TITLE

Student Marklist Report: A Menu-Driven C Application for Adding, Sorting, and Searching Student Records

1) Introduction

This project is a simple, menu-driven C program that helps maintain a small in-memory list of students and their marks. It supports three core operations:

1. Add Student – capture a student's name and marks.
2. Sort by Marks – display all students sorted by their marks in ascending order.
3. Search by Name – look up a particular student by exact name match and display their marks.

The program uses a fixed-size array to store records and demonstrates foundational programming concepts: structures, arrays, loops, decision-making, functions, and basic algorithms (Bubble Sort and Linear Search)

---

2) Project Team Members

- P. Venu Gopal
- B. Ganesh
- Jai Manikanta
- K. Sriram
- Vasu
- Hemanth

---

3) Global Variables

- Structure Definition: `struct Student` holds `name[50]` and `int marks`.
- Global Array: `struct Student students[MAX];` — fixed capacity list.
- Global Counter: `int count;` — tracks current number of students.

---

4) Module Breakdown

1. Input Module (`addStudent`)
    - Reads `name` (no spaces) and `marks`, appends to array.
2. Sorting Module (`sortByMarks`)
    - Sorts records in ascending order using Bubble Sort (stable for equal marks).
3. Searching Module (`searchByName`)
    - Performs Linear Search over stored names using `strcmp`.
4. UI/Driver Module (`main loop`)
    - Menu rendering, user choice handling, and function dispatch.

5) Algorithms Used

5.1 Bubble Sort (Ascending by marks)

- Idea: Repeatedly swap adjacent out-of-order elements. Stable when using > comparison only.
- Pseudocode:

```
Run
Copy code
for i from 0 to count-1:
    if strcmp(students[i].name, searchName) == 0:
        return i
return -1
```

- Time Complexity: O(n) worst; Space: O(1).

5.2 Linear Search (Exact name match)

- Idea: Scan each record; compare `name` via `strcmp` until match found.
- Pseudocode:

```
Run
Copy code
for i from 0 to count-1:
    if strcmp(students[i].name, searchName) == 0:
        return i
```

- `return -1`
- Time Complexity: O(n) worst; Space: O(1).

---

6) Detailed Module Descriptions

6.1 `addStudent()`

- Responsibility: Append a new Student to the list if capacity allows.
- Inputs: `name` (no spaces), `marks` (int).
- Outputs: Console confirmation or error if full.
- Edge Cases: `count == MAX` (list full).

6.2 `sortByMarks()`

- Responsibility: Sort `students[0..count-1]` by marks ascending.
- Inputs: None (uses global `students`).
- Outputs: Sorted list printed to console.
- Edge Cases: `count == 0` (prints header; no records).

6.3 `searchByName()`

- Responsibility: Locate a student by exact name.
- Inputs: `searchName` (no spaces).
- Outputs: Found record or “not found” message.
- Edge Cases: Empty list; name not present; case sensitivity (`Ravi ≠ ravi`).

6.4 `main()`

- Responsibility: Display menu, read user's numeric choice, call corresponding functions, repeat until Exit.
- Inputs: User's numeric choice.
- Outputs: Operation results or an "Invalid choice" message.

---

7) Sample Inputs & Outputs (Console Transcripts)

7.1 Adding Students

Run

Copy code

```
--- Student Mark Sheet Analyzer --

1. Add Student

2. Sort Students by Marks

3. Search Student by Name

4. Exit

Enter your choice: 1

--- Add Student --
```

```
Enter Name (no spaces): Ravi

Enter Marks: 85

Student added successfully.
```

7.2 Sorting by Marks

```
Run

Copy code

Enter your choice: 2

--- Students Sorted by Marks --

Name: Anya, Marks: 72

Name: Ravi, Marks: 85

Name: Sam, Marks: 92
```

7.3 Searching by Name (Found)

```
Run

Copy code

Enter your choice: 3

Enter student name to search: Ravi
```

```
--- Student Found --

Name: Ravi

Marks: 85
```

7.4 Searching by Name (Not Found)

```
Run

Copy code

Enter your choice: 3

Enter student name to search: Zara

No student found with name Zara.
```

7.5 Max Capacity Reached

```
Run

Copy code

... (after adding 100 students) ...

Enter your choice: 1

--- Add Student --

Enter Name (no spaces): TestUser
```

```
Enter Marks: 50
```

```
Student list is full!
```

