual destructor
- `virtual void operator*= (double scale) = 0`: Scale the derived object
- `virtual Base* copy() const = 0`: Deep copy the derived object and return the pointer to the new copy
- `virtual void draw(Bitmap& map, int method) const = 0`: Draw the derived object onto the map via the given method

You will have to implement these four functions in all the five derived classes.

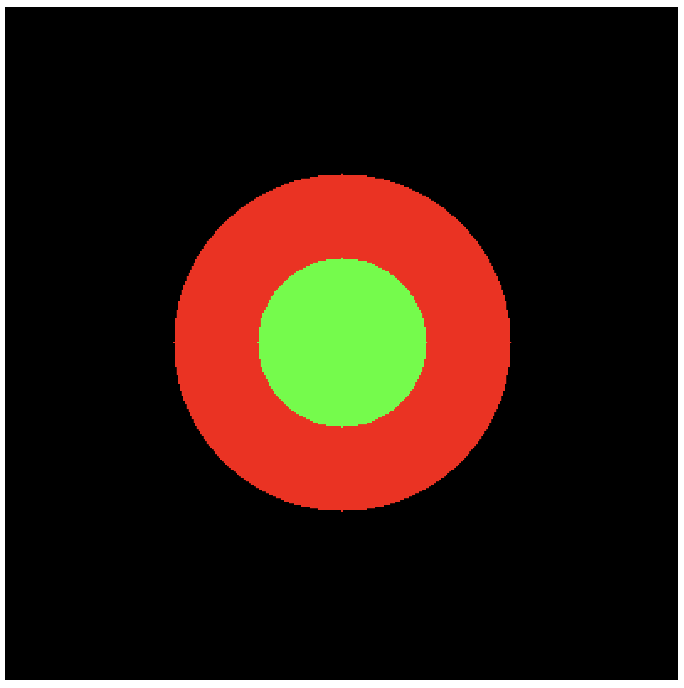
Circle

Implement class `Circle` in `circle.h`. A `Circle` object is determined by the center (the reference point) and radius of the circle. Implement the four virtual functions from the base class.

Example

```
#include "circle.h"
using namespace std;

int main() {
    Bitmap map(400, 400);
    Circle red_circle(Point(0, 0), 100, RGB(255, 0, 0));
    Base* green_circle;
    green_circle = red_circle.copy();
    green_circle->setColor(RGB(0, 255, 0));
    *green_circle *= 0.5;
    red_circle.draw(map, 0);
    green_circle->draw(map, 0);
    map.save("circle.bmp");
    return 0;
}
```

Output (circle.bmp)



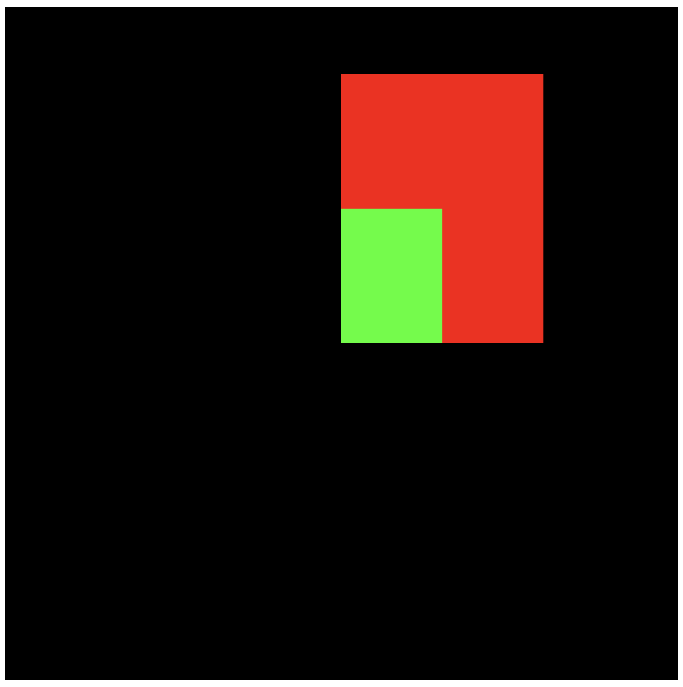
Rect

Implement class `Rect` in `rect.h`. A `Rect` object is determined by the width and height of the rectangle. The reference point of a rectangle is the left-bottom corner of the rectangle. Similar to class `Circle`, implement the four virtual functions from the base class.

