THE UNIVERSITY OF SYDNEY

COMP2017 9017 Assignment 1

Due: 23:59 10 March 2024

This assignment is worth 5% of your final assessment

Task Description

In this assignment you will write and test software to support a simple geometry processing pipeline that is to be used as part of a stream processing framework. Facial recognition is a more sophisticated form of the same problem, however, more complex and dynamic.

You will use your prerequisite knowledge of procedural programming and your newfound knowledge of C to write two separate but related programs.

`Generator`: Write a Java or Python program to generate a set of 2D points satisfying a particular neighbourhood constraint. This is known as the `Generator` program.

`Searcher`: Write a C program that calculates the three closest 2D points and outputs whether these form a triangle. This is known as the `Searcher` program.

If correctly implemented, the `Generator` produced by this assignment should be interoperable with the `Searcher` program.

We strongly recommend reading this entire document at least twice. You are encouraged to ask questions on Ed after doing so. Be sure to **search first**, and check for updates of the document. If the question has not been asked before, make sure your question post is of **"Question"** post type and is under **"Assignment" category** → **"A1"** subcategory. As with any assignment, make sure that your work is your own[1], and that you do not share your code or solutions with other students.

It is important that you continually back up your assignment files onto your own machine, flash drives, external hard drives and cloud storage providers (as private). You are encouraged to submit your assignment regularly while you are in the process of completing it.

Prerequisites

You must first complete these online lessons:
https://edstem.org/au/courses/14786/lessons/49216/slides/334701
https://edstem.org/au/courses/14786/lessons/49520/slides/334708

[1] Not GPT-3/4's, ChatGPT's or copilot's, etc.

Implementation Details

There are three components to this assignment; the Java or Python program that generates points, the C program that performs the triangle search, and the software development environment.

The `Generator` program accepts command line arguments and generates output to `stdout`. The `Searcher` receives input from `stdin` and produces output to `stdout`. The software development environment is tying it all together.

Generator: A program to generate N 2D points

Write a Java or Python program to accept three parameters `mindist`, `N`, and `rseed` from command line arguments. The program generates a random set of `N` 2D points. Each point that is created is within an X range $[-50, 50]$, and a Y range $[-50, 50]$ (100×100 area). Additionally, each point must be guaranteed to be a minimum distance of `mindist` from all points generated so far. Euclidean distance is the metric used. The set of 2D points with `mindist` distance between them are printed to `stdout` in any order. `rseed` is *optional*, it is the integer value used to initialise the pseudo random number generator, once, before the first point is generated.

The format of the output requires floating point numbers with exactly 2 decimal places:

```
<x>, <y>
<x>, <y>
<x>, <y>
...
<x>, <y>
```

For example `N`=3, `mindist`=2:

```
1.34, 15.60
-41.00, 0.20
24.22, -49.89
```

The post conditions of executing this program with `N`=3,`mindist`=2:

- no two points in this are within distance 2.00 of one another.
- each point is represented as two floating point numbers in ASCII form separate by a comma
- exactly 3 points are presented to `stdout`, where each appears on a line
- No other formatting is permitted (white space or other characters)

Where `N` is less than zero, return from the program and print nothing to `stdout`. Print `"N less than zero"` to `stderr`. The program must exit with value -1.

Where $mindist < 0$ or $mindist > 10$, return from the program and print nothing to `stdout`. Print `"mindist outside range"` to `stderr`. The program must exit with value -2.

Where $N > \frac{10000}{\pi\ mindist^2}$, return from the program and print nothing to `stdout`. Print `"point saturation"` to `stderr`. The program must exit with value -3.

For Python3 implementations: python import `sys`,`argparse` can be used for command line arguments, `random` for generating random numbers, and `math` for using sqrt method in Euclidean measures. For Python3, in your repository, The file `is_generator_python.txt` must contain `true`.

For Java implementations: Java may use `java.util.Random` for generating random numbers. `Math` for using sqrt method in Euclidean measures, Double and Integer are useful classes for command line arguments processing. For Java, in your repository, the file `is_generator_python.txt` must contain `false`.

Additional libraries are not necessary for this assignment, however, they can be used. These can only be imported from within the language supported on Ed (see `python3 --version` or `java --version`). You cannot include any extra libraries, components, frameworks or source code in the git repository.

