Problem Set

Trinity University ACM
High School Programming Competition
April 11th, 2009

Problem 0 – Bell's Curve

To combat grade inflation, Dr. Hardgrade is a strong believer in distributing grades according to quartiles. That means the top 25% of the class gets an A, the next down gets a B, the next a C, and the lowest 25% of the class will always fail. He has asked you to write a program for him that will take in a class roster with grades and tell him what letter grade each student should get.

Input: The input will begin with a number telling you how many inputs you need to consider. Each input will start with a line that has a number, S, that tells you how many students are in the class. That will be followed by S lines that have student names and grades. The student names will be single words. The number of students in each class will always be a multiple of four. No two students will have the same name or grade. (S<101)

Output: For each input set you need to print out a header telling which course it is followed by the students in the order they were read in with the grades that they received. The header should read "Course #" where # is the input number beginning at 1.

Sample Input:

```
2
4
Mark 65
Amy 72
Jen 98
John 84
8
Mike 23
Tim 96
Tony 87
Roger 73
Ralph 106
Janet 88
Meghan 95
Maddie 42
```

Sample Output:

```
Course 1
Mark F
Amy C
Jen A
John B
Course 2
Mike F
Tim A
Tony C
Roger C
Ralph A
Janet B
Meghan B
Maddie F
```

Problem 1 - Toll Road Terror!

Dr. Lewis is driving the ACM Programming team to a competition in Louisiana. However, while driving through Houston, he has managed to find himself in a maze of toll roads. Since Dr. Lewis' motto is "A penny saved is a penny earned" he gives the task of making an algorithm to find the cheapest route from one location to another to the team. In this case it's you.

Input: Input to this problem will begin with a single line containing an integer, N, where N is the number of data sets that will be entered. Each data set will begin with a single line containing one integer, M, that tells you the number of roads. The second line will contain two characters ('A' - 'Z') separated by a space. The first character represents the starting location and the second represents the destination. That will be followed by M "road" lines which contain two letters ('A' - 'Z') and an integer, each separated by a space. The first character represents starting location for that toll road, the second represents the ending location, and the integer, C, is how much it costs to travel on that road. Given any two locations, there can be no more than one road between them and it costs the same amount to travel in either direction on that road. All roads are two way.

$0<M<=325$
$0<C<3000$

Output: For each data set, print out the lowest cost to get from the starting location to the ending location.

Sample Input:
```
2
3
A C
A B 2
B C 3
A C 7
6
A D
A B 4
A C 1
A D 6
B D 2
B C 1
C D 6
```

Sample Output:
```
5
4
```

Problem 2 - Wheels of Glory

Your crazy computer science teacher, Dr. Riedell, has run into a problem. He lost all the grades for all his students and there is only one day left in the school year. He decides to rent the skating rink for the day and determine the students' grades by how well they perform in an endurance test. You get to write a program that will determine your grade based on the grading requirements and how fast you can skate.

Input: Input will begin with a number, N, that represents how many datasets will be inputed. Each dataset will be on one line containing first an integer, M, representing the number of minutes to skate, an integer S representing how many seconds it takes you to complete a lap, an integer, A, representing how many laps it takes to get an A, an integer, B, representing how many laps it takes to get a B, and lastly an integer, C, representing how many laps it takes to get a C.

0 <= M <= 10,000,000
0 < S <= 10,000,000
A >= B >= C

Output: For each dataset, output the letter grade they will receive in the class. If a student doesn't receive at least a C, print out F. Only whole laps count. For example, if a student completes one lap every 18 seconds, they will only have completed three laps if the time limit were a minute.

Sample Input:
```
4
10 60 10 8 7
10 60 11 8 7
0 200 12 8 5
20 12 200 150 50
```

Sample Output:
```
A
B
F
C
```

Problem 3 – Schemers

One of the earliest programming languages was a language called LISP (which some argue stands for Lots of Insipid, Stupid Parentheses). LISP was, and is, commonly used in the field of AI. One of the derivatives of LISP is a language called Scheme that is used fairly commonly in education because it is a remarkably simple programming language. For this problem, we want you to write a Scheme interpreter and use it to run some programs. Of course, we don't expect you to have a fully functional Scheme implementation. You only have to do a small subset of the language that is described below.