Example

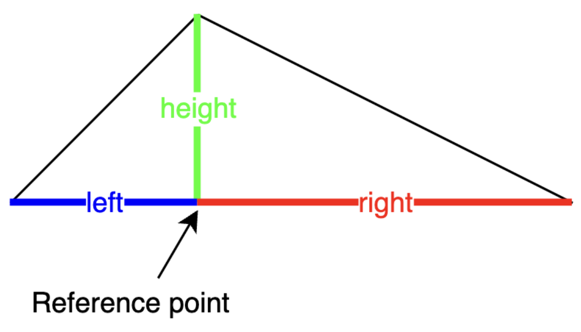
```
#include "rect.h"
using namespace std;

int main() {
    Bitmap map(400, 400);
    Rect red_rect(Point(0, 0), 120, 160, RGB(255, 0, 0));
    Base* green_rect;
    green_rect = red_rect.copy();
    green_rect->setColor(RGB(0, 255, 0));
    *green_rect *= 0.5;
    red_rect.draw(map, 0);
    green_rect->draw(map, 0);
    map.save("rect.bmp");
    return 0;
}
```

Output (rect.bmp)



Triangle

Implement class `Tri` in `tri.h`. A `Tri` object is a triangle determined by `left`, `right`, and `height`. The reference point of a triangle is the projection point of top vertex onto the base edge.



For convenience, there are two restrictions on this triangle class:

1. The base edge is always horizontal.

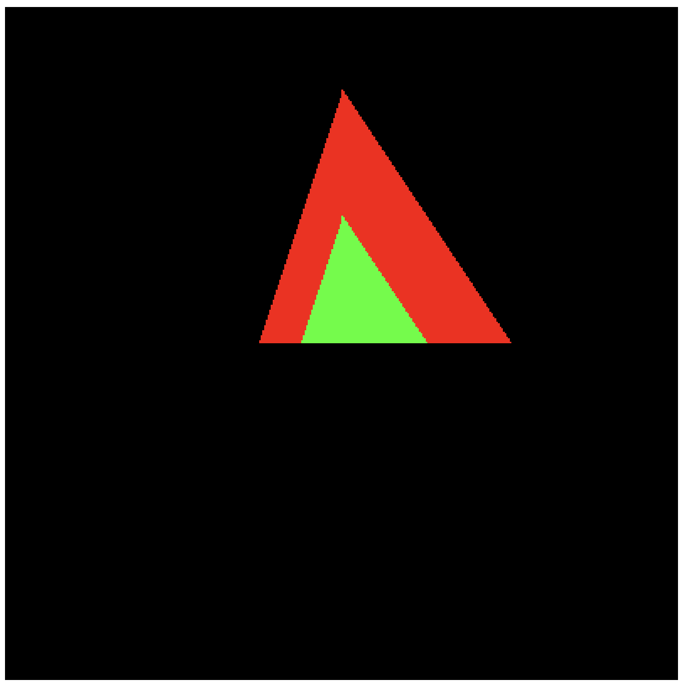
2. The reference point is always on the base edge.

Similar to class `Circle` and `Rect`, implement the four virtual functions from the base class.

Example

```
#include "tri.h"
using namespace std;

int main() {
    Bitmap map(400, 400);
    Tri red_tri(Point(0, 0), 50, 100, 150, RGB(255, 0, 0));
    Base* green_tri;
    green_tri = red_tri.copy();
    green_tri->setColor(RGB(0, 255, 0));
    *green_tri *= 0.5;
    red_tri.draw(map, 0);
    green_tri->draw(map, 0);
    map.save("tri.bmp");
    return 0;
}
```

Output (tri.bmp)



Oval

Implement the class `Oval` in `oval.h`. An `Oval` object is an oval determined by two radii,`radius,X` and `radiusY`. The reference point of an oval is its center.