Assumptions: assume that command line arguments can be presented in any order

```
java GenPoints -N=<number> -mindist=<mindist> -rseed=<rseed>
java GenPoints -rseed=<rseed> -mindist=<mindist> -N=<number>
```

If any necessary command line arguments are missing, or cannot be parsed correctly, print "invalid arguments" to `stderr`. The program must exit with value -4.

Searcher: Program to search for a triangle

Write a C program that accepts a set of 2D points from `stdin`. The program will calculate the three closest points using Euclidean space distance metric. The program will print to `stdout`: the number of points read, the three closest points, and also print whether it forms a triangle or not. Only the first valid 1,000 input points can be received and processed by the program. If an input line does not follow expectations of format (up to 2 decimal places) or range, it is skipped and not counted in the number of points read. The range and formatted expected is based on the `Generator` program output.

Three points form a triangle if the area of that triangle is greater than `0.001`.

In this assignment, the three closest points are considered to be those which form the smallest sum of triangle side lengths (Euclidean). Consider three points, p, q, r, they can form a triangle having side lengths a, b, c. The three closest points are those whose sum $a+b+c$ is minimised among all possible triangles from the set of input points. Where equal sized triangles are found, either are accepted as the correct response.

The format of the output is as follows (points use 2 decimal places):

```
read <N> points
<x>, <y>
<x>, <y>
<x>, <y>
This is [not] a triangle
```

Example 1

Given an input of the following points:

```
0, 0
1, 0
0, 1
2, 2
5, 5
```

The output is:

```
read 5 points
0.00, 0.00
1.00, 0.00
0.00, 1.00
This is a triangle
```

Example 2

Given an input of the following points:

```
0.00, 0
1.00, 0.0
2.00, 0.00
3.00, 3
```

The output is:

```
read 4 points
0.00, 0.00
1.00, 0.00
2.00, 0.00
This is not a triangle
```

Example 3

Given an input of the following points:

```
0.03, 0.00
```

The output is:

```
read 1 points
0.03, 0.00
This is not a triangle
```

C implementation restrictions

- No variable length arrays (VLA)
- No dynamic memory (malloc or other alloc related function calls, including subversive approaches such as strdup() or getline())
- No additional 3rd party includes or libraries
- gcc not clang (OS X)
- The C flags provided in the Makefile for compilation are to be used but not modified.

Assume that input for `Searcher` has a maximum of 2 decimal places, but it can be fewer as is shown in the above examples.

Software development environment

Git

Use git for your submission via Ed. No other form of submission is accepted. There must be at least 5 git submissions total. At least 5 git submissions *must* show a meaningful change of the code with an appropriate and descriptive comment. These will be inspected and graded accordingly. The comments can be used to describe any major change of algorithm, implementation or testing procedure, or even first attempt. Many git submissions are expected, and you are encouraged to do this in your first assignment. You need to place a note of your most 5 significant git commits in a README.md file.

Your git repository should be clean. It should not be polluted with non-source code files. It should be limited to source files (.c files, and .py/.java), testing files (in the form .in and .out or others for arguments under 'tests/'), Makefile, and README.md or README.txt plain text documents only. Multimedia files, application specific files .docx, .pdf, or other formats, or compiled binary files, are not permitted.

If you are to include your own tests, place them in a directory called `tests`. The total size of 'tests' directory should not exceed 50KB.

Not following these requirements will result in marks being deducted.

Make

Compilation and execution of programs must use the Makefile provided. You are to make changes to this Makefile as appropriate.

The following make commands will be run in your directory [2].

`make` - is the default rule that should execute the equivalent of `make build` - compiles all your program binaries
`make clean` - removes any files that are not source files. These files may be created by compilation process, or by executing the compiled programs with data in tests.
`make sample` - execute a sample program -N=20,-mindist=2,-rseed=3
`make test` - run your own tests. You can place your own scripting instructions here to self check.

gcc

`gcc` is the compiler you must use in your Makefile to compile the `Searcher` program[3]. The flags to be used with the compilation have already been provided and do not need modification.

The programs must compile on Ed using the `make` or `make build` command. Python programs must compile and not run with these commands.

[2] We advise not to use recursive make or multiple directories in this assignment
[3] no other compiler. Mac OS X gcc is also excluded

Running the Code

Your C code must make use of a Makefile. As can be seen in the template, the `make` command will be run while setting up your assignment with no arguments. It is expected that both your `Generator` and `Searcher` programs should compile only.

