COMP1038 Coursework - Airstrike Planner

Introduction

This is the COMP1038 Coursework. It is worth **45% of the module mark**. The deadline for this exercise is **16:00 on Wednesday 25th of December 2019**.

Read the entire document before beginning the exercise.

If you have any questions about this exercise, please ask in the Q&A forum on Moodle, after a lecture, in a lab, or during the advertised office hours. Do not post your program or parts of your program to Moodle as you are not allowed to share your coursework programs with other students. If any questions requires this exercise to be clarified then this document will be updated and everyone will be notified via Moodle.

Version History

- Version 1.0 - 2019-11-16 - Original version.

Submission

You must submit a single C source code file containing all your code for this exercise. This file must be called `airstrikeplanner.c` and must not require any other files outside of the standard C headers which are always available. The first line of the file should be a comment which contains your student ID number, username, and full name, of the form:

```
// 6512345 zy12345 Joe Blogs
```

The file must compile without warnings or errors when I use the command

```
gcc -std=c99 -lm -Wall airstrikeplanner.c -o airstrikeplanner
```

This command will be run on our Linux server `cslinux`. If it does not compile, for any reason, then you will lose all the marks for testing (common reasons in the past have been submitting a file with the wrong filename, or developing your solution on your personal computer without having tested it on our Linux server). If the file compiles but has warnings then you will lose some marks for not correcting the warnings.

The completed source code file should be uploaded to the Coursework Submission link on the COMP1038 Moodle page. You may submit as many times as you wish before the deadline (the last submission before the deadline will be used). After the deadline has passed, if you have already submitted your exercise then you will not be able to submit again. If you have not already submitted then you will be allowed to submit **once**.

Late submissions: Late submissions will lose 2 percentage points **per hour**, rounded up to the next whole hour. This is to better represent the large benefit a small amount of extra time can give at the end of a programming exercise. No late submissions will be accepted more than 50 hours after the exercise deadline. If you have extenuating circumstances you should file them before the deadline.

Plagiarism

You should complete this coursework on your own. Anyone suspected of plagiarism will be investigated and punished in accordance with the university policy on plagiarism (see your student handbook and the University Quality Manual). This may include a mark of zero for this coursework.

You should write the source code required for this assignment yourself. If you use code from other sources (books, web pages, etc), you should use comments to acknowledge this (and marks will be heavily adjusted down accordingly). *The only exception to this is the dynamic data-structures (linked lists and others) developed during the lectures and tutorials; you may use these, with or without modification, without penalty as long as you add a comment in your program saying you have taken them from the lectures or tutorials and saying how you have modified it (or not modified it). If you do not acknowledge their source in a comment then it will be regarded as potential plagiarism.*

You must not copy or share source code with other students. You must not work together on your solution. You can informally talk about higher-level ideas but not to a level of detail that would allow you all to create the same source code.

Remember, it is quite easy for experienced lecturers to spot plagiarism in source code. We also have automated tools that can help us identify shared code, even with modifications designed to hide copying. If you are having problems you should ask questions rather then plagiarize. If you are not able to complete the exercise then you should still submit your incomplete program as that will still get you some of the marks for the parts you have done (but make sure your incomplete solution compiles and partially runs!).

If I have concerns about a submission, I may ask you to come to my office and explain your work in your own words.

Marking

The marking scheme will be as follows:

- *Tests (60%):* Your program should correctly implement the task requirements. A number of tests will be run against your program with different input data designed to test if this is the case for each individual requirement. The tests themselves are secret but general examples of the tests might be:
  - Does the program work with the example I/O in the question?
  - Does the program work with typical valid input?
  - Does the program correctly deal with input around boundary values?
  - Does the program correctly deal with invalid input (both invalid files and values)?
  - Does the program handle errors with resources not being available (eg, malloc failing or a filename being wrong)?
  - Does the program output match the required format?
  - Does the program output an appropriate 2D map when required?

