Computer Science & Engineering 171
Assignment #2: Object-Oriented Programming

Due: May 14th at 6:00 pm

In this assignment, you will use C++ and Smalltalk, two object-oriented languages discussed in class. Use the Rocky 9 machines in the Engineering Computing Center for this assignment. The Smalltalk interpreter is `gst` (you will need to execute "`setup smalltalk`" first) and the C++ compiler is `g++`.

The practicum will be done in class in teams of two or three. Your score will be based on your participation. The homework is to be done individually and submitted using Camino. You score will be based on correctness and coding style. The practicum and the homework are each worth 5 points.

1 Practicum

1.1 Binary Search Trees

On Camino you will find an implementation in C++ of a binary search tree. Translate this program to Smalltalk, keeping the same class and operation names. Equivalent code for the function `main` in C++ is provided for you as `main.st`. Call the file with your class definitions `Tree.st`. You can run both files through the interpreter using `gst Tree.st main.st`.

Goal: To learn about classes as a mechanism for information hiding.

1.2 The 8-Queens Problem

In chess, a queen can move any number of squares horizontally, vertically, or diagonally. The ***8-queens problem*** is the problem of trying to place eight queens on an empty chessboard in such a way that no queen can attack any other queen. This problem is intriguing because there is no efficient algorithm known for solving the general problem. Rather, the straightforward algorithm of trying all possible placements is most often used in practice, with the only optimization being that each queen must be placed in a separate row and column:

1. Starting with the first row, try to place a queen in the current column.
2. If you can safely place a queen in that column, move on to the next column.
3. If you are unable to safely place a queen in that column, go back to the previous column, and move that queen down to the next row where it can safely be placed. Move on to the next column.

On Camino you will find my solution to the 8-Queens Problem in the file `Queens.cpp`. My solution provides the following generalizations:

1. Solves the n-Queens Problem rather than the 8-Queens Problem. The chessboard is of size $n \times n$ rather than of size 8×8, and n queens are to be placed on the board.
2. Computes and displays the ***total*** number of solutions possible. Rather than finding one solution and exiting, my program will find all possible solutions and write the total number of possible solutions to standard output.
3. Solves the n-***Pieces*** Problem rather than the n-Queens Problem. Given any kind of chess piece, not just a queen, my program displays the total number of solutions possible for placing n of those pieces on the board such that no piece can attack any other piece.

You will need to write a file called `Queens.h` that contains the class and function definitions for each chess piece. Write a base chess piece class with which my algorithm works. The base class should provide at least the following methods:

- `int row() const`: returns the row number of this piece
- `int column() const`: returns the column number of this piece
- `void place(int row, int col)`: places this piece on the board given a row number and a column number
- `bool menaces(const Piece *p) const`: given another piece, returns true if this piece can attack the given piece, and false otherwise

Now write classes using virtual multiple inheritance for each of the following chess pieces:

- rooks: A rook can move any number of squares horizontally or vertically, but cannot move diagonally.
- bishop: A bishop can move any number of squares diagonally, but cannot move horizontally or vertically.
- knight: A knight can move two spaces in either a horizontal or vertical direction and an additional space in the other direction.
- queens: A queen can move any number of squares horizontally, vertically, or diagonally. Thus, it can move as either a rook or a bishop.
- amazon: An amazon can move either as a queen or as a knight.

Goal: To generalize a problem using object-oriented techniques.

Hints: Write and test each piece in the order listed.

1.3 Coding Guidelines

- C++ requires explicit memory deallocation, whereas Smalltalk uses garbage collection.
- C++ does not automatically initialize instance variables to a default value, whereas Smalltalk initializes all variables to `nil`.
- By convention, Smalltalk classes understand the `new` message, which is inherited and parameterless, and use mutators on the object to initialize their state. Classes can also be written to have custom "constructors," but mutators are typically still needed to modify the state of the created instance.
- Since all the functions required for the 8-Queens problem are rather small, they can easily be written in the header file.
- In order for the position of a piece to be kept hidden and only available to subclasses, it should be declared as `protected`. Also, decide which functions need be polymorphic, and which have no need to be polymorphic and overridden in subclasses.
- Since the base class is not meant to be instantiated, its `menaces` function should be a pure virtual function (i.e., no function body should be provided, and instead it should be assigned zero in its declaration).

2 Homework

Build upon the provided solutions to the practicum (and not your own solutions) when solving the homework.

2.1 Binary Search Trees

Extend the Smalltalk implementation by implementing a method called `maximum` to compute and return the maximum value in itself. Your method should answer `nil` if the tree is empty. Call your program `Tree.st` and submit it using Camino.

2.2 The 8-Queens Problem

Modify the solution to the 8-Queens problem so it does not use multiple or virtual inheritance. You may not duplicate the logic to detect if a piece menaces another piece. For example, you may not duplicate the logic for `Rook` or `Bishop` in `Queen`.

Instead, you must replace inheritance with composition, a common technique for mimicking multiple inheritance of implementation in languages in which it is not supported. If we have two classes, `A` and `B`, rather than stating `B` "is an" `A` by using inheritance, we state that `B` "has an" `A` by using composition. For example, `Queen` can still inherit from `Rook` but must now contain an instance of `Bishop`:

```
class Queen : public Rook {
    Bishop b;

public:
    /* rest of implementation */
};
```

In addition to providing a `menaces` function, you will also need to override the `place` function and change its declaration so it is polymorphic. Call your file `Queens.h` and submit it using Camino.