The make sample and make test should only compile if necessary, and instead only be used for execution.

The computer grading of your assignment will be using a base installation of Linux with gcc, make, Python3 and Java. There are no guarantees about python or Java packages importable (if needed). Check Ed workspaces to confirm.

Each program must compile. Each program must be able to execute simple test cases. The final test cases will involved both programs being linked together using pipe operator | `stdout` to `stdin` and will be run as (Java example):

```
java GenPoints -N=20 -mindist=2 -rseed=3 | ./smallest_triangle
```

Submission and Testing

You will be submitting your code via git using Ed.

If you have any questions about git usage, feel free to check the Git Lesson, the relevant manual pages, or ask on Ed.

After the assignment is released, a small number of test files will be made available. Correctness tests will be provided to ensure that your code can execute fundamental examples. These tests will not be the complete set of tests run against your code. You may write your own test cases to extend on this, (and this is strongly encouraged). You should put these tests in the 'tests' directory in your repository.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not follow the assignment description or if your code is unnecessarily or deliberately obfuscated.

Marking Details

This assignment will be subjected to manual marking and auto marking.

- Code style and quality will account for 20% of your mark. This will encompass code style, readability and your solution approach. Please follow C coding guidelines provided on Ed, PEP8 for Python, and Google Java style guides.
- Git commits and their comments will account for 20% of your mark. Commit messages have to be informative. There is no prescribed guideline as it is not intended to be in great detail. If the commit changes do not reflect the description or vice versa, or the description cannot be understood by a learned person of the English language, marks will be deducted.

- Test cases made by you will account for 20% of your mark. The testing script run by your `make test` should include a comment to indicate the type and nature of test being performed. Tests types should classify if it is testing `Generator`, `Searcher`, or both. They can also specify if it is a normal, edge or corner case. It is recommended to have at least 5 test cases for the three cases `Generator`, `Searcher`, or both. It is recommended to create/update the file `is_generator_python.txt` for `Generator` testing.
- Auto marking will account for 40% of your mark, test cases will be visible, hidden, or only assessed after the due date. The exact percentage will depend on the test cases and are split approximately in three ways.

You are strongly encouraged to comment your code and follow style guidelines for all your programs. Meaningful variable names will greatly improve the readability of your program.

Deductions

These need to be stated:

- You will receive zero marks if you provide a solution requiring restrictions listed for the C program. Up to 100% deduction of test cases for `Searcher`.
- You will receive zero marks if there is no submission.
- You will receive zero marks for automatic test cases if your program fails to compile or execute.
- Style marking is only applied for reasonable attempts. This means that each program should successfully compile and run with `make sample` without error(s).
- Marks are deducted for unclean repositories; up to 20% deduction of total mark.
- Marks are deducted for lack of meaningful commits, or comments on commits; up to 50% deduction of total mark.
- Marks are deducted for lack of randomness; up to 50% deduction of test cases `Generator`.
- Using or overusing unnecessary libraries for `Generator` program. The farther the code is from being interpret-able, the worse it is. You should not assume that numpy, itertools, regular expressions, specific classes or inheritances structures are understood by the grader; up to 20% deduction of `Generator` program. Any special or sophisticated math logic is permitted and should be clarified in code comments.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

Changes

Last updated 2024-02-29 16:46:30

2024-02-29

- typo "rseed" to "-rseed"

2024-02-28

- The scaffold makefile comment "-N=20 -mindist=5 rseed=3" should be ignored, please use the description comment "-N=20 -mindist=2 -rseed=3" for the `make sample` target.
- To aid with Generator testing, a file called `is_generator_python.txt` must be located in the same directory as the Makefile. If the Generator uses python, it should contain `true`. Otherwise if it uses Java, it should contain `false`.

2024-02-19

- Tests size increased to 50KB
- Further clarification that dynamic memory of indirect uses of alloc() family such as strdup() or getline() are not permitted.

2024-02-19

- Saturation threshold changed from $\frac{10000}{mindist}$ to $\frac{10000}{\pi\ mindist^2}$
- Missing arguments for Generator now has a stderr message and return value
- Update marking guide for git commits - informative, not overly detailed descriptions.
- Clarified closest points and edge case for equal sized sum of side lengths.