  As noted in the submission section, **if your program does not compile then you will lose all testing marks**.
- *Appropriate use of language features (30%):* Your program should use the appropriate C language features in your solution. You can use any language features or techniques that you have seen in the course, or you have learned on your own, as long as they are appropriate to your solution. Examples of this might be:

    - If you have many similar values, are you using arrays (or equivalent) instead of many individual variables?
    - Have you broken your program down into separate functions?
    - Are all your function arguments being used?
    - If your functions return values, are they being used?
    - If you have complex data, are you using structures?
    - Are you using loops to avoid repeating many lines of code?
    - Are your if/switch statements making a difference, or is the conditions always true or false making the statement pointless?
- *Source code formatting (10%):* Your program should be correctly formatted and easy to understand by a competent C programmer. This includes, but is not limited to, indentation, bracketing, variable/function naming, and use of comments.

Task

A nuclear war is unavoidable between two superpowers on the Planet E. As a Major responsible for nuclear airstrike, your task is to write an interactive, menu-driven program that will act as an airstrike planner. The targets of airstrikes will be given in files that your program will need to load. This program should be able to show if one or some targets are within the damage zone of a nuclear missile.

When the program runs, it should display the following main menu to the user and prompt them to enter a menu option:

```
1) Load a target file
2) Show current targets
3) Search a target
4) Plan an airstrike
5) Execute an airstrike
6) Quit
Option:
```

If the user enters the number of an option, the program should perform that option then return to the main menu to let the user select another option. If the user enters something which is not a valid option then the program should print `"Unknown option."` then print `"Option: "` again for the user to select another option. The user may enter any input at this, or any other prompt, in the program, terminated by pressing the return key (newline character). Your program must deal with this appropriately, accepting valid input and rejecting invalid input according to the particular prompt.

If the user selects the "Load a target file" option, then the program should prompt `"Enter a target file: "` and wait for a file name entered by the user. When a file name is entered, your program should open the file and read in all target data if it is valid or prompt `"Invalid file."` if it is not.

A target is in the format of "name value value", where name is a string that contains English chars and digits only, and two values denote the latitude and longitude of a location. The latitude and longitude values must be within the range of `[0, 100]`. There are one or more space `' '` in between name and values. The length of a name must be no more than 15 chars. The length of a value must be no more than 15 digits including the decimal point `'.'`

Every valid target file contains one or more targets in a line, with one or more space `' '` in between different targets. Some examples of valid target files are,

```
"a01 30.5678 15.3421 a02 45 17.5673"
"539  30.5678 15.3421    53a 0 017.5673  "
"0 30.5678 15.3421   1 5 17.5673 "
```

Examples of invalid target files are,

```
"a01 30.5678 15.3421 a02 45 17.5673 a03"
"a01 30.5678 a02 45.4532 17.5673 "
"A0 30.5678 5 13.4578 17.5673 34.6754"
"AC12345678902019X 30.5678 13.4578 "
"A1 101.5678 13.4578 "
"A1 -1.5678 13.4578 "
"A1 0.5678 13.4578333333333 "
```

The targets should be stored in a data-structure based on dynamic memory allocation(for example a linked list). Every time your program loads a valid target file, the targets should be added into the same data-structure. At the same time, your program should check whether a new target is conflict with those exist targets. Two targets are considered as conflict if the distance between them is smaller than **0.1**. The distance here means the length of straight line segment between the two points in X-Y coodinates. When a new target is conflict with any exist targets, it should be discarded. So your data-structure should **not** contain any conflict targets.

The program should then return to the main menu.

If the user selects the "Show current targets" option, then the program should print two things if the data-structure contains targets:

1. Text based information about the targets and
2. A 2D map of planet E with all the targets .

For text based information, information about all targets should be printed, one target per one line, with a space between name, latitude, and longitude. For the 2D map, you are to use your creativity to design and display the map by using chars within a 80 x 40 (width x height) area. The location of all targets should be displayed on the map.

If the data-structure is empty, the program should do nothing and return to the main menu.

If the user selects the "Search a target" option, then the program should prompt `"Enter the name: "` and allow the user to enter the name of a target. If a target with the entered name exists, the program should print the target. Otherwise, it should print `"Entry does not exist."` If the user presses return without entering any other characters, then this option should do nothing else and return to the main menu.

If the user selects the "Plan an airstrike" option, then the program should allow the user to enter the predicted collision point and damage zone of a nuclear missile. The damage zone is a round area centered on the collision point. And then the program should print all the targets that locate inside the damage zone.

