Homework #2
Due Friday, March 4, 11:59 PM
60 points total

Write your programs in C using the environment of your choice. Submit your code via Blackboard as a zip file. We'll discuss what files to include in class. **See the posted documentation guidelines - for this assignment some problems will have points assigned to documentation. Also please give your files meaningful file names (e.g. problem1.c or gameshow.c, etc.) instead of the default such as source.c**

1) (10 points) Arrays

Professor Krunk has 20 students in his class. For purposes of this exercise you can assume each student is identified by a number between 1 and 20. Krunk's (not very ethical) grading scheme is to randomly assign an A to 5 students, a B to 5 students, and a C to everyone else. Write a program that uses array(s) to generate the randomly assigned grades. If the program is run a second time it should generate a different set of grades. There are a number of ways to generate a solution, you are free to use any that you like as long as the program uses at least one array.

2) (10 points) Inventory

In class we went over an example of a program that tracked an inventory of product ID's and product names. Add another option to the program that inputs a product name (keep names as one word) and prints out the corresponding product ID. If the name is not in the inventory then a message should be printed that the name was not found.

3) (10 points) Debugging

This program requires you to use an integrated debugger. Debugging was discussed in this lecture: debugging.pdf. zyBooks does not include a debugger so you will need to use something else. The following code was written by Michael Pawka as part of a obfuscated C programming contest, and was judged as hardest to read! You don't need to understand how the code work - it has intentionally been written to be hard to figure out how it works. It shows that C allows an extreme of obfuscated, hard-to-read code. Here is the C file, you can right-click it to save it to your computer:

- pawka.c

You should be able to run the program and it will produce some interesting output. To do:

1. Use an integrated debugger in any IDE of your choice and set a breakpoint on line 21. Turn in a screenshot of all the variables that are active on that line along with their values.
2. Step through the program and it will return back to line 21. Turn in another screenshot of all the variables that are active on that line for the repeat execution along with their values.

4) (10 points) Babylonian Algorithm

The Babylonian algorithm to compute the square root of a number *n* is:

1. Make initial guess of *answer = n / 2*
2. Compute *r = n / answer*
3. Set *answer = (answer + r) / 2*
4. Go back to step 2 and repeat many times

Write a function with an input parameter that is the number *n* as a double. The function should iterates through the Babylonian algorithm 1000 times and return the answer. The answer should be calculated as a double. Note

that more iterations are needed as *n* increases, so for large values of *n*, 1000 iterations may not give an accurate answer. In main, call your function several times to test it and output the results.

5) (10 points) Functions

You have probably noticed that if you try to read into an int variable using scanf but type some text instead, then the program will do unpredictable things and generally not work correctly. In this exercise you will write a function to safely read an int by first reading it as a string.

1. Write a function named `safeReadInt` that takes no parameters and returns an `int`. The function should read input into a large string using scanf, convert the string into an int, and return the int. Use the `atoi` function to convert from a string to an int. Here is a reference on atoi if you want to learn more. (There is a safer way to do the conversion, strtoul, but we'll skip that for now). Write some code in `main` that calls and tests your function.
2. A disadvantage of using `atoi` is that it returns 0 if the string can't be converted to an integer, but there is no way to tell if the user intended to type 0 or you are getting 0 because of an error. Modify the function so that it loops through the input string and forces the user to type new input if any of the characters typed in are not digits. This will require some type of loop. If all of the characters are digits then `atoi` can be invoked and returned. Recall that '0' through '9' have numerical ASCII values that are contiguous, so you can check for a char < '0' and a char > '9' to see if it is not a digit.

6) (10 points) Separate Compilation

In this exercise you are to create a separate .h and .c file for a small math library. Your math library should have two functions:

- `double babylonianSquareRoot(double n);` - Your square root function from problem 4
- `double absoluteValue(double n);` - You need to write this one, it should return the absolute value of *n*

Put the prototypes into an appropriate .h file, and the implementation in an appropriate .c file. Then create a separate main.c file that includes the .h file and tests both of the functions.