CS161 Final Project (100+ points)
09 July, 2014

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 passing grade on this project 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!

(+) Assignment reevaluation components:

You may list requests at the top of your report for re-evaluation of assignments 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 assignments,

3. list where in this final project you will demonstrate understanding of the requirements,

   (**Note**: if the requirement overlaps with a final project requirement, list that number here and then mention it in your final project report)

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!)

   (**Also Note**: if you request and then demonstrate that you clearly understand something now that you did not demonstrate earlier, you may be given back up to half of the points you missed earlier)

(30) Report Components (not necessarily the same as a normal assignment):

You need to write a report briefly demonstrating each of the source code's requirements:

1. (05) Explain what your program can do and what motivated you to pick that topic or theme. No, this does not need to be as involved an explanation as I normally require for projects or assignments; Yes you can include your design work, but should have the simple explanation first.

2. (10) Overall style of code and use of program: Your code should be of a consistent style, be easy to read, and fairly easy to follow the flow through your code. Be sure to indent, have good identifiers (variables, functions, classes, …), and do your best to make your code easy to read through (both inside a single class or function as well as the general structure of your whole program). Your program should also be easy to use, informing a user when input is required and validating any input.

3. (15) Reflect upon all the techniques you used, what all you learned for the project or in the class in general, and whether you approach problems a little differently now that you have gone through all of the materials for this class.

(20) Requirements listing component:

Requirements listing: You will be required to demonstrate in your code where each requirement is located, so use the following template to show where a requirement is. Take special note that there is a leading zero on requirements below 10 to make them easier to search for.

Requirement listing template:

```
//-------------------------------------------------------------------
   Requirement #01: Demonstrate simple output
//-------------------------------------------------------------------
cout << "some message" << endl;
```

(50) Program requirements (generally 2 points each, though the extra credit items can be more variable):

1. Demonstrates **simple output** (cout or printf is fine),
2. Demonstrates **simple input** (cin or getline are fine, better would be to use both :)!),
3. Demonstrates **explicit type casting** (a couple ways to do this),
4. Demonstrates **conditional** statements (should be easy, we use these all the time),
5. Demonstrates **logical operators** (specifically &&, ||, !, or ==, should also be easy)
6. Demonstrates at least one **bitwise operator** (more specifically &, |, ^),

   (for example, 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;
   ```

   if the effect is not obvious right away, write out each variable on a piece of paper, write the starting value next to each variable, run your code, print the variables and their values, write out on paper what the variables and values look like after running the code, and look at how the variable values have changed.
)
7. Demonstrates at least one **loop** (preferably a for, while, or do-while loop)
8. Demonstrates at least one **random number**,
9. Demonstrates understanding of the three general **error categories** we talked about (syntax, logic, and run-time), and yes you should explain this in commented out code rather than submit incorrect or non-compiling code,
10. Demonstrates at least one **debugging** "trick" that we have learned throughout the class (print statements, input verification, checking 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**)
11. Demonstrates at least one **function** that you define (should be easy considering other requirements...),
12. Demonstrates general **functional decomposition** to reduce how large a single section of code is or to make the plan or algorithm obvious for your program; yes, this means you should have a smaller main function and a fair number of functions and that you need to organize your functions into some easy to follow order,

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

14. 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 or other complicated process involving passing values and references around),

15. Demonstrates **function overloading**,

16. Demonstrates at least one **std::string** variable **and** at least one c-style string (character array terminated with the null character ('\0')),

17. Demonstrates some form of **recursion**,

18. Demonstrates at least one **multi-dimensional array**,

19. Demonstrates at least one **dynamically declared array**,

20. Demonstrates at least one **command line argument** (make sure you have a usage statement if no arguments are given to the program, regardless of whether you allow the program to continue without arguments!),

21. Demonstrates definition and use of at least one **struct**,

22. Demonstrates definition and use of at least one **class** and at least one **object**,

23. Demonstrate at least one **pointer to an array**,

24. Demonstrate at least one **pointer to a struct**,

25. Demonstrate at least one **pointer to an object**,

26. (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!)

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

Double-check check-list:

1. Did you write a report with program explanation as well as descriptions of the lessons learned and the techniques used in the class and final project?

2. Does your code meet all the requirements on the list?

3. Is your program obvious to compile and run as well as easy to use?

4. Is your code well commented, consistently following an easy to read style, and have the requirement listing comments using the provided template?