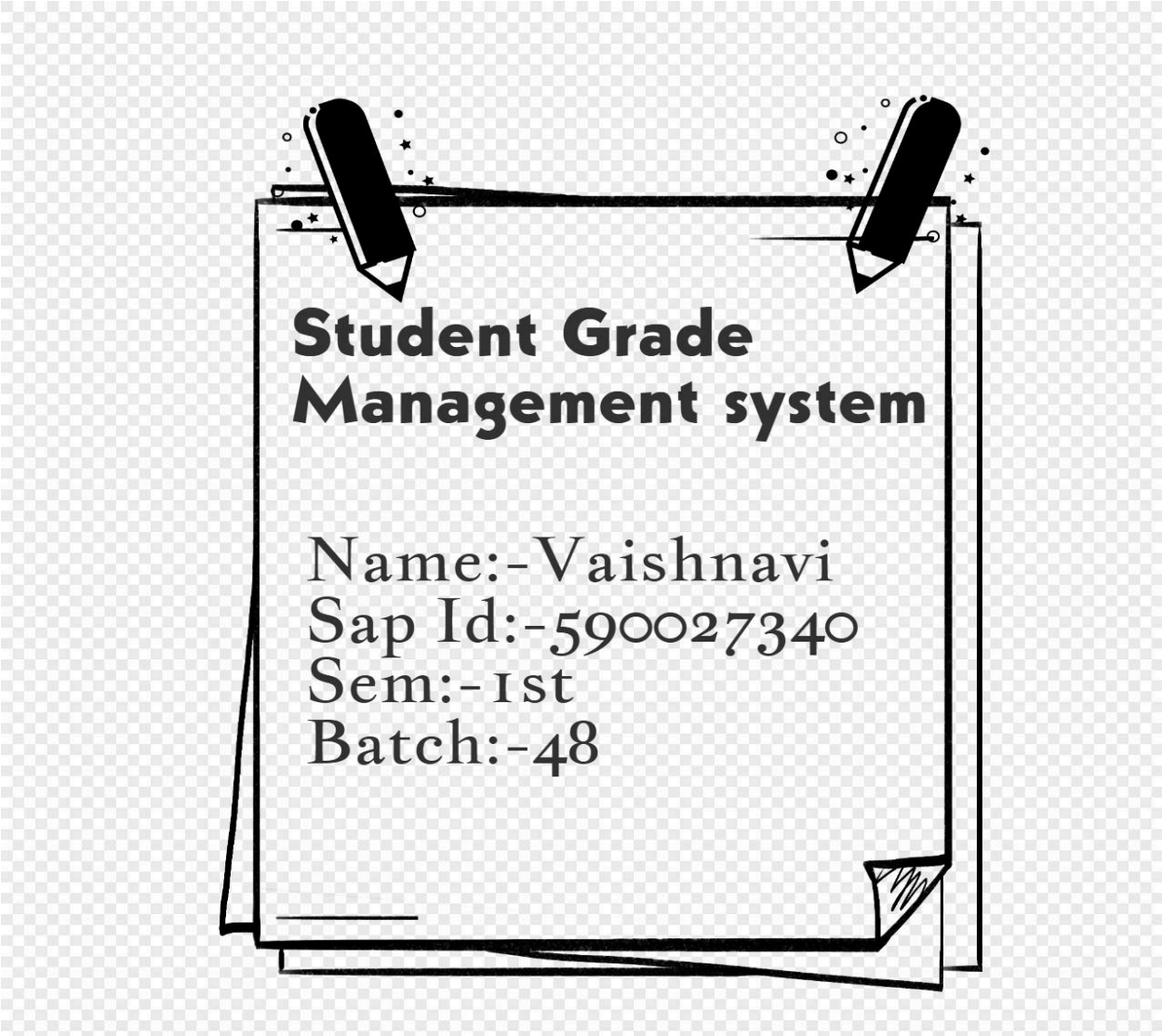
UPES

UNIVERSITY OF TOMORROW

---------- PROJECT of PROGRAMMING in C----------



Issue Description:

The project is about an efficient and robust way of management of student's academic performance data . Managing students academic records (subjects marks percentage,assigning grade) is a fundamental requirement of any educational institute . Performing these tasks manually (excel or other spreadsheets) is time-consuming and prone to human error **and lacks the structural integrity** needed for long-term data management.

The challenge addressed by this project is to make a single, reliable structure capable of accurately capturing this info and stores it accordingly.This system must overcome the limitations of flat-file processing by incorporating secure input handling and implementing a **safe method for record removal** that ensures the integrity and coherence of the remaining data set."

Objective of Project:

The main objective is to build a functional Student Grade Management system in C.