---

8) Test Cases

TC#	Precondition	Input	Expected Output
1	Empty list	Sort	Prints header; no records; no crash
2	Empty list	Search name Ravi	No student found with name Ravi.
3	Space available	Add: Ravi 85	Student added successfully.; count=1
4	Some records exist	Add: Anya 72, Sam 92	Records appended; count increases
5	3 records	Sort	Order: Anya (72), Ravi (85), Sam (92)
6	Records with equal marks	Add: Zara 92 then Sort	Sam remains before Zara (stable ties)
7	Records exist	Search exact Ravi	Displays Ravi’s marks

8	Records exist	Search ravi (lowercase)	Not found (case-sensitive)
9	MAX=100	Add until 100, then add one more	Student list is full!
10	Invalid menu Choice	9	Invalid choice. Try again.
11	Name with spaces	Try "Mary Jane"	Only Mary read; document limitation
12	Marks out of range	Add marks -5 or 150	Accepted by code; note validation enhancement

---

9) Limitations (Current Code)

- Names cannot contain spaces (scanf("%s")).
- No input validation on marks (negative or >100 allowed).
- Exact, case-sensitive name matching only.
- Volatile storage: data lost on program exit (no file/database).
- Fixed capacity also there

---

10) Future Enhancements

- Allow spaces in names (use `fgets` or `scanf("%49[^\n]s", name)` and trim).
- Validate marks (e.g., enforce 0–100; reject invalid input safely).

- Persistent storage (file I/O: save/load CSV or binary; or SQLite).
- Update/Delete operations; prevent duplicate names if desired.
- Multiple subjects, totals, averages, grades, ranks, toppers.
- Sort options (descending, by name alphabetically) and stable sort guarantee.
- Search improvements (case-insensitive, prefix search, binary search after sorting).
- Dynamic sizing (use `malloc/realloc` or a linked list).
- User experience (clearer menus, input prompts, error messages).

---

11) Conclusion

The Student Marklist Report project demonstrates core C programming skills using structures, arrays, and foundational algorithms. It provides a functional baseline for managing a small set of student records with add, sort, and search operations. While intentionally simple, the code offers a strong platform for learning and for future extension into a robust mark management system with validation, persistence, richer queries, and enhanced user experience.

---

12) (Optional) Suggested Code Hygiene Tweaks

- Maintain consistent indentation and brace style.
- Consider checking return values to detect invalid inputs.
- Report the number of records when printing sorted lists.
- Encapsulate globals behind accessors or convert to a simple module.

- If you want, we can also provide an enhanced version of the code with input validation, names with spaces, file save/load, and case-insensitive search.

13) Program Code

```
#include <stdio.h>
#include <string.h>
#define MAX 100

struct Student {
    char name[50];
    int marks;
};

struct Student students[MAX];
int count = 0;

void addStudent() {
    if (count >= MAX) {
        printf("Student list is full!\n");
        return;
    }
    printf("---Add Student---\n");
    printf("Enter Name: ");
    scanf("%s", students[count].name);
    printf("Enter Marks: ");
    scanf("%d", &students[count].marks);
    count++;
    printf("Student added successfully.\n");
}

void sortByMarks() {
    for (int i = 0; i < count-1; i++) {
        for (int j = 0; j < count-i-1; j++) {
            if (students[j].marks > students[j+1].marks) {
                struct Student temp = students[j];
                students[j] = students[j+1];
                students[j+1] = temp;
            }
        }
    }
```

```
    printf("---Students Sorted by Marks---\n");
    for (int i = 0; i < count; i++) {
        printf("Name: %s, Marks: %d\n", students[i].name, students[i].marks);
    }
}

void searchByName() {
    char searchName[50];
    printf("Enter name to search: ");
    scanf("%s", searchName);
    for (int i = 0; i < count; i++) {
        if (strcmp(students[i].name, searchName) == 0) {
            printf("---Student Found---\n");
            printf("Name: %s\nMarks: %d\n",
                  students[i].name, students[i].marks);
            return;
        }
    }
    printf("Student not found!\n");
}

int main() {
    int choice;
    while(1) {
        printf("\n--- Student Marklist System ---\n");
        printf("1. Add Student\n");
        printf("2. Sort by Marks\n");
        printf("3. Search by Name\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: addStudent(); break;
            case 2: sortByMarks(); break;
            case 3: searchByName(); break;
            case 4: return 0;
            default: printf("Invalid choice!\n");
        }
    }
}
```