For convenience, there are two restrictions on this oval class:

1. The `radiusX` represents the horizontal radius.
2. The `radiusY` represents the vertical radius.

Additionally, the `Oval` class supports gradient colors. Implement the `draw` function to handle gradient coloring from `color1` to `color2`.

The color transitions from color1 to color2 based on the distance ratio from the reference point of the oval (normalized distance to a range of `[0, 1]`):

$$\text{distance ratio} = \sqrt{\frac{x^2}{\text{radiusX}^2} + \frac{y^2}{\text{radiusY}^2}},$$

and the relationship between color (r, g, b) and distance ratio is given by the following formula:

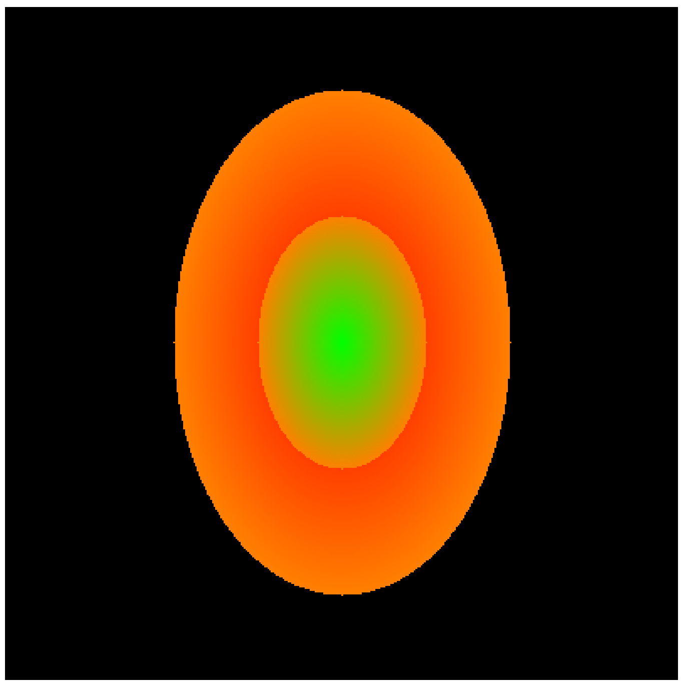
$$\text{color\_gradient} = \text{color1} + (\text{color1} - \text{color2}) \times \text{distance ratio}$$

Similar to class `Circle` , `Rect` , and `Tri` , implement the four virtual functions from the base class.

Example

```
#include "oval.h"
using namespace std;

int main() {
    Bitmap map(400, 400);
    Oval red_grad_oval(Point(-30,-30), 100.0, 150.0, RGB(255,0,0), RGB(255,128,0));
    Base* green_grad_oval;
    green_grad_oval = red_grad_oval.copy();
    green_grad_oval->setColor(RGB(0, 255, 0));
    *green_grad_oval *= 0.5;
    red_grad_oval.draw(map, 0);
    green_grad_oval->draw(map, 0);
    map.save("oval.bmp");
    return 0;
}
```

Output (oval.bmp)