- **Data Persistence:** Implement File I/O to save and load student data permanently.
- **Validation:** Develop routines to validate all user inputs. This prevents bad data and program crashes.
- **Modularity:** Separate the code into `.h` and `.c` files. This enhances project readability and maintainability.
- **Memory Management:** Use `malloc` for dynamic memory allocation. This allows each student to have a variable number of subjects.
- **Reporting:** Accurately calculate percentages and assign letter grades.
- **Record Management:** Implement a **Read-Rewrite-Replace mechanism**. This achieves safe, permanent record deletion from the data file.

Software used:

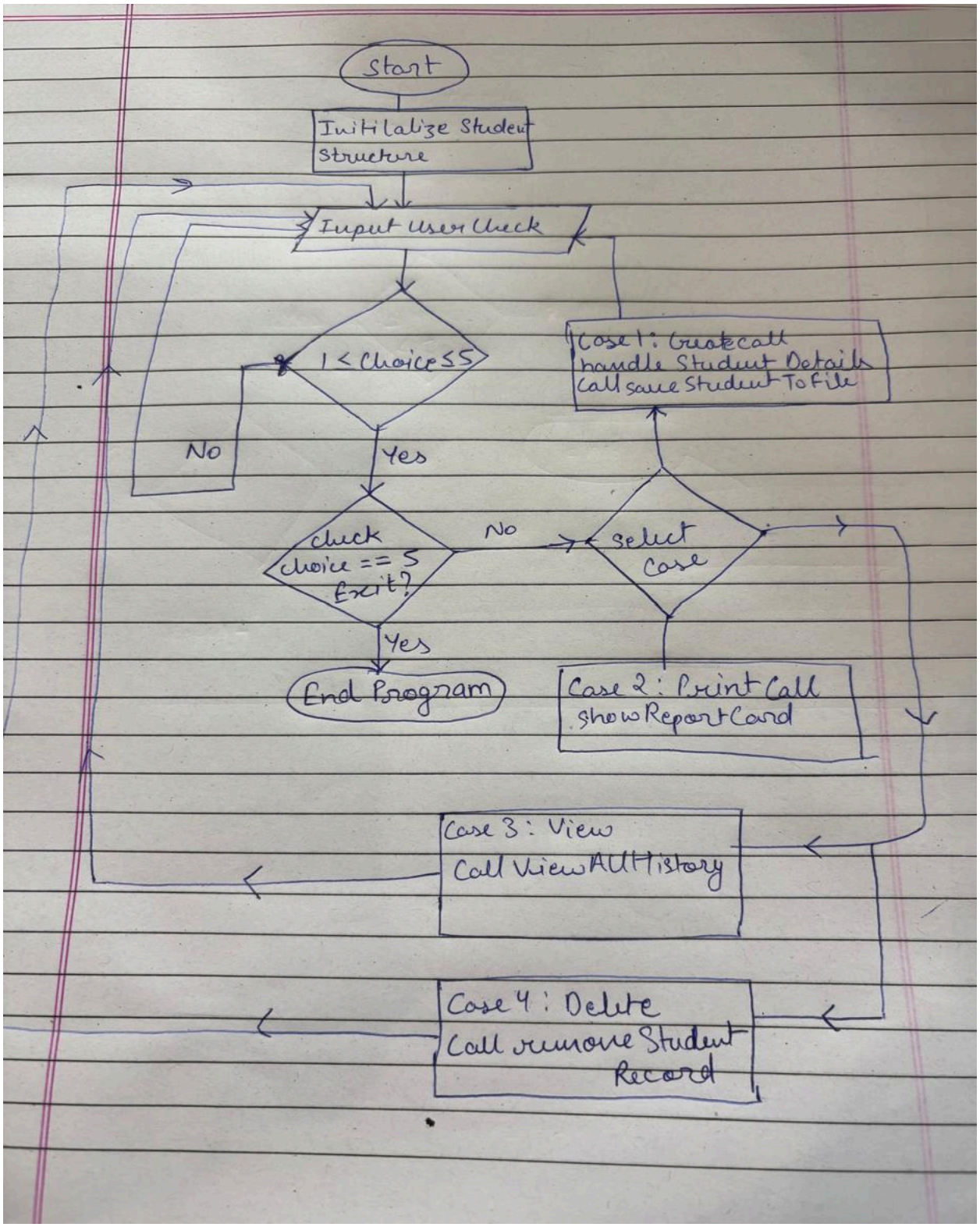
- Operating System: Microsoft Windows 10/11
- Integrated Development Environment (IDE)/Text Editor: Visual Studio Code (VS Code)
- Compiler Toolchain: MinGW (Minimalist GNU for Windows)
- Language - C (As it is project of C programming)

ALGORITHM:

