NBKR Institute of Science and Technology

Project report

Project:-Online Exam Rank List

Department:-Computer Science Engineering

Section:-E

Year:-I

Semester:-II

Date:-May 01,2025

Submitted to:-Ashok Selva Kumar

Submitted by:-

1)SK.Rahill (24KB1A05JD)
2)SK.Mohammad faizil(24KB1A05HV)
3)S.Anil (24KB1A05KG)
4)D.Ravi Kumar (24KB1A05KX)

ACKNOWLEDGEMENT:-

I would like to express my sincere gratitude to all those who have supported and guided me throughout the development of this project.

- **[Instructor's Name]** – For providing invaluable guidance and support during the development of this project. Their expertise and encouragement were instrumental in shaping the project's success.
- **[Classmates' Names]** – For their collaborative efforts, constructive feedback, and assistance in various stages of the project. Their teamwork contributed significantly to the project's completion.
- **[Family Members' Names]** – For their unwavering support, understanding, and encouragement throughout this endeavor. Their belief in me provided the motivation to overcome challenges.
- **[Online Communities/Forums]** – For offering valuable insights and solutions to technical queries, enhancing the quality and efficiency of the project.

Certainly! Here's an abstract tailored for your C program that manages student records and sorts them by score in descending order:

Abstract:-

This C program implements a student record management system using structures to encapsulate student data, including their names and scores. The program allows for the input of multiple student records, stores them in an array of structures, and then sorts the records based on student scores in descending order using the bubble sort algorithm.

Key Features:

- **Student Structure:** Defines a structure to hold each student's name and score.
- **Dynamic Input:** Prompts the user to enter the number of students and their respective details.
- **Sorting Mechanism:** Employs the bubble sort algorithm to arrange student records in descending order based on scores.
- **Output Display:** Displays the sorted list of students along with their ranks and scores.

Algorithm Overview:

1. **Input Phase:** The program first asks the user to input the number of students and then iteratively collects each student's name and score.
2. **Sorting Phase:** Using the bubble sort algorithm, the program compares adjacent student scores and swaps their positions if they are in the wrong order. This process continues until the entire list is sorted.

3. **Output Phase:** After sorting, the program prints the students' names and scores in descending order, along with their respective ranks.

Introduction:-

This C program demonstrates the use of structures to manage student data, specifically their names and scores. It allows users to input multiple student records, stores them in an array of structures, and then sorts these records in descending order based on the students' scores using the bubble sort algorithm.

Key Features:

- **Student Structure:** Defines a structure to hold each student's name and score.
- **Dynamic Input:** Prompts the user to enter the number of students and their respective details.
- **Sorting Mechanism:** Employs the bubble sort algorithm to arrange student records in descending order based on scores.
- **Output Display:** Displays the sorted list of students along with their ranks and scores.

Usage:

Upon execution, the program will:

1. Request the number of students.
2. Allow input of each student's name and score.
3. Sort the student records in descending order of scores.
4. Display the sorted list with ranks, names, and scores.

This program serves as a foundational example for handling and processing structured data in C, showcasing the practical application of structures and sorting algorithms.

Objectives:-

1. **Data Organization Using Structures**
   Utilize a struct to encapsulate student information, including their name and score, allowing for efficient storage and retrieval of data.
2. **Dynamic Input Handling**
   Prompt the user to input the number of students and their respective details, ensuring flexibility to handle varying numbers of records.

3. **Sorting Algorithm Implementation**
   Implement the bubble sort algorithm to arrange student records in descending order based on scores, facilitating the generation of a rank list.
4. **Rank List Generation**
   Display the sorted list of students along with their ranks and scores, providing a clear and ordered presentation of the data.
5. **User Interaction and Output**
   Ensure the program interacts effectively with the user, providing prompts for input and displaying results in a user-friendly format.

