YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 427: Object-Oriented Programming — *Handout #6*

Professor M. J. Fischer — *October 24, 2016*

Problem Set 5

Due before midnight on Monday, October 31, 2016.

1 Assignment Goals

1. Produce an application with more than one class, appropriately split into multiple files.
2. Learn how to use a constructor and friend function to produce a semantically valid non-trivial data structure

2 The Game of Kalaha

Kalaha (also known as "Kalah") is a game in the ancient Mancala family of board games that has become popular in the Western world. There are many variations of the game. We will consider the one described on the Kalah Wikipedia page.

It is played on a rectangular board as illustrated in Figure 1.

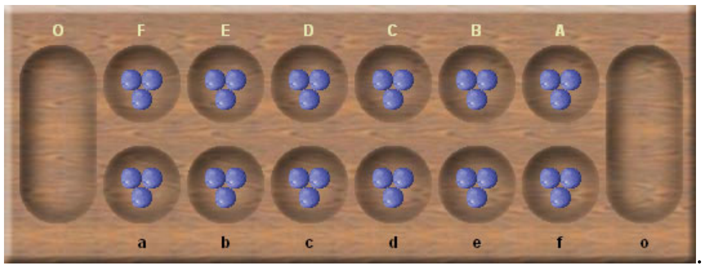

Figure 1: A Kalaha board. (Image from `http://www.iggamecenter.com/images/info/kalah/2.jpg`).

The pits along the top and bottom are called *houses*. The two larger pits at the ends are called *stores*. The two players are called *North* and *South*. North owns owns the six houses on the top the board as well as the store to the left. South owns the six hours on the bottom of the board as well as the store to the right.

Kalaha can be played with any number of houses and any number of initial seeds. The game illustrated in Figure 1 is the $(6,3)$ game. The $(6,4)$ game is often played as well. We will allow any (reasonable) number of houses and initial seeds.

```
              N1    N2    N3    N4    N5    N6
             [ 4]  [ 4]  [ 4]  [ 4]  [ 4]  [ 4]
   N0  [ 0]                                       [ 0]  S0
             [ 4]  [ 4]  [ 4]  [ 4]  [ 4]  [ 4]
              S6    S5    S4    S3    S2    S1
```

Figure 2: A text-based representation of a $(6,4)$ Kalaha board.

Figure 2 is a text-based representation of a $(6, 4)$ Kalaha board. Each pit is represented by a pair of square brackets enclosing a number of seeds. The top and bottom rows show the house labels; the middle row shows the two stores N0 and S0, respectively. Note that the pits are labeled in increasing order going *clockwise* around the board, starting with the store, but play proceeds in the *counterclockwise* direction.

Play alternates between the two players, with South playing first. She begins by picking up all of the seeds in one of her non-empty houses. Starting with the next pit and proceeding in *counterclockwise* order, she *sows* the seeds by dropping one seed in each pit except for her opponent's store, which she skips. When the last seed is dropped, one of three things happens:

1. If the last seed is dropped in her own store, then she gets another turn.
2. If the last seed is dropped in her own empty house and the opponent's house opposite[1] hers is non-empty, she *captures* all of the seeds in both houses, places them in her store, and her turn ends.
3. Otherwise, nothing special happens and the turn ends.

The game ends when all of one player's houses become empty. At this point, the opponent moves all of the seeds in her houses to her store. The player with the most seeds in her store wins. If they both have the same number, the game is a tie.

3 Problem

In the next few problem sets, you will be implementing a Kalaha game system. For this problem, you will implement two classes that represent the *structure* of the Kalaha board.

The board consists of a number of pits. Each pit has an *owner* (North or South) a pit number relative to its owner, and a string label as shown in Figure 2. Pit 0 is the store, pits 1...`numHouses` are the houses.

The pits on the board stand in relationships to one another. The *next* pit is the next one in counterclockwise order. So the next pit from N3 is N2, the next pit from S1 is S0, and the next pit from S0 is N6. The *opposite* house is the one across the board from it, so the opposite house from N3 is S4. Opposite is only defined for houses. These relationships should be represented by two pointers in each pit – the next pointer and the opposite pointer. (The opposite pointer in each store can be set to `nullptr`.)

Corresponding to this structure, you should implement types `Player`, `PitFrame`, and `BoardFrame`, each with their own `.cpp` and `.hpp` files. `Player` can be an `enum`. `PitFrame` and `BoardFrame` are classes.

Class `PitFrame` should implement several public functions:

```
PitFrame( Player owner, int pitNum, string label );
Player getOwner() const;
int getPitNum() const;
string getLabel() const;
PitFrame* next() const;
PitFrame* opposite() const;
ostream& print(ostream& out) const;
```

[1]For example, in Figure 2, houses S2 and N5 are opposite each other.

Class `BoardFrame` should implement several public functions:

```
BoardFrame(int numHouses);
PitFrame* getPitFrame( Player pl, int pitNum ) const;
ostream& print( ostream& out ) const;
```

In addition, you should implement a test program in `main.cpp` that takes a single command line argument `numHouses` and creates a board of that size. It should then test each of the functions in the two classes. A test consists of preparing and printing the arguments, calling the function, and printing the result.

Your test program should check that all of the links are properly set. First, it should walk all around the board, print out each pit as it passes, and stop when it gets back to the starting pit. Second, it should visit each house and print the label of the house and its opposite.

To print a pit, you should print the following, nicely formatted: the owner, the pit number, the label, the label of pit `*next()`, and the label of pit `*opposite()`. To print a board, you should print each of the pits on the board.

4 Programming Notes

Your `BoardFrame` constructor has two jobs to do:

1. It must instantiate the pits.
2. It must set private pointers `next_p` and `opposite_p` in each `PitFrame` to point to the appropriate `PitFrame` created in step 1. This is because `next()` and `opposite()` should do nothing more than return their respective points; they should not be dealing directly with subscripts.

Because a pit must be created before a pointer can point at it, step 2 cannot be done in the pit constructor since the pit constructor is called in step 1 before all of the pits have been created. Rather, the board constructor must set the links. But this presents a problem since the pointers are private in `PitFrame`.

`BoardFrame` and `PitFrame` are an example of *tightly-coupled* classes.[2] *Do not* make the link pointers public or create setter functions for them. Rather, create a forward declaration for `BoardFrame` in class `PitFrame`, and then have `PitFrame` give friendship to class `BoardFrame`.

Other conditions on your code for this assignment:

1. Do not use of standard library containers such as `vector`.
2. Do not use public data members.
3. You may use `new[]` and `delete[]`. Do not use `malloc()` and `free()`. (They have no place in modern C++ programs.)
4. Extend `operator<<()` to types `Player`, `PitFrame`, and `BoardFrame`.

5 Grading Rubric

Your assignment will be graded according to the scale given in Figure 3 (see below).

[2] See Chapter 8 of the textbook.

#	Pts.	Item
1.	1	A well-formed `Makefile` or `makefile` is submitted that specifies compiler options `-O1 -g -Wall -std=c++11`.
2.	1	Running `make` successfully compiles and links the project and results in an executable file `testKalaha`.
3.	6	`testKalaha` performs all required tests.
4.	8	`testKalaha` produces correct output when run with argument 3 and with argument 6.
5.	3	All of the instructions in sections 3 and 4 are followed.
6.	1	All relevant standards from previous problem sets are followed regarding good coding style, submission, identification of authorship on all files, and so forth.
	20	Total points.

Figure 3: Grading rubric.