ECS40 Winter 2017 2017-02-08

Homework 2 due Fri 2017-02-17 at 20:00

Use the handin directory hw2 to submit your work

Shape editor

Note: the concepts and software developed in Homework2 may be used in Homework3.

Description

In this assignment you will write a program that manipulates geometric shapes.

We will consider the following set of shapes, named O, I, L, S, X, U (the names are reminiscent of the shapes). We can represent shapes by drawing the squares forming the shape as shown in Figure 1:

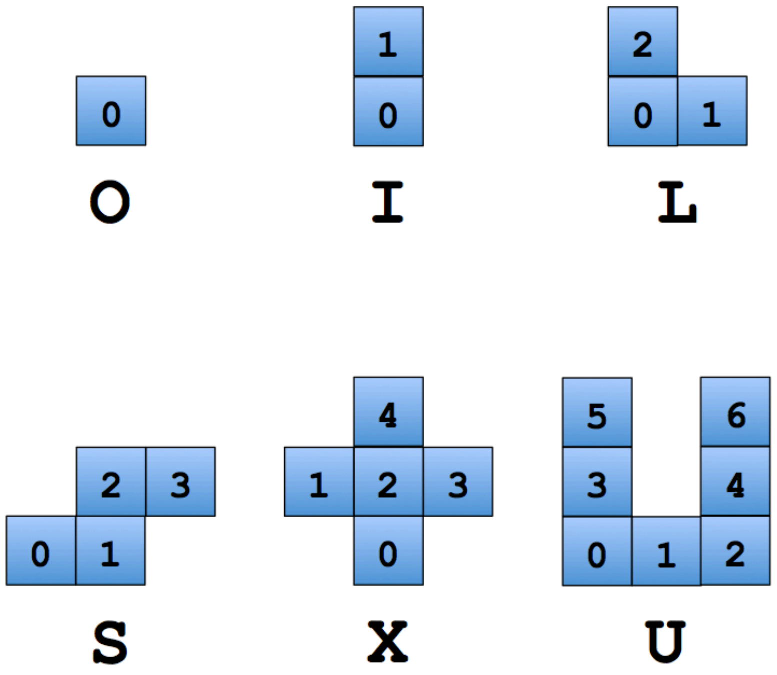

Figure 1

The cells (or squares) forming a shape are indexed by a number starting from 0, as shown in Fig 1. A shape can be placed on a board by specifying the position of its "zeroth cell" in the x-y plane. In Figure 2 below, shapes have been placed on a board as follows:

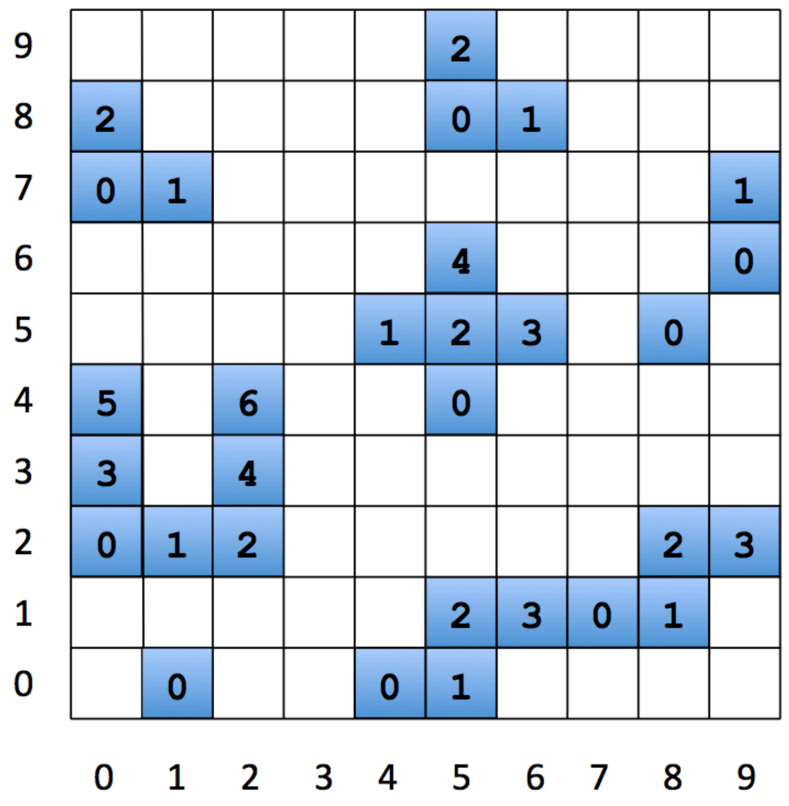

Figure 2

- A shape of type "L" placed at (0,7)
- A shape of type "L" placed at (5,8)
- A shape of type "I" placed at (9,6)
- A shape of type "X" placed at (5,4)
- A shape of type "O" placed at (8,5)
- A shape of type "U" placed at (0,2)
- A shape of type "O" placed at (1,0)
- A shape of type "S" placed at (4,0).
- A shape of type "S" placed at (7,1).