SYSTEM REQUIREMENTS:-

Software:-Turbo C++,Dev C++

Hardware:-Minimum ram,i3,processor

Methodology:-

This program employs fundamental concepts of C programming, including structures, arrays, and sorting algorithms, to manage and display student records efficiently. The methodology is structured as follows:

1. Data Representation Using Structures

- **Structure Definition**: A Student structure is defined to encapsulate each student's information, specifically their name (a string of up to 50 characters) and score (a floating-point number representing the student's marks).
- typedef struct {
- char name[50];
- float score;
- } Student;

2. Dynamic Data Input

- **User Interaction**: The program prompts the user to input the number of students (n) and then iteratively collects each student's name and score.
- for (int i = 0; i < n; i++) {
- printf("Enter name of student %d: ", i + 1);
- scanf("%s", students[i].name);
- printf("Enter score of %s: ", students[i].name);
- scanf("%f", &students[i].score);
- }

3. Sorting Mechanism

- **Bubble Sort Algorithm**: The program implements the bubble sort algorithm to arrange the student records in descending order based on their scores. The algorithm compares adjacent elements and swaps them if they are in the incorrect order, repeatedly passing through the list until no swaps are needed.
- void sortDescending(Student students[], int n) {
- Student temp;
- for (int i = 0; i < n - 1; i++) {
- for (int j = i + 1; j < n; j++) {
- if (students[i].score < students[j].score) {
- temp = students[i];
- students[i] = students[j];
- students[j] = temp;
- }
- }
- }
- }

Note: Bubble sort is a simple comparison-based algorithm with a time complexity of $O(n^2)$, making it suitable for small datasets or educational purposes .

4. Displaying the Sorted List

- **Output Format**: After sorting, the program displays the student records in descending order of scores, assigning ranks accordingly.
- printf("\n--- Rank List ---\n");
- for (int i = 0; i < n; i++) {
- printf("Rank %d: %s - %.2f\n", i + 1, students[i].name, students[i].score);
- }

Project Description:-

Overview:

This project implements a Student Ranking System in C, utilizing structures to manage student data and sorting algorithms to rank students based on their scores. The system allows users to input student details, processes the data, and outputs a ranked list of students.

Key Features:

- **Student Data Representation:** Utilizes a Student structure to encapsulate each student's name and score.
- **Dynamic Input Handling:** Prompts the user to input the number of students and their respective details.
- **Sorting Mechanism:** Employs the bubble sort algorithm to arrange student records in descending order based on scores.
- **Ranked Output Display:** Displays the sorted list of students along with their ranks and scores.

Methodology:

1. **Data Structure Definition:** Define a Student structure with fields for the student's name and score.
2. **Input Collection:** Prompt the user to enter the number of students and then collect each student's name and score.
3. **Sorting Algorithm:** Implement the bubble sort algorithm to sort the student records in descending order based on scores.
4. **Output Display:** After sorting, display the ranked list of students with their names and scores.

Sample Output:

Enter the number of students: 3

Enter name of student 1: Alice

Enter score of Alice: 85

Enter name of student 2: Bob

Enter score of Bob: 90

Enter name of student 3: Charlie

Enter score of Charlie: 80

--- Rank List ---

Rank 1: Bob - 90.00

Rank 2: Alice - 85.00

Rank 3: Charlie - 80.00

Conclusion:

This project demonstrates the application of structures and sorting algorithms in C to manage and process student data efficiently. It serves as a foundational example for handling and organizing structured data in programming.

Algorithm Steps:-

Step:-Start
step2:- Input Student Data

- Prompt the user to enter the number of students (n).
- For each student (from 1 to n):

- Input the student's name.
- Input the student's score.

Step3:-Sorting Students by Score (Descending Order)

- Implement the Bubble Sort algorithm to sort the students array:
  - For each student i from 0 to n-2:
    - For each student j from i+1 to n-1:
      - If students[i].score < students[j].score:
        - Swap students[i] and students[j].

This sorting ensures that students with higher scores appear first in the list.

Step4:-Display Sorted Rank List

- Print the header --- Rank List ---.
- For each student i from 0 to n-1:
  - Print the rank (i+1), student's name, and score.

