AE1PGA Coursework 4 - Drone Checker

Introduction

This is the fourth AE1PGA Coursework. It is worth **35% of the module mark**. It requires you to write a program which will monitor the movements of drones and detect if they are entering no-fly zones. The deadline for this exercise is **16:00 on Thursday 21st of December 2017**.

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 - 2017-12-07 - Original version.

Submission

You must submit a single C source code file containing all your code for this exercise. This file must be called `dronechecker.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 -Wformat -Wwrite-strings dronechecker.c -o dronechecker
```

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 4 Submission link on the AE1PGA 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**.

Remember that you can only access the Linux server from the designated labs in SEB and SSB and that these labs are also heavily booked for teaching. **Do not** wait until the last moment to submit as you may find you cannot get into any of the rooms to access your source code files. Not being able to access the machine due to lack of planning is **not** an extenuating circumstance. If you choose to use the VDI then the same rules apply. Equipment occasionally breaks down, is full, inaccessible or unreliable. You need to plan ahead to allow time for foreseeable things outside of your control going wrong.

Late submissions: AE1PGA late submission policy is **different** from the standard university policy. Late

submissions will lose 5 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 24 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 prompt function or the dynamic data-structures developed during the lectures; you may use these, with or without modification, without penalty as long as you add a comment saying you have taken them from the lectures 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 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?

  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. See the lecture notes, and the example programs for examples of correctly formatting programs.

Late Submissions: see submission section above.

Task

Your task is to write a program that will check that the flight plan of a drone does not enter any no-fly zones. A representation of the no-fly zones will be stored in a file that your program will need to load. The flight plan of the drone will also be stored in a different file you will need to load. Both of these filenames will be specified as command line parameters (first the no-fly zone filename, then the flight plan filename). Your program should then check the flight plan to make sure the drone does not enter any no-fly zones during it's flight.

If the program is run without the correct number of command line parameters, it should exit with the error message "Invalid command line arguments. Usage: <noflyzones> <flightplan>". Both filenames may be relative or absolute pathnames which can be understood by `fopen`. If either file cannot be opened, use `perror` to print the error message "Cannot open X file." together with the operating system specific error message and exit with exit code 1, where X is either "no-fly zone" or "flight plan" as appropriate. Check the no-fly zone file is accessible and valid before checking the flight plan file is accessible and valid.

The format of the no-fly zone file is as follows.

- Lines are separated by a single newline character '\n'.
- Lines that start with the character '#' are comments and their contents should be ignored until the end of the line.
- Blank lines (lines with no characters) should be ignored.
- Each line that describes a circular no-fly zone will contain 3 integers separated by one or more whitespace characters. Each integer will be greater than or equal to zero and less than 10,000.
    - The first number is the x coordinate of the center of the no-fly zone.
    - The second number is the y coordinate of the center of the no-fly zone.
    - The third number is the radius of the no-fly zone.
