12. Sudoku Helper: Details and Documents

CSCI 4526 / 6626 Fall 2022

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.
- You have used an enum class 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:

- The 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 12 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.

I am proud of this. In your submission, start by telling me (briefly) about what you feel you contributed to this effort that is especially well done or creative. There is potential extra credit here.

Details: 6 points.

- (1) A manifest: that is, a list of all the files needed to run your application (alphabetical order). and any other files that you are including in the final package. You need at one output to prove that the makefile works for you and the output is still correct after the adjustments listed below.
- (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 ctors and inline functions many times in your program.
- (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 that works on my system. Make it yourself, following the examples– do not send me a machine-generated version. the goal is for you to understand what a makefile is and how to make one. I will use it to compile your program.

  Documentation: 3 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 in WORD: it takes forever and is not portable to my machine. It is easy to do by hand, but try to be neat and make your writing large and neat enough to read.
  - (1) Some other kind of graphic documentation, such as a call chart or an interaction diagram or a flow chart of some complex part of your logic.
  - (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. If you don't have a real scanner for the diagrams, PLEASE get a CamScanner app (it is free) and set it to black-and-white reproduction. Please do not send simple snapshots; they are very hard to read because of low contrast, brown background, and distorted perspective.