🔁 Bubble Sort Algorithm (Descending Order)

The Bubble Sort algorithm works by repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order. The pass through the list is repeated until the list is sorted

Steps:

1. Compare each pair of adjacent elements.
2. If the current element is smaller than the next, swap them.
3. Repeat the process until no swaps are needed.

Time Complexity:

- Best Case: O(n) (when the list is already sorted)
- Average and Worst Case: $O(n^2)$

Space Complexity:

- O(1) (in-place sorting)

✅ Example Walkthrough

Given the following student data:

Name	Score
Alice	85
Bob	92
Charlie	78
Diana	88

After Sorting:

Rank	Name	Score
1	Bob	92
2	Diana	88
3	Alice	85
4	Charlie	78

This sorting ensures that the highest scorer is ranked first, followed by the others in descending order of their scores.

PROGRAM CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 70       //it is used for max marks should students can be secured

// Define the Student structure
typedef struct {
   char name[50];
   float score;
} Student;

// the main logic or function part to the code  based on arrangement of students in descending order
void sortDescending(Student students[], int n) {
   Student temp;
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (students[i].score < students[j].score) {
            // Swapping of their prespective marks theobubble sort
            temp = students[i];
            students[i] = students[j];
            students[j] = temp;
         }
      }
   }
}

int main() {
   Student students[MAX];
   int n;

   printf("Enter the number of students: ");
```

```
    scanf("%d", &n);

    // wants to give the data of students
    for (int i = 0; i < n; i++) {
        printf("Enter name of student %d: ", i + 1);
        scanf("%s", students[i].name);
        printf("Enter score of %s: ", students[i].name);
        scanf("%f", &students[i].score);
    }

    // Sort the students based on their perspective ranks in descending order.
    sortDescending(students, n);

    // the atudents list of their perspective ranks
    printf("\n--- Rank List ---\n");
    for (int i = 0; i < n; i++) {
        printf("Rank %d: %s - %.2f\n", i + 1, students[i].name, students[i].score);
    }

    return 0;
}
```

OUTPUT:-

```
Enter the number of students: 5
Enter name of student 1: ashok
Enter score of ashok: 55
Enter name of student 2: ambati
Enter score of ambati: 54
Enter name of student 3: bhaskar
Enter score of bhaskar: 43
Enter name of student 4: hamid
Enter score of hamid: 32
Enter name of student 5: mohammad
Enter score of mohammad: 67

