CS165 Final Project

Now to code your way through all the class topics ☺! We will be cursory in grading these and somewhat forgiving in interpretation or execution, but make sure it obviously meets the requirements before adding extras. A good grade on this problem does not require actually meeting all these requirements, but most are so easy that you should be able to do it in a couple comments, a couple lines of code, and some careful calculation or output formatting.

This project should help you in remembering most of the coding, logic, and vocabulary that was related to programming in this class.

Be sure it is easy to read the output of your program as well as the code!

You may list requests at the top of your code for re-evaluation of projects that you missed points on. You must:

1. list which projects you want to have re-evaluated at the top of the file that contains the main,
2. list which requirements you wish to have re-evaluated from those projects,
3. list where in this project you will demonstrate understanding of the requirements,
4. include some code somewhere in this project that demonstrates that you understand how to do the past requirement you are trying to meet (with explanation as to why it meets the past project's requirements that you missed),

   (**Note**: Yes, these can overlap with the requirements of this project!)

Please create and submit a source file (.cpp for C++) for a program that has or does the following:

(**These should each be explicitly labeled with the requirement number and some indicator (perhaps a line with several asterices or hyphens), and would ideally be used in the order they are required (so much so that it will be worth extra credit if you do so!)**)

(**2 bullet points worth of extra credit**): does all of the following in the order that they appear (makes it much easier to grade!)

1. Demonstrates understanding of **binary** numbers (comments may be used to help explain the code, but the output should explain something about the numbers),
2. Demonstrates understanding of **two's complement** numbers (comments may be used to help explain the code, but the output should explain something about the numbers),
3. Uses some **pre-defined macro** from an existing include or library (such as INT_MAX or perhaps there is a PI value somewhere),
4. Uses some form of **simple output** (cout or printf is fine),
5. Uses some form of **simple input** (cin or getline are fine, better would be to use both :)!),
6. Uses some form of **type casting** (a few ways to do this, some are implicit some are explicit),
7. Uses some form of **conditional** (should be easy, we use these all the time),

8. Uses some form of **logical** or **bitwise operator** (more specifically &, |, ^, &&, ||, !, ==),

   (see what the following does by adding print statements to see a neat binary trick:
   int x = 10;
   int y = 25;
   x = x ^ y;
   y = x ^ y;
   x = x ^ y;
)

9. Uses at least one **loop**,

10. Uses at least one **random number**,

11. Demonstrates understanding of the three general **error categories** we talked about (syntax, logic, and run-time),

12. Demonstrates some form of **debugging** "tricks" that we have learned throughout the class (print statements, input verification, checks for bad conditions (divide by zero anyone?), asks users for clarification, uses menus to filter user input into an easy to read format (number instead of word or phrase), or any other trick that can help catch, report, or reduce bugs; **just be sure to explain why your tricks help reduce bugs**)

13. Uses at least one **function** that you define (should be easy considering other requirements...),

14. Generally uses **functional decomposition** to reduce how large a single section of code is or to make the plan or algorithm obvious for your program,

15. Demonstrates how **scope** of variables works (how a specific variable only is accessible from within its defined block of code),

16. Demonstrates the different **passing mechanisms** (in C++ this means pass by value and pass by reference, but you may also choose to show how passing the value of a pointer can allow you to reference the memory of values that will live after the function where the pointer is passed by value),

17. Demonstrates **function overloading**,

18. Uses at least one **string** variable (std::string or c-style string),

19. Uses some form of **recursion**,

20. Uses at least one **multi-dimensional array**,

21. Uses at least one **dynamically declared array**,

22. Uses at least one **command line argument** (make sure it is obvious how to use it!),

23. Defines and sues at least one **struct**,

24. Defines and uses at least one **class** to create and use an **object**,

25. Attempts to use a **pointer to an array** (we used these a bit, so should not be too bad),

26. Attempts to use a **pointer to a struct** (I think most of you used these some),

27. Attempts to use a **pointer to an object** (This may be new to you, but it is similar to an array or a struct),

28. Uses its own **namespace** (use your engr username or some fun name for this),

29. Use a **header file** you write,

30. Use a **makefile** that we use to compile the project on flip,

31. Uses at least one **vector**,

32. Defines classes covering the use of a **default constructor**, a **copy constructor**, and a **destructor** (you have to make these to prove that you understand them!),

33. Demonstrates **overloaded operators**,

34. Uses some form of **file IO** (and none of that hard-coded filename-in-a-string stuff!!),

35. Uses other classes for some neat effects such as things from **STL** or **boost** libraries (be explicit with comments)(**these will help you greatly in future classes!!**),

36. Uses **inheritance**,

37. Uses **polymorphism**,

38. Uses **exceptions** (does not have to fill the whole huge project with exceptions for all the error handling),

39. (extra credit) does **something awesome** perhaps a game, a text editor, a spreadsheet editor (could be a basic interface showing the current spreadsheet and allows edits such as D13 $27.03 or you could go even farther and add calculator functionality, showing a string for a calculation in a "cell" and having a formula-view or a value-view (toggle with a menu option) where you store a string for a calculation but process the string when updated to generate the value-view (I would strongly recommend only simple calculations for your formula parser such as +, -, *, /, % and only using integral types, floating point types, and strings)... it sounds like so much fun I might just go do that for the next round of videos!)

40. (extra credit) can you think of **something else** that would be good for extra credit, then list it with a comment and we will consider it :)!