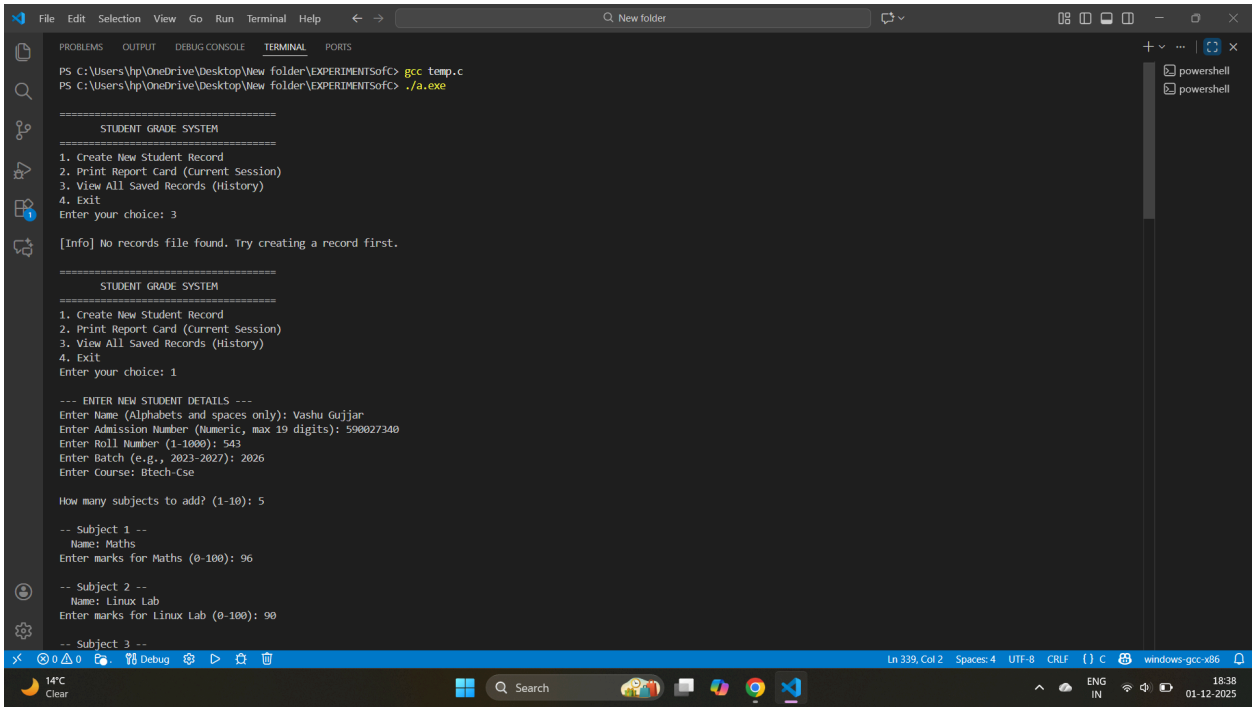
1. *Main Program Flow* :-

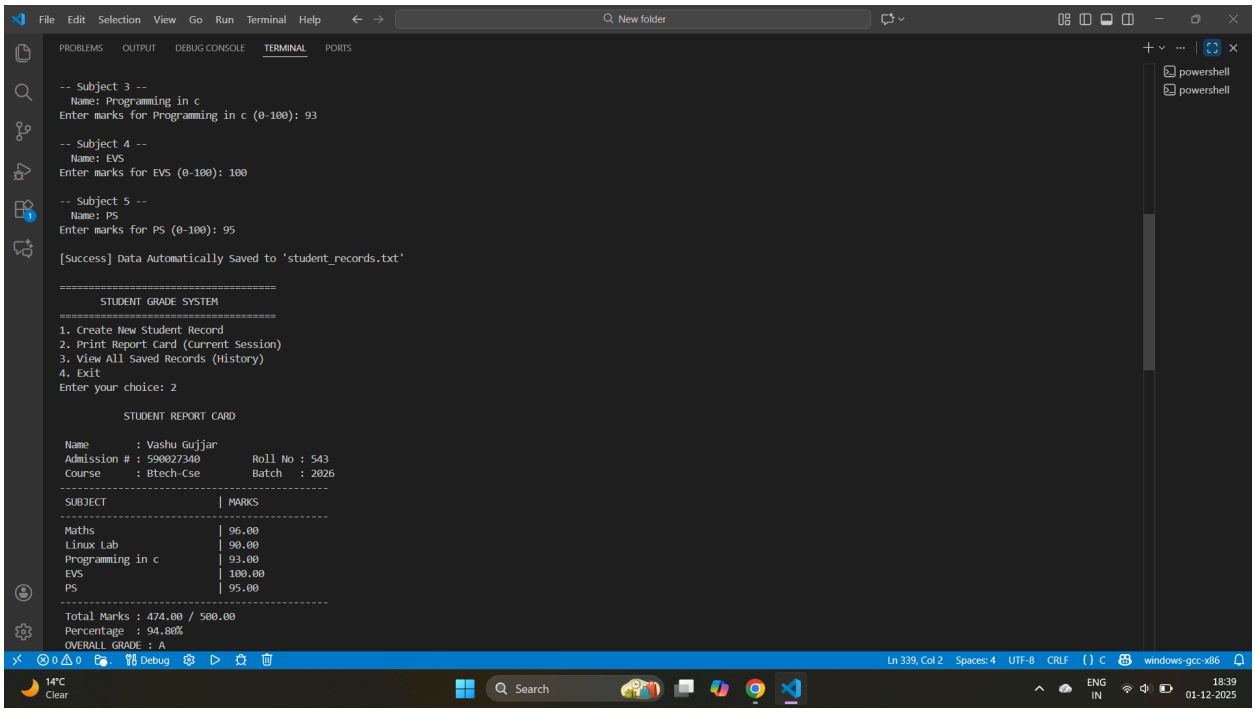
1. START: Initialize the `Student` structure (`std`) and set the dynamic pointer (`std.subjects`) to NULL.
2. Display Menu: Show the user the five main options (Create, Print, View History, Delete, Exit).
3. Get Input: Accept the user's choice (1-5).
4. Validate Choice: Check if the input is a valid number between 1 and 5. If invalid, display an error and return to Step 2.
5. Process Choice:
    - Case 1 (Create): Call `freeStudentMemory` (cleanup old data), call `handleStudentDetails` (get new input), and call `saveStudentToFile` (write to disk).
    - Case 2 (Print): If `std.subjects` is not NULL, call `showReportCard` to display the current in-memory record. Otherwise, show an error.
    - Case 3 (View History): Call `viewAllHistory` to display the entire contents of `student_records.txt`.
    - Case 4 (Delete): Call the `removeStudentRecord` function (detailed in Algorithm 2).
    - Case 5 (Exit): Proceed to Step 6.
6. Loop/Exit: If the choice was not 5, return to Step 2.
7. Final Cleanup: If `std.subjects` is not NULL, call `freeStudentMemory`.
8. END: Terminate the program.