--- Rank List ---
Rank 1: mohammad - 67.00
Rank 2: ashok - 55.00
Rank 3: ambati - 54.00
Rank 4: bhaskar - 43.00
Rank 5: hamid - 32.00
```

Testing and Validation :-

To ensure the correctness and robustness of the student rank sorting program, it's essential to conduct comprehensive testing. This involves verifying the program's functionality under various scenarios, including edge cases and performance considerations.

1. Functional Testing
Objective: Verify that the program correctly sorts students based on their scores in descending order and displays the rank list accurately.
Test Cases:

Test Case No.	Input Data	Expected Output	Remarks
TC1	3 students: Alice (85.5), Bob (90.0), Charlie (78.0)	Rank 1: Bob - 90.00Rank 2: Alice - 85.50Rank 3: Charlie - 78.00	Standard case with distinct scores
TC2	2 students: Alice (90.0), Bob (90.0)	Rank 1: Alice - 90.00Rank 2: Bob - 90.00	Case with identical scores
TC3	4 students: Alice (95.0), Bob (85.0), Charlie (95.0), Dave (80.0)	Rank 1: Alice - 95.00Rank 2: Charlie - 95.00Rank 3: Bob - 85.00Rank 4: Dave - 80.00	Handling of ties in scores
TC4	1 student: Alice (88.0)	Rank 1: Alice - 88.00	Single student scenario
TC5	5 students: Alice (70.0), Bob (85.0), Charlie (90.0), Dave (75.0), Eve (80.0)	Rank 1: Charlie - 90.00Rank 2: Bob - 85.00Rank 3: Eve - 80.00Rank 4: Dave - 75.00Rank 5: Alice - 70.00	Multiple students with distinct scores

Procedure:
1. Input the number of students.
2. Enter each student's name and score.
3. Verify that the output matches the expected rank list.

Expected Outcome: The program should display the students in descending order of their scores, with appropriate handling of ties.

2. Boundary Testing
Objective: Ensure the program handles edge cases effectively.
Test Cases:

Test Case No.	Input Data	Expected Output	Remarks
TC6	0 students	Error message: "No students entered."	Handling of zero input
TC7	70 students with varying scores	Correct rank list for 70 students	Maximum input size
TC8	70 students with identical scores	All students ranked equally	Handling of identical scores

Procedure:
1. Test with zero students.
2. Test with the maximum number of students (70).
3. Test with all students having identical scores.

Expected Outcome: The program should handle these scenarios gracefully, providing appropriate messages or outputs.

3. Performance Testing
Objective: Assess the program's efficiency with a large number of students.
Test Case:

Test Case No.	Input Data	Expected Outcome	Remarks
TC9	70 students with random scores	Program executes within acceptable time limits	Performance under load

Procedure:
1. Input 70 students with randomly generated scores.
2. Measure the execution time.

Expected Outcome: The program should execute within a reasonable time frame, demonstrating its efficiency.

---

4. Usability Testing
Objective: Evaluate the user-friendliness of the program.
Test Case:

Test Case No.	Input Data	Expected Outcome	Remarks
TC10	User inputs data as prompted	Clear prompts and understandable outputs	User interface clarity

Procedure:
1. Interact with the program as an end user.
2. Assess the clarity of prompts and outputs.

Expected Outcome: The program should be intuitive and easy to use.

5. Code Review and Static Analysis
Objective: Ensure code quality and adherence to best practices.
Focus Areas:
- **Code Structure:** Clear and modular code with appropriate comments.
- **Variable Naming:** Descriptive and consistent naming conventions.
- **Error Handling:** Proper handling of invalid inputs and edge cases.
- **Memory Management:** Efficient use of memory, especially when handling large inputs.

Expected Outcome: The code should be well-structured, maintainable, and free of obvious errors.

Conclusion:
By conducting these tests, we can verify that the student rank sorting program functions correctly under various scenarios, handles edge cases effectively, performs efficiently, and provides a user-friendly experience.

LIMITATIONS:-

Limitations of the Given Code:

1. **Fixed Maximum Array Size (MAX = 70)**:
    - The program uses a fixed maximum number of students (MAX = 70). This is a limitation because the program cannot handle more than 70 students. If a user wants to input more than 70 students, the program will not be able to accommodate them, unless the maximum size is manually changed in the code.
2. **Input Constraints**:
    - The program assumes that students' names will not exceed 50 characters. If a student's name exceeds this length, it could lead to a buffer overflow or truncation of the name. This is a limitation in the scanf call for name input.
    - No input validation is performed for the number of students (n) or the scores. For example, the program would fail if the user enters a non-numeric value for the score.
3. **No Error Handling**:
    - The program doesn't handle any invalid inputs (e.g., negative scores, non-numeric input for scores). It would crash or give incorrect results if the input is invalid.
    - It also doesn't check if the number of students is within the allowed limit (MAX). If the user enters a number greater than MAX, it could result in undefined behavior.
4. **No Memory Allocation for Dynamic Input**:
