CS103L – Abundant Commas

1 Introduction

In this assignment you will develop an algorithm to perform some common tasks:
1.) Finding numbers that meet the mathematical definition of an "abundant" number and
2.) beautify the output of an input integer by adding commas (i.e. 1234567 => 1,234,567).
This program will require you to utilize the following knowledge and skills:

- variables
- input and output
- conditionals
- for and while loops
- simple algorithms

2 Program Descriptions and Requirements

Abundant. The first problem you will solve is to identify so-called "abundant" numbers. Mathematically, an "abundant" number is one where the **sum** of its proper divisors (positive divisors other than the number itself) is greater than the number. For example, n=12 is abundant because the sum of its divisors, 16 (1+2+3+4+6) > 12. The number n=15 is NOT abundant because its proper divisors do NOT sum to more than 15 (i.e. 1+3+5=9 <= 15).

Beyond determining whether a single number is "abundant", this assignment asks that you take any sequence of positive integers, separated by spaces and ending with a 0 to indicate the last value in the sequence, and find a.) *HOW MANY* numbers in the sequence are abundant and b.) List the 3 *MOST* abundant numbers and their abundance.

For example: If we provide the input `18 15 12 48 32 36 0` your program should produce the following output:

```
Abundant number count: 4
Top 3 most abundant numbers:
48 : 76
36 : 55
18 : 21
```

The first value of the most 3 abundant numbers (e.g. 48) should be the number with the highest sum of proper divisors (e.g. 76), the 2nd number listed should have the 2nd highest sum, etc.

Commas. The second problem you will solve is to read in an integer (any valid 64-bit integer = **`long long`** type) and output the same number but with commas inserted. So if the user entered `-1234567890` your program should output `-1,234,567,890`. Commas should appear after every three significant digits (provided more digits remain) starting from the decimal point and working left toward more significant digits. If the number entered does not require commas, do not add any. For example, if the input is `234` you should output `234`. The input `0` should produce output `0`. Note in the example above that the number can be positive or negative. Your output must maintain the case of the input.

This problem could be easily solved if you were allowed to read the information in as a string or character by character. However, the input will be read in as a **single integer** and you will need to use appropriate arithmetic to isolate and print each digit of the number one at a time and output commas at appropriate locations.

To repeat, you **may NOT** receive input character by character or as a string. You **may NOT** use arrays, strings, vectors, or any other composite. You **may ONLY** use type: `bool`, `long long`, and `(unsigned) int`. *The point of this assignment is for you to determine an algorithm to isolate the digits one at a time from most-significant digit to the least and output them one at a time, inserting commas where appropriate*.

3 Skeletons and I/O Format

Each program will be developed separately on Vocareum. When you start each part on Vocareum, a skeleton will be copied into your work folder. You may not change what is provided in the skeleton but may only add code to the indicated areas. Failure to follow this direction will lead to receiving a 0 on that part of the program.

Abundant program: We have provided some declarations to store the number of abundant numbers and the three numbers (n1, n2, n3) with the largest "abundance" (i.e. sum of proper divisors) (a1, a2, a3). Also, we have provided all the output necessary to prompt the user for the sequence and to output the results. You will need to add appropriate input (cin) statements but should not add more output (cout) statements in the final submission. By providing these outputs at the beginning and end, we can ensure your output format matches the expected format of our automated checkers. The provided comments in the skeleton give additional information and serve as requirements (i.e. you must follow the directions given in the comments).

Commas program. We provided only a basic skeleton with some comments. You must write the remainder of the program from scratch. Be sure you follow the rules stated earlier about NOT reading the input character by character or as strings, vectors, etc.

Your program should output a prompt to the user to enter an integer on a single line. It should then receive the input in a `long long` variable (64-bit integer). It should then out the resulting number with commas on a **single line** which is the **last line** of output.

4 Building and Running your Code

To compile and run your code, you will need to use the Vocareum terminal/command line. Review your labs for the appropriate g++ command to compile your code into an executable. You should then be able to test and run your program by executing the command `./abundant` or `./commas` at the terminal prompt. Continue editing, compiling, and running your code until you believe it works, testing it with several input scenarios that would exercise various cases in your code.

5 Style

In addition to correctness, you must follow ALL the code style guidelines posted here: http://bytes.usc.edu/cs103/coursework/style.

6 Readme

Each part of the program will have a readme.txt file with a few reflection questions. Please take a few moments to fill those out. You do not need to write volumes but your answers should be thoughtful to receive full credit.

7 Submission

Once you feel your code is correct, you can click submit and Vocareum will run a brief automated script to provide a basic sanity test. You may review the output report from that script. If it indicates failures, you should attempt to fix the issues and resubmit. The report will indicate what inputs where used that failed to lead to correct behavior so that you can duplicate it yourself by running from the terminal. You can submit as many times as you like. When we grade your assignment, we will use more input cases to test its correctness for nominal and edge cases. We strongly recommend you think about potential values that may cause your program to malfunction and ensure your code works correctly for those cases.

8 Q&A

Q: For **abundant**, do we have to worry about non-numeric inputs (letters, etc.) in the input, negative numbers, or an input sequence not terminated by 0?
A: No we will ensure valid inputs.

Q: For **commas**, do we have to worry about input numbers larger than the range of a long long?
A: No, we will not input numbers larger than the given range of a long long.

Q: Can we use functions for this assignment?
A: You can if you like, but they are not required, and don't help in any serious way.

9 Grading Scheme

Total = 20 pts. (10 pts. per assignment)