Scheme, like LISP before it, is a prefix language. That means that operators come before their arguments. Also, expressions are fully parenthesized. So the expression to add 3 and 5 is (+ 3 5). One advantage of this is that you can have more than two arguments. For example, (+ 3 4 5 6 7) is perfectly happy Scheme for 3+4+5+6+7. You need to implement the operators +, -, *, and / and be able to nest them. So for 3*2+8/4 you should be able to parse the Scheme expression (+ (* 3 2) (/ 8 4)). Note how things are nested.

The second thing you need to implement is the if construct and with it some simple boolean expressions. The if construct in Scheme looks like the following (if *cond then else*) where *cond* is any valid Scheme expression that is true or false, *then* is the Scheme expression that is evaluated and returned if *cond* happens to be true, and *else* is a Scheme expression that is evaluated and returned if *cond* is false. For the conditional expressions you only need to have <, >, and =. So you can say things like (< 2 3) which would be true. Note that there are spaces between the <, 2, and 3. All Scheme expressions have spaces between the tokens.

The last thing you need to be able to handle is the define command. Define lets you give names to things. It is kind of like declaring variables, but a bit more powerful. The simplest form is (define *name value*) where *name* is the name of the thing you are defining and *value* is any Scheme expression. So (define eight (+ 5 3)) will make it so that eight evaluates to 8. The other form of define (define (*name varlist*) *expression*). This form defines a function with the name *name.* The *varlist* is a space separated list of parameters. The *expression* is any Scheme expression and it can use the variable names that are part of the *varlist*. Using this syntax, the expression (define (square x) (* x x)) defines a function that calculates squares. If after doing that the expression (square 5), will have a value of 25.

Input: The input for this program will be a bunch of Scheme using the rules described above. To keep things simple define will only be used as the top level expression. Note that expressions can be split across multiple lines. The parentheses will tell you where expressions end. The input will end with the expression (quit). All numbers should be treated as ints. No function invocation will ever require more than 100 recursive calls to execute so you shouldn't worry about stack overflows or identifying tail recursion. All inputs will follow the rules outlined above. There will not be syntax errors in the input.

Output: Your output should be a print a line with the value of any expression that isn't a define.

Sample Input:

```
8
(+ 5 7)
(define num 5)
```

```
num
(define (square x) (* x x))
(square 9)
(square (* 5 2))
(if (< (square 5) 30) 1 0)
(define (fact n)
        (if (< n 2)
                1
                (* n (fact (- n 1)))))
(fact 3)
(define five 5)
(+ 2 (fact five))
(define (sum a b) (+ a b))
(sum 2 4)
(quit)
```

Sample Output:

```
8
12
5
81
100
1
6
122
6
```

Problem 4 - Rock, paper, scissors.

Your friends are bored in class and they have decided to play rock, paper, scissors by texting their selections to a judge. They have also decided that you are to be the judge. The problem is, you want to actually get a good grade in the class and pay attention to what the teacher is saying. To solve this dilemma, you write a program to do the judging for you.

Input: The input will start with a number of different contests that you have to consider, N. After that will be N lines, each with two words on it. The words will be either “rock”, “paper”, or “scissors”.

Output: For each contest you should print who won or if it was a tie. If there is a tie you print the word “tie”. If not, print the selection of the winning player.

Sample Input:
```
5
rock paper
rock rock
scissors paper
paper scissors
rock scissors
```

Sample Output:
```
paper
tie
scissors
scissors
rock
```

Problem 5 – Bits-to-ASCII

In a fit of incredibly geekiness, your friends have decided that you should start encrypting messages you send back and forth by putting them into a binary representation of the ASCII values. After all, who is going to try to find meaning in a whole bunch of seemingly random 1s and 0s? To make it even tougher for people to find, you decide you will put the messages into web pages as comments so people will only see them if they view the source of your HTML files. Your friend is going to write the code that converts the original message over to binary bits. Your job is to make a program that will take a string of ones and zeros and print out the original message.

