College of Engineering and Technology (CEAT)
Department of Computer Science and Engineering

Assignment on Programming in C
Fall Semester 2025
Program: Bachelor of Computer Science and Engineering (BCSE)
Course Instructor: Md. Min Khayer

Course No.: CSC 183
Course Title: Programming in C **Full Marks: 100**
Section: B, C **Submission Date: 12-01-2025**

Target CO3, POa: Apply knowledge of mathematics, natural science, engineering fundamentals and an engineering specialization

CO3	Identifying appropriate techniques of the programming language to solve a particular problem.	
POa	**K1, K2, K3**	

Note:1) This assignment should be undertaken by the students individually. **2)** Assessment of this assignment will be done through the rubrics.

Objective:
To evaluate students' ability to apply basic mathematical reasoning and engineering fundamentals in designing and implementing a modular C program using arrays and functions to simulate a Simple Banking Transaction System. The system should support basic banking operations while ensuring correctness, validation, and logical consistency.

Scenario:

A commercial bank wants to develop a Simple Banking Transaction System to assist bank officers in managing daily customer transactions. The system should handle up to 20 customers and support basic banking activities such as deposits, withdrawals, balance checking, and transaction history tracking. The system must also detect and prevent invalid operations such as negative deposits and overdraft attempts.

Expected Features of the System:

Data Entry & Storage

- Store customer information such as:
  - Customer name
  - Account number
  - Account balance
- Use arrays to store customer data.
- Maintain a transaction history of the last 10 operations for each customer using arrays.

Banking Operations

The system must support the following operations through a menu-driven interface:

- Deposit money: Validate input to ensure the deposit amount is positive.
- Withdraw money: Ensure that withdrawal does not exceed the current balance.
- Check account balance
- View transaction history: Display the last N transactions for a selected customer.

All operations must be implemented using separate functions to ensure modularity.

Error and Exception Handling

The program must handle the following cases:

- Prevent zero or negative deposit amounts.
- Prevent withdrawals that exceed available balance.
- Prevent invalid numeric inputs.
- Ensure transaction history does not exceed the last 10 records.

Reporting Features

- Display a summary table showing:
  - Account number
  - Customer name
  - Current balance
- Allow bank officers to view transaction history for any selected customer.

Design Considerations and Constraints

Requirement-Level Considerations and Constraints:

Customer Limit: Bank officers prefer unlimited customers, but system constraints limit the number to 20.

Transaction History: Full transaction history is desired, but memory limitations restrict storage to the last 10 transactions.

Withdrawal Rules: Customers may request overdraft flexibility, but bank policy strictly prohibits negative balances.

Numerical Accuracy: Financial values require accurate calculations while maintaining program simplicity.

Technical-Level Considerations and Constraints:

Data Storage: Use arrays (fast access, fixed size) vs dynamically allocated structures (flexible but more complex).

Precision: Use float for speed vs double for accuracy in calculations.

Input Method: Read from console (simple, interactive) vs read from file (automated, reusable).

Error Handling: Strict input validation improves reliability but increases code length.

Task for Students:

1. Conflict Analysis

- Identify and analyze all requirement-level and technical-level conflicts.
- Choose one approach for each conflict and justify your choice.
- Compare your chosen approach with at least one alternative.

2. System Design

Prepare the following design artifacts:

a) Flowchart

The flowchart should represent:

- Main menu
- Deposit operation
- Withdrawal operation

- Balance checking
- Transaction history display
- Program exit

b) Pseudo-code

Write pseudo-code for the following functions:

- deposit()
- withdraw()
- showBalance()
- addTransaction()

3. Implementation

The C program must:

- Use arrays to store customer data and transaction history.
- Use functions for each banking operation.
- Validate all user inputs.
- Follow structured programming principles.
- Be menu-driven and user-friendly.