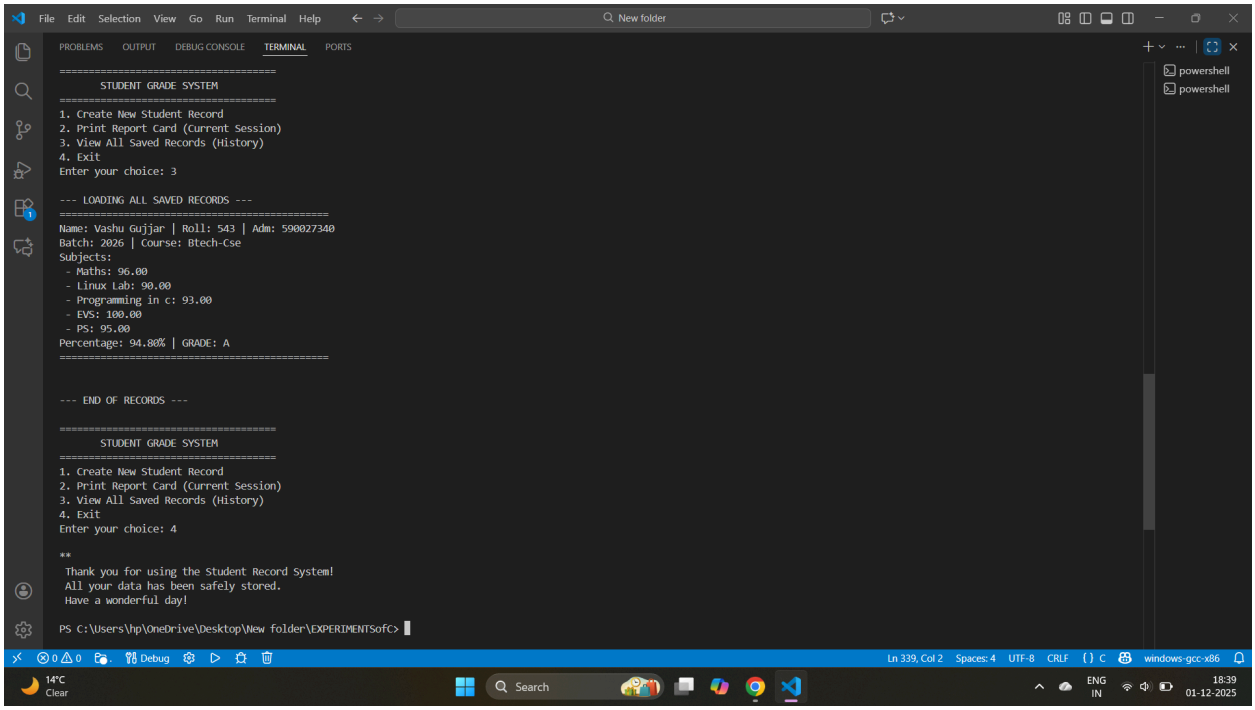
FLOWCHARTS:

