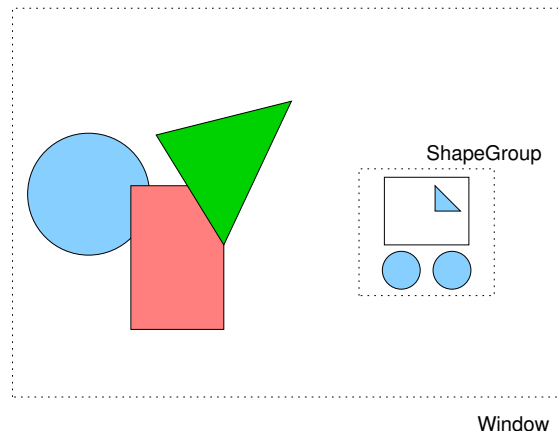


CEng 242 Homework 5

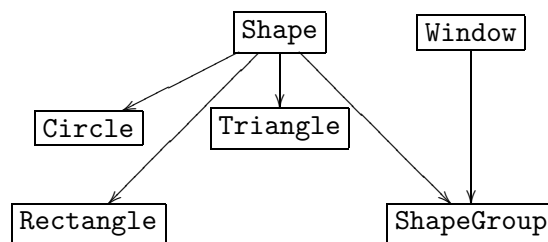
Due 21st May 2003

In this homework you will implement a simple back-end for a drawing package. In this package there are 3 basic shapes **Rectangle**, **Circle** and **Triangle**. Also it is possible to group shapes in **ShapeGroup** objects.

Drawing tool works in a **Window** providing the events and user interaction. Each shape has an offset (position) within the window by x and y values. In addition it has a *depth* value determining the ordering of the intersecting shapes. So that when a window is drawn shapes will be drawn from the largest depth to smallest one. In addition each shape contains a *color* attribute. In the following picture there is a **Window** with a **Circle**, a **Rectangle**, a **Triangle**, and a **ShapeGroup**, where **ShapeGroup** has a **Rectangle**, a **Triangle** and 2 **Circles**. Depth of the triangle is the smallest and the circle is the largest so that they are ordered in this way.



In the homework, you will use the following class hierarchy:



Class **Shape** contains the following members:

```
struct Point { int x, int y;};
enum Color {WHITE,RED,BLUE,GREEN,MAGENTA,CYAN,YELLOW,PINK,BROWN,ORANGE};
struct Event {
    enum evttype {INSIDE,DRAW,RAISE,LOWER,SCALE,MOVE,SWITCH} ev;
    Point evpoint;
    union {    // Anonymous union, all members are available to Event
        Point point;
        double scale;
    };
};
```

```

class Shape {
protected:
    int x,y;
private:
    int depth;
    Color color;
public:
    Shape(int,int,int,Color);
    virtual int inside(Point) = 0;
    virtual void draw() = 0;
    virtual void scale(double) = 0;
    void move(Point);
    Color getcolor();
    int getdepth();
    virtual int handle(Event);
};

```

In subclasses all virtual functions will be implemented. `inside()` returns true if the given point is inside the boundaries of the shape. `draw()` prints a message on standard output about the primitives to be used.

The other member functions are described as follows:

`Shape(int x, int y,int d,Color c)`

Constructs the shape in the offset (*x,y*), depth *d* and color *c*.

`move(Point p)`

Sets the offset of the object to the new point *p*.

`getcolor()`

Returns the color of the shape.

`getdepth()`

Returns the depth of the shape.

`handle(Event ev)`

Handles the given event by accessing the appropriate members. Handler meanings are described below .

The other shapes implement the `inside()` function according to their types (i.e. a circle implements it by checking the distance of the point is less than the radius). They have the following members:

Circle: `int radius;`

 constructor will be `Circle(x,y,r,depth,color)`

Rectangle: `int height,width;` constructor will be `Rectangle(x,y,width,height,depth,color)`

Triangle: `Point p2,p3;`

p2 and *p3* are 2 coordinates relative to the shape offset (*x,y*), representing the second and third corners of the rectangle.

 constructor will be `Triangle(x,y,p2.x,p2.y,p3.x,p3.y,depth,color)`

`scale(double s)` operation changes all relative coordinates of the shape: `r` for `Circle`, `width` and `height` for `Rectangle`, two point coordinates for `Triangle` are multiplied and rounded by the scale factor `s`.

