Assignment 2 - NXDOMAIN*

This assignment is worth **20%** of your overall grade for the course.

Due: Week 11, Sunday, 22 Oct 2023 at 23:59, Sydney local time.

Task Description

You are going to implement a simplified DNS infrastructure that contains a DNS recursor, some DNS servers, a launcher program that generates configurations for the DNS servers and a verifier that validates the configurations of the DNS servers.
When the DNS servers are running, any end user can ask the recursor to resolve a hostname. Then, the recursor initiates a chain of DNS queries to the DNS servers via TCP connection. Eventually, the recursor shall collect responses from the DNS servers, and resolve the hostname to a valid identifier or `NXDOMAIN` to the end user.

You need to thoroughly test your solution. The testing quality will be inspected by the examiner and a testing coverage tool, Coverage.py.

On the high level, the DNS server is capable of the following:

1. Start up according to the given command-line arguments.
2. Accept DNS queries from the recursor via TCP.
3. Process the queries and reply to the recursor via TCP.
4. Log the server activities on the standard output.
5. Parse commands sent from the user via TCP and update the server.

The recursor can:

1. Start up if the root DNS server is specified in command-line argument.
2. Read and validate a domain in the standard input.
3. Query a chain of DNS servers, and receive responses.
4. Resolve the domain and output the response in the standard output.

Note that the DNS infrastructure in the context of this assignment will be running on the same device and is incapable of running on different physical devices. This is an intentional design and therefore the complexity of communication over the Internet is omitted in the system.

The assignment has five tasks, namely:

1. Implement the simplified DNS recursor program.
2. Implement the simplified DNS server program.
3. Implement the launcher program that generates DNS server configurations.
4. Implement the verifier program that validates the equivalency of configurations.
5. Test whether the DNS server fulfils the implementation requirements and achieve a high coverage.

The implementation must be made in Python, whereas the testing can be assisted with Bash script. You will be provided a test suite to assist in developing your solutions.

Please read the whole specification to better understand these tasks.

Before attempting this assignment, it would be a good idea to familiarise yourself with socket programming, DNS server in general and the testing tool. A strong understanding of these concepts is essential to completing this assignment.

Some implementation details are purposefully left ambiguous; you have the freedom to decide on the specifics yourself. Additionally, this description does not define all possible behaviour that can be exhibited by the system; some error cases are not documented. You are expected to gracefully report and handle these errors yourself.

You are encouraged to ask questions on Edstem. Make sure your "Question"-typed post is under "Assignment2" category.

As with any assignment, make sure that your work is your own, (not GPT-3/4's, ChatGPT's or copilot's, etc.) and that you do not share your code or solutions with other students.

All documentation in your submission, including comments, README files, and so on, must be written in English. Penalties may apply otherwise.

* `NXDOMAIN` stands for a non-existent domain and represents an error DNS message received by the Recursive DNS server (the client) when the requested domain cannot be resolved to an IP address. In other words, an `NXDOMAIN` error message simply indicates that the domain does not exist.

Background and Glossary

What is DNS?

The Domain Name System (DNS) is the phone book of the Internet. Humans access information online through domain names, like `www.google.com` or `www.apple.com`. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses, so browsers can load Internet resources.

Each device connected to the Internet has a unique public IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorise IP addresses such as 172.217.24.46 (in IPv4).

However, in the context of this assignment, we want to reduce the complexity of communication over the Internet, therefore we simplify the DNS server and implement the simplified DNS server. On the high level, we weaken the DNS functionality and assume all simplified DNS servers listen on the local network interface named `localhost`. Moreover, each simplified DNS server on `localhost` occupies a unique port which other servers use to connect the server.

Domain vs. Hostname

The term "domain" and "hostname" are interchangeable throughout this document.

How does DNS work?

The process of DNS resolution involves converting a hostname (such as `www.example.com`) into a computer-friendly IP address (such as `192.168.1.1`). In this assignment, all simplified DNS servers reside on the same local machine, hence they share the same name/IP address (i.e, `localhost`/`127.0.0.1`). Consequently, the unique port owned by each simplified DNS server is their unique identifier, instead of an IP address.

An IP address is given to some device on the Internet, and that address is necessary to find the appropriate Internet device - like a street address is used to find a particular home. Analogously, a port can identify a simplified DNS server in this assignment.

