Coding Test

Please read the following carefully as your solution may be rejected if you do not follow these instructions.

1. This is a timed test. You have 3 hours to complete your solutions from the time you receive this message to the time you email your response. There are no bonus points for turning your solution in early, so please check your work thoroughly if you have time remaining. If you do not finish in time, please submit what you have so far and provide an explanation in your email of what work remains to be done.

2. You must submit a solution to each of the problem statements contained in this email.

3. We will accept solutions in either Java (JDK 7 or earlier), C++ (ANSI C++ 14882-2003), C (ANSI C 99), or Python 2.7.x. You may only use standard, operating system neutral libraries and you must write your code to be portable. You do not need to use the same language for all problems.

4. Your solution will be judged primarily on the following:

- i) Completeness:
  - Does the code compile?
  - Does it run and solve the problem statement?
- ii) Correctness:
  - Does the code correctly pass the test set given?
  - Does the code correctly pass a more comprehensive test suite?
- iii) Clarity:
  - Is the logic simple and clear?
  - Does it contain helpful comments?
- iv) Craftsmanship:
  - How is the software design and algorithm?
  - How is the coding style?

5. If you have questions about the problem set, please use your best judgment and carefully document your assumptions in your code.

6. When you have your final solution to all problems, please reply to this email with your solution as a .zip, .tar, or .tar.gz attachment. Do not include any binaries, only source code, as this might trigger our spam filters. If you have problems sending your solution by email, please upload your files on Dropbox (https://db.tt/Tr3zYiLS) and send us the shared link.

Thank you for participating in the code test!

____

****
PROBLEM STATEMENT 1
****

The sentence "A quick brown fox jumps over the lazy dog" contains every single letter in the alphabet. Such sentences are called pangrams. You are to write a method getMissingLetters, which takes a String, sentence, and returns all the letters it is missing (which prevent it from being a pangram). You should ignore the case of the letters in sentence, and your return should be all lower case letters, in alphabetical order. You should also ignore all non US-ASCII characters.

The code you submit must contain a method that conforms to the expected method signature, as follows:

Java Definition
Public Class: MissingLetters
Method signature: public String getMissingLetters(String sentence)

C++ Definition
Function signature: string getMissingLetters(const string& sentence)

C Definition
Function signature: char *getMissingLetters(const char *sentence)
Here the return value is a dynamically allocated string,
which can be free'd using free(result). If the function fails
because it cannot allocate memory it should return NULL.

Examples: (Note that in the examples below, the double quotes should not be considered part of the input or output strings.)

0) "A quick brown fox jumps over the lazy dog"
Returns: ""
(This sentence contains every letter)

1) "A slow yellow fox crawls under the proactive dog"
Returns: "bjkmqz"

2) "Lions, and tigers, and bears, oh my!"
Returns: "cfjkpquvwxz"

3) ""
Returns: "abcdefghijklmnopqrstuvwxyz"

****
PROBLEM STATEMENT 2
****

A collection of particles is contained in a linear chamber. They all have the same speed, but some are headed toward the right and others are headed toward the left. These particles can pass through each other without disturbing the motion of the particles, so all the particles will leave the chamber relatively quickly.

You will be given the initial conditions by a String init containing at each position an 'L' for a leftward moving particle, an 'R' for a rightward moving particle, or a '.' for an empty location. init shows all the positions in the chamber. Initially, no location in the chamber contains two particles passing through each other.

We would like an animation of the process. At each unit of time, we want a string showing occupied locations with an 'X' and unoccupied locations with a '.'. Create a class Animation that contains a method animate that is given an int speed and a String init giving the initial conditions. The speed is the number of positions each particle moves in one time unit.

The method will return an array of strings in which each successive element shows the occupied locations at the next time unit. The first element of the return should show the occupied locations at the initial instant (at time = 0) in the 'X', '.' format. The last element in the return should show the empty chamber at the first time that it becomes empty.

Java Definition
Class: Animation
Method: animate
Method signature:
```
public String[] animate(int speed, String init)
```
Your method should be public

C++ Definition
Function signature:
```
vector<string> animate(int speed, const string& init)
```

C Definition
Function signature:
```
char** animate(size_t* steps, int speed, const char* init)
```
If you choose to implement the C definition, you must allocate memory in the function animate and return the result as an array of char *. You should also place the length of the array in "steps". The allocation should be done in a way that the caller can:
```
  for (size_t i = 0; i < steps; i)
      free(result[i]);
  free(result);
```
to free the storage you allocated, without memory leaks.
The function should return NULL if it fails to allocate memory.

You may assume the following constraints:
- speed will be between 1 and 10 inclusive
- init will contain between 1 and 50 characters inclusive
- each character in init will be '.' or 'L' or 'R'

Examples (Note that in the examples below, the double quotes should not be considered part of the input or output strings.)

0) 2, "..R...."
Returns:
```
{ "..X....",
  "....X..",
  "......X",
  "......." }
```

The single particle starts at the 3rd position, moves to the 5th, then 7th, and then out of the chamber.

1) 3, "RR..LRL"
Returns:
```
{ "XX..XXX",
  ".X.XX..",
  "X.....X",
  "......." }
```

At time 1, there are actually 4 particles in the chamber, but two are passing through each other at the 4th position

2) 2, "LRLR.LRLR"
Returns:
```
{ "XXXX.XXXX",
  "X..X.X..X",
  ".X.X.X.X.",
  ".X.....X.",
  "........." }
```

At time 0 there are 8 particles. At time 1, there are still 6 particles, but only 4 positions are occupied since particles are passing through each other.

3) 10, "RLRLRLRLRL"
Returns:
```
{ "XXXXXXXXXX",
  ".........." }
```

These particles are moving so fast that they all exit the chamber by time 1.

4) 1, "..."

Returns:

```
{ "..." }
```

5) 1, "LRRL.LR.LRR.R.LRRL."

Returns:

```
{ "XXXX.XX.XXX.X.XXXX.",
  "..XXX..X..XX.X..XX.",
  ".X.XX.X.X..XX.XX.XX",
  "X.X.XX...X.XXXXX..X",
  ".X..XXX...X..XX.X..",
  "X..X..XX.X.XX.XX.X.",
  "..X....XX..XX..XX.X",
  ".X.....XXXX..X..XX.",
  "X.....X..XX...X..XX",
  ".....X..X.XX...X..X",
  "....X..X...XX...X..",
  "...X..X.....XX...X.",
  "..X..X.......XX...X",
  ".X..X.........XX...",
  "X..X...........XX..",
  "..X.............XX.",
  ".X...............XX",
  "X.................X",
  "..................." }
```