Fifth Annual
High School Programming Contest

May 16, 2009

Each problem in this packet contains a brief description, followed by two example executions of a successful implementation.

The example input and output shown for each question should be regarded only as a suggestion. We will test your programs on the examples provided, as well as other sample input test cases. In the examples, the text that is underlined serves as the input to the program as entered by the user.

If your program fails, the result returned by the submission system will state the counterexample test case that caused your program to be judged incorrect. Each incorrect answer will be assessed a 20 minute penalty. As noted in the rules, the overall time will be considered as a tie breaker (with total number of problems solved serving as the initial ranking metric).

Please pay close attention to the directions in each problem description. In some cases, assumptions are stated about the limitations of the input, which are designed so that you do not have to consider difficult cases or perform input validation.

Problem 1:
Telephone Number Helper

Some companies like to cite an 800 number that has a special connection to something about their business (e.g., 1-800-FLO-WERS). Create a program that allows a user to enter a phone number in the format of 1-XXX-XXX-XXXX, where "X" is an alphanumeric character. All parts of the phone number that are alphabetic characters should be converted to a number based on the following standard mapping found on most phone keypads:

You may assume that the input is perfectly formed (dashed separators always provided, number starts with 1, and no illegal characters are given). Thus, you do not need to perform any specific validation checks on the input, in order to make your program simpler.

Your input and output should resemble the following examples.

Example 1:

```
Input phone number: 1-800-FLO-WERS
The real phone number is 1-800-356-9377
```

Example 2:

```
Input phone number: 1-205-GOU-ABGO
The real phone number is 1-205-468-2246
```

Problem 2:
Sound Volume Manipulator

Sounds are often digitized using a sample rate that ranges in value from -32,768 to 32,767. There can be 44,000 such samples in a second of recording on a CD. The larger the sample value, the louder the sound. The more quickly samples change value, the more high-pitched the sound.

Write a program that reads an array of samples representing a sound file and doubles the volume of the recording. Keep in mind that the maximum value is 32,767 (and -32,768 in the other direction). If a value is beyond these two limits, the volume is clipped to the max (i.e., a doubled sample value of 40,000 is clipped to 32,767).

Your program will not know the size of the sample stream and must handle one or more streams of values. A stream is just a list of integers entered on a single line– in the Example 2 below, the longer list wrapped to the next line.

Example 1:

```
Input stream: -10000 0 10000 20000 10000
Output stream: -20000 0 20000 32767 20000
```

Example 2:

```
Input stream: 1 2 4 8 16 32 64 128 256 512 1024 2048 4096
8192 16384 -32768
Output stream: 2 4 8 16 32 64 128 256 512 1024 2048 4096
8192 16384 32767 -32768
```

Problem 3:
Chromakey Converter

A color image is often stored as a matrix of RGB pixels, where R is the intensity of red, G is the intensity of green and B is the intensity of blue, intensity ranging from 0 to 255 (e.g., (0, 0, 0) is black, (127, 127, 127) is gray, (255, 255, 255) is white, (255, 0, 0) is very red, (0, 255, 0) is very green, and (0, 0, 255) is very blue).

Chroma key works by taking a picture with a blue background and copying a different background into the blue pixels (e.g., how weatherpersons stand in front of a blue screen that is converted to a map behind them showing weather patterns).

Write a program that reads two sets of RGB pixels; the first set of pixels being the picture with a particular subject in front of a blue background, and the second set of pixels having the new background that is to be copied into the pixels that are blue in the first set. Your program should copy the new background pixels onto the corresponding blue pixels of the first set of pixels.

A pixel in the original image should be replaced by the second image if the sum of the red and green values is smaller than the blue value. For example, the value (50, 100, 200) would be considered blue and in need of replacement.

You do not have to check the correctness of the input – you may assume the input is correctly formed and that each of the two sets has the same number of pixels. A comma separates the individual pixels and a semicolon separates the two sets of pixels. The input is entered as a single input (the examples below span lines due to word wrap).

Example 1:

```
Input pixels: (125, 125, 125), (10, 20, 30), (50, 100, 200); (47, 63, 72), (81, 10, 196), (38, 220, 10)
Output image: (125, 125, 125), (10, 20, 30), (38, 220, 10)
```

Example 2:

```
Input pixels: (0, 0, 255), (0, 0, 0), (128, 128, 255); (47, 63, 72), (38, 220, 10), (255, 0, 0)
Output image: (47, 63, 72), (0, 0, 0), (128, 128, 255)
```

Problem 4:
Space Coordinator

Assume that you have been hired by an office planning firm to write a program that helps to determine if the coordinates of specific office furniture overlaps. The firm would like to express the coordinates of various pieces of office furniture by specifying the lower-left coordinate and the top-right coordinate of the furniture (assuming all furniture is rectangular).

The firm wants to make sure that an office space plan does not have overlapping furniture. You must write a program that receives as first input the number of pieces of furniture to be assigned to the room, and then the individual coordinates of each piece of furniture (e.g., 1 1 2 2 would be furniture with a bottom-left corner at <1, 1> and a top-right corner at <2, 2>). The output should indicate whether the set of furniture pieces overlap, or not. You may assume that every office is 10 x 10 spaces wide.

Example 1:

```
Furniture coordinates:
3
0 0 2 2
1 3 3 5
4 1 7 3

Result: No overlap
```

Example 2:

```
Furniture coordinates:
4
4 4 7 7
3 3 5 5
6 2 7 3
1 1 2 2

Result: Overlap
```

Problem 5:
Perfect Number Checker

A positive integer is said to be a perfect number if it is equal to the sum of its positive divisors less than itself. For example, 28 is perfect, because

$$28 = 1 + 2 + 4 + 7 + 14$$

On the other hand, 12 is not perfect, because

$$12 != 1 + 2 + 3 + 4 + 6$$

You are to write a program that prompts the user to enter a positive integer and responds by reporting whether or not the given number is perfect.

Example 1:

```
Enter a positive integer: 12
12 IS NOT perfect.
```

Example 2:

```
Enter a positive integer: 28
28 IS perfect
```

Problem 6:
Bob's Triangle

Pascal's Triangle is constructed by forming a new digit as the sum of the digits above it in the triangle. The first few lines are:

```
          1
        1   1
      1   2   1
    1   3   3   1
```

Mr. Pascal had a cousin named Bob. Bob wasn't as mathematically oriented as his famous kinsman, but Bob wanted a triangle of his own. Your task is to determine the pattern for Bob's triangle (understanding the pattern is a key to your solution), then write a program that can generate it.

Allow the user to input the number of rows of Bob's triangle to generate, and then display those requested number of rows, Your output may be left-aligned, as shown below.

Example 1:

```
Number of rows: 5

Output:
1
1 1
2 1
1 2 1 1
1 1 1 2 2 1
```

Example 2:

```
Number of rows: 6

Output:
1
1 1
2 1
1 2 1 1
1 1 1 2 2 1
3 1 2 2 1 1
```

This page intentionally left blank for scratch paper.