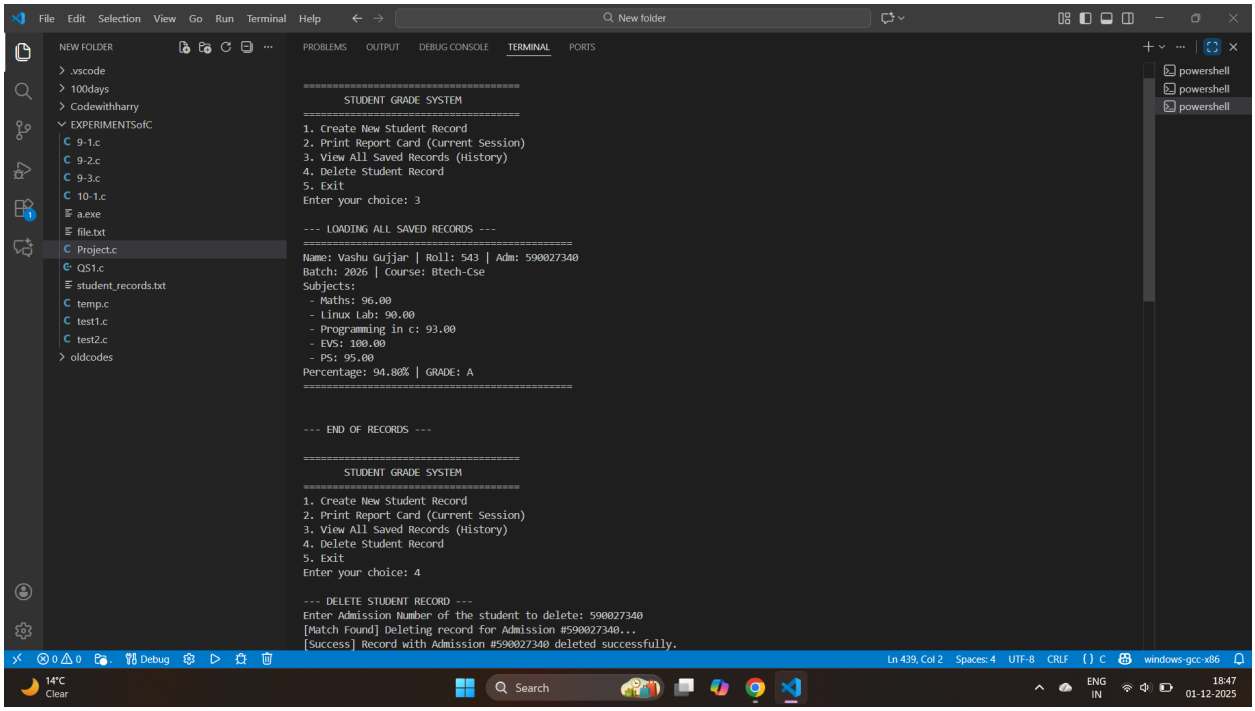

Start
Initialize Student Structure
Input User Choice
1 < Choice ≤ 5
No
Yes
Case 1: Create call handle Student Details Call save Student To file
check choice == 5 Exit?
No
select case
Yes
End Program
Case 2: Print Call show Report Card
Case 3: View Call View All History
Case 4: Delete Call remove Student Record

OUTPUTS:


File Edit Selection View Go Run Terminal Help
New folder
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\hp\OneDrive\Desktop\New folder\EXPERIMENTSofC> gcc temp.c
PS C:\Users\hp\OneDrive\Desktop\New folder\EXPERIMENTSofC> ./a.exe
STUDENT GRADE SYSTEM
1. Create New Student Record
2. Print Report Card (Current Session)
3. View All Saved Records (History)
4. Exit
Enter your choice: 3
[Info] No records file found. Try creating a record first.
STUDENT GRADE SYSTEM
1. Create New Student Record
2. Print Report Card (Current Session)
3. View All Saved Records (History)
4. Exit
Enter your choice: 1
--- ENTER NEW STUDENT DETAILS ---
Enter Name (Alphabets and spaces only): Vashu Gujjar
Enter Admission Number (Numeric, max 19 digits): 590027340
Enter Roll Number (1-1000): 543
Enter Batch (e.g., 2023-2027): 2026
Enter Course: Btech-Cse
How many subjects to add? (1-10): 5
-- Subject 1 --
Name: Maths
Enter marks for Maths (0-100): 96
-- Subject 2 --
Name: Linux Lab
Enter marks for Linux Lab (0-100): 90
-- Subject 3 --
powershell
powershell
Ln 339, Col 2 Spaces: 4 UTF-8 CRLF C windows-gcc-x86
14°C Clear
Search
ENG IN
18:38 01-12-2025


