National University of Computer & Emerging Sciences, Karachi
Fall-2025 School of Computing (BSCS, BSSE, BSCY, BSAI)

Assignment # 3

Subject: Programming Fundamentals CS-1002	**Post Date: 19/11/2025**
Total Marks: 60	**Due Date:** 30/11/2025

Course Instructors: Javeria Farooq, Sobia Iftikhar, Abeeha Sattar, Saif ur Rehman, Syed Ahmed, Faisal Ali, Fizza Mansoor, Talha Shahid, Abuzar Zafar, Uzma Raza

Instructions

- All Programs should be submitted in softcopy and should have .c file
- Plagiarism (copying/cheating) and late submissions result in a zero mark.It should be obvious that submitting your work after the due date will result in zero points being awarded.
- Plagiarism (copying/cheating) and late submissions result in a zero mark.

Question 1: A bank wants to calculate the total repayment amount for a loan using a recursive approach. The loan decreases each year after paying a fixed installment, and interest is applied annually to the remaining loan. Design a recursive function calculateRepayment(loan, interestRate, years) that prints the repayment schedule year by year in the format "Year X: Remaining loan = Y" and calculates the total repayment for n years. Identify the base case (loan fully repaid or no years remaining) and the recursive case (repayment for remaining years). Using this function, trace the repayment for a loan of 100,000 at 5% yearly interest over 3 years, and explain how recursion helps track the loan state across years. Extend the function logically to handle extra yearly payments and discuss why recursion is suitable for such incremental calculations.

Question 2: A space research organization is developing a simulation to track the fuel levels of a spacecraft as it travels between multiple planets. During its journey, the spacecraft consumes a fixed amount of fuel at each planetary stop due to thrust and atmospheric entry. However, when it passes through certain gravitational zones, it regenerates some fuel because of gravitational assists that reduce energy usage. Additionally, after every fourth planetary stop, the spacecraft undergoes a "solar recharge," where its solar panels absorb extra energy from nearby stars, increasing its fuel reserves. The mission control team wants to model this behavior using a recursive function calculateFuel(int fuel, int consumption, int recharge, int solarBonus, int planet, int totalPlanets) that prints the remaining fuel after each planet in the format "Planet X: Fuel Remaining = Y", and determines whether the spacecraft can complete its journey. The base case occurs when either the fuel is exhausted or all planets have been visited, and the recursive case handles fuel adjustments for each subsequent planet.

Question 3: ABC Technologies is a mid-sized software company that wants to build a simple system to manage its employees' records. The HR department wants to store and analyze employee data such as Employee ID, Name, Designation, and Salary. The company plans to use this system to make quick HR decisions like salary review, promotions, and employee lookup.

ABC Technologies needs a structure that can hold an employee's information. The HR team wants to enter data for multiple employees at once. Write a C Code to input records for n employees using a loop.

The HR manager wants to view all employee records in a readable format. Design a function displayEmployees() that takes an array of employee structures and displays all employee details neatly in a table-like format. The finance department needs to know who earns the highest salary to evaluate seniority and bonuses. Write a function findHighestSalary() that finds and displays the details of the employee with the highest salary. The HR wants to be able to search employees either by ID or by name.

Design a function searchEmployee() that can:

- Search by ID (exact match), or
- Search by Name (case-sensitive string comparison).

If the company later wants to give a 10% salary bonus to employees whose salary is below a certain threshold (say, 50,000), Explain how you could modify your existing program using a function with structure arrays passed by reference to update salaries directly.

Question 4: You are tasked with designing a **Library Shelf System** that can store a limited number of books. Each book has a **unique ID** and a **popularity score**. The shelf can only hold a fixed number of books (**capacity**), and the system must ensure that the **least recently accessed book is removed** when the shelf is full and a new book is added.

The system supports two operations:

1. **ADD x y** – Add a book with ID x and popularity score y.
   - If the book already exists on the shelf, update its popularity score.
   - If the shelf is full, remove the **least recently accessed book** before adding the new one.

2. **ACCESS x** – Access a book with ID x.
   - If the book exists on the shelf, return its popularity score.
   - If the book is not on the shelf, return -1.

Implement the shelf system in C using **structures and pointers** (arrays can be used, no advanced data structures like hash maps or linked lists required).

Input Format:

- First line: Two integers capacity and Q (number of operations).
- Next Q lines: Each line contains one operation:
  - ADD x y
  - ACCESS x

Output Format:

- For each ACCESS operation, print the popularity score or -1 if the book is not present.

Input	Ouptu
2 5 ADD 101 50 ADD 102 30 ACCESS 101 ADD 103 40 ACCESS 102	50 -1

Question 5:

Minimal Line-Based Text Editor Buffer

FAST University Karachi is currently working on a lightweight text-editing tool that can be used inside command-line environments for quick code notes, log entries, and configuration files. As part of the developer team, you are required to help build a minimal line-based text editor. This editor should allow users to type lines of text, store them efficiently, edit them, and save them for later use.

Because the editor must handle an unknown number of lines at runtime, you cannot rely on fixed-size arrays. Instead, the system should use Dynamic Memory Allocation (DMA) and pointers to grow or shrink the memory as needed. The editor will store each line separately, using a dynamic array of char* pointers so memory is only allocated for what the user actually types.

