Problems for Op 2019

By Dr. Robert W. Hasker
Friday 22 November 2019

Notes: Unless a problem specifies otherwise, you may assume that all user inputs are valid. Also, the precise wording of your output is not critical as long as it is completely clear.

1. Slippery pairs (10 points)

Prompt the user for two letters and a line of text. Count the number of times the two letters appear in the text where the letters can be in either order but must appear next to each other. Output this count. Assume all letters are in lower case. For example, if the two letters are 'i' and 'e' and the input is

    their fields are in the hidden foreign forest

Then the result is 3, counting the 'ei' in 'their' and 'foreign' and the 'ie' in 'field'.

2. Shifting people (10 points)

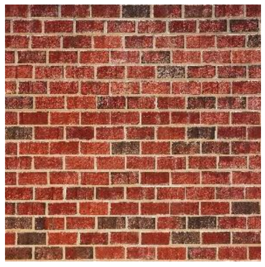
It is well understood that vehicles like busses are more efficient than cars at moving people. Write a program to see how much of a difference there is. The program will read four values in the following order:

1. Number of passengers carried by vehicle 1
2. Miles per gallon for vehicle 1
3. Number of passengers carried by vehicle 2
4. Miles per gallon for vehicle 2

The program will then print the more efficient vehicle, "vehicle 1" or "vehicle 2", followed by the average number of gallons that vehicle would consume per passenger when travelling 100 miles, assuming that the vehicle is always fully loaded. If the two results are the same, either vehicle number can be printed. Display the number of gallons as a floating-point value with at least six digits of precision.

3. Bricks on a wall (10 points)

The standard way of laying bricks for a wall is to stagger them as shown in the following picture:



The Half Brick Works company produces bricks in two sizes, full and half, where full bricks are 8 inches in length (after accounting for the mortar between the bricks) and half bricks are 4 inches in length. When the mortar has been added, all bricks are 2¼ inches high. You are to compute the number of bricks needed to build a wall of a specific height and length.

Read the length of the wall in feet. Fractions of a foot are allowed. Round the width up to the next full four inches. For example, if the length is specified as 12.2 feet then you would build a wall that is 12 feet 4 inches in length.

Read the height of the wall in feet as well. Again, fractions of a foot are allowed. However, in this case you round down. That is, if an additional row of bricks would make the wall too tall, assume that row of bricks would be replaced by wood or another material. Thus, the number of rows will be determined by the number of full, 2¼"-high bricks that can fit within the space.

Assume the left-most brick on the first row is a full brick; this means that if all bricks in the bottom-most row are full, the second row will have half bricks on each end. If the bottom-most row has one half-brick, then the second row will start with a half brick and end with a full brick.

The program is to read the height and width and display the number of full bricks followed by the number of half bricks required to build the wall. Label each output as either full or half bricks; the exact text is up to the team.

4. Peak to peak (20 points)

You've been hired to capture video of travelling between mountain peaks using a helicopter. The helicopter has limited range, so your goal is to visit the peaks in a reasonably short pattern. Prompt the user for the name of a file and read the coordinates from this file. Each peak is on its own line and is written as three values: the distance east, the distance north, and the height. There are no commas or other punctuation between the values. The distances east and north are in kilometers, the height is in meters. All three values are floating-point values, and distances are no more than 100 kilometers and heights are no more than 10,000 meters. The helicopter always starts at 0 east and 0 north at a height of 0 meters. All distances take into account both the change in east and north coordinates as well as the change in elevation. Each flight from one peak to another is done as a straight line. Read the entire file; you are guaranteed that every line will have three numbers on it.

The path is determined by finding the closest unvisited peak at each point. Start by finding the peak closest to 0 east, 0 north, 0 meters and travel to it. Next, find the peak which is closest to the first peak and travel to it. Continue on, always finding the closest peak without ever visiting any peak twice. Finish by returning to the starting point. The program is to output the total distance traveled. Peaks can be very close together, but no two peaks have exactly the same coordinates. All measures are positive. Display the total distance traveled in kilometers with four digits after the decimal. Trailing zeros do not have to be printed.

5. Lucky 13 (20 points)

Write a program that reads a file containing a 5 by 5 grid of integers and displays all triples (combinations of 3 integers) that sum to 13. The program will start by prompting the user for a file name and then reading the 16 integers from that file. It will then search for triples that add up to 13, where each triple must be three numbers in a row horizontally or three numbers in a row vertically. The program is to display the triples in any order, but each individual triple is to be sorted from low to high. Separate the numbers on the output by commas and spaces. Note that numbers in the grid may be both negative and positive. If the program reads a file containing

```
8   9   1   2   7
4   10  7   6   1
1   5   4   8   9
0   -3  12  4   16
5   20  9   18  99
```

then it would print the contents of *one* of the following boxes:

```
-3, 4, 12

1, 4, 8
```

or

```
1, 4, 8

-3, 4, 12
```

You may assume all integers are in the range -32,768 to 32,767.

