# Project 6

Programming and Algorithms II CSCI 211

## Objectives

- Practice using Inheritance
- Practice using Polymorphism virtual methods
- Practice with 2-dimensional arrays

## Overview

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

**Start early and make daily progress.**

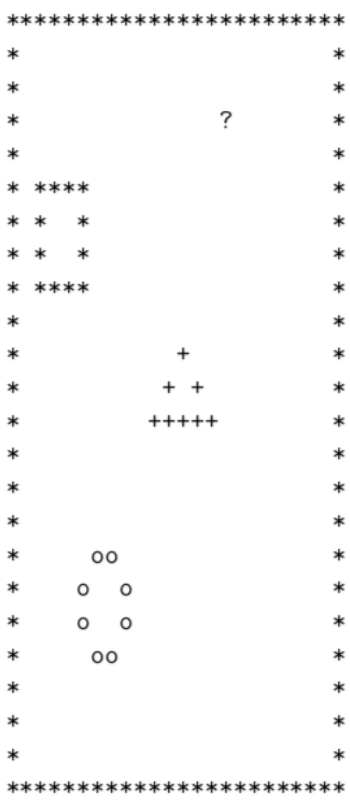
The starter pack includes main.cpp with all input code implemented (you don't need to implement the input code this time). You are also given a Makefile, grid.h, and grid_test.cpp, which shows a simple example using class Grid, which you can use as a guide for implementing all of your shape classes.

Continued on next page…

The main function reads shape descriptions (square, point, triangle, circle) and draws them onto a 2-dimensional Grid (character array) via a draw method. When all the shapes have been drawn onto the grid the program will then output the grid to stdout via a Grid::print method.
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:



Note: The outer "border" shown above is actually just the "square 0 0 24" object rendering. You don't need to draw a separate border. Also note that squares won't really look square for the output because of the differences in horizontal and vertical spacing for character output.

Continued on next page…

# Program Requirements

You must use the exact structure, function names and filenames given below or your program will not compile with the provided main.cpp & Makefile. You are not allowed to change main.cpp.

Class Grid contains a private 60x24 (60 wide and 24 high) grid of characters (a 2D array of char).

**grid.h, included in the starter pack, includes the complete definition for class Grid. Don't forget to add your name etc. to the comment at the top of the file before submission.** You need to create and implement grid.cpp.

Initialize Grid so that all characters are spaces. Class Grid must provide a set method so that the shapes can set individual characters in the grid. The set method 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 method: void Grid::print(). This method must output the grid to standard out. Grid::print() must 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. Shape must store the x and y origin that the shape is to be drawn from (m_x and m_y). Shape must have a single constructor that takes x and y as arguments. In addition to the constructor, Shape must have a single member function draw() that must have take single argument, a reference to a Grid object (void draw(Grid &grid)). The Shape::draw method must be a 'pure virtual' method.

You can make a **virtual** method into a **pure virtual** method by assigning it to 0 in the base class declaration, as shown below. Pure virtual methods MUST be overridden by any inheriting subclasses (they just declare an interface that must be implemented by the subclass).

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

**Warning: Don't copy / paste directly from this document (or any PDF file) into your source code. You can end-up with hidden non-ASCII characters that will cause TurnIn to think your source code files are in binary / executable format, and TurnIn won't accept them.**

Class Square inherits from class Shape. Square must provide a single constructor that has three integer arguments: x, y, and size. Square must also override the base-class's virtual draw() method. The draw() method will draw an x by x size square into the Grid using the character '*'.
For example, if size = 4, Square::draw would draw the following shape onto the Grid:

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

The x,y pair will determine the rectangle origin. 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 the square shown above would be drawn onto the grid by calling grid.set(5,6,'*'). The next * would be drawn onto the grid by calling grid.set(6,6,'*').

Continued on next page…

Class Square can be used like this:

```
Grid grid;
// Square is located at x = 5, y = 7 and is 10x10 characters in size
Square square(5,7,10);
square.draw(grid);
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.

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 left-most 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).

Continued on next page…

# General Requirements

1. Avoid code duplication.
2. Use a **single** return statement for all functions & methods that return a value.
3. Follow these variable naming rules:
    - Local variables are all lowercase, with underbars between words, e.g. my_array.
    - Class member variables begin with m_ and are all lowercase, with underbars between words, e.g. m_count.
    - Class names and Struct names are "camel case", starting with an uppercase letter, e.g. IntStack.
    - Constants are ALL CAPS.
4. Use descriptive names for all classes, structs, functions, methods, parameters, variables and types.
5. Do not declare any non-constant **global** variables.
6. Do not submit commented-out code.
7. Do not submit unused / un-called code.
8. Code blocks are properly aligned.
9. No more than three submissions to TurnIn (use run_tests to validate locally). Come see me if you are having trouble with your TurnIn submissions (e.g. all local tests are passing but not all TurnIn tests are not passing).

## 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
```

Each student will complete and submit this assignment individually.