When a user wants to resolve a hostname, a translation must occur between the hostname that the user supplies, e.g., `www.google.com`, and the port identifier necessary to locate the hostname `www.google.com`.

In order to understand the process behind the DNS resolution, it's important to learn about the different components a DNS query must pass between. Starting from the next section, we will focus on the simplified DNS in the context of this assignment rather than a working DNS on the Internet. We remind the reader to temporarily forget about what they have learnt about a general DNS, though understanding the interaction between Internet DNS components should help the reader to understand how the simplified DNS works.

There are four components involved in resolving a domain:

1. The DNS recursor. The recursor can be thought of as a librarian who is asked to go find a particular book somewhere in a library. The DNS recursor is a server designed to receive queries from client. Typically, the recursor is then responsible for making additional requests in order to satisfy the client's DNS query.
2. Root nameserver. The root server is the first step in translating (resolving) human-readable host names into port identifiers. It can be thought of like an index in a library that points to different racks of books - typically it serves as a reference to other more specific locations.
3. TLD nameserver. The top level domain server (TLD) can be thought of as a specific rack of books in a library. This server is the next step in the search for a specific identifier, and it hosts the last portion of a hostname.
4. Authoritative nameserver. This final nameserver can be thought of as a dictionary on a rack of books, in which a specific name can be translated into its definition. The authoritative nameserver is the last stop in the nameserver query. If the authoritative name server has access to the requested record, it will return the identifier for the requested hostname back to the DNS recursor (the librarian) that made the initial request.

Quick Example

To resolve a hostname `alice.bob.carol.dan.eve`, a client first inputs the hostname to the DNS recursor. Then, the DNS recursor connects and queries the root nameserver `.` about the identifier of the TLD nameserver `eve`. Upon receiving a port of the TLD server `eve`, the recursor connects to the server `eve` and asks which port the authoritative nameserver `dan.eve` owns. Finally, the DNS recursor connects to server `dan.eve` and asks for the identifier of the hostname `alice.bob.carol.dan.eve`.

If the recursor encounters any `NXDOMAIN` error from a server or experiences a time-out during the whole process, it should immediately return a string `NXDOMAIN` to the client (e.g., print `NXDOMAIN` to the `stdout` of the recursor program) as an error message, and stop resolving the current hostname.

Valid Hostname Syntax

The format of a hostname is valid when it can be written in the form of

```
C.B.A
```

Where `A` and `B` are non-empty strings that only contain alphanumeric characters, `a` to `z`, `A` to `Z`, and `0` to `9` and the Hyphen character, `-`.
`C` is a non-empty string that contains alphanumeric characters, the Hyphen character, the Dot character, `.`, such that `C` does not start or end with the Dot character.

Partial-hostname / Partial-domain

- A hostname is valid if it can be interpreted by the `C.B.A` syntax.
- A partial-hostname is valid if it cannot be interpreted by the `C.B.A` syntax, however still can be interpreted by the syntax `A` or `B.A`.

The reader will notice that `google.com` is not a valid hostname but a valid partial-hostname and `www.google.com` is a valid hostname. Indeed, this is true by the design of this system.

Recursor Behaviours

To start the recursor program, the user needs to supply the port of the root nameserver as the first command-line argument, and the time-out in second as the second argument to the recursor.

```
python3 recursor.py root_port timeout
```

When the recursor successfully connects to the root server, it awaits the user to input a domain. The user will input a hostname to query and end the line by entering a "newline" character `\n`, e.g., by pressing "Enter" on the keyboard.
Then the recursor validates the formation of the domain. If the hostname is invalid, the program outputs `INVALID` on the standard output. Otherwise, it starts resolving the domain in six steps:

1. The recursor queries a DNS root nameserver `.` which is known to it as a command-line argument. For example, the port assigned to the root is `1026`. When the user is looking up for `alice.bob.carol.dan.eve`, the recursor sends `eve\n` to the root server over TCP at the port `1026`.
2. The root server then responds to the recursor with the port of the TLD DNS server. When looking up for `alice.bob.carol.dan.eve`, the response from the root is a port owned by the `eve` TLD nameserver. E.g., the recursor receives `1028\n` from the root.
3. The recursor then makes a request to the `eve` TLD by sending `dan.eve\n` to port `1028` over TCP.
4. The `eve` TLD server then responds with the port of the authoritative nameserver `dan.eve`. E.g., the recursor receives `1030\n` from `eve`.
5. The recursor sends a query to the authoritative nameserver `dan.eve` by sending `alice.bob.carol.dan.eve\n` to port `1030` over TCP.
6. The port for `alice.bob.carol.dan.eve` is then returned to the recursor from the nameserver `dan.eve` over TCP. E.g., the recursor receives `1080\n` from `dan.eve`.