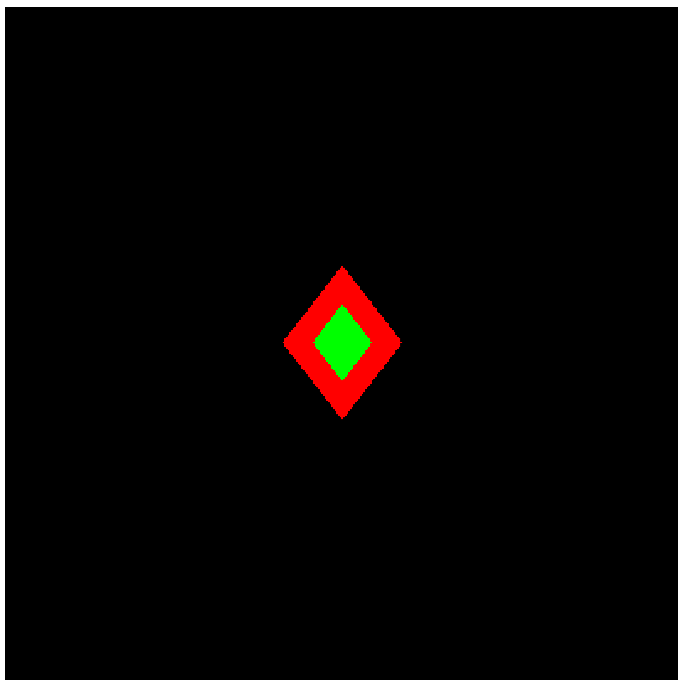
Diamond

Implement class `Diamond` in `diamond.h` . A `Diamond` object is a diamond shape defined by width and height. The reference point of the diamond is the center point of the diamond. The diamond is aligned such that the top and bottom vertices lie on the vertical axis, and the left and right vertices lie on the horizontal axis.
Similar to class
`Circle` , `Rect` , `Tri` , and `Oval` , implement the four virtual functions from the base class.

Example

```
#include "diamond.h"
using namespace std;

int main() {
    Bitmap map(400, 400);
    Diamond red_diamond(Point(-100,-30), 50.0, 70.0, RGB(255,0,0));
    Base* green_diamond;
    green_diamond = red_diamond.copy();
    green_diamond->setColor(RGB(0, 255, 0));
    *green_diamond *= 0.5;
    red_diamond.draw(map, 0);
    green_diamond->draw(map, 0);
    map.save("diamond.bmp");
    return 0;
}
```

Output (diamond.bmp)



Handle

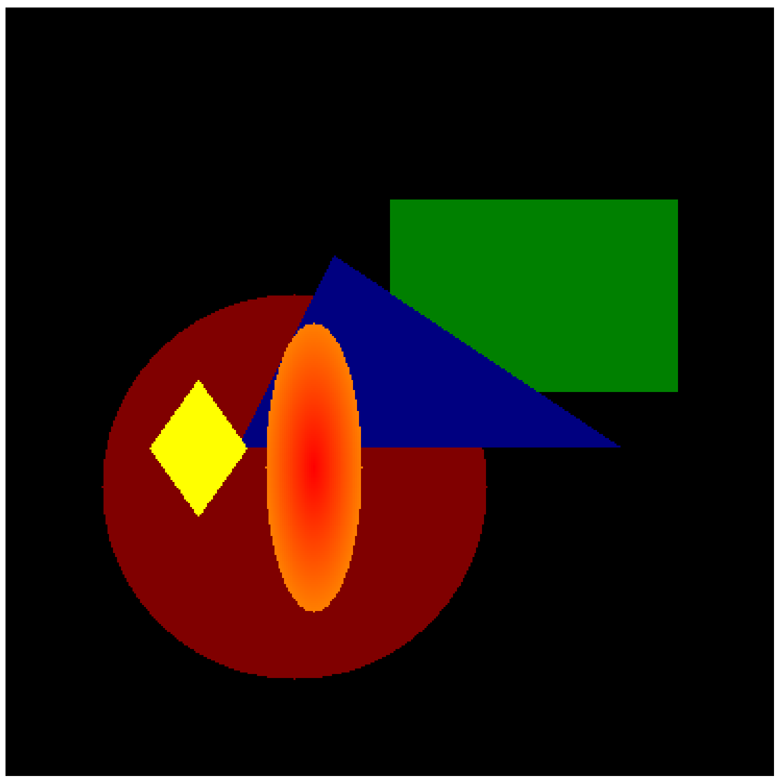
Implement Class `Handle` to handle class `base`. First, read `desktop.h`, which is a wrapper of `bitmap.h`. Then, implement `handle.h` to support the following functionalities.

```
#include <iostream>
#include "point.h"
#include "bitmap.h"
#include "handle.h"
#include "desktop.h"

using namespace std;

int main() {

    // Circle
    Handle a(Point(-50,-50), 100, RGB(128,0,0));
    // Rect
    Handle b(Point(0, 0), 150, 100, RGB(0,128,0));
    // Triangle
    Handle c(Point(-30,-30), 50, 150, 100, RGB(0,0,128));
    // Oval
    Handle d(Point(-40,-40), 25.0, 75.0, RGB(255,0,0), RGB(255,128,0));
    // Diamond
    Handle e(Point(-100,-30), 50.0, 70.0, RGB(255,255,0));


    Desktop desk(400,400);
    desk << a << b << c << d << e;
    desk.save("1.bmp");

    desk.clear();
    desk + a + b + c + d + e;
    desk.save("2.bmp");
    // shallow copy
    b = a;

    // copy-on-write
    b += Point(50,0);

    b *= 1.2;

    d *= 0.5;
    d.setColor(RGB(255,0,0));

    desk.clear();
    desk + a + b - d;
    desk.save("3.bmp");

    // No problem with using array
    desk.clear();
    int num = 30;
    Handle *list = new Handle[num];

    list[0] = Handle(Point(0,0), 300.0, 300.0, RGB(255,0,0));
    double u = 255.0;
    for (int i=1; i<num; ++i) {
        list[i] = list[i-1] * 0.9;
        u *= 0.9;
        list[i].setColor(RGB(int(u),0,0));
    }

    for (int i=0; i<num; ++i)
        desk + list[i] - (list[i]*0.95);

    desk.save("tunnel.bmp");
    delete []list;

    return 0;
}
```

