Problem Set 4

Due by 11:59 pm on Friday, October 11, 2019.

1 Assignment Goals

1. Learn how to model external objects using object-oriented design principles.
2. Learn how to use a constructor and friend function to produce a semantically valid non-trivial data structure.

2 The Game of Kalah

Mancala is an ancient family of board games that are still played today. There are many variations of the game. We will consider the version described on the Kalah Wikipedia page that has become popular in the Western world. It is played on a rectangular board as illustrated in Figure 1.

![Figure 1: A Kalah board with labeled pits.](image)

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 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.

Kalah can be played with any number of houses and any number of initial seeds. Each player in the (6, 3) game has 6 houses and starts with 3 seeds in each. The (6, 4) game is often played as well. We will allow any (reasonable) number of houses and initial seeds.

Figure 2 is a text-based representation of a (6, 4) Kalah 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 decreasing order going counterclockwise around the board, starting from the leftmost house of each player.

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 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 couple of problem sets, you will implement a Kalah game system. For this problem, you will implement the classes and types used to represent the Kalah board and the state of a game in progress, but not the game play algorithm itself.

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

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 pit is the one across the board from it, so the opposite pit 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 pointers in the stores should be set to nullptr.

Corresponding to this structure, you should implement types Player, Pit, and Board, each with its own .hpp and .cpp files (if needed). Player should be an enum type. Pit and Board are classes.

Class Pit should have a private data member contents to hold the number of seeds currently in the pit and private pointers next and opposite that point to the next and opposite pits, respectively. Pit will also need private data members to store its owner, number, and string label. The Pit constructor will be given the owner, number, string label, and initial contents as parameters.

The only game actions that involve contents are:

- Initialize contents with the starting number of seeds.
- Scoop up and return the number of seeds in the pit, leaving the pit empty.
- Drop a seed into the pit.
- Test if the pit is empty.
- Print the contents as a decimal number.

\(^1\)For example, in Figure 2, houses S2 and N5 are opposite each other.
These simple operations capture exactly what is needed for playing the game and not more. They could all be implemented using getter and setter functions, but in keeping with OO-principles, they should instead be defined as public inline functions in class Pit.

The Board(numHouses, numSeeds) constructor should create a board for a Kalah game with the specified number of houses and number of initial seeds. To this end, it should first allocate a single array in dynamic memory large enough to contain all of the pits that are needed. Each pit in the array will be automatically constructed using the pit’s default constructor. Then the board constructor should go through the pits one by one, initializing each pit appropriately.

In addition, you should implement a test program in main.cpp that takes two command line arguments numHouses and numSeeds and creates a board of that size and initial contents. It should then test each of the functions in classes Pit and Board. 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 data members 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 of pit *next(), the label of pit *opposite(), and the contents. To print a board, you should print each of the pits on the board in counterclockwise order starting from South’s leftmost house.

4 Programming Notes

Because the pits are all constructed when the array is created using the default Pit constructor, they must be configured later with their semantically-meaningful data. Only the Board constructor can do this since it is the class with knowledge of the whole board, how the board is numbered, and so forth. However, Board cannot modify private data members of Pit since they are private, so we have a problem. This is an example of tightly-coupled classes, which is discussed in the textbook.

Do not make the Pit’s data members public or create individual setter functions for them. Rather, create a Pit constructor that initializes all of its data members according to values passed to it as arguments. Because the pits in the pit array have already been initialized by the default constructor, this new constructor cannot be used to reinitialize them. Instead, Board uses this constructor to construct a temporary fully-initialized Pit, and that temporary pit is then copied or moved into the pit array slot to ready it for use. Because Pit does not have dynamic extensions, it makes no difference whether it is copied or moved into the array.

Other conditions on your code for this assignment:

1. Do not use standard library containers such as vector.
2. Do not use public data members.
3. Use new[] and delete[] only for allocating the Pit array in Board. Do not use malloc() and free(). (They have no place in modern C++ programs.)
4. Extend the output operator<<() to types Player, Pit, and Board.

5 Grading Rubric

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

---

2See Chapter 8 of the textbook.
<table>
<thead>
<tr>
<th>#</th>
<th>Pts.</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1</td>
<td>All relevant standards from previous problem sets are followed regarding good coding style, submission, identification of authorship on all files, and so forth.</td>
</tr>
<tr>
<td>2.</td>
<td>5</td>
<td>All of the instructions in sections 3 and 4 regarding code structure are followed.</td>
</tr>
<tr>
<td>3.</td>
<td>1</td>
<td>Running <code>make</code> successfully compiles and links the project and results in an executable file <code>testKalah</code>.</td>
</tr>
<tr>
<td>4.</td>
<td>6</td>
<td><code>testKalah</code> performs all required tests.</td>
</tr>
<tr>
<td>5.</td>
<td>6</td>
<td><code>testKalah</code> produces correct output when run with arguments (6,3) and with (3,4).</td>
</tr>
<tr>
<td>6.</td>
<td>1</td>
<td><code>valgrind</code> shows no memory leaks and no memory management errors.</td>
</tr>
</tbody>
</table>

20 Total points.

Figure 3: Grading rubric.