- Non-blank, non-comment lines that do not match that format are invalid. In this case, the program should print the error message "No-fly zone file invalid." and exit with exit code 2.
- There is no limit on the number of no-fly zones in the file. There may be zero. The radius of each no-fly zone will always be greater than zero. You can assume no no-fly zones overlap.

The format of the flight plan file is as follows.

- Lines are separated by a single newline character '\n'.
- Lines that start with the character '#' are comments and their contents should be ignored until the end of the line.

- Blank lines (line with no characters) should be ignored.
- Each line that describes a waypoint will contain 2 integers separated by one or more whitespace characters. Each integer will be greater than or equal to zero and less than 10,000.
    - The first number is the x coordinate of the center of the way point.
    - The second number is the y coordinate of the center of the way point.
- Non-blank, non-comment lines that do not match that format are invalid. In this case, the program should print the error message "Flight plan file invalid." and exit with exit code 3.
- There is no limit on the number of way-points in the file. There will always be at least 2. No two consecutive way-points will have the same coordinates.

The drone will move in a straight line between each of the way-points given in the flight plan file. The flight plan is only valid if the drone never enters any of the restricted areas. To determine if the drone has entered a restricted area, you need to check if any of the line segments between the waypoints intersect any of the restricted areas. For a given line segment,

- if one or both of the way-points are inside a restricted area then the flight plan is invalid.
- if neither way-point is inside a restricted area but the line segment intersects (crosses through) a restricted area then the flight plan is invalid.

To check if a given line segment intersects a circle, you can use the following equations. Given a line segment with start point coordinates (A_x, A_y) and end point coordinates (B_x, B_y), and a circle of radius r with it's center at point (C_x, C_y), the line segment intersects the circle if the quadratic equation $Xt^2 + Yt + Z = 0$ has 1 or more real roots between 0 and 1 (inclusive) when solving for t where,

$$X = A_x^2 - (2 * A_x * B_x) + B_x^2 + A_y^2 - (2 * A_y * B_y) + B_y^2$$

$$Y = (2 * C_x * B_x) - (2 * C_x * A_x) - (2 * C_y * A_y) + (2 * C_y * B_y) + (2 * A_x * B_x) + (2 * A_y * B_y) - (2 * B_x^2) - (2 * B_y^2)$$

$$Z = C_x^2 - (2 * C_x * B_x) + C_y^2 - (2 * C_y * B_y) - r^2 + B_x^2 + B_y^2$$

In order to find out the roots, use the normal quadratic formula, substituting for the value of the formulas above,

$$Discrim = Y^2 - (4 * X * Z)$$

$$t = (-Y \pm sqrt(Discrim)) / (2 * X)$$

If *Discrim* (the discriminant) is less than zero then there are no real roots. If it is greater than or equal to zero, there may be one or more roots. If either of those roots are between 0 and 1 (inclusive), then the line segments intersects the circle. This also includes the line being tangential to the circle, which is also not a valid flight plan. Note that these formulas are not valid if the line segment is entirely within the circle. **Be careful with converting the maths into code.** In particular, make sure your equations have appropriately parenthesis and the plus or minus signs are correct, and remember about integer versus floating point division.

If the flight plan is valid, it should print "Flight plan valid." and exit with code 0. If the flight plan is not valid, it should print "Invalid flight plan." and "Enters restricted area around X, Y." where X,Y are the coordinates of the center of the first no-fly zone entered. It should then exit with code 4.

If the program needs to exit because it cannot allocate memory, it should print the error message "Unable to allocate memory." and exit with exit code 5. If the program needs to exit for any other reason not covered in this sheet, print an appropriate error message and exit with exit code 6.

Example input/output

Given the following no-fly zone file (nofly.txt):