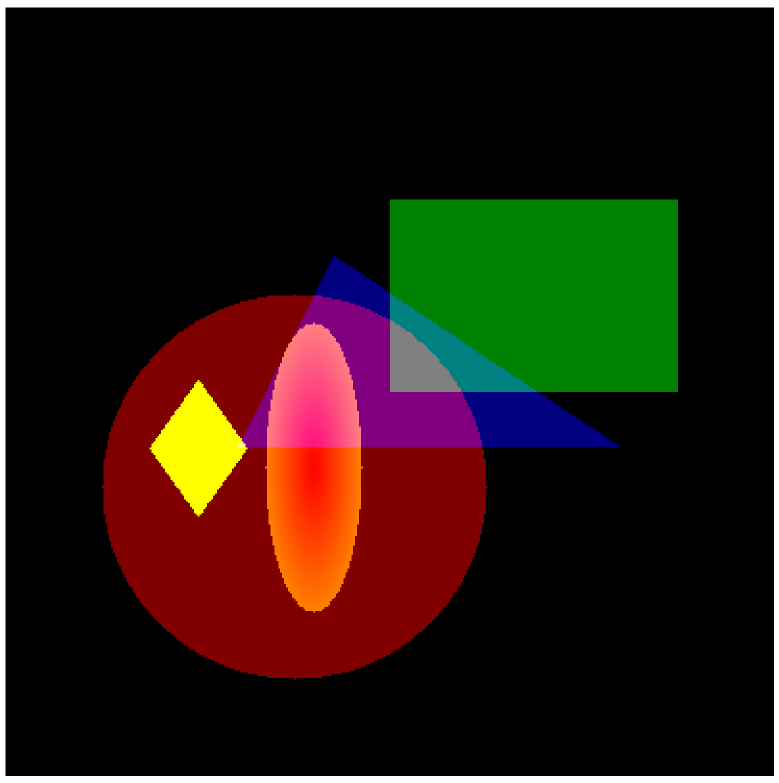
This code snippet is an example that will be used to test your final result. That is to say, every example except this one is intermediate, meaning that it is fine that they are not compilable once you finish `handle.h`, since there exists some expected modification of the the previous header files.