Finally, the resolve prints `1080\n` to the standard output and await the next input from user.

If the user needs to exit the recursor program, an end-of-file character will be supplied (e.g., the hotkey `Ctrl-D`).

Start timing from the moment the recursor sends a message to the root server, if the accumulated query time overruns the time-out given by the command-line argument, the recursor will experience a time-out. In this case, the recursor should immediately print a string `NXDOMAIN` to the standard output as an error message, and stop resolving the current hostname.

When the recursor receives a `NXDOMAIN\n` over TCP from a server during the query process, the recursor should immediately print a string `NXDOMAIN` to the standard output as an error message, and stop resolving the current hostname.

Error Handling

- If there are not enough or too many command-line arguments, the recursor should output `INVALID ARGUMENTS\n` and exit. Note that `timeout` can be an integer or float number, and `root_port` should be within the valid range (Refer to the section "Invalid Port Number").
- If the recursor cannot connect to the root server, the recursor should output `FAILED TO CONNECT TO ROOT\n` and exit.
- If the recursor cannot connect to the TLD server, the recursor should output `FAILED TO CONNECT TO TLD\n` and exit.
- If the recursor cannot connect to the authoritative server, the recursor should output `FAILED TO CONNECT TO AUTH\n` and exit.

Server Behaviours

Note that the "server" in this section can be any of the servers described above. That is, the simplified DNS servers are the same program, despite they may depend on different configurations.

To start the server program, the user needs to supply a file path of the server's "single" configuration file as the first command-line argument to the server.

```
python3 server.py config_file
```

Except for the first line, each line of the server configuration file is a record. Each record consists a domain or a partial-domain and a port identifier, delimited by comma. The first line of the configuration contains the port number that the server occupies.

- Example Server Configuration ("Single" Configuration)

  ```
  1025
  com,1234
  org,2587
  www.google.com,8987
  google.com,8888
  ```

A server ("single") configuration is invalid if:

- its first line has an invalid port;
- it contains invalid records;
  - the record has an invalid domain or partial-domain;
  - the record has an invalid port identifier;
- it contains contradicting records.
  - records contradict when they have the same (partial-)domain but different ports.

For the interactions between a server and a recursor, they are stated in the previous section, therefore won't be reiterated at here.

If any server does not hold the record of a valid (partial-)hostname being queried, the server should send `NXDOMAIN\n` as the reply to indicate such (partial-)hostname cannot be resolved.

Logging

The server will log resolving history on the standard output. When it replies to a recursor, it logs a line on `stdout`:

```
resolve HOSTNAME to ID
```

where the `HOSTNAME` refers to the inbound hostname and `ID` refers to the port identifier that this server translates to from the hostname. If a hostname is not on the record of this server, its `ID` will be presented as `NXDOMAIN`.

Commands

A user can alter the records of a server after it starts running by sending two types of commands to the program via the TCP. Invalid commands will trigger an error `INVALID\n` to be printed to the standard output.

A command will not change the configuration file which is being supplied to a running server. A command will temporarily change the server's records and these changes caused the commands will be lost when the server is shutdown. The changes will not persist if the server is restarted with the same configuration.

- `ADD` command

  ```
  !ADD HOSTNAME PORT\n
  ```

  This command should add the record of `HOSTNAME,PORT` to the server. If the server already holds a record of `HOSTNAME`, this command will overwrite the existing record. If the `HOSTNAME` is invalid, this command will do nothing. If the `PORT` is invalid or has already been used by another `HOSTNAME`, this command will do nothing.

- `DEL` command

  ```
  !DEL HOSTNAME\n
  ```

  This command should remove the record of `HOSTNAME` from the server. If the server does not hold a record of `HOSTNAME` or the `HOSTNAME` is invalid, this command will do nothing.

- `EXIT` command

  ```
  !EXIT\n
  ```

  This command should shut down the server.