To begin, the editor will start with a small default capacity for storing lines. As the user adds more lines, the program must automatically expand the underlying structure using realloc. Each new line typed by the user should be stored in exactly the right amount of memory, no more, no less.

The editor must also support insertion and deletion of lines at any position. When inserting a new line, the program should shift pointer locations (not the whole strings) to make room. When deleting a line, the program should free its memory and compact the pointer array using memmove or a manual loop.

To optimize memory usage, the editor should include a shrinkToFit feature that reduces memory allocation to exactly match the number of stored lines. Additionally, the project must allow the user to save all lines to a file and load them later, rebuilding the dynamically allocated structure in memory.

Your program must handle long input lines safely, check all malloc/realloc results, and display proper error messages. Good comments and clear explanation of how pointer-based memory management reduces wastage compared to large static arrays are required.

In the end, the editor should be tested with different cases: many short lines and a few very long lines as stress-test scenarios for memory handling.

Task to Perform:

a. Implement a line-based text editor using dynamic memory allocation.

b. Use a dynamic array of char* pointers to store all lines.

c. Allocate an initial capacity using malloc, and grow the array using realloc.

d. For each new line, allocate exact-sized memory (strlen+1).

e. Implement functions:

    insertLine(index, text)

    deleteLine(index)

    printAllLines()

    freeAll()

f. Use char** to pass and update the lines array between functions.

g. When inserting, shift pointers—not the actual strings—to create space.

h. When deleting, free the string and compact the array using memmove or manual shifting.

i. Implement shrinkToFit using realloc to reduce unused capacity.

j. Add file support:

    i. Save the buffer to a file

    ii. Load from a file (rebuild all dynamic allocations)

k. Handle long input lines safely and check all memory allocation results.

l. Print error messages and gracefully exit when memory allocation fails.

m. Comment your code explaining why dynamic allocation is more efficient than fixed-size rows.

[Hint :a. In this program you need to include <error.h> file for error handling.

b. Read the functions and their concepts like malloc, realloc, free, char*, strlen, strcpy, memmove, fopen, fclose, fprintf, fgetc]

Question 6:

IEEE / ACM Membership Registration Manager

FAST University Karachi is developing a digital membership registration system for IEEE and ACM student chapters. This system will manage student registrations for the BS Computer Science, Software Engineering, Cyber Security, and Artificial Intelligence batches.

As a member of the developer team, you are required to build a program that can register students, update their information, and retrieve membership records, all while persisting data to a file using standard C file handling techniques.

Each student record should contain the following information:

1) Student ID (integer, unique)
2) Full Name (char[100])
3) Batch (string: CS, SE, Cyber Security, AI)
4) Membership Type (IEEE / ACM)
5) Registration Date (YYYY-MM-DD)
6) Date of Birth (YYYY-MM-DD)
7) Interest in (IEEE/ACM/Both)

System Behavior

The system should behave as follows:

1. All student records must be saved to a file (members.dat) so that data persists after program exit.
2. On program startup, the system should load existing records from the file.
3. After adding or updating records, the system must save the updated database.
4. The system should validate student ID uniqueness to prevent duplicate registrations.
5. It should be possible to generate batch-wise reports, for example, listing all AI students registered for IEEE or ACM or for Both.

File Handling:

1. Binary file operations: fopen, fwrite, fread, fclose for efficient storage.
2. Appending records to an existing file without overwriting previous data.
3. Reading all records to display reports or check for duplicates.
4. Updating records safely, either in-place or using a temporary file.
5. Deleting records by rewriting remaining records to a new file.
6. Error checking for all file operations to handle failures gracefully.

Tasks to Perform

1. Design the Student Record Struct

2. Implement File-Based Functions

You must create the following functions to manage persistent storage:

1. loadDatabase(const char *filename) to Load all student records from the file into memory.
2. saveDatabase(const char *filename) to Save all student records from memory to the file.
3. addStudent(Student s, const char *filename) to Append a new student record to the file.
4. updateStudent(int studentID) to Update a student's batch or membership type.
5. deleteStudent(int studentID) to Remove a student from the file.

3. Implement In-Memory Operations

While the program is running:

1. Maintain a dynamic array of Student records using malloc and realloc.
2. Check for duplicate student IDs before adding a new record.
3. Provide functions to display all students or generate batch-wise lists.

4. Build a Menu-Driven Interface

Your program should allow the user to:

1. Register a new student.
2. Update membership type or batch.
3. Delete a student registration.
4. View all registrations.
5. Generate batch-wise reports.
6. Exit the program (ensuring the database is saved).

5. Stress-Test File Handling

1. Add 20–30 student records across all batches.(Best use your class mates Data and Random Data)
2. Delete 5–10 random students.
3. Update 5 random records.
4. Verify that file integrity is maintained and no data is lost.

6. Implement Error Handling

1. Handle cases where the file cannot be opened or written.
2. Print meaningful error messages for the user.
3. Ensure there are no memory leaks in the dynamic arrays used for in-memory operations.