Project 6

PROGRAMMING AND ALGORITHMS II
CSCI 211, FALL 2018

Objectives

- Practice using Inheritance
- Practice using Polymorphism
- Use a `virtual` function
- Practice with 2D arrays

Overview

The goal of this program is to provide practice in implementing inheritance in C++.

You will create 5 shape classes: Shape (the base class), Square, Point, Triangle, Circle, and a Grid class. Each shape class will draw itself onto a Grid object.

PROTIP: Start early and work daily.

You are given the main() so you do not have to worry about parsing the program input. You are also given a Makefile. Download these from BBL.

The main program will read shape descriptions (square, point, triangle, circle) and draw them onto a Grid. It will then draw the grid. For example, the following input:

```
square 0 0 24
square 2 5 4
triangle 10 10
circle 5 16
point 15 3 ?
```

Must produce the output:

```
************************
*                      *
*                      *
*               ?      *
*                      *
* ****                 *
* *  *                 *
* *  *                 *
* ****                 *
*                      *
*           +          *
*          + +         *
*         +++++        *
*                      *
*                      *
*                      *
*     oo               *
*    o  o              *
*    o  o              *
*     oo               *
*                      *
*                      *
*                      *
************************
```

Program Requirements

You must be careful to use the exact structure, functions names and filenames or your program will not compile with the main I provide.

Class Grid contains a private 60x24 (60 wide and 24 high) grid of characters (a 2D array of char). Initially the grid will be initialized so that all characters are spaces (this is done in Grid's only constructor `Grid()`). Class `Grid` must provide a set function so that the shapes can set individual characters in the grid. This function could have the prototype: `void Grid::set(int x, int y, char c)`. If the `(x,y)` values are inside the grid, set the `(x,y)` character in the grid to character `c`. If the `(x,y)` values are outside the grid, do nothing.

Class Grid must also provide a print function: `void Grid::print()`. This function must draw the grid to standard output. `Grid::print()` should first draw all the characters in row 0 followed by an `endl`. Then all the characters in row 1 followed by an `endl`. And so on.

Class Shape must be used as the base class for classes Square, Triangle, Circle, and Point. It must store the `x` and `y` location that the shape is to be drawn (I called the variables `m_x` and `m_y`). It will have a single constructor that takes `x` and `y` as arguments. In addition to the constructor, it will have a single member function `draw()` that must have a single argument, a reference to a Grid object (`void draw(Grid &grid)`). This function must be a pure virtual function (learn this term, it will be on the next exam).

You can make a virtual function into a pure virtual function by assigning it to zero in the base class declarations. Pure virtual functions MUST be overridden by the sub class before the sub class can be instantiated.

```
class Shape
{
    public:
        ...
        virtual void draw(Grid &grid) = 0;  // this is a pure virtual function
        ...
};
```

Class Square will inherit class Shape. It will provide a single constructor that will have three integer arguments: `x`, `y`, and `size`. It must also implement the `draw()` function. The `draw()` function will draw a size `x` size square into the Grid using the character '*'. For example, if `size = 4`, it would draw:

```
****
*  *
*  *
****
```

The `x,y` will determine where in the Grid the rectangle is drawn. For example, if `x = 5, y = 6`, the * in the top left corner of the square would be drawn at grid location 5,6. The square (and all the other shapes) will draw into the grid by calling `Grid::set()`. The * in upper left corner of this square would be drawn by calling `grid.set(5,6,'*')`. The next * would be drawn by `grid.set(6,6,'*')`.

Class Square can be used like this:

```
int main()
{
    Grid my_grid;
    // Square is located at x = 5, y = 7 and is 10x10 characters in size
    Square my_square(5,7,10);
    my_square.draw(my_grid);
    my_grid.print();
}
```

Class Triangle works just like class Square, but all triangles are the same size, and it uses the character '+'. Thus, the constructor only takes `x` and `y`: `Triangle::Triangle(int x, int y)`. Triangles must be drawn as shown below.

```
  +
 + +
+++++
```

Position the triangle so its top is in row y and its left most `+` is in column x. The # in the next diagram shows where the (x,y) point is.

```
# +
 + +
+++++
```

Class Circle works just like class Triangle, but uses 'o' (the lowercase letter o, not the number 0). All circles are the same size and must be drawn as shown below. Once again, put its top row in row `y`, and its leftmost column in col `x` (just like with triangle).

```
 oo
o  o
o  o
 oo
```

Class Point's constructor takes the same `(x,y)` as the other shapes and a character: `Point(int x, int y, char c)`. It will draw the single character (`c`) at the `location (x,y)`.

Tips & Tricks

This program has many classes and files. However, they are all very simple. Once you get one shape working, creating the other shapes is easy. Make sure you get one shape completely working before you start creating the others.

PROTIP: It is best to start with the square.

The origin (x = 0, y = 0) of the grid is at the upper left. This makes the grid easy to draw. As an example, a square of size 2 drawn at (x,y) = (0,0) would occupy the following spots in the grid:

```
0,0
0,1
1,0
1,1
```

General Requirements

The first lines of all your files (both .h and .cpp) must contain the following comments:

```
// filename
// last name, first name
// ecst_username
```

Programs without your name in all files will not be graded.

Comment your program. You should have block comments that explain each function. You should also comment for each line or short code block.

Format your code. Make sure your code blocks align. There is no required indentation strategy, but be consistent throughout your program.

Use descriptive variable names. Avoid ambiguous or short variable names, with the exception of loop counters (e.g., `i, j, k`).

Use a capital letter to name classes in your code (e.g., `Grid`). Use all lower case for your file names (e.g., `grid.h`).

Submission

Submit your program at Turn-in. This assignment is worth 125 points.

`grid.h`	`grid.cpp`
`shape.h`	`shape.cpp`
`triangle.h`	`triangle.cpp`
`square.h`	`square.cpp`
`circle.h`	`circle.cpp`
`point.h`	`point.cpp`

PROTIP: Submit only your C++ files. Do not submit your object file or your executable program. Do not archive (e.g., `zip`) your files.

Each student will complete and submit this assignment individually. Submit your work before the deadline as late work is not accepted (except using your late days).

PROTIP: If you do not finish in time, turn in whatever work you have. If you turn nothing, you get a zero. If you turn in something, you receive partial credit.