Note that a complete command must start with an Exclamation character `!`, hence it cannot be a valid hostname or partial-hostname. If an input over TCP to the server is neither a complete command nor a valid (partial-)hostname, it will log `INVALID\n` on `stdout`.

Handle "incomplete" messages

The server should **only** start processing an inbound message if the message is terminated by a new-line character, `\n`.

If the server receive an "incomplete" message, that is, a message over TCP however not terminated by `\n`, the server should keep the "incomplete" message in a buffer. The server should then accumulate all the "incomplete" messages until a `\n` is finally received by the server. This implies that the server should not start validating the message until the message is completed.

Examples:

- An input message `!EXI` over TCP to the server is incomplete, and the server should wait for more inputs and look forward to a `\n` to end this incomplete message. If `T\n` is received immediately after, then the server should quit as expected.
- An input message `abc\n!EXIT\n` should be interrupted as one invalid hostname followed by a complete `EXIT` command.
- An input message `abc\n!E` should be interrupted as one invalid hostname followed by an incomplete message that needs more incoming inputs and a `\n` to be completed.

When a TCP client (including a recursor) disconnects from the server, the buffer of "incomplete" messages should be cleared.

Error Handling

- If there are not enough or too many command-line arguments, the server should output `INVALID ARGUMENTS\n` and exits.
- If `config_file` does not exist, cannot be read or is invalid, the server should output `INVALID CONFIGURATION\n` to `stdout`, exit.

Launcher Behaviours

The launcher can break down a "master" DNS configuration to a collection of "single" server configuration files.
A "master" configuration should be equivalent to the collection of "single" configurations that the launcher generates from the "master".

Syntactically, a "master" configuration is a special kind of "single" configuration which does not permit any record including a partial-domain.

- Example "Master" Configuration

```
1024
www.google.com,8987
```

A "master" configuration is invalid if:

- its first line has an invalid port;
- it contains invalid records;
    - the record has an invalid domain;
    - the record has an invalid port identifier;
- it contains contradicting records.
    - records contradict when they have the same domain but different ports.

Note that it can be assumed that the "master" configuration file always has at most 21504 records.

To start the launcher program, the user needs to supply the file path of the "master" configuration file as the first command-line argument to the launcher.
The second command-line argument is the directory path where the program should generate a collection of "single" configuration files.

```
python3 launcher.py master_file directory_of_single_files
```

Then, the launcher program may generate server configuration files, also called the "single" configuration files. "Single" configurations may include both domains and partial-domains, and can be used to start up a collection of simplified DNS servers.

The naming convention of the generated "single" configuration files is unrestricted - their names can be purely random. The launcher can name the generated files freely. (This is not an update, but a reminder of FAQ Launcher Q3 that was previously released in v2.)

Note that these DNS servers should be able to resolve all hostnames in the "master" configuration to their corresponding port identifiers. For example, three "single" configuration files can be made:

- `root.conf`

  ```
  1024
  com,1025
  ```

- `tld-com.conf`

  ```
  1025
  google.com,1026
  ```

- `auth-google.conf`

  ```
  1026
  www.google.com,8987
  ```

Note that, if a hostname does not exist in the "master" file, then the servers that are set up by all generated "single" files should not be able to resolve such a hostname.

Error Handling

- If there are not enough or too many command-line arguments, the launcher should output `INVALID ARGUMENTS\n` and exit.
- The launcher should first validate the given "master" file before generating "single" configuration files.
  - If the master file does not exist or is invalid, it should output an error message `INVALID MASTER\n` to `stdout`, terminate the generation and exit.
  - If the `direcotry_of_single_files` does not exist or cannot be written to, the launcher should output an error message `NON-WRITABLE SINGLE DIR\n` to `stdout`, terminate the generation and exit.

Note that a launcher is allowed to overwrite existing files on disk.

Verifier Behaviours

The verifier program can compare a "master" file and a collection of "single" files, and output whether they are exactly equivalent.

To start the verifier program, the user needs to supply the file path of a "master" configuration file as the first command-line argument, and the directory path that contains a collection of "single" configuration files as the second command-line argument.

```
python3 verifier.py master_file direcotry_of_single_files
```

The naming convention of the "single" configuration files to be compared with the "master" file is unrestricted - their names can be purely random. The verifier should assume "single" configuration files has random names. (This is not an update, but a reminder of FAQ Verifier Q1 & Launcher Q3 that was previously released in v2. The verifier should be able to verify what the launcher generates, which can be "single" files with random names, hence the verifier should be capable of testing "single" files with random naming.)

