Assignment Brief: Incremental Programming Logbook

Course Title: Programming for Engineers (Level 5)

Deadline: 13th Decemeber 23:00

Objective:

In this assignment, you will demonstrate your understanding of various C programming concepts by incrementally building a program. You will be provided with a `main.c` file containing commented-out method calls. Your task is to implement the necessary functions by creating a `myfunctions.h` header file and a `myfunctions.c` implementation file. You will uncomment each part section in `main.c` as you complete it, remember to build the `main.exe` to test and submit your work incrementally to Git with suitable commit messages.

After completeting the `C` part of the logbook, you will continue with the `Python` part, note that like `C` you need to uncomment where appropriate and implement the functionality. You should create a `myfucntions.py` and import it into the `main.py` script, `import myfunctions`.

1. You can download the logbook exercises from here:

- Github classroom
  - https://classroom.github.com/classrooms/114398065-uniofgreenwich-elee1147
- Github classroom Assignment
  - https://classroom.github.com/a/dcEp-FSq
- Clone the repository i.e:

  ```
  $ git clone git@github.com:uniofgreenwich/ELEE1147_Logbooks_YOURGITHUBUSERNAME.git
  ```

- The contents of the repo will be as follows:
  - `ELEE1147_Logbook`
    - `.git/`
    - `C/main.c`
    - `Python/main.py`
    - `Brief.md`
    - `README.md`

2. Instructions:

Git Workflow:

1. **Continual git workflow**
   - For each completed step you must add and commit all changes.
   - Ensure all commit messgages follow the standard
   - as there are 9 parts to complete, there should be a **minimum** of 9 commits.
2. **Update the README.md**

- The README.md needs to be updated with the following sections:
  - Introduciton
    - Describe the purpose of the repo, keep it less than 100 words.
    - *hint: Look at the brief*
  - Prerequisites
    - What do you need to run the code in this repo, include version numbers of software/packages
    - *hint: what is available on the university machines that we have/will be using for this module.*
  - Author
    - Add your Github User name as a hyper link to your profile

Intentionally left blank

C: Implement Part 1: Basic Math Operations with Different Data Types

- Uncomment the Part 1 section in `main.c`.
- Create two new files: `myfunctions.h` and `myfunctions.c`.
- In `myfunctions.h`, add prototypes for the part 1 functions in the now uncommented section
- In `myfunctions.c`, implement the above functions to perform basic math operations.

C: Implement Part 2: Bitwise Operations

- Uncomment the Part 2 section in `main.c`.
- In `myfunctions.h`, add prototypes for the following functions in the now uncommented section
- In `myfunctions.c`, implement the above functions to perform bitwise operations.

C: Implement Part 3: Memory Management

- Uncomment the Part 3 section in `main.c`.
- In `myfunctions.h`, add prototypes for the following functions in the now uncommented section.
- In `myfunctions.c`, implement the above functions to dynamically allocate memory for an array and initialize it with random values.

C: Implement Part 4: Data Structures

- Uncomment the Part 4 section in `main.c`.
- In `myfunctions.h`, add prototypes for the following functions in the now uncommented section.
- In `myfunctions.c`, implement the above functionality and build data structures called struct

Python: Implement Part 5: Basic Python Operations

- Uncomment the Part 5 section in `main.py`.
- Create one new file: `myfunctions.py`.
- In `myfunctions.py`, implement the above functions to perform basic math operations.

Python: Implement Part 6: Basic Python Bit Wise Operations

- Uncomment the Part 6 section in `main.py`.
- In `myfunctions.py`, implement the above functions to perform basic bitwise operations.

Python: Implement Part 7: Memory Management Simulation (Lists in Python)

- Uncomment the Part 7 section in `main.py`.
- In `myfunctions.py`, implement the above functions to perform memory management operations.

Python: Implement Part 8: Objects and Classes

- Uncomment the Part 8 section in `main.py`.
- In `myfunctions.py`, implement the above functions to perform Object and class operations.

