COMP1312 Programming I (UoSM) 2025/2026
Coursework I

This Coursework is worth **30%** of the total marks for this module. The deadline is **8th December 2025, 17:00 (Malaysia Time)**, handin via the blackboard site. The work is expected to take **10 to 30 hours**, can vary based on your familiarity with C and problem-solving skills.

Coursework Overview:
In this coursework, you must **demonstrate** a ***Banking System Application*** using ***C programming***. You have the freedom to choose to work with any Integrated Development Environment (IDE). This exercise aims to emphasize fundamental programming concepts, such as how to make efficient use of the ***various data types and variable types***, ***arrays***, ***loops***, ***conditionals***, ***functions, and user inputs***. You will write a program (as a deliverable of this coursework) that enables users to ***create new bank account, perform deposit, withdrawals, remittance and delete bank account.***

Coursework Requirements

The essential requirement of this coursework is to demonstrate the ***Banking System Application*** *by* **using ONLY native C Programming features and methods such as <stdio.h> <stdlib.h> <stdint.h> <stdbool.h> <string.h> <ctype.h> <time.h> <errno.h> and <limits.h>,** you are not allowed to use third party libraries. The coursework requirements are as follows:

1. Menu & Session

- Implement a text-based menu that repeatedly prompts the user to select from the available options.
- Allow users to perform **FIVE (5)** operations, **Create New Bank Account**, **Delete Bank Account**, **Deposit, Withdrawal and Remittance**.
- Text menu loops until Exit.
- Menu must support numbered and keyword selections (e.g. "1" or "create", "deposit", etc.).
- Show session info (date/time, count of loaded accounts) at start.
- Append an entry for each action ("create account", "deposit", "withdrawal", etc.) to transaction.log which should be in a directory called "**database**".

2. Create New Bank Account

- When creating a new bank account, prompt users to input details such as **Name**, **Identification Number (ID)**, **Type of Account (Savings** or **Current) and 4-digit PIN.**
- Generate a **bank account number** by generating a random number between 7 and 9 digits. Ensure uniqueness by checking an index file (stored in "**database**" directory).
- Initial balance defaults to 0.00.
- Store all the information above in a file (.txt format).
- One file represents one bank account (e.g. **TWO (2)** bank accounts will have **TWO (2)** files, 5471252.txt, 7904612.txt).
- All .txt files should be placed in a directory called "**database**".

3. Delete Bank Account

- Retrieve all banking account numbers from the index file and allow users to choose and delete from the current list of existing bank account number.
- Require the user to confirm the account number + the last 4 characters of ID number + 4-digit PIN.
- After deleting a bank account, all information (current balance, personal information, banking number etc) should no longer be available in the file. The file should also be deleted from the "**database**" directory.

4. Deposit

- Authenticate with account number + 4-digit PIN.
- Allow users to deposit an amount into an existing bank account.
- Amount must be > RM0 and ≤ RM50,000 per operation.
- The current balance should be updated in the file after the deposit.

5. Withdrawal

- Authenticate with account number + 4-digit PIN.
- Show available balance.
- Allow users to withdraw an amount from an existing bank account.
- The current balance should be updated in the file after the withdrawal.

6. Remittance

- Authenticate with sender with 4-digit PIN.
- Allow users to transfer an amount from an existing bank account to another existing bank account.
- Validate both accounts exist and are distinct.
- The current balance of both affected accounts should be updated after the remittance and the new current balance should be updated in the file respectively.
- Additionally, the following rules apply:
  - Savings to Current account will incur a 2% remittance fee (e.g. Transfer amount: RM100, remittance fee: RM2).
  - Current to Savings account will incur a 3% remittance fee (e.g. Transfer amount: RM100, remittance fee: RM3).
  - The remittance fee shall be deducted from the current balance of the sender.

Important:
While no specific error handling/edge cases (examples are provided only for understanding) are mentioned in the coursework description, it is essential that ***your program is robust and handles unexpected or invalid inputs***. ***<u>You are responsible for considering and addressing potential edge</u>***