For example, if the `master_file` contains

```
1024
www.google.com,8987
```

and the `direcotry_of_single_files` contains the three "single" files, namely `root.conf`, `tld-com.conf`, and `auth-google.conf`.

- `root.conf`

  ```
  1024
  com,1025
  ```

- `tld-com.conf`

  ```
  1025
  google.com,1026
  ```

- `auth-google.conf`

  ```
  1026
  www.google.com,8987
  ```

Then the program should output `eq\n` in the standard output and exit.
However, if we alter the `auth-google.conf` so that it contains

- `auth-google.conf`

  ```
  1026
  www.google.com,898700
  ```

then the program should output `neq\n` in the standard output and exit.

Error Handling

- If there are not enough or too many command-line arguments, the verifier should output `invalid arguments\n` and exit.
- The verifier should first validate the given "master" file before generating "single" configuration files. If the master file does not exist or is invalid, it should output `invalid master\n` to `stdout`, abort further comparison and exit.
- If the `direcotry_of_single_files` does not exist or cannot be read, the verifier should output `singles io error\n` to `stdout`, abort further comparison and exit.
- If some "single" file is invalid, the verifier should output `invalid single\n` to `stdout`, abort further comparison and exit.

Other Error Handling

Invalid Port Number

Only ports in the range of `[1024, 65535]` are considered valid. An invalid port should result in an appropriate error message.

`OSError: Address already in use`

The servers should always make use of `SO_REUSEADDR` flag to prevent such error from occurring. See also at https://docs.python.org/3/library/socket.html

Testing

The examiner will change the working directory to the root of your git repository and execute `bash tests/run.sh` to run **ALL** your test cases.

You are expected to write a number of test cases for all implemented programs, however **ONLY** the testing for the server program will be assessed. You are expected to test as many execution paths as possible for your code.

We will provide you with one sample test case, but it does not test all the functionality described in the assignment. It is important that you thoroughly test your code by writing your own tests.

Unit tests are not required and not assessable.

You must place all of your test cases in the `tests/` directory. The examiner needs to be able to run your tests and the coverage of your tests must be output at the end of their execution. Run the sample test script `tests/sample_test.sh` to check the expectation. E.g.,

```
$ bash tests/sample_test.sh
server starts listening
fake recursor sends EXIT
server accepts connection
server receives b'!EXIT\n'
server exits
Name                     Stmts   Miss  Cover   Missing
------------------------------------------------------
tests/sample_server.py      15      0   100%
------------------------------------------------------
TOTAL                       15      0   100%
$ bash tests/sample_test_diff.sh
fake recursor sends EXIT
```

```
Name                     Stmts   Miss  Cover   Missing
------------------------------------------------------
tests/sample_server.py      15      0   100%
------------------------------------------------------
TOTAL                       15      0   100%
1,4d0
< server starts listening
< server accepts connection
< server receives b'!EXIT\n'
< server exits
```

`Coverage.py` will be used to generate the coverage of your own tests and the coverage rate will determine some testing marks.

However, the testing marks will not only be determined by the coverage, but also by the functionalities that the program can achieve and by how many functionalities are tested. E.g., the logging and commands are expected to be tested. The examiner will inspect functionality testing manually.
If the server program achieves ZERO functionality (e.g., it outputs "hello" and exits) and it achieves 100% coverage rate, the awarded marks for testing will not be full mark.

End-to-end (input/output) testing is welcome to test the functionalities. Each pair of end-to-end test case could have a `<name>.in` input file and a `<name>.out` output file. We recommend that the names of your end-to-end test cases are descriptive so that the reader can easily know what each case is testing. E.g. `exit_1.in`, `exit_1.out`.

You should have a brief description of your tests, and how they can be run, in `README.md`. Please keep it concise.

Implementation

The recursor, server, launcher and verifier need to be implemented in Python. A set of scaffold files will be provided. You are expected to write legible code with good style, e.g. PEP 8. Testing and non-assessable components can be done with Bash script.