Output

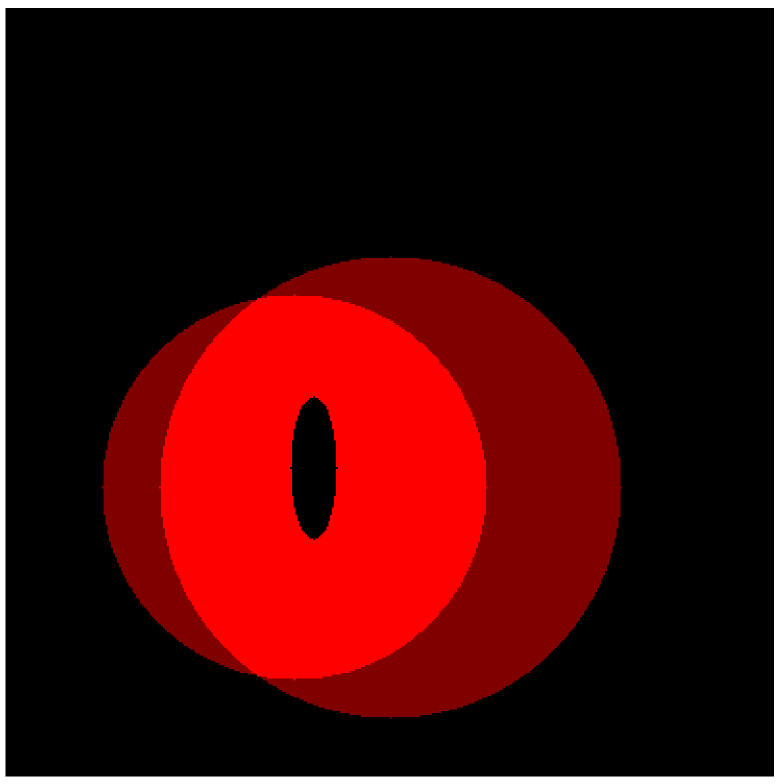
1.bmp



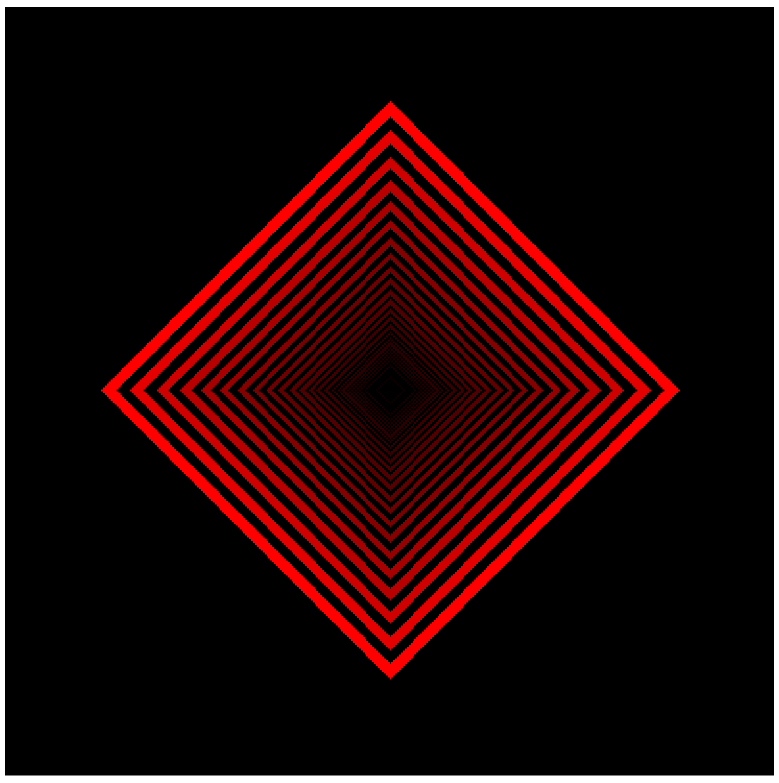
2.bmp



3.bmp



tunnel.bmp



Desktop

The class `Desktop` is a wrapper of the class `Bitmap` (figure) and `Handle` (objects to plot). Do not modify this file.