    - The program uses static arrays (Student students[MAX]), which is fine for small, fixed-size datasets. However, for larger datasets or more flexible input, dynamic memory allocation (e.g., using malloc or calloc) could be more efficient and scalable.
5. **Bubble Sort Inefficiency**:
    - The sorting algorithm used is bubble sort, which has a time complexity of $O(n^2)$. For larger datasets, this sorting method is inefficient. More efficient algorithms like quicksort or mergesort (O(n log n)) would improve performance.
6. **No Handling of Equal Scores**:
    - While the program sorts students based on scores in descending order, it doesn't handle the case where multiple students have the same score. In such cases, the program simply lists them in the order they were entered, which may not be the most logical or desired behavior.

FUTURE ENHANCEMENT:-

Here are some possible future enhancements for the code you provided:

1. Dynamic Memory Allocation:

- Instead of using a fixed size array (MAX = 70), you can allow the user to enter the number of students dynamically, and allocate memory based on that input using malloc or calloc. This would make the program more flexible and scalable for larger numbers of students.

2. Improved Sorting Algorithm:

- **Quicksort or Mergesort**: Replace the inefficient Bubble Sort with a more efficient algorithm such as **Quicksort** or **Mergesort**. These algorithms have average time complexities of O(n log n), which would improve performance especially when handling large datasets.

Example of Mergesort or Quicksort would be more efficient than Bubble Sort.

3. Error Handling and Input Validation:

- Implement better input validation for scores, names, and the number of students. Ensure that the input is valid (e.g., positive scores, proper name lengths) to prevent crashes or incorrect data entry.
- If a user enters a non-numeric value for score or an invalid number of students, the program should prompt the user to re-enter the data.

4. Support for More Flexible Score Formats:

- Allow users to input scores with varying precision (e.g., 3 decimal places), and format the output accordingly.
- You can use %.2f in printf for formatting, but this can be made more flexible by accepting a precision value as input from the user.

5. Ability to Modify or Delete Data:

- After entering the student data, provide the option to modify or delete a student's information before sorting. This feature would make the program more user-friendly.

6. Better Output Formatting:

- Improve the output formatting by adding features like column alignment, and headers. For example, the output could be displayed in a table-like format:
- Rank | Name | Score
- ----------
- 1 | John | 95.00
- 2 | Sarah | 92.50

Conclusion:-

The provided C code successfully implements a system for ranking students based on their scores. Here's a conclusion summarizing its functionality and potential improvements:

Conclusion:

1. **Functionality**:
    - The program allows users to input student names and scores.
    - It sorts the students in descending order based on their scores using a basic bubble sort algorithm.
    - It displays the ranked list of students with their respective scores.

2. **Key Features**:
    - The program uses a Student structure to store names and scores.
    - It applies a sorting algorithm (Bubble Sort) to order students by their scores.
    - The program handles input for the number of students and each student's information.
    - The result is printed in a clear, ranked format.
3. **Limitations**:
    - The code uses Bubble Sort, which is inefficient for large datasets ($O(n^2)$ time complexity). Optimizing the sorting algorithm (e.g., using Merge Sort or Quick Sort) would be beneficial for better performance.
    - The program doesn't handle edge cases like invalid input (e.g., non-numeric scores or incorrect number of students).
    - It uses static memory allocation (MAX size), which can be improved with dynamic memory allocation for more flexibility.

REFERENCES:-

Here are some references that can help you understand and enhance the code:

1. **C Programming - Arrays and Structures**:
    - Learn about arrays and structures in C programming to better understand how data is handled, including how structures like Student are used.
2. **Sorting Algorithms (Bubble Sort)**:
    - The bubble sort algorithm is used in this code to sort the students by their scores. You can explore more about bubble sort and other sorting algorithms to optimize the program.
3. **C Programming Input and Output**:
    - The program uses basic I/O functions such as scanf and printf. Learning about I/O operations in C will help you understand how to handle user input and output in your program.
4. **C Memory Management (Static vs Dynamic Memory)**:
    - The program uses static memory allocation with the MAX constant. To further enhance it, you could switch to dynamic memory allocation (e.g., using malloc) to allow for flexible handling of different numbers of students.
5. **Improving Sorting Efficiency**:

- Bubble sort is not the most efficient sorting algorithm. You can replace it with more efficient algorithms like quicksort or merge sort.

6. **C Error Handling and Input Validation**:
    - Error handling and input validation are important in any program to avoid unexpected behavior. You can improve the program by adding error checks for user input.