Marking Guidelines

Grading Criteria	Wtg	Excellent 5	Above average 4	Average 3	Fair 2	Low 1	0	Max Mark (≈Wtg * Score)
Menu & Session	1	All menus are available and works excellently without any errors or bugs. Menus can be exited. Menus support both numbered and keyword selections. Session information is displayed. Each transaction is logged and stored in a transaction.log file	All menus are available and work but with very minor errors and/or bugs. Menus can be exited. Most menus support both numbered and keyword selections. Session information is displayed. Most transaction is logged and stored in a transaction.log file	Most menus are available but with some errors and/or bugs. Menus can be exited. A partial of menus support both or either numbered and keyword selections. A partial of session information is displayed. A partial of transaction is logged and stored in a transaction.log file	Few menus are available but with a number of errors and/or bugs. Menus can be exited. Very few menus support both or either numbered and keyword selections. No session information is displayed. Most transaction are not logged but are stored in a transaction.log file	Few menus are available but with a concerning number of errors and/or bugs. Menus are not able to exit. Very few menus support either numbered or keyword selections. No session information is displayed. Most transaction are not logged but are stored in a transaction.log file however, with discrepancy in transaction information	No menus are available. No session information is displayed. No transactions are logged and are not stored in a transaction.log file.	**5**
Create New Bank Account	3	All required personal information are prompted and stored accurately in a file. Generated bank account number are all unique and not repetitive	All required personal information are prompted and stored accurately in a file with minor bugs. Generated bank account number are all unique and not repetitive	Some required personal information are prompted and partially stored in a file with minor bugs. Some bank account numbers are not unique	Some required personal information are prompted but not stored in a file. Some bank account numbers are not unique	Missing many required personal information and information partially stored in a file. No bank account number is generated	Feature not implemented.	**15**
Delete Bank Account	3	The bank account along with all the information are successfully and accurately deleted from the file without any trace.	The bank account along with all the information are successfully and accurately deleted from the file with minor bugs	The bank account along with some of the information are deleted from the file with minor bugs	Partial deletion of data with many bugs such as wrong account deleted	Deletion of banking account is performed but the file is not updated.	Feature not implemented.	**15**

Deposit	**3**	Deposit process is successfully, and accurately performed and current balance is updated in the file.	Deposit process is successfully, and accurately performed and current balance is updated in the file with minor bugs	Deposit process is successful, but current balance is not accurate. Current balance is updated in the file.	Deposit process is successful, but current balance is not accurate. Current balance is inaccurately updated in the file.	Deposit is performed but the file is not updated.	Feature not implemented.	**15**
Withdrawal	**3**	Withdrawal process is successfully, and accurately performed and current balance is updated in the file.	Withdrawal process is successfully, and accurately performed and current balance is updated in the file with minor bugs	Withdrawal process is successful, but current balance is not accurate. Current balance is updated in the file.	Withdrawal process is successful, but current balance is not accurate. Current balance is inaccurately updated in the file.	Withdrawal is performed but the file is not updated.	Feature not implemented.	**15**
Remittance	**4**	Remittance process is successfully, and accurately performed and current balance is updated in the file.	Remittance process is successfully, and accurately performed and current balance is updated in the file with minor bugs	Remittance process is successful, but current balance is not accurate. Current balance is updated in the file.	Remittance process is successful, but current balance is not accurate. Current balance is inaccurately updated in the file.	Remittance is performed but the file is not updated.	Feature not implemented.	**20**
Error Handling and Validation	**3**	Handles all invalid inputs and edge cases flawlessly. Program never gets stuck or produces unexpected behaviour due to bad input.	Most input errors and edge cases are handled, but the program may have minor issues with unusual inputs.	Basic validation present, but program behave unexpectedly due to bad input.	Little error handling or validation, frequent issues with invalid input, causing the program to behave incorrectly.	No input validation or error handling, program fails to handle invalid input effectively.	No error handling implemented.	**15**
Total	**20**							100

Any work submitted after the deadline's time will be subject to the standard University late penalties unless an extension has been granted, in writing by the Senior Tutor, in advance of the deadline. Details on the University's late penalties can be found here:

- https://www.southampton.ac.uk/~assets/doc/quality-handbook/Late%20Submission.pdf

Edge Cases

In the context of a Banking System program, here are a few examples of "bad inputs" or edge cases that the students' code should be able to handle:

1. Non-numeric Input

When users are expected to input number (e.g., deposit money), they might enter something that isn't a number:

- **Edge Case**: The user enters a string instead of a number.
  - **Bad Input**: "four hundred"
  - **Expected Handling**: The program should catch this and ask for valid numeric input instead of crashing.
  - **Example Error Message**: *"Invalid input. Please enter a number."*

2. Insufficient Funds

When users withdraws/remit money, there could be an insufficient amount of money in the current balance:

- **Edge Case**: The user enters an amount to withdraws/remit
  - **Bad Input**: entering 400 but the current balance is showing 100.
  - **Expected Handling**: The program should notify the user that the there is an insufficient amount.
  - **Example Error Message**: *"Insufficient funds. Please try again."*

3. Invalid Option in Menu

When selecting from the menu, the user might input an invalid option.

- **Edge Case**: The user enters a number or character that is not part of the available menu options.
  - **Bad Input**: Typing 7 when the menu only has options 1–5.
  - **Expected Handling**: The program should display an error message and re-prompt the user.
  - **Example Error Message**: *"Invalid option. Please select a valid menu option."*

Handling these bad inputs gracefully ensures that the program doesn't crash or behave unexpectedly, and instead informs the user of the mistake and prompts them for correct input.