```
# no fly zones
# x y radius
500 100 55
300 98 90
456 365 44
```

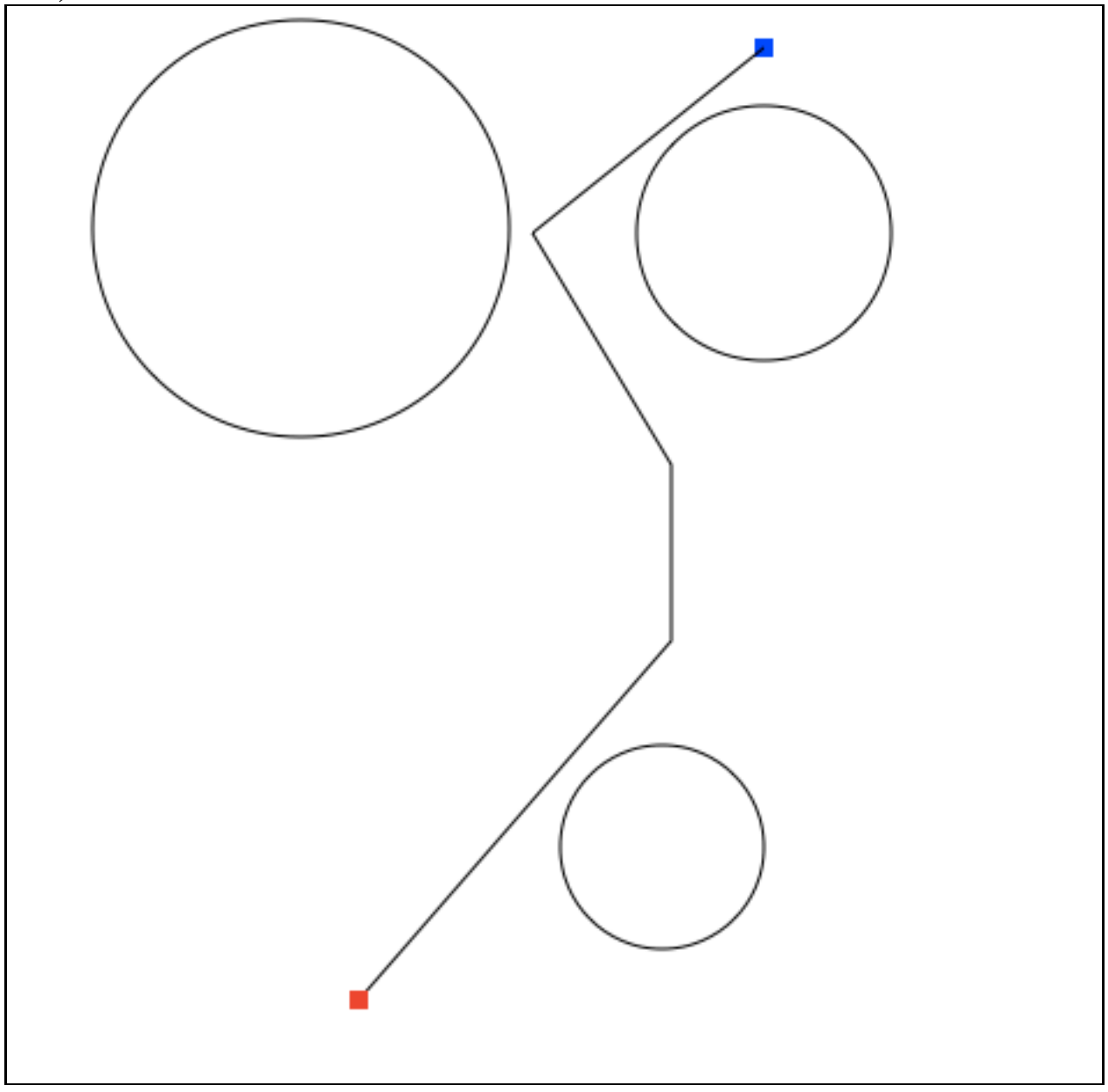
and flight plan file (flightplan.txt):