Input: The input to the problem will begin with a single number on a line telling you how many input sets there are. Each input set will be a single line with a string of 1s and 0s. The string will have a length that is a multiple of 8 because there are 8 bits in each byte you need for ASCII values. All strings will be 800 characters or less in length.

Output: For the output, you should print one line for each input. The line will be the decoded text for the input values.

Sample Input:
```
4
01000011011000010110111000100000011110010110111101110101
01110010011001010110000101100100
011000100110100101101110011000010111001001111001
010000010101001101000011010010010100100100111111
```

Sample Output:
```
Can you
read
binary
ASCII?
```

Problem 6 – Bruiser and Scratch

You recently bought this cool little WiiWare puzzle game called Bruiser and Scratch. While you are pretty good at it, you want to know if the solutions you are coming up with are optimal. After all, they have trophies for optimal solutions. Being the programmer that you are, you figure that you should be able to write a program that tells you the number of moves in an optimal solution to a board. You decide to start with a simplified version of the game where you just check if a board is solvable.

The way Bruiser and Scratch is played is that you move around a character on a 2-D grid and push pieces. Normally there are multiple types of pieces, but for your simplified model you will just use one. Your goal is to get all the pieces joined together. You can push pieces horizontally or vertically, and if a piece moves off one edge of the board it reappears on the other side. When you push the piece, it will slide until it hits something. If it doesn't hit anything it comes around and hits you which is bad. If it hits another piece, it will merge with that piece.

At the beginning you will have pieces, represented by an A, spread around the board. When all the pieces have been merged into a single piece, you beat the level. If you get to a point where there are no more pushes that merge pieces you have hit an impasse. Let's run through an example board.

```
#########
#A#A#A#A#
#########
```

This 9x3 board starts with four pieces. The # signs are used to represent open squares. If you push the left most piece to the right, the board will change to the following.

```
#########
###A#A#A#
#########
```

The two As merged. If you give two more pushes to the right all the pieces will be merged and you have won. So this board can be solved. Now let's consider a different board.

```
#A#######
####AA###
#########
```

In this board, the top A can be pushed in any direction (you are allowed to stand at an edge and push a piece on the other side), but because it would never hit anything, such a push would hit you and make you lose the board. This board does not allow you to get all three pieces together. The best you can do is get down to two pieces.

Input: The input will begin with a single line that has in integer, N, telling you how many boards to consider. Each board input will begin with a number, R, telling you how many rows the board. That will be followed by R lines of text, each of the same length. The lines will have only # and A characters for blank spaces and pieces respectively. No board will have more than 10 pieces to start with.

Output: You should output one line for each board telling you how many pieces you can get the board down to.

Sample Input:

```
2
3
#A#######
#A##AA###
#########
4
#A#######
####AA###
##A######
#AA######
```

Sample Output:

```
1
2
```

Problem 7 – Like Rabbits

It's Easter and you had the great idea of buying your sweetheart two new born rabbits, a male and female. It doesn't take long until you realize you've made a huge mistake. These rabbits are not your ordinary rabbits, they are mutants who mature and give birth super fast and to make things worse, they breed.... like rabbits. After they've been born it takes 1 hour for the rabbits to mature, during the second hour the rabbits give birth to a male and female rabbit. The rabbits continue to reproduce and give birth to a single male and female rabbit every hour from then on. You make a mad dash to the pet store to buy food for all of your rabbits but you need to know how much to get (or whether you should consider buying a bigger house while you're out). To do this, you write a program that will compute how many rabbits there will be at the specified hour.