HW2 Assignment

The goal of HW2 is to implement classes representing shapes. A given program will use this class to create shapes at arbitrary locations in the x-y plane and move them. The program uses a function to move shapes and to detect if shapes overlap. The board is assumed to be the entire x-y plane.

The class **Shape** is an abstract base class from which six classes are derived, named: **O**, **I**, **L**, **S**, **X**, **U**. A detailed specification of the class **Shape** is given below. You are given the header file **Shape.h** (at http://web.cs.ucdavis.edu/~fgygi/ecs40/homework/hw2). Your task is to write the implementation file **Shape.cpp** containing the implementation of all classes. You are not allowed to modify the file **Shape.h** (Do not insert your name or SID in it since we will use the cmp command to check that the file is the same as the original). Your class implementation file **`Shape.cpp`** should compile without warning using the command

```
$ g++ -Wall –c Shape.cpp
```

You are also given a program **`testShape.cpp`** and a **`Makefile`** (same web site as above). These two files should not be modified. The **`testShape.cpp`** program tests the functionality of the **`Shape`** class. You should be able to build the executable **`testShape`** using the command

```
$ make
```

You will use the **`testShape`** executable and the input and output test files provided on the web site to check the functionality of your classes and verify that your program reproduces the test output *exactly*. Use the diff command to compare your output file with the output file provided. Note that these test files do not check all the functionality and you are responsible for verifying your implementation. Other test files will also be used when grading your implementation. The **`testShape`** program reads input from **`stdin`** and writes to **`stdout`**. It can be used e.g. as

```
$ ./testShape < test1.in
```

It can be assumed that input will consist of two lines. Each line consists of one character followed by two integers. The character and integers are separated by white space as in e.g.
`X 2    -3`

Specification of the Shape class

We describe here the detailed specification of the **`Shape`** class. The positions of the cells occupied by a shape are stored in two integer arrays whose size depends on the shape, i.e. a shape occupies the cells located at (x[0],y[0]), (x[1],y[1]), etc...

Study carefully the file **`Shape.h`**. Some members are public and some protected. Some members are virtual. Make sure you understand why. Study the file **`testShape.cpp`** in order to understand how the classes will be used.

Public members

`virtual char name(void) = 0;`
This pure virtual function returns the name of the class instance as a character, i.e. **`O`**, **`I`**, **`L`**, **`S`**, **`X`**, **`U`**.

`void print(void)`
This function prints information about the location of the shape by showing the position of the cells that the shape occupies:

`name` **`at (`*****`x[0]`*****`,`*****`y[0]`*****`) (`*****`x[1]`*****`,`*****`y[1]`*****`) ...`**

For example, from Figure 2:

```
X at (5,4) (4,5) (5,5) (6,5) (5,6)
```

`bool overlap(const Shape &s)`
This function should return **`true`** if the shape calling the function overlaps with the shape **`s`**. This can be checked by comparing the arrays x and y of both shapes. If there is no overlap, i.e. no cells are shared by the two shapes, the function should return **`false`**.

`void move(int dx, int dy)`
This function moves the shape by a displacement (dx, dy). This function modifies the values of the arrays x and y. Note that this function is the same for all shapes and need not be implemented as a virtual function.

`static Shape *makeShape(char ch, int posx, int posy)`
This function is a "shape factory". It is used to generate instances of shapes. It creates an instance of the appropriate kind of shape (specified by the character **`ch`**, with values **`'O'`**, **`'I'`**, **`'L'`**, **`'S'`**,**`'X'`**, **`'U'`**) at the given position (posx,posy) using the operator **`new`**, and returns a pointer to the instance. If the character **`ch`** is not one of the allowed letters, the function should throw an **`invalid_argument`** exception. The error printed by the program should correspond to the example given in one of the test files.

protected members

`int *x,*y`
The arrays holding the coordinates of the cells occupied by the shape.

Submission

Create a tar file named **`hw2.tar`** containing the files **`Shape.h, Shape.cpp, testShape.cpp`** and **`Makefile`**. Do not compress the tar file. The files **`Shape.h, testShape.cpp`** and **`Makefile`** must be identical to the files provided. Include your name and Student ID in a comment at the top of the **`Shape.cpp`** file. Submit your project using:

```
$ handin cs40 hw2 hw2.tar
```