Sudoku 2: Non-Polymorphic Derivation

CSCI 4526 / 6626, Fall 2016

1 Goals

This assignment asks you to implement a class named Square that is derived from State, and Board (a 9 by 9 array of Squares). Parts (data and functions) will be added to these classes as the project develops.

- To derive a class from another class.
- To compose a class within another class.
- To use constructors, destructors, inline functions and non-inline functions.
- To use simple inheritance and ctor initializers.
- To learn what delegation is.
- To use trace comments in constructors and destructors to build an appreciation for how the language works.

2 One Square on a Sudoku Board

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. The Square class, derived from State, will store all the other information about the Square: its row and column numbers and its relationships to other squares in the same row, column, and box. 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.

Put the Square class in the same pair of files as the State class. 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 a separate pair of files.

Modification to the State class: Make all of the data members of State protected, not public. Introduce a default constructor that can be called from the default constructor of Square.

The Square class. Derive the Square class from the State class. This creates a new class that has all the members of a State followed by all the new members defined for Square. The syntax is:

```
class Square :  public State {...}
```

Terminology: Square is the *derived class* and State is the *base class*. This week, the data members of a Square are just the Square's row and column coordinates (digits between 1 and 9). We will add more data members in future weeks.

Square Functions.

- A constructor that will initialize the Square and the three members of the State. The constructor must have three parameters: a digit or a dash (char) and the row and column numbers (short ints) of the Square. To initialize the base object (the state of this square), you need to use a ctor that passes the char from the parameter list of the Square constructor to the State constructor. 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 base class, State. The syntax is:

`Square::Square ( parameters ) : State( initializer ) { ...` 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.`

- A constructor with no parameters that will initialize the Square but *will* print a trace comment, "Constructing empty square 3, 1". You need this no-parameter constructor in order to declare an array of Squares (the Board).
- `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 this prototype: `ostream& print( ostream& );` that will print the data members of the Square and its base object, the State. Return the stream parameter. Delegate printing the state to the State class, that is, let Square::print() call State::print(). My print function produces this output: `Square [4, 0] Value: - Possible: --765--21`
- 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.

Inherited Functions. Because Square is inherited from State, all of the public and protected function members of State are inherited. Protected functions can be called from Square, and public functions can be called from anywhere in the program. There is no need to re-implement `move` and `erase` in the Square class. They are already present.

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. Call the State test first, then the Square test.

To test a constructor, declare at least two variables of type Square and supply parameters in the declaration. This week, it is enough to declare single squares. Your destructor will be tested automatically when your object declarations go out of scope. 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 26. 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 This is an easy program. FOLLOW THE INSTRUCTIONS and don't complicate it. Its purpose is stated at the top of this handout. Email me or Michael Wanser for help promptly if you are confused about the instructions or you have trouble of any sort. Try to have something to hand in by the due date.

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.