`Window` class is like a container of several shapes and it has a definition like the following:

```
class Window {
protected:
    int width,height;
private:
    Shape *shapes[100];
    int nshapes;
public:
    Window(int,int);
    ~Window();
    void addShape(Shape &);
    void redraw();
    int handle(Event);
};
```

`Window` is constructed with an empty list of `Shapes` kept in `shapes` and `nshapes` members. `addShape(Shape & s)` adds a new shape `s` in the window. `redraw()` draws all of the shapes in the windows. In order to display the depths correctly, it should draw the objects sorted in descending order of depth. `handle(Event ev)` will send the event to the shape containing the event coordinates. In order to do so `inside()` member function will be used. Again, since a point can be inside of multiple shapes the topmost (smallest depth) shape will be tried first. The first shape returning a true inside report will be capture the event (Its `handle()` method will be called).

`ShapeGroup` class is a container of other shapes but it is a shape itself. So it inherits from both `Window` and `Shape`. A `ShapeGroup` has an additional boolean member called `eventpass` (0 as default). When `eventpass` is true, it passes all events that it is taking to its subshapes, transparently. When `eventpass` is false, it captures and handles the event itself. There is no theoretical limit on the nesting of `ShapeGroups`. Inside test for a `ShapeGroup` is same with the rectangle defining the `Window` of it. Constructor of `ShapeGroup` will be `ShapeGroup(x,y,width,height,depth)`.

Events

For all events, if event is handled by the `Window`, the `Window` determines the shape containing the event point stored at the `Event::evpoint` by making the `inside()` tests (from smallest to largest depth value) and sends the event to handler of the corresponding shape. If no shape returns true for inside test, reports that the event is not handled by returning 0. Otherwise, if a window matches, returns 1. Similarly if the event point is outside of the window area (event point dimensions are greater than width and/or height).

Shapes also return 1 for a succesful handle case and 0 for not successful.

DRAW

`Shape` subclasses simply call their self `draw()`.

RAISE

`Shape` subclasses simply decrement the `depth`.

LOWER

Shape subclasses simply increment the depth.

SCALE

Shape subclasses call their self `scale()` with the `Event::scale` value in the event structure.

MOVE

`Window` takes the relative move position from the `Event::point` value in the event structure.

Shape subclasses set their offset with the new offset value given in the `Event::point` value in the event structure similarly.

SWITCH

Only handled by a `ShapeGroup` type shape. It changes the `eventpass` flag. Other shapes return 0 for this event.

A `ShapeGroup` class object handles events according to `eventpass` member. If it is true (non-zero), it passes all events as if it were a `Window`, it only modifies the `Event::evpoint` value in the event, converts it to offset relative (all shapes included in the group has relative offset and depths) values. In case of a `SWITCH` event, if `eventpass` is false, makes it true. If it is true, it first tries to send this event to its window. If the matched shape does not handle it or no shape matches the point, it makes the `eventpass` false.

When `eventpass` is false, it handles events to modify its `Shape` properties. In case of `SCALE`, it only changes the `height` and `width`, does not reflect the scale operation on the contained shapes.

draw() functions

Draw functions for different shapes produces different outputs:

Circle

```
drawCircle(x,y,r,Color)
for example 'drawCircle(10,10,7,BLUE)'
```

Rectangle

`drawRectangle(x,y,x+width,y+height,Color)`
for example '`drawRectangle(10,5,30,20,RED)`'

Triangle

```
drawPolyLine(x,y,p2.x,p2.y,p3.x,p3.y,x,y,Color)
for example 'drawPolyLine(5,5,10,15,15,10,5,5,GREEN)
```

ShapeGroup

ShapeGroup first outputs `translateoffset(x,y)`. Then it calls the `redraw()` of its Window superclass to draw contained shapes. Then it outputs `translateoffset(-x,-y)` for example:

```
translateoffset(100,50)
drawCircle(5,8,4)
drawRectangle(20,40,30,50)
translateoffset(-100,-50)
```

You should try to use minimum number of additional member functions, minimum member access permissions and your object oriented design should be as elegant as possible. No public member should be added to classes in the given hierarchy. Your programs will also be evaluated by your class definitions.

Similar to the previous homework, put only class prototype (no inline definition) and required type definitions in `hw5.h`, include this file in `hw5.cpp` and in your for yourself only main program code `hw5main.cpp`. Your codes should be compiled as:

```
g++ -c hw5.cpp
```

```
g++ -o hw5main hw5main.cpp hw5.o
```

Sample main code will be given in the following days.