File Edit Selection View Go Run Terminal Help
New folder
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
-- Subject 3 --
Name: Programming in c
Enter marks for Programming in c (0-100): 93
-- Subject 4 --
Name: EVS
Enter marks for EVS (0-100): 100
-- Subject 5 --
Name: PS
Enter marks for PS (0-100): 95
[Success] Data Automatically Saved to 'student_records.txt'
STUDENT GRADE SYSTEM
1. Create New Student Record
2. Print Report Card (Current Session)
3. View All Saved Records (History)
4. Exit
Enter your choice: 2
STUDENT REPORT CARD
Name : Vashu Gujjar
Admission # : 590027340 Roll No : 543
Course : Btech-Cse Batch : 2026
SUBJECT | MARKS
Maths | 96.00
Linux Lab | 90.00
Programming in c | 93.00
EVS | 100.00
PS | 95.00
Total Marks : 474.00 / 500.00
Percentage : 94.80%
OVERALL GRADE : A
powershell
powershell
Ln 339, Col 2 Spaces: 4 UTF-8 CRLF C windows-gcc-x86
14°C Clear
Search
ENG IN
18:39 01-12-2025





SOURCE CODE:

```c
#include <stdio.h>
#include <string.h>
```

```c
#include <stdlib.h>
#include <ctype.h>

// --- Constants ---
#define MAX_NAME_LEN 50
#define MAX_ADM_LEN 20
#define FILE_NAME "student_records.txt"

// --- Structures ---

typedef struct {
    char sub[30];
    float marks;
} Subject;

typedef struct {
    char name[MAX_NAME_LEN];
    int roll_No;
    char adm_No[MAX_ADM_LEN];
    char batch[40];
    char course[20];
    int subcount;
    Subject *subjects;
} Student;

// --- Function Prototypes ---

// Utility/Validation
void cleanInputBuffer();
int checkNameValidity(const char *name);
int checkIfNumeric(const char *ptr);
float enterCorrectMarks(const char *subname);
char getGrade(float percentage);

// Main System Logic
void handleStudentDetails(Student *s);
void showReportCard(const Student *s);
void freeStudentMemory(Student *s);
void saveStudentToFile(const Student *s);
void viewAllHistory();
```

```
void removeStudentRecord();
```

```
int main() {
    // Student structure definition is now inherited from the header
    Student std;
    std.subjects = NULL;

    int choice;
    do {
        printf("\n=======================================\n");
        printf("       STUDENT GRADE SYSTEM          ");
        printf("\n=======================================\n");
        printf("1. Create New Student Record\n");
        printf("2. Print Report Card (Current Session)\n");
        printf("3. View All Saved Records (History)\n");
        printf("4. Delete Student Record\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");

        char choiceBuffer[10];
        if (fgets(choiceBuffer, sizeof(choiceBuffer), stdin) != NULL) {
            if (checkIfNumeric(choiceBuffer)) {
                choice = atoi(choiceBuffer);
            } else {
                choice = 0;
            }
        } else {
            choice = 5;
        }

        switch (choice) {
            case 1:
                if (std.subjects != NULL) {
                    printf("\n[Info] Clearing current memory for new
entry...\n");
                    freeStudentMemory(&std);
                }
                handleStudentDetails(&std);
```

```
                saveStudentToFile(&std);
                break;
            case 2:
                if (std.subjects != NULL) {
                    showReportCard(&std);
                } else {
                    printf("\nERROR, No record in memory to print\n");
                }
                break;
            case 3:
                viewAllHistory();
                break;
            case 4:
                removeStudentRecord();
                break;
            case 5:
                printf("\n**\n");
                printf(" Thank you for using the Student Record System!
\n");
                printf(" All your data has been safely stored.
\n");
                printf(" Have a wonderful day!
\n");
                printf("\n");
                freeStudentMemory(&std);
                break;
            default:
                printf("Invalid choice. Choose a number between 1 to
5\n");
        }
    } while (choice != 5);

    return 0;
}

// --- Utility/Validation Functions ---

void cleanInputBuffer() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
```

```
}

int checkNameValidity(const char *name) {
    if (strlen(name) == 0) return 0;
    for (int i = 0; name[i] != '\0'; i++) {
        if (!isalpha((unsigned char)name[i]) && !isspace((unsigned
char)name[i])) {
            return 0;
        }
    }
    return 1;
}

int checkIfNumeric(const char *ptr) {
    char temp[50];
    strncpy(temp, ptr, 50);
    temp[49] = '\0';

    temp[strcspn(temp, "\n")] = '\0';

    if (strlen(temp) == 0) return 0;

    for (int i = 0; temp[i] != '\0'; i++) {
        if (!isdigit((unsigned char)temp[i])) return 0;
    }
    return 1;
}

float enterCorrectMarks(const char *subname) {
    char buffer[50];
    float marks = 0.0f;
    int valid = 0;

    do {
        printf("Enter marks for %s (0-100): ", subname);
        if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
            buffer[strcspn(buffer, "\n")] = 0;

