School of Computing & Information Technology

CSCI251/CSCI851 Advanced Programming Autumn 2019

Assignment 1 (Worth 10%)

Due 11:55pm Friday 5^{th} April 2019. (End of Week Five)

Overview

This assignment is to be implemented using procedural programming. The overall program should process animals through a veterinary (Vet) clinic. This is not supposed to be a sensible simulation of such a clinic, and does not comply with typical operating practices for such centres.

General code notes

These are some general rules about what you should and shouldn't do.

1. Your assignment should be organised into:
   (a) A driver file containing your `main()` function.
   (b) A header file containing the prototypes for the functions you write.
   (c) An implementation file containing the implementations of your functions.
2. Provide a text file `Readme.txt` with instructions for compiling your code on Banshee into the executable `VET`.
3. **If your code doesn't compile on Banshee you will likely receive 0 for the assignment.**
4. You are not allowed to use classes.
5. You can use structs, but without member functions.
6. Within your code, be consistent in your tabbing style. We want readable code.
7. Include sensible volumes of commenting.
8. Use appropriate variable names.
9. Don't leak memory.
10. Your `main()` function should make it clear what is going on, and shouldn't be too large.

Run structure

Once your program is compiled into the executable `VET`, it must run as follows:

```
$ ./VET limit Animals.txt Vets.txt Problems.txt Treatments.txt Output-file
```

The Vet clinic runs on a strict quota system and closes after handling the number of animals specifed by `limit`. It also closes when there are no animals left to see. The animals are ordered in the `Animals.txt` file but otherwise there is no sense of time in this program, so it's possible every animal could be dealt with by the same vet.

Errors should be reported to standard error. Runtime progress should be reported to standard out. A summary of what happens to each animal should be passed to whatever file is specified on the command line as the `Output-file`.

The 4 txt files listed on the command line are text data files. When you load the data files, you should output the content in a sensible format to make it clear that your input processing works. The data file names shouldn't be hard coded.

If errors are detected in reading the line of a file you should report the problem and ignore the line. This means, for example, that the reference to the fourth entry/line of a file will actually be tied to the fifth line if exactly one of the earlier lines has a problem.

- You should deal with the animals on a one by one basis, in the order specified in the input file.
- Each animal will need to be checked out by a randomly chosen vet.
- The vet will attempt to diagnose the correct problem, with the percentage chance of doing so being some sensible function of the vet's quality and the problem determination complexity. You should specify your detection function in your `Readme.txt` file.
- If the vet fails to identify the correct problem, the vet will guess at the problem, and is equally likely to specify any of the possible problems including the correct one.
- Once an animal is determined to have a particular problem, the vet will apply the treatment associated with that problem.
- The chance of the treatment working should be some sensible function of the vet's quality and the problem treatment complexity. If it's the wrong treament, the chance of it working should be 25% of the chance when using the correct treament. You should specify your success function in your `Readme.txt` file.
- Once given the treatment, the animal leaves, whether the treatment is successful or not.

Sensible with respect to the functions above means that increasing or decreasing the arguments should change the chance of success in a common sense way, so either a higher quality vet, lower problem determination complexity, or a lower problem treament complexity, give better chances of success. Here goes an example giving numbers indicative of the trend required, you do not need to have these specific numbers. You can have percentages greater than 100, for example a problem with very low determination complexity might always be identified by a decent vet.

You don't have to take a 50, 50, 50 as your baseline or have the two parameters equally weighted; so 75,75 doesn't have to give a 50% chance.

Quality	Problem determination complexity	Chance of successful diagnosis	Comment
50	50	50	This is the baseline.
75	50	60	Chance increased.
50	25	60	Chance increased.
50	75	40	Chance decreased.
25	50	40	Chance decreased.
75	75	50	Balanced again.
25	25	50	Balanced again.

Inputs

Four data files will be provided. The general syntax of those files is described here, and examples are provided on Banshee in `/share/cs-pub/251/Assignments/One/`. Those files are in Unix format so if you copy them over to Windows and back to Unix there may be additional characters, particularly the end of line ^M that may appear.

The four data files are as follows, with the comman and full stops used to seperate fields and rows.

1. `Animals.txt`: No more than 20 entries.

   ```
   Name:Type:Registration:Problem.
   ```

   Example:

   ```
   Alice:Anaconda:10323:1.
   Boris:Bull:23456:2.
   Calvin:Cat:01320:3.
   Dusty:Dinosaur:00001:5.
   Eddie:Eagle:57429:4.
   ```

   The Problem value corresponds to the Problems listed in the file `Problems.txt`. Registration is a 5 digit animal registration string, note that it can have leading 0's.

2. `Vets.txt`: No more than 5 entries.

   ```
   Name:Quality.

   Boris Bluenose:77.
   Geraldine Gardner:73.
   Alice Average:50.
   Frankie Fisher:75.
   Marlene Marvellous:90.
   Harold Hopeless:10.
   ```

   The name cannot be empty. The quality is a percentage in the range 1 to 100.

3. `Problems.txt`: No more than 20 entries.

```
Name:Problem determination complexity:Problem treatment complexity:Treatment.
```

Problem determination complexity and problem treatment complexity are percentages in the range 1 to 100. The lower the problem determination complexity, the easier it is to recognise the problem. The higher the problem treatment complexity, the harder it is to apply the treatment successfully.

The Treatments corresponds to the Treatments listed in the file `Treatments.txt`.

```
Hungry:70:15:1.
Broken limb:30:50:2.
Fleas:10:10:3.
Organ failure:50:75:4.
Extinct:1:100:5.
```

4. `Treatments.txt`: No more than 20 entries.

```
Name.
```

```
Feed the animal.
Set limb and paster cast.
Flea powder.
Organ transplant.
Genetic reconstruction.
```

If a given data file has an incorrectly formatted line, you should report the problem and ignore that line.

Output

Errors should be reported to standard error. Runtime progress, included the file loading output, should be reported to standard out.

A summary of what happens to each animal should be passed to whatever file is specified on the command line as the `Output-file`. This should include the vet seen, the actual problem, the problem diagnosed, the treatment applied, and whether the treatment was successful.

Notes on submission

Submission is via Moodle.

Your code must compile on Banshee with the instructions you provide. If it doesn't you will likely be given zero for the programming part of this assignment.

Please submit your source, so .cpp and .h files, and your Readme.txt file directly to Moodle. There shouldn't be other files or directories and they shouldn't be in a zip file.

The `Readme.txt` file should contain your compilation instruction and the functions mentioned earlier.

1. The deadline is 11:55pm Friday 5^{th} April 2019. (End of Week Five).
2. Late submissions will be marked with a 25% deduction for each day, including days over the weekend.
3. Submissions more than three days late will not be marked, unless an extension has been granted.
4. If you need an extension apply through SOLS, if possible **before** the assignment deadline.
5. Plagiarism is treated seriously. Students involved will likely receive zero.