```
# flight plan
# x y
500 20
400 100
460 200
460 276
325 431
```

Example:

```
zlizpd3 $ ./dronechecker nofly.txt flightplan.txt
Flight plan valid.
zlizpd3 $
```

Visual representation of flight, (0,0) off the top-left corner (for illustration; your program should not draw this):



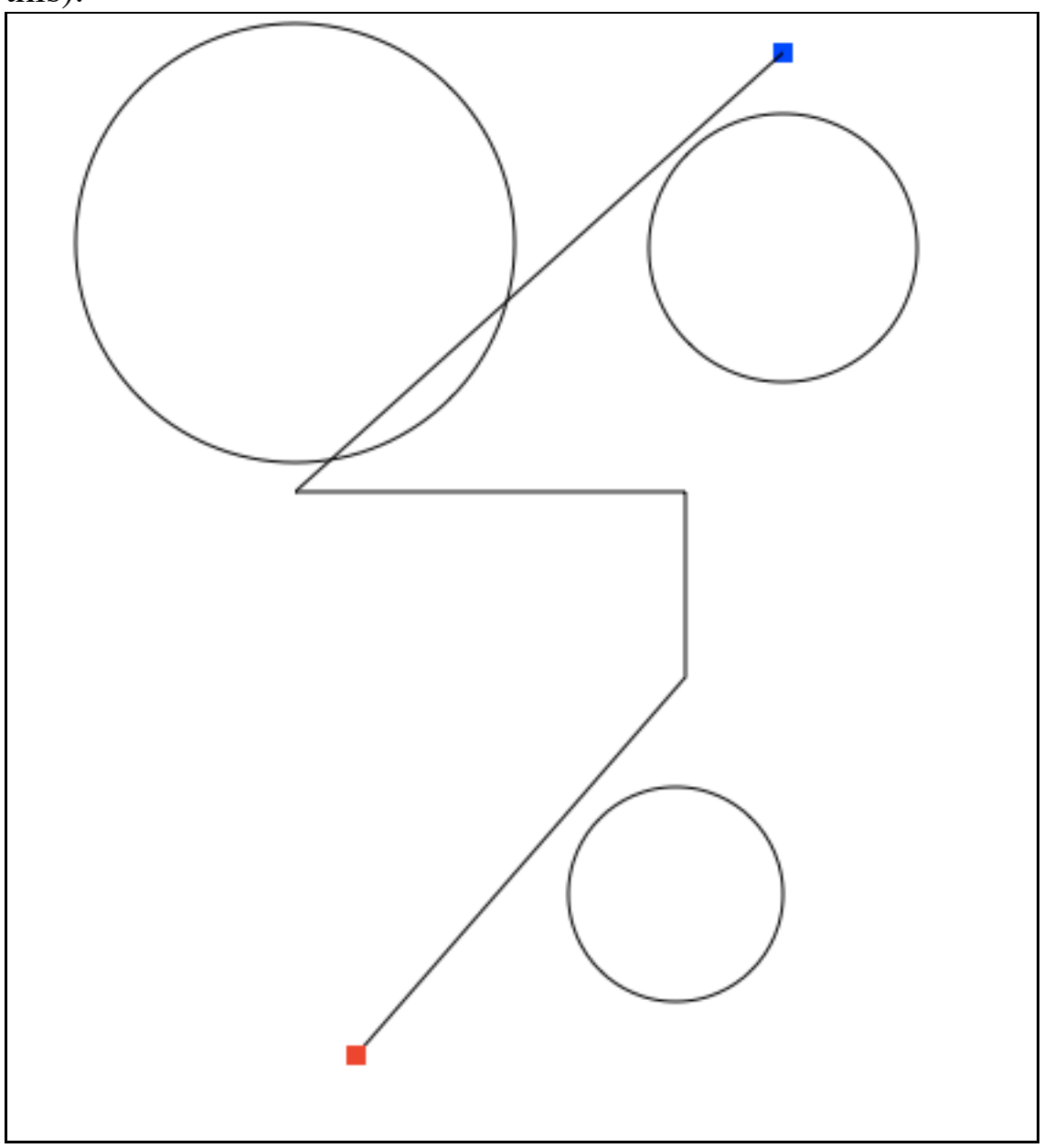
Given the previous no-fly zone file and a new flight plan file (flightplan2.txt):

```
# flight plan
# x y
500 20
300 200
460 200
460 276
325 431
```

Example:

```
zlizpd3 $ ./dronechecker nofly.txt flightplan2.txt
Invalid flight plan.
Enters restricted area around 300,98.
zlizpd3 $
```

Visual representation of flight, (0,0) off the top-left corner (for illustration; your program should not draw this):



More examples of valid and invalid flight plans with different no-fly zones will be made available 1 week after the exercise has been released. Do not wait until that time to start your solution. You should be able to design and implement your program, and create your own test data, from the specification above.

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.

- There is a difference between a *line* (which is infinitely long) and a *line segment* (which has a specific start and end point). Be careful with online references that use the wrong equations.
- 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.
- On Linux, you can check the exit code of your program by running `echo $?` as the next command after your program has exited.
- The exercise does not need the full 2 weeks given to complete it. You should be able to finish it, including a few problems and debugging, in roughly a single week (10 hours non-contact time per week for this module). You have been given more time than that so you can fit it in around your other courseworks and studying. Please plan ahead and do not leave it until the last moment.

END