CS352 Assignment #3  
Four Programming Problems  
(Correction #1)

Due Date:  Wednesday, March 25th at 7:00 PM

Turnin:

Submit all files to the cs352s15assg3 turnin directory. Recall that this command looks like

turnin cs352s15assg3 <list of files>

You may list the files individually, or, if they are the only .c files in your directory, you may just use the ‘*.c’ as the second argument.

You may check what files have been submitted by typing “turnin -ls cs352s15assg3”.

Your scripts will be put through a series of test cases and compared to ours using diff. This means that things like spelling, punctuation and whitespace are very important if you would like to pass all the test cases without needing to appeal. Additionally, your filenames will need to match exactly in order for the script to pick it up. If you have any questions about the output of your scripts or the format of the filenames, please stop by our office hours, post on Piazza or email us.

Remember, no late work will be accepted.

The following rules apply to all programs:

1. **Do not use anything covered after chapter 10 in the King textbook. No string variables, no structs, no pointers, etc.**
2. Each program should be a single C file.
3. Do not #include anything beyond <stdio.h>.
1. Ascending Order (ascending.c)
Please write a program that will read a series of numbers from user input and find the longest ascending run.
1. Your program will then print the numbers in the longest run.
2. Each printed number will be on its own line (i.e., followed by a newline).
3. There will not be any additional spacing before or after the number.
4. You will read in at most 100 numbers, separated by whitespace (e.g., spaces, tabs, newlines).
5. All the numbers will be regular C “int” integers.
6. All the numbers will be positive.
7. There may be no numbers in the sequence, which would result in no output.
8. Ascending means “strictly greater than”. This means a run of 1, 2, 2, 3 is broken into two smaller runs of size two: {1,2} and {2,3}.
9. If there is a tie, print the first run of maximum length.
10. The rules above imply that the longest ascending subsequence could be as short as one number.

2. Bowling Scores (bowling.c)
Please write a program that will score a single bowling game (http://slocums.homestead.com/gamescore.html).
1. The program will read a sequence of integers that represent how many pins were knocked down by each roll of the ball.
2. The sequence of numbers will be separated by whitespace (e.g., spaces, newlines, etc.)
3. The program will print one integer, followed by a newline. That integer will represent the bowling score for that sequence of rolls.
4. If the input has errors (e.g., too few or too many numbers, illegal numbers, etc), the program should exit with status 1 without printing anything.
5. Upon successful completion, the program should exit with status 0.
3. **Postfix arithmetic (postfix.c)**

Please write a program that will evaluate postfix arithmetic expressions on integer values. Postfix operations are evaluated using a stack of values.

1. The program will read in a postfix arithmetic expressions and evaluate them.
2. Postfix expressions are composed of integer literals and the operators, +, -, *, /, and, possibly, white space.
3. All integer literals will be non-negative
4. All integer values may kept in “int” variables.
5. You may assume the evaluation stack will never be more than 10 deep.
6. Operators have C semantics.
   a. E.g., “3 4 +” is an expression that would evaluate to 7.
   b. “3 4 5+*” would print 27.
7. The input will be terminated when there is no more input, at which point the stack of values will be printed starting with the value deepest in the stack.
   a. Each number should be printed on a separate line, terminated by a newline, with no other characters of output.
   b. E.g., “3 4 + 5” would print two lines of output: 7 and 5 (in that order).
8. If there are no input expressions, then there is no output.
9. If there is an error in the input, the program should exit silently with status 1.
10. If there were no errors, the program should exit with status 0.

4. **Multi-Precision Addition (multi.c)**

Please write a program that will add two large numbers.

1. Your program will read these two potentially huge numbers from user input.
2. The numbers will be separated by a newline.
3. The program will then add these two numbers and print out their sum, followed by a newline. There will be no additional spacing before/after the sum.
4. The numbers will be of the form digits.digits (with at least one before and one after the decimal point.)
5. The input and the answer will not have more than 50 digits to the left of the decimal point nor 50 digits to the right.
6. Your answers must not lose any digits---these values will not fit in C’s given types, so you will need to construct them out of arrays.

Solutions that use “includes” to solve this problem will not receive credit.