An example I/O:

```
1) Load a target file
2) Show current targets
3) Search a target
4) Plan an airstrike
```

```
5) Execute an airstrike
6) Quit
Option: 1
Enter a target file: target1.txt
Option: 2
a05 17.234500 20.567800
a06 15.340000 28.342000

[2D map of planet E goes here]

Option: 4
Enter predicted latitude: 15.34
Enter predicted longitude:  24.6743
Enter ratio of damage zone: 5
a05 17.234500 20.567800
a06 15.340000 28.342000
Option:
```

If the user selects the "Execute an airstrike" option, then the program should allow the user to enter the targeted collision point and damage zone of a nuclear missile. If there is not any target within the damage zone, the program should print `"No target aimed. Mission cancelled."` Otherwise, the program should print `"X target destroyed."` where `X` should be the number of targets that locate inside the damage zone. The program should print all the targets that locate inside the damage zone. After that, those targets should be removed from the data-structure.

An example I/O:

```
1) Load a target file
2) Show current targets
3) Search a target
4) Plan an airstrike
5) Execute an airstrike
6) Quit
Option: 1
Enter a target file: target1.txt
Option: 2
a05 17.234500 20.567800
a06 15.340000 28.342000

[2D map of planet E goes here]

Option: 5
Enter targeted latitude: 15.34
Enter targeted longitude:  24.6743
Enter ratio of damage zone: 5.0
2 target destroyed.
a05 17.234500 20.567800
a06 15.340000 28.342000
Option: 2
Option:
```

If the user selects the "Quit" option, the program should exit. Before exiting, the program should explicitly free any memory it has allocated that it hasn't yet freed. (Note that most operating systems will recover all unfreed memory from a program when it exits but this exercise requires you to free allocated memory yourself in order to demonstrate you can do it).

If the program needs to exit because it cannot allocate memory, it should print the error message "`Unable to allocate memory.`" and call `exit(-1)`. If the program needs to exit for any other reason, print an appropriate error message and call `exit(-2)`.

Example input/output

Given the following two target files (target1.txt and target2.txt):

```
a05 17.2345 20.5678 a06 15.34  28.342
```

```
0001  55.4853 32.6544   0002 17.2345 20.5678 a3 45.0232 023.447658
```

Example:

```
zlizpd3 $ ./airstrikeplanner
1) Load a target file
2) Show current targets
3) Search a target
4) Plan an airstrike
5) Execute an airstrike
6) Quit
Option: 2
Option: 1
Enter a target file: target1.txt
Option: 2
a05 17.234500 20.567800
a06 15.340000 28.342000

[2D map of planet E goes here]

Option: 1
Enter a target file: target2.txt
Option: 2
a05 17.234500 20.567800
a06 15.340000 28.342000
0001 55.485300 32.654400
a3 45.023200 23.447658

[2D map of planet E goes here]

Option: 6
zlizpd3 $
```

Example:

```
zlizpd3 $ ./airstrikeplanner
1) Load a target file
2) Show current targets
3) Search a target
4) Plan an airstrike
5) Execute an airstrike
6) Quit
Option: a
Unknown option.
Option: 1
Enter a target file: target1
Invalid file.
Option: 1
Enter a target file: target1.txt
Option: 3
```

```
Enter the name:
Option: 3
Enter the name: a06
a06 15.340000 28.342000
Option: 3
Enter the name: a03
Entry does not exist.
Option: 5
Enter targeted latitude: 15.34
Enter targeted longitude:  24.6743
Enter ratio of damage zone: 1
No target aimed. Mission cancelled.
Option: 5
Enter targeted latitude: 15.34
Enter targeted longitude:  28
Enter ratio of damage zone: 1
1 target destroyed.
a06 15.340000 28.342000
Option: 6
zlizpd3 $
```

Hints

- If you are given a file name, that file might not exist or you might not have permission to read it!
- Remember to free any memory which you no longer need. Your program should not have any memory leaks (dynamically allocated areas of memory which are no longer reachable). You will need to consider how the responsibility for allocated data transfers as your program runs.
- The internal representation of your data does not have to be in the same format as the input files (ie, text). It should be whatever format is easiest for your program to do what it needs to do with the data. For example, you can add into your data-structure whatever data types that you think is necessary to complete the task.
- Your program should be able to handle some, if not all, invalid input entered by users.

END