Input:
Input to the problem will begin with a line containing a single integer, N. This integer will represent the number of problem sets to solve. Each of the following n lines will contain a single integer, H, representing how many hours have passed since purchasing the new born rabbits. (N<100, H<30)

Output:
For each input set, output a line containing a single integer that represents the number of rabbits you have at that hour. Note that you get the initial bunnies as immature newborns.

Sample Input:
```
3
0
4
7
```

Sample Output:
```
2
6
26
```

Problem 8 - Coding Your Way Into A Blizzard

Blizzard Entertainment has hired you to help them finish StarCraft 2 before the end of the decade. In order to outdo their own success of placing the original StarCraft as a mainstay of RTS tournaments, they have decided to add some native support for tournament adjudication. Your boss believes it is folly to spend time playing out a game that has already been definitively won by one side, and is merely being dragged out by the other.

You've been asked to create a simple program that will determine if such a point has been reached in a game. To do this, you are to calculate the total power of all units on one side, and compare them to the total power of all units of the opponent. If the difference of the totals exceeds a supplied threshold, the game should be declared over and the player with the larger total the victor. You have been asked only to model this mode for 1v1 matches.

Here are the strengths for each unit:

Zerg
1. Zergling – 10
2. Hydralisk – 35
3. Mutalisk – 29
4. Ultralisk – 250
5. Baneling – 125

Terran
1. Marine – 15
2. Firebat – 25
3. Reaper – 9
4. Ghost – 75
5. Siege Tank – 300

Protoss
1. Zealot – 18
2. Immortal – 100
3. Dark Templar – 40
4. Archon – 33
5. Carrier – 485

Input: The input will begin with a single number, N, on a line representing how many matches you should consider. Each input will consist of three lines of text. The first two give comma separated lists of the units of player 1 and player 2 respectively. Each unit specification will have an integer (U<1000) followed by the name of the unit type. The third line for a match is the margin, M, that has to be reached to consider one side to have a victory. (M<=10000)

Output: For each match you will print one of the following: “Player 1 wins.”, “Player 2 wins.”, or “Too close to call.” It is too close to call if the difference between the sides is less than required margin.

Sample input:

```
2
```

```
9 Zealot, 5 Immortal, 1 Carrier, 3 Archon, 4 Dark Templar
15 Marine, 10 Firebat, 2 Ghost, 10 Reaper, 5 Siege Tank
500
1 Carrier, 1 Immortal
58 Zergling
10
```

Sample output:

```
Player 2 wins.
Too close to call.
```

Problem 9 - Room Usage

Your computer science teacher is responsible for scheduling the computer lab when various groups want to use it. For some reason, it is actually in high demand and the requested times can be quite odd. What your teacher has learned is that there is no way to please everyone and the way to minimize complaints is to satisfy the most requests. So instead of trying to maximize how much time the room is in use, the goal of scheduling has become creating a schedule that maximizes the number of different requests that are granted.

The task of dealing with the room scheduling is really getting in the way of teaching. Since you want your teacher to have time to teach you some more advanced concepts, you volunteer to write a program that will do the scheduling. For the first cut we just want a program that will tell us how many requests we will be able to satisfy. So the program will be given a set of events with start and finish times (in military time) and you have to return the maximum number of non-overlapping events that can be scheduled.

Input: The input for this problem will begin with a single line telling you how many different days of scheduling you have to consider, N. That will be followed by N inputs sets. The input for each day will start with a line that has a single number, R, on it representing the number of requests that day. That will be followed by R lines. Each line will have a start time and a stop time in military format of HH:MM. Events can not be scheduled back-to-back. There has to be at least a minute between events. The events will be given to you in random order. (N<100, R<100)

Output: Your output will have N lines. Each line should have a single number telling the maximum number of requests that can be granted for that day's schedule.

Sample Input:

```
2
5
08:30 09:30
10:00 20:00
10:30 12:30
18:00 21:00
15:15 15:16
6
10:00 11:00
11:00 12:00
12:00 13:00
13:00 14:00
11:15 11:45
13:05 23:00
```

Sample Output:

```
4
4
```