6. Poet's cipher (20 points)

You and a friend have agreed to use poems to send short messages. An encrypted message consists of a string of digits. The digits are grouped into quadruples (groups of four), where the first two give a line number, the third gives a word number, and the last gives the letter within that word. The first letter on the first word on the first line is '0111'. The reader is left to figure out where the breaks are between individual words. Your program will read a poem as lines of text followed by the word END (in all caps) followed by a string of digits representing the message. Your program is to print the message based on the poem (with no spaces between the letters). For example, given the E.E. Cummings poem

maggie and milly and molly and may
went down to the beach (to play one day)

and maggie discovered a shell that sang
so sweetly she couldn't remember her troubles, and

milly befriended a stranded star
whose rays five languid fingers were;

and molly was chased by a horrible thing
which raced sideways while blowing bubbles: and

may came home with a smooth round stone
as small as a world and as large as alone.

For whatever we lose (like a you or a me)
it's always ourselves we find in the sea

Then the code 0113 0253 0321 0423 0613 0712 0824 1062 1112 1223 translates to 'gameonenow'. What "game one now" indicates is a bit of a mystery, but not one that your program must solve; the goal of the program is to simply translate the code back to text. See prob6_maggie.txt for a sample input file.

Write a program that prompts the user for a file, reads poem lines from the file until it encounters END, then reads groups of digits until it encounters the end of the file or characters that are not digits. It then decodes the message and writes it out. Be sure to label the resulting message with something like '`message: `'. The program is to decode one message and quit. You can assume the line numbers, word numbers, and letter numbers are always valid.

7. Cryptic poets (40 points)

For this problem you will compute the opposite of *Poet's Cipher*: given a file containing a poem followed by "END" and followed by single piece of text (all the words of the message, possibly with spaces between them), identify the sequence of codes that result in that message or print "NOT POSSIBLE" if there is no acceptable code. Codes must obey the following constraints:

- All spaces are to be removed before encoding words.
- Use the lines in sequence; for example, do not jump from line 5 back to line 4. However, you *can* jump ahead multiple lines, going (say) from line 5 to 8.
- Use just one word from each line and use just one letter from each word. That is, once a letter has been found in a line, the next letter must come from the next line (or later).
- Any words past the 9th word on a line are ignored; this is so word numbers can be encoded as given in problem 6.
- Any letters past the 9th letter in a word are ignored; this is so letter numbers can be encoded as given in problem 6.
- The line number must be 99 or less.

If a poem can encode a piece of text in more than one way, pick the encoding with the smallest line numbers. That is, encoding 0234 0581 would be preferred over 0311 0423. In addition, if two encodings use the same line numbers, prefer the encoding with the smallest word numbers. This means that 0234 0581 would be preferred over 0241 0581.

The sample file prob7_maggie.txt contains the maggie and milly and molly poem followed by the text "game one now". Running your solution on this file should print

```
0113 0253 0321 0423 0613 0712 0824 0974 1052 1121
```

Note this is slightly different than the ciphertext in problem 6; that ciphertext was generated with a different algorithm.

8. Lucky 99 (40 points)

Write a version of Lucky 13 that reads files containing any size of input grid of integers (where both the width and height are the same) and displays any sequence of two or more numbers that sum to 99. A sequence can be adjacent vertical numbers, adjacent horizontal numbers, or adjacent diagonal numbers. Diagonals are at 45 degrees to the rows and columns, either rising or falling. For rising diagonals, move up one position each time you move right one position. For falling diagonals, move down one position for each position moved right.

Unlike Lucky 13, do *not* sort the values within the sequences that sum to 99. Horizontal sequences and diagonals are displayed left to right and vertical sequences top to bottom.

While sequences are unchanged, do sort the sequences lexicographically:

a, b, c

would come before

u, v, w, x

if any one of the following is true:

- a < u
- a = u and b < v
- a = u and b = v and c < w
- a = u, b = v, and c = w (since then the first list is shorter than the second but the start matches)

In general, one would compare until one finds corresponding items that do not match and then use the comparison of the next item to determine which comes first.

You can assume the input files contain just numbers in the range -32768 to 32676, and that every row has the same number of integers.

9. Maximum thievery (40 points)

A cat burglar has scoped out a line of houses on a quiet street and has determined the value of the jewels (in thousands of dollars) stored at each house. The burglar wishes to raid the houses on this street during the week around Christmas, but they cannot steal from two houses that are next to each other because their residents might notice and then be waiting to catch the burglar. Read the row of values and display the maximum haul in thousands of dollars. For example, if the input is

7 2 1 0 2 8 2 4

then the output will be

20

since the burglar can steal $20,000 in jewelry from the houses with $7,000, $1,000, $8,000, and $4,000 in jewels. Note that this is not just a matter of stealing from every other house; in this example, the burglar skipped from the house with $2,000 in jewels to the one with $8,000 in jewels. Simply selecting the highest value houses also does not work: the input 1 9 15 11 also results in printing 20. Some streets are very long and contain hundreds of houses; your solution must find the maximal value for those in less than a minute. The total take will be less than $2 trillion, so the computed result will fit in a 32-bit integer. No house has negative value.