You are free to use all built-in functions of Python. You are **NOT** allowed to import **ANY** Python modules except:

- `pathlib`,
- `random`,
- `signal`,
- `sys.argv`,
- `socket`,
- `time`,
- `typing`.

If you want to use an additional module which will not trivialise the assignment, please ask on Ed. The allowed library list may be extended. Related announcement will be posted accordingly.

You should **NOT** preserve any data temporarily or permanently in any actual files on disk, except for the files to be generated.

TCP should be used for all connections in this assignment.

Please be mindful that breaching the above restrictions may result in serious deductions for the entire assignment.

How to start a DNS infrastructure?

1. Prepare a "master" configuration.
2. Generate a collection of "single" configurations by the launcher.
3. Start the DNS servers by the "single" configuration files, manually or automatically by your own script (non-assessable component).
4. Start a recursor, and start resolving hostnames.
5. Shut down the recursor and the servers when you finish querying.

Submitting your code

An Ed Lesson code challenge is available for you to test and submit your code. Public test cases will be released up to **Week 10, Sunday, October 15th**. Additionally, there could be a set of unreleased test cases which will be run against your code after the due date.

You will need to use `git` to submit your code. To learn what `git` is and how to set it up (and SSH key) properly, please refer to the Git Lesson.

Where to start

Note: the ordering of these suggestions does not necessarily imply the ordering in which you should consider them.

- Understand what DNS is in the context of this assignment. Drawing on a piece of paper may be helpful.
- Understand how components in this assignment interact with each other. Drawing on a piece of paper may be helpful.
- Understand the structure of "master" and "single" configuration files. Design a feasible method to convert "master" configuration to some "single" configurations.
- Consider edge cases.
- Be familiar with basic Python file I/O.
- Be familiar with Python socket programming. Use `sample_server.py` as a basic reference. Consult the official Python documentation and examples: https://docs.python.org/3/library/socket.html and https://docs.python.org/3/howto/sockets.html.

Marking Criteria

This assignment is worth **20 marks** of your overall grade for the course.

Auto-test on Edstem

An automatic testing system on Edstem will mark your code submission. A mark will be given based on the auto-tests passed on Edstem (15/20). There will be public test cases made available for you to test against, but there will also be extra non-public tests used for marking. Success with the public tests doesn't guarantee your program will pass the private tests.

Auto-tests can be categorised into four tasks. As a condition, you will NOT be able to receive marks for Task 4 unless all public test cases in Task 1 and 2 have been completed.

- Task 1 (4 marks)

  To complete this task, you need to implement the recursor program.

- Task 2 (6 marks)

  To complete this task, you need to implement the server program.

- Task 3 (2 marks)

To complete this task, you need to implement the launcher program.

- Task 4 (3 marks)

  To complete this task, you need to implement the verifier program.

Manual Inspection

A manual mark will be given for overall style, quality, readability, etc (2 marks).

Self-testing

You are expected to write your own tests and submit them with your code. A mark will be given based on the coverage and the manual inspection of your tests (3 marks).

Your test cases must cover as many execution paths as possible and should be designed to test specific features. These test cases are for the executable code written in Python for the server program, e.g., Task 2.

Friendly note and important dates

Sometimes we find typos or other errors in specifications. Sometimes the specification could be clearer. Students and tutors often make great suggestions for improving the specification. Therefore, this assignment specification may be clarified up to **Week 10, Sunday, October 15th**. No major changes will be made. Revised versions will be clearly marked, and the most recent version announced to the class via Ed Discussions.

The assignment will be official due on October 22nd 2023, Week 11 Sunday, at 23:59, Sydney local time. Late penalty applies if you submit your work late. Please refer to the late policy on the Unit Outline.

November 5th 2023 is the last date to discontinue fail a unit of study in Semester 2 (S2C). Please also refer to the official information provided by the University on discontinuing: https://www.sydney.edu.au/students/discontinue-unit-of-study/dc.html

Frequently Asked Question

* Note: this section is initiated by the staff.

** Moved to FAQ.md.

Warning

Any attempts to deceive the automatic marking system will result in an immediate zero for the entire assignment.

Negative marks can be assigned if you do not properly follow the assignment description, or your code is unnecessarily or deliberately obfuscated.

All documentation in your submission, including comments, README files, and so on, must be written in English. If not, penalties may apply.

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

Major changes will be announced here.