High School Programming Contest

May 13, 2006

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

You will submit an answer by copying the source code of your solution onto a floppy disk, which you must then pass on to one of the volunteers in your assigned room. Your answer will be graded and a result form will be returned to you.

The example input and output shown for each question should be regarded only as a suggestion. We will test your program on the example provided, as well as two other sample input test cases.

If your program fails, the result sheet 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 secondary tie breaker (with total number of problems solved 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.

Problem 1:
Merging at the Brick Yard

The Indianapolis 500 race is just a few weeks away. They need your help in creating a program that will provide the order in which the cars are to appear at the starting line (ordering is according to the number placed on the car).

There are two garages where the race cars are stored. Each garage will form a line of 5 cars that are sorted by the number on the car. As the cars from the two garages merge to form a single start line, your program must maintain the ordering such that the smallest numbered cars are earlier in the line than those with larger numbers.

Note: Some cars may share the same number, and the numbers range from 1 to 10. You may assume that there are always 5 cars in each garage prior to the merging and each garage is already sorted.

Example 1:

```
Enter the order for garage 1: 1 1 2 3 4
Enter the order for garage 2: 2 2 3 4 5

The final merged order is:

1 1 2 2 2 3 3 4 4 5
```

Example 2:

```
Enter the order for garage 1: 1 2 3 4 5
Enter the order for garage 2: 5 6 7 8 9

The final merged order is:

1 2 3 4 5 5 6 7 8 9
```

Problem 2:
Circle Intersection

In this problem you must write a program that determines if two circles intersect each other. The input to your program will be the <x, y> coordinates for the center of each circle, followed by the radius of each circle. The output will state whether the circles overlap, do not overlap, or are tangential (i.e., tangential circles touch each other at just one common point).

Example 1:

```
Enter the coordinates and radius for circle 1: 10 10 3
Enter the coordinates and radius for circle 2: 10 6 1

The circles are tangential.
```

Example 2:

```
Enter the coordinates and radius for circle 1: 8 8 3
Enter the coordinates and radius for circle 2: 8 4 2

The circles overlap.
```

Problem 3:
Diamond Printer

This program will print a shape on the screen using asterisks ("*") characters. The user will be prompted to enter an ODD number between 1 and 99 (you may assume test cases will not be beyond this range and that all test cases will be odd numbers). The shape that will be printed resembles a diamond, where the number provided by the user represents the number of *'s printed on the middle line. The line above and below will be centered and will have 2 less *'s than the middle line. This reduction by 2 *'s for each line continues until a line with a single * is printed at the top and bottom of the figure.

Example 1:

```
Enter an odd number (1-100): 1

*
```

Example 2:

```
Enter an odd number (1-100): 5

  *
 ***
*****
 ***
  *
```

Problem 4:
Three-Frame Bowling

In bowling, there are 10 pins per frame and a bowler has two chances per frame to knock down all pins. In general, the score for each frame is simply the number of pins knocked down on two rolls of the ball. If all ten balls are knocked down by the second ball, a "spare" is recorded – the score for that frame is the 10 + the number of pins knocked down by the first ball of the next frame (for example, a player rolls a spare in the first frame; with the first ball of the second frame, the player knocks down seven pins. The first frame, then, gets 17 points).

If a bowler knocks down all 10 pins with the first ball of the frame, a "strike" is recorded - the score for that frame is 10 + the number of pins knocked down in the next 2 rolls (for example, the bowler who rolls three strikes in a row in the first three frames gets credit for 30 points in the first frame).

If a bowler gets a strike in the last frame, the score for that frame can't be recorded before rolling twice more. Similarly, if a bowler rolls a spare in the last frame, one more roll is required before the final score can be tallied.

This problem requires you to implement a program that keeps track of a mini-bowling tournament that has just three frames. The perfect score for mini-bowling is 90 (5 strikes in a row). If a bowler rolls a strike, your program should be smart enough to ask for input on the next frame (rather than waiting for a second roll in the same frame – see the first frame of example 2).

Example 1:

```
Frame 1, ball 1: 8
Frame 1, ball 2: 2
Frame 2, ball 1: 0
Frame 2, ball 2: 9
Frame 3, ball 1: 2
Frame 3, ball 2: 7

Final score: 28
```

Example 2:

```
Frame 1, ball 1: 10
Frame 2, ball 1: 9
Frame 2, ball 2: 1
Frame 3, ball 1: 10
Extra ball 1: 7
Extra ball 2: 2

Final score: 59
```

Problem 5:
Verifying the 8-Queens

The 8-Queens problem in chess is to place 8 queens on a chess board such that none of the queens is threatening any of the others. A chess board is an 8 x 8 matrix of squares and queens may move any direction along a row, column or diagonal. The problem is to input the 8 columns of the queens on the rows of a chess board, with 1 being the first column and 8 being the last, e.g. 1 2 3 4 5 6 7 8 means the queens are along the diagonal, which would not be a valid solution.

Example 1:

```
Enter board configuration: 3 5 2 8 1 7 4 6
This is a valid configuration.
```

Example 2:

```
Enter board configuration: 1 8 2 5 3 7 4 6
This is NOT a valid configuration.
```

Problem 6:
Stack Machine Interpretation

Many virtual machines, e.g. the Java VM, are based on the notion of a run-time stack to hold data. This is like a stack of plates where data may only be removed from or added to the top. Code is of the form:

```
PUSH 1
PUSH 2
ADD
RESULT
=> 3            (output from stack)
```

PUSH 1 pushes a 1, PUSH 2 pushes a 2 and ADD removes the top 2 values on the stack, adds them and pushes the result (so the 1 and 2 would be replaced by 3). RESULT is a special operation that will display the current value at the top of the stack.

Write an interpreter for such stack machine code. In addition to PUSH which will always be followed by an unsigned integer, implement ADD, SUB, MUL, DIV, and RESULT operations. The operators may be abbreviated to A, S, M, D, and R. All operands will be integers. You may assume integer division when implementing DIV.

Example 1:

```
Enter virtual machine commands:
P 1
P 2
A
P 3
M
P 1
S
P 2
D
R
=> 4
```

Example 2:

```
Enter virtual machine commands:
P 1
P 2
P 3
P 4
P 5
M
A
P 6
S
A
A
P 10
D
R
=> 2
```