4. Testing

- Create at least 5 test cases:

Test Case Type	**Example**
Zero deposit attempt	Deposit = 0
Overdraft attempt	Withdraw > Balance
Multiple transactions	Several deposits and withdrawals
Boundary test	More than 10 transactions
Invalid input	Negative or non-numeric input

- Record results in: Test Case → Expected Output → Actual Output → Pass/Fail.

5. Documentation (Report)

- Submit a concise project report (PDF) including:
  - Introduction
  - Conflict analysis & chosen solutions

- Flowchart & pseudo-code
- Source Code listing with explanations
- Test case results
- Conclusion

Submission Requirements:

Soft Copy submission:

- Project report (PDF) with CEP Justification
- .c source code file
- Flowchart and pseudo-code (PDF/images)
- All in a ZIP file named: StudentID_Section uploaded to Google Classroom.

Hard Copy submission: Submit the hardcopy of your project assignment by 12th January, 2026.

Provide CEP Justification

Complex Engineering Criteria	Knowledge Profile	Your Justification by mentioning Section number/ line number of code/chapter number
	K1 (Engineering fundamentals knowledge)	
	K2 (Mathematics & data analysis)	
	K3 (specific knowledge)	

The following rubrics will be used for assessing the submission. Please see the next page:

Programming in C

Assessment Rubrics

Assessment Criteria	Sub-Criteria (KPA Level)	Weight (%)	Excellent 80-100	Good 60-79	Satisfactory 45-59	Insufficient Below 45
Problem Analysis	Application of Natural Science in problem analysis	10	Natural science principles are applied correctly and effectively to analyze the problem.	Some natural science principles are applied, but with minor gaps.	Few natural science principles are applied, but with gaps.	Little or no use of natural science principles in analysis.
	Application of computer and information science in problem analysis	10	Relevant computer and information science concepts are applied fully to understand the problem.	Some relevant concepts are applied, but not fully.	Few relevant concepts are applied, but partially.	Concepts are mostly missing or incorrectly applied.
	Formulation of Engineering Fundamentals in problem analysis	30	Engineering fundamentals are clearly formulated and applied to the analysis.	Fundamentals are reasonably formulated or applied.	Fundamentals are partly formulated or applied.	No clear application of engineering fundamentals.
	Identifying raised technical conflicts	50	All possible	Some technical	Few technical	Very few or no

	in problem analysis		technical conflicts are identified.	conflicts are identified.	conflicts are identified.	conflicts are identified.
Program Design	Formulation of Engineering Fundamentals in program design	20	Engineering fundamentals are accurately applied to develop the program design.	Engineering fundamentals are fairly applied to develop the program design.	Engineering fundamentals are poorly applied to develop the program design.	Little or no use of fundamentals in design.
	Resolving raised technical conflicts in program design	20	All raised conflicts are effectively addressed in the design.	Some conflicts are addressed.	Few conflicts are addressed.	Very few or no conflicts are addressed.
	Standard of the proposed design	60	The proposed design is accurate and meets requirements.	The proposed design is somewhat accurate with minor issues.	The proposed design is to some extent accurate with major issues.	The proposed design is inaccurate or incomplete.
Functionality	Implement the features using the concepts of engineering fundamentals	20	All features are implemented correctly using relevant engineering fundamentals.	Some features are implemented correctly.	Fewfeatures are implemented correctly.	Very few or no features are implemented correctly.

	The raised conflicts are resolved by implementing the functionalities	80	All raised conflicts are resolved by the implemented functionalities.	Some raised conflicts are resolved.	Fewconflicts are resolved.	Very few or no conflicts are resolved by the implemented functionalities or no relevant functionalities are found
Error Handling	Managing the raised errors/bug/inconsistencies of the system	100	All inconsistencies are managed effectively.	Some inconsistencies are managed.	Fewinconsistencies are managed.	Very few inconsistencies are managed.
Documentation	Completeness and clarity of documentation	100	Documentation is complete, clear, and well-organized.	Documentation is satisfactorily complete but lacks clarity.	Documentation is poorly complete or lacks clarity.	Documentation is incomplete or missing.