2. Sudoku Helper: Square composes State

CSCI 4526 / 6626, Fall 2022

1 Goals

- To compose a class within another class.
- To use constructors, destructors, inline functions and non-inline functions.
- To learn what delegation is.
- To use trace comments in constructors and destructors to build an appreciation for how the language works.
- To implement two interacting classes:
  - Square
  - State

  Parts (data and functions) will be added to these classes as the project develops.

2 One Square.

Put the Square class in the same pair of files as the State class, with State first. In this case, I believe it will make comprehension and debugging easier if these two closely related classes are kept together. If you strongly disagree about this design issue, use two pairs of files.

Addition to the State class.
Your State class will need a getValue() function to be called by Square. Your Square class might also need one, later, to be called from Game.

Square Private Data Members.
We are unable, at this time, to handle all of the properties of a Square; more parts will be added later as more classes are added to the project.

- A State object. The State class contains all the information that will be saved during play to support operations of undo, redo, save to a file, and restore from a file.
- Two short integers: the row and column numbers of this Square.
- A vector listing all the neighboring Squares (same row, column, or box) will be added later.

Square Public Functions. Wherever it makes sense, make functions inline. If they are one statement, maybe two, not more, and fit on one line (maybe broken onto two lines), put the function entirely within the header file instead of a prototype.

- A default null constructor.
- A constructor with three parameters to initialize the Square: a digit or a dash (char) and the row and column numbers of the Square (short ints). Use a ctor to supply the character to the constructor of State.

  A ctor is an initialization expression that can be used to initialize any class member. It is written after the parameter list of a constructor and before the opening brace. There are three situations in which a ctor *must* be used, but we will deal with two of them later. For now, we need a ctor in the Square constructor to call the constructor of the State member.

The syntax is:

`Square::Square ( params ) : State-member-name( params ) { ...`

Finally, add a trace output to this constructor that will print a comment each time the constructor is used. Example: `cerr << Square 3, 1 constructed.`

- `Square::~Square`. A destructor. Since you are not using dynamic allocation in this class, the destructor has no essential work to do. It could be a "do nothing" function. However, please put an output statement in it so that you will know every time a Square gets deleted. My destructor produces output like this: `Deleting square 3, 1`
- `Square::print()`. Write a public function with prototype: `ostream& print( ostream& );` that will print all the data members of the Square. Print the State by using *delegation*, that is, by calling the print() function in the State class. Return the stream parameter. My print function produces this output: `Square [4, 0] Value: - Possible: --765--21`
- `Square::mark()`. A public function that calls State's mark function with a potential new value for the square. Marking a square should change its possibility list. In your test, I want to see the poslist before and after marking to verify the change.
- Outside the class but inside the .hpp file , declare an inline method for the output operator:

  `ostream&  operator <<( ostream&, Sqare& );`

  It must call your *print()* function with the appropriate parameter. Don't worry right now about how or why this works; just follow the example. This definition allows you to output a Square as easily as you output an integer or a string.

3 Testing

For every class you write, you should create a unit test to test all of its functions and to cause every possible error comment to appear in your output. In this program, you have two classes, so you need the unit test from P1 plus a new test for Square. Put these functions in the same file as your main program. In main, instantiate a Square object, which has a State object inside it. Call the State test first, then the Square test.

To test a constructor, declare at least two variables and supply different parameters in the declaration. Since you are not using *new* in the declaration, the Squares will be deleted automatically at the end of main(), and your destructor will be tested automatically.

Test the print functions by using the `<<` operator.

In the documentation for your test plan, say what input or parameter you will supply for each test and what output or program behavior should be produced. Incorporate this test plan into your two unit test functions (`testSquare()` and `testState()`).

Due September 10. Make a folder for turning in your work. The name of the folder should include both your name and the problem number (Example: FischerP2). Put copies of your test plan, your source code (.cpp and .hpp files) and your output into this directory, then zip it up. Use email to my home to turn in your zip file.

Advice FOLLOW THE INSTRUCTIONS and don't complicate it. If you are confused about the instructions or you have trouble of any sort email me and enclose a current copy of your code and screenshots of your problem. Hand this in by the due date or be prepared to give me a written reason for lateness..

Do not go to the internet to find solutions. They do not exist. Even if you can find a working Sudoku Helper program, it will not meet the requirements of this assignment and will not help you learn the material.