11: Details and Documents

CSCI 4526 / 6626 Fall 2016

1 The Project So Far

Functionality. You have implemented a complete menu-driven game with a graphic interface and two variants of Sudoku. The menu allows you to backup and redo and also to save the game and continue it later.

Techniques. If you followed the instructions, all major parts of the C++ language have been used:

- Classes whose data members are all private.
- Your classes have .hpp and .cpp files, with `#include` statements in the .hpp file, except for the one `#include` that was placed in the .cpp file to break a circular reference.
- You used a class member that was a reference type.
- Inline functions have been used for one-line definitions.
- Initializations in the class declaration and ctors have been used to initialize data members.
- You use stack allocation for most things and used dynamic allocation when necessary.
- You defined data structures yourself, including a pair of classes that have a circular relationship. You have also used a data structure from the standard template library.
- You have learned about forward declarations and have used an enum type effectively.
- Extensions of the `<<` operator.
- Non-polymorphic class derivation was used to define State and Square, and also Board and DiagBoard. You used casts to negotiate the derivation relationship.
- Polymorphic derivation was used to define exception classes. Virtual functions were used effectively to handle exceptions.
- A text file was used to input the game board.
- A binary file was used to save the state of the game.

Style. If you followed my guidance and examples, your code has these properties:

- Parts of your classes are sensibly divided between the .cpp and .hpp files.
- It is neat, readable, and not clumsy or repetitive.
- Comment lines are used to improve readability.
- It is not filled with lots of blank lines that have no purpose.
- It DOES have one blank line to separate each group of related statements.

2 Now What?

Sudoku assignment 11 does not ask you to write new code, but you may find yourself modifying your existing code to "clean it up" or add some required feature.

Details: 6 points.

- (2) Use at least one operator extension in addition to `<<` and `>>`. The obvious one to use is subscript (`operator[]`, but there may be other good opportunities.
- (2) Use const meaningfully and in a lot of places. There are several ways to use const, and I expect to be able to find the first five in your code. The last is rarely needed.
  - A const return value. (Sometimes used for * and `&` returned types.)
  - A const parameter.
  - A const implied parameter. (The const goes after the () and before the {}.)
  - A const class member.
  - A const local variable.
  - A const pointer, that is, a pointer that cannot be moved: `MyType* const = & myVar;`
  - (2) A makefile.

Documentation: 4 points.

- (1) A UML diagram that shows only classes and the relationships between them. Show all of your classes on one page. DO NOT use an app to do this job for you. Do not even try to do it is WORD: it takes forever and is not portable to my machine. It is easy to do by hand.
- (2) A call chart, showing a box for every function you wrote. For each function, draw a line that shows what other functions it calls. If the function is virtual, write a "V" on that line. This may easily go onto two pages. Do this yourself.
- (1) A proof of any kind that your code does not have memory leaks. If you can run it under Linux, use ValGrind to produce the proof.

3 Submission.

Submit electronically, as usual. PLEASE get or borrow a CamScanner app. I understand that they are widely available and free. If your phone cannot run it, borrow a friend's phone long enough to turn in the project. Please do not send simple snapshots; they are very hard to read because of low contrast and distorted perspective.