Submission:

- Ensure that you have committed each part section incrementally with appropriate commit messages.
- Push your final changes to the remote repository.
- The final commit hash is the text you place in the submission point.

Intentionally left blank

Grading Criteria:

- **Commit Quality (20%):** Each commit message should clearly describe the changes made.
- **Incremental Submission (20%):** Each part section should be committed and pushed incrementally.
- **Correctness C (20%):** The functions should work correctly as per the requirements of each part.
- **Correctness Python (20%):** The functions should work correctly as per the requirements of each part.
- **Code Quality (20%):** Code should be well-organized and commented where necessary.

Full Rubic Below:

Commit Quality 20%

Criteria	Description
Exceptional (80-100%)	All commit messages are clear, descriptive, and well-structured, providing detailed insight into the changes made.
Excellent (70-79%)	Commit messages are clear, descriptive, and structured well.
Very Good (60-69%)	Commit messages are clear and mostly descriptive, with minor issues.
Good (50-59%)	Most commit messages are clear and somewhat descriptive.
Satisfactory (40-49%)	Commit messages are present but may be vague or inconsistent.
Fail (30-39%)	Commit messages are minimal, often unclear or lacking in detail.
Fail (0-29%)	No commit messages, or messages are unclear and lack description.

Intentionally left blank

Incremental Submission 20%

Criteria	Description
Exceptional (80-100%)	Each part section is committed and pushed incrementally, with well-timed and consistent commits.
Excellent (70-79%)	Incremental submissions are consistent and well-timed for each part section.
Very Good (60-69%)	Incremental submissions are mostly consistent, with minor issues.
Good (50-59%)	Most part sections are committed incrementally, but there may be minor inconsistencies.
Satisfactory (40-49%)	Some part sections are committed incrementally, but with gaps or inconsistencies.
Fail (30-39%)	Few part sections are committed incrementally; most changes are made in large chunks.
Fail (0-29%)	No incremental submission; commits are made all at once or infrequently.

Correctness C 20%

Criteria	Description
Exceptional (80-100%)	All C functions work correctly as per the part requirements, with no errors.
Excellent (70-79%)	C functions are correct and meet the requirements, with very few or no errors.
Very Good (60-69%)	C functions are mostly correct, with minor errors.
Good (50-59%)	Most C functions work correctly, with some errors.
Satisfactory (40-49%)	Some C functions work correctly, but there are significant errors.
Fail (30-39%)	Few C functions work correctly; many errors present.
Fail (0-29%)	The C functions do not work correctly or the code does not meet the requirements.

Intentionally left blank

Correctness Python 20%

Criteria	Description
Exceptional (80-100%)	All Python functions work correctly as per the part requirements, with no errors.
Excellent (70-79%)	Python functions are correct and meet the requirements, with very few or no errors.
Very Good (60-69%)	Python functions are mostly correct, with minor errors.
Good (50-59%)	Most Python functions work correctly, with some errors.
Satisfactory (40-49%)	Some Python functions work correctly, but there are significant errors.
Fail (30-39%)	Few Python functions work correctly; many errors present.
Fail (0-29%)	The Python functions do not work correctly or the code does not meet the requirements.

Code Quality 20%

Criteria	Description
Exceptional (80-100%)	Code is exceptionally well-organized, modular, and commented, demonstrating a high level of understanding.
Excellent (70-79%)	Code is well-organized, modular, and well-commented throughout.
Very Good (60-69%)	Code is well-organized and mostly well-commented, with minor issues.
Good (50-59%)	Code is mostly well-organized and commented, but with some inconsistencies.
Satisfactory (40-49%)	Code organization and commenting are inconsistent, leading to confusion.
Fail (30-39%)	Code is poorly organized and lacks sufficient comments.
Fail (0-29%)	Code is disorganized, lacks comments, and is difficult to understand or follow.