COP3223 Section 1: Intro to C - Programming Assignment #2

Due date: Monday, Feb 11, 2019

Problem A: Abundant Numbers (abundant.c) (25 points)

Some computer scientists really care about divisibility. They especially like numbers that have lots of divisors. In particular, they like numbers with proper divisors that sum to greater than the number itself. These numbers are called, "abundant numbers." A proper divisor of a number is any number strictly less than it that divides evenly into it. For example, 6 is a proper divisor of 12, but 12 is not. 12 is an example of an abundant number because 1+2+3+4+6 = 16 and 16 is greater than 12. As this example shows, one way to find out all the proper divisors of a number is to try each integer (starting at 1) out until you reach the number divided by 2. (It's impossible for any number in between 7 and 11 to divide evenly into 12 because it is guaranteed that the result of the division is less than 2 since we are dividing by a greater number, and greater than 1, since we are not dividing by the number itself.)

Write a program that reads in a list of numbers, and for each number, determines and prints out whether or not that number is abundant.

Input Specification

1. The first integer input will be a positive integer, *n*, indicating the number of test cases coming next.
2. The next *n* inputs are single positive integers each, and for each you are to determine whether the number is abundant or not.

Output Specification

Output a line with one of the two the following formats for each input number:

```
Test case #t: X is abundant.

Test case #t: X is NOT abundant.
```

where t represents the number of the test case (starting with 1), and X represents the number for that case.

Sample

Here is one sample output of running the program. Note that this test is NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given. (User input is in italics).

Sample Run

```
Please enter n followed by n numbers: 3 6 12 17

Test case #1: 6 is NOT abundant.
Test case #2: 12 is abundant.
Test case #3: 17 is NOT abundant.
```

Problem B: Roller Coaster Redesign (coaster2.c)(50 points)

Your boss has noticed that maximizing the length of the train does not always maximize the number of passengers. She's come up with a great idea for you to improve your program, so that you can calculate the actual maximum number of passengers the roller coaster can support. Her idea is as follows:

Let's say the maximum length of a train is 55. Then, we can simply start by trying out a train of length 10 (one car), and seeing how many passengers such a train would support. Then, we can try a train of length 18 (two cars) and recalculate the number of passengers this design would support. If this is better than the best design we've seen so far, simply save this new value. Continue in this fashion, until we've tried all possible trains. In this situation, we would try trains of lengths 10, 18, 26, 34, 42, and 50. (We stop here because the next train, of length 58, would be too long.) In each of these candidate lengths, compute the following quantity: total rollercoaster passengers divided by the total length of all the trains. Store each value in an array, so you will be storing as many values as there are lengths. Finally, compute the average of those values, and print it out as a final answer.

The user will input the same information as was inputted in assignment #1 part C.
Note that this time you will not be inputting the number N, and you will not print the message about the surplus. This time, however, your program should output the actual number of maximum people the ride can support, the number of cars per train that achieves this maximum, and the final average of the passengers/length ratio.

Do the problem in small stages. First, ensure that you are getting the correct enumeration of the various train-lengths, then (for each train length) compute how many trains can fit on the track; that gives the number of people on the track simultaneously, check if this beats the max-so-far, if so, replace max. For each train length choice, you need to compute the ratio of people to cumulative car length, and (it is a good idea to) store this in a one-dimensional array that can store decimal values. Finally, compute the average of the ratios, and print that out.

Input Specification
1. The total length of the track will be a positive integer (in feet) less than 10000.
2. The maximum length of a train will be a positive integer in between 10 and 100, inclusive.

Output Specification
The output should consist of three lines. The first line outputs the maximum number of passengers on the roller coaster at any one time with a single statement of the following format:

```
Your ride can have at most X people on it at one time.
```

The second line should output the number of cars in the train that achieves this maximum with a single statement of the following format:

```
This can be achieved with trains of Y cars.
```

Note: If there are multiple ways in which the maximum number of people can be supported, output the smallest number of cars that achieves this maximum. (Thus, if both 2 cars and 4 cars lead to 100 people on the ride and this is the maximum, then your program should output 2 cars.)

The third line should simply output the average value of people to length, as:

```
AVG Ratio: XXXX.XXX
```

Output Samples
Two sample outputs of running the program are included below. **Note that these samples are NOT a comprehensive test.** You should test your program with different data than is shown here based on the specifications given above.

Sample Run #1
What is the total length of the track, in feet?
1000
What is the maximum length of a train, in feet?
42
Your ride can have at most 112 people on it at one time.
This can be achieved with trains of 4 cars.
AVG Ratio: 0.451

(Note: The maximum is achieved when each train has 4 cars on it, and each car has 4 people, at most. Thus, 16 people can sit in one train. This train has a length of 34 feet, and since 34 feet x 7 = 238 feet, which is less than 25% of the total track length, this

means that exactly 7 trains can be placed on the track at the same time. Thus, 16 people/train x 7 trains = 112 people total.)

Sample Run #2

```
What is the total length of the track, in feet?
4025
What is the maximum length of a train, in feet?
89
Your ride can have at most 480 people on it at one time.
This can be achieved with trains of 6 cars.
```

AVG Ratio: 0.467

Problem C: Practice Strings (lastnames.c)(15 points)

Read in n, then n lastnames, and check to see if the first in the list is ever repeated again.

Sample Run #1

```
Enter n, followed by n Last names (each last name must be a
single word):
5 Reagan Bush Clinton Bush Obama
First name in list is not repeated.
```

Sample Run #2

```
Enter n, followed by n Last names (each last name must be a
single word):
4 Bush Clinton Bush Obama
First name in list is repeated.
```

Part D: Embed all the above Parts in a menu (menu.c) (10 points)

Write this similar to bankinclass.c

Deliverables

Four source files:

1) *abundant.c*, for your solution to problem A
2) *coaster2.c* for your solution to problem B
3) *lastnames.c* for your solution to problem C
4) *menu.*c for your solution to part D.

All files are to be submitted over WebCourses2, Canvas.

Restrictions

Although you may use other compilers, your program must compile and run using Dev C++. Your programs should include a header comment with the following information:

your name, course number, section number, assignment title, and date. Include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness.

2) Your programming style. Even if your program works perfectly, if your programming style is poor, you could get 2% deducted from your grade.

3) Compatibility to Dev C++ (in Windows). If your program does not compile in this environment, you will get a **sizable** deduction from your grade.