            int isFloat = 1;
            int dotCount = 0;
```

```
            int len = strlen(buffer);

            if (len == 0) isFloat = 0;

            for (int i = 0; i < len; i++) {
                if (!isdigit((unsigned char)buffer[i])) {
                    if (buffer[i] == '.' && dotCount == 0) {
                        dotCount++;
                    } else {
                        isFloat = 0;
                        break;
                    }
                }
            }

            if (isFloat) {
                marks = atof(buffer);
                if (marks >= 0.0f && marks <= 100.0f) {
                    valid = 1;
                } else {
                    printf("  [Error] Marks must be between 0 and
100.\n");
                }
            } else {
                printf("  [Error] Invalid input. Please enter numbers
only.\n");
            }
        } else {
             break;
        }
    } while (!valid);

    return marks;
}

char getGrade(float percentage) {
    if (percentage >= 90.0f) return 'A';
    else if (percentage >= 80.0f) return 'B';
    else if (percentage >= 70.0f) return 'C';
    else if (percentage >= 60.0f) return 'D';
```

```
    else if (percentage >= 50.0f) return 'E';
    else return 'F';
}

// --- Main System Logic Functions ---

void handleStudentDetails(Student *s) {
    char buffer[100];
    int valid = 0;

    printf("\n--- ENTER NEW STUDENT DETAILS ---\n");

    do {
        printf("Enter Name (Alphabets and spaces only): ");
        if (fgets(s->name, MAX_NAME_LEN, stdin) != NULL) {
            s->name[strcspn(s->name, "\n")] = 0;
            if (checkNameValidity(s->name)) valid = 1;
            else printf("Error: Name must only contain letters and
spaces.\n");
        } else break;
    } while (!valid);

    valid = 0;
    do {
        printf("Enter Admission Number (Numeric, max %d digits): ",
MAX_ADM_LEN - 1);
        if (fgets(s->adm_No, MAX_ADM_LEN, stdin) != NULL) {
            s->adm_No[strcspn(s->adm_No, "\n")] = 0;
            if (checkIfNumeric(s->adm_No)) valid = 1;
            else printf("Error: Admission Number must be numeric.\n");
        } else break;
    } while (!valid);

    valid = 0;
    do {
        printf("Enter Roll Number (1-1000): ");
        if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
            if (checkIfNumeric(buffer)) {
                s->roll_No = atoi(buffer);
                if (s->roll_No > 0 && s->roll_No <= 1000) valid = 1;
```

```
                else printf("Error: Roll Number must be in range
1-1000.\n");
            } else printf("Error: Invalid input. Digits only.\n");
        } else break;
    } while (!valid);

    printf("Enter Batch (e.g., 2023-2027): ");
    fgets(s->batch, 40, stdin);
    s->batch[strcspn(s->batch, "\n")] = 0;

    printf("Enter Course: ");
    fgets(s->course, 20, stdin);
    s->course[strcspn(s->course, "\n")] = 0;

    valid = 0;
    do {
        printf("\nHow many subjects to add? (1-10): ");
        if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
            if (checkIfNumeric(buffer)) {
                int count = atoi(buffer);
                if (count > 0 && count <= 10) {
                    s->subcount = count;
                    valid = 1;
                } else {
                    printf("Error: Subject count must be between 1 and
10.\n");
                }
            } else {
                printf("Error: Invalid input. Enter a number.\n");
            }
        } else break;
    } while (!valid);

    s->subjects = (Subject*)malloc(s->subcount * sizeof(Subject));
    if (s->subjects == NULL) {
        printf("Memory Allocation Failed!\n");
        exit(1);
    }

    for (int i = 0; i < s->subcount; i++) {
```

```
        printf("\n-- Subject %d --\n", i + 1);
        printf("  Name: ");
        fgets(s->subjects[i].sub, 30, stdin);
        s->subjects[i].sub[strcspn(s->subjects[i].sub, "\n")] = 0;

        s->subjects[i].marks = enterCorrectMarks(s->subjects[i].sub);
    }
}

void showReportCard(const Student *s) {
    printf("\n");
    printf("           STUDENT REPORT CARD           \n");
    printf("\n");
    printf(" Name        : %-30s\n", s->name);
    printf(" Admission # : %-15s   Roll No : %d\n", s->adm_No,
s->roll_No);
    printf(" Course      : %-15s   Batch   : %s\n", s->course, s->batch);
    printf("-----------------------------------------------\n");
    printf(" %-25s | %-10s \n", "SUBJECT", "MARKS");
    printf("-----------------------------------------------\n");

    float total = 0.0f;
    for (int i = 0; i < s->subcount; i++) {
        printf(" %-25s | %-10.2f \n",
               s->subjects[i].sub,
               s->subjects[i].marks);
        total += s->subjects[i].marks;
    }
    float percentage = (total / (s->subcount * 100.0f)) * 100.0f;
    char overallGrade = getGrade(percentage);

    printf("-----------------------------------------------\n");
    printf(" Total Marks : %.2f / %.2f\n", total, (float)s->subcount *
100.0f);
    printf(" Percentage  : %.2f%%\n", percentage);
    printf(" OVERALL GRADE : %c\n", overallGrade);
    printf("\n");
}

void saveStudentToFile(const Student *s) {
```

```
    FILE *fp = fopen(FILE_NAME, "a");

    if (fp == NULL) {
        printf("\n[Error] Could not open file for writing.\n");
        return;
    }
    float total = 0.0f;
    for (int i = 0; i < s->subcount; i++) total += s->subjects[i].marks;
    float percent = (total / ((float)s->subcount * 100.0f)) * 100.0f;
    char overallGrade = getGrade(percent);

    fprintf(fp, "================================================\n");
    fprintf(fp, "Name: %s | Roll: %d | Adm: %s\n", s->name, s->roll_No,
s->adm_No);
    fprintf(fp, "Batch: %s | Course: %s\n", s->batch, s->course);
    fprintf(fp, "Subjects:\n");
    for (int i = 0; i < s->subcount; i++) {
        fprintf(fp, " - %s: %.2f\n", s->subjects[i].sub,
s->subjects[i].marks);
    }
    fprintf(fp, "Percentage: %.2f%% | GRADE: %c\n", percent,
overallGrade);
    fprintf(fp, "================================================\n\n");

    fclose(fp);
    printf("\n[Success] Data Automatically Saved to '%s'\n", FILE_NAME);
}

void viewAllHistory() {
    FILE *fp = fopen(FILE_NAME, "r");
    int ch;

    if (fp == NULL) {
        printf("\n[Info] No records file found. Try creating a record
first.\n");
        return;
    }

    printf("\n--- LOADING ALL SAVED RECORDS ---\n");
    while ((ch = fgetc(fp)) != EOF) {
```

```
        printf("%c", ch);
    }
    printf("\n--- END OF RECORDS ---\n");

    fclose(fp);
}

void removeStudentRecord() {
    FILE *fp, *ft;
    char admToDelete[MAX_ADM_LEN];
    char buffer[512];
    int found = 0;

    printf("\n--- DELETE STUDENT RECORD ---\n");
    printf("Enter Admission Number of the student to delete: ");

    if (fgets(admToDelete, MAX_ADM_LEN, stdin) == NULL) return;
    admToDelete[strcspn(admToDelete, "\n")] = 0;

    if (strlen(admToDelete) == 0) {
        printf("[Error] Admission Number cannot be empty.\n");
        return;
    }

    fp = fopen(FILE_NAME, "r");
    if (fp == NULL) {
        printf("\n[Info] No records file found or error opening file.\n");
        return;
    }

    ft = fopen("temp_records.txt", "w");
    if (ft == NULL) {
        printf("[Error] Could not create temporary file.\n");
        fclose(fp);
        return;
    }

    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        if (strstr(buffer,
"==========================================") != NULL) {
```

```
            if (fgets(buffer, sizeof(buffer), fp) == NULL) break;

            char *admPtr = strstr(buffer, "Adm: ");

            if (admPtr != NULL) {
                admPtr += 5;
                char currentAdm[MAX_ADM_LEN];
                sscanf(admPtr, "%s", currentAdm);

                if (strcmp(currentAdm, admToDelete) == 0) {
                    found = 1;
                    printf("[Match Found] Deleting record for Admission
#%s...\n", admToDelete);
                    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
                        if (strstr(buffer,
"============================================") != NULL) {
                            break;
                        }
                    }
                    continue;
                }
            }

            fprintf(ft,
"============================================\n");
            fprintf(ft, "%s", buffer);

        } else {
            fprintf(ft, "%s", buffer);
        }
    }

    fclose(fp);
    fclose(ft);

    if (found) {
        if (remove(FILE_NAME) != 0) {
            printf("[Fatal Error] Could not delete original file.\n");
            return;
```

```
        }

        if (rename("temp_records.txt", FILE_NAME) != 0) {
            printf("[Fatal Error] Could not rename temporary file.\n");
            return;
        }
        printf("[Success] Record with Admission #%s deleted
successfully.\n", admToDelete);
    } else {
        remove("temp_records.txt");
        printf("[Info] No record found with Admission #%s.\n",
admToDelete);
    }
}

void freeStudentMemory(Student *s) {
    if (s->subjects != NULL) {
        free(s->subjects);
        s->subjects = NULL;
    }
}
```

Conclusion:

This project demonstrate permanent data storage via file I/O and efficient dynamic memory allocation.The system successfully implement Read-Rewrite-Replace algorithm for secure record deletion,ensuring file integrity.
The modular code structure (using .h and .c files) makes the system easily maintainable.

Future Enhancement:

The project can be improved by adding these key features:

1. **Record Modification (Update):** Implement a function to **change existing student details or marks** (the missing "Update" feature).
2. **Weighted Grading:** Enhance the grading logic to use **multiple weighted components** (like assignments and exams) to calculate the final subject marks.
3. **Data Retrieval:** Add functions to **search for specific student records** and to **sort the entire history by final percentage**.