DSA Lab Assignment

Linked List

Date: 21-08-2023

NAME : Pratyush Parth

ROLL No. : 122AD0050

BRANCH : Artificial Intelligence & Data Science

Q1. Create a Linked List of Students with following parameters:

a. Name

b. Roll Number

c. 5 Marks of the student

d. Average

PSEUDOCODE :

```
1.Define a structure named "std" with fields: name (string), rln (integer), marks (array of integers), avg (float), and next (pointer to std structure)

2.Define a global integer variable "c" and initialize it to 0

3.Function createstd(name, rln, marks):
   3.1 Allocate memory for a new "std" structure named "newstd"
   3.2If allocation fails, print an error message and exit
   3.3 Copy the "name" to newstd's name field
   3.4 Set newstd's rln to the provided rln
   3.5 Copy the elements of the "marks" array to newstd's marks array
   3.6 Calculate the sum of marks and store it in "sum"
   3.7 Calculate the average by dividing "sum" by 5 and store it in newstd's avg field
   3.8 Set newstd's next pointer to NULL
   3.9 Return newstd

4.Function displaystd(std):
    4.1 Print "Name:", std's name
```

```
 4.2 Print "Roll Number:", std's rln
 4.3 Print "Marks:", std's marks array
 4.4 Print "avg:", std's avg
 4.5 Print "--------------------"
5. Function addstd(head, newstd):
 5.1 If head is NULL:
     Set head to newstd
 5.2 Else:
     Initialize a pointer "current" to head
 5.3 Iterate until current's next pointer is NULL:
     Move current to the next std structure
 5.4 Set current's next pointer to newstd
6. Function main():
 6.1 Initialize a pointer "head" to NULL
 6.2 Add students using addstd and createstd functions, with name, roll number, and marks
 6.3 Initialize a pointer "current" to head
 6.4 Iterate while current is not NULL:
     Call displaystd with the current std structure
     Move current to the next std structure
 6.5 Return 0
7. Call the main function to start the program
```

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
struct Students
{
    char name[50];
    int rollno;
    int marks[5];
    float avg;
    struct Students* next;
};
float calculateAverage(int marks[])
{
    int sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum += marks[i];
    }
    return (float)sum / 5;
}
void append(struct Students** headRef, struct Students* newStudent)
{
    if (*headRef == NULL)
    {
        *headRef = newStudent;
    }
    else
    {
        struct Students* current = *headRef;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = newStudent;
    }
}
void display(struct Students* head)
{
    struct Students* current = head;
    while (current != NULL) {
        printf("Name: %s\n", current->name);
        printf("Roll Number: %d\n", current->rollno);
        printf("Marks: ");
        for (int i = 0; i < 5; i++)
        {
            printf("%d ", current->marks[i]);
        }
        printf("\nAverage: %.2f\n", current->avg);
        printf("----------\n");
        current = current->next;
    }
}
int main()
{
    struct Students* head = NULL;
    struct Students* student1 = (struct Students*)malloc(sizeof(struct
Students));
    printf("Details of student: \n");
    strcpy(student1->name, "ABCDE");
    student1->rollno = 12345;
    student1->marks[0] = 75;
    student1->marks[1] = 98;
    student1->marks[2] = 76;
    student1->marks[3] = 86;
    student1->marks[4] = 82;
    student1->avg = calculateAverage(student1->marks);
    student1->next = NULL;
    append(&head, student1);
    display(head);
    free(student1);
    return 0;
}
```

OUTPUT :

```
Details of student:

Name: ABCDE

Roll Number: 12345

Marks: 75 98 76 86 82

Average: 83.40

----------
```

Q2. Do the following operations on the Linked List

a. Adding Data of students from Roll number 1 to 25 (All the data for a student)

PSEUDOCODE :

```
1.struct Student
   1.1char name[50]
   1.2int roll_number
   1.3int marks[5]
   1.4float average
   1.5struct Student* next
2.function createStudent(name, roll_number, marks)
   2.1allocate memory for newStudent
   2.2if newStudent is NULL
      print "Memory allocation failed."
      exit program
   2.3copy name into newStudent->name
   2.4set newStudent->roll_number to roll_number
   2.5for i = 0 to 4
      set newStudent->marks[i] to marks[i]
   2.6initialize sum to 0
   2.7for i = 0 to 4
      add marks[i] to sum
   2.8set newStudent->average to sum / 5
   2.9set newStudent->next to NULL
   2.10return newStudent
3.function displayStudent(student)
   3.1print "Name:", student->name
   3.2print "Roll Number:", student->roll_number
   3.3print "Marks:",
    3.4for i = 0 to 4
        print student->marks[i]
    3.5print "Average:", student->average
    3.6print "--------------------"
4.function lengthoflist(head)
    4.1set c to 0
    4.2set ptr to head
    4.3while ptr is not NULL
        increment c by 1
        if c is 1 or 4 or 23 or 25
            call displayStudent(ptr)
        set ptr to ptr->next
5.function appendStudent(head, newStudent)
    5.1if head is NULL
        set head to newStudent
    5.2else
        set current to head
        while current->next is not NULL
            set current to current->next
        set current->next to newStudent
6.function main()
    6.1set head to NULL
    6.2call appendStudent(head, createStudent("A", 1, {85, 90, 78, 92, 88}))
    6.3call appendStudent(head, createStudent("B", 2, {92, 88, 76, 85, 90}))
    ...
```

6.4call appendStudent(head, createStudent("Y", 25, {85, 90, 78, 92, 88}))

6.5set current to head

6.6call lengthoflist(current)

6.7return 0

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
int c=0;
struct std
{
    char name[50];
    int rln;
    int marks[5];
    float avg;
    struct std* next;
};
struct std* createstd(char name[], int rln, int marks[])
{
    struct std* newstd = (struct std*)malloc(sizeof(struct std));
    if (newstd == NULL)
    {
        printf("Memory allocation failed.");
        exit(1);
    }
    strcpy(newstd->name, name);
    newstd->rln = rln;
    for (int i = 0; i < 5; i++)
    {
        newstd->marks[i] = marks[i];
    }
    float sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum += marks[i];
    }
    newstd->avg = sum / 5;
    newstd->next = NULL;
    return newstd;
}
void displaystd(struct std* std)
{
    printf("Name: %s\n", std->name);
    printf("Roll Number: %d\n", std->rln);
    printf("Marks: ");
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", std->marks[i]);
    }
    printf("\n");
    printf("avg: %.2f\n", std->avg);
    printf("--------------------\n");
}
void addstd(struct std** head, struct std* newstd)
{
    if (*head == NULL)
    {
        *head = newstd;
    }
    else
    {
        struct std* current = *head;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = newstd;
    }
}

int main()
{
    struct std* head = NULL;
    addstd(&head, createstd("A", 1, (int[]){85, 90, 78, 92, 88}));
    addstd(&head, createstd("B", 2, (int[]){92, 88, 76, 85, 90}));
    addstd(&head, createstd("C", 3, (int[]){78, 92, 88, 76, 85}));
    addstd(&head, createstd("D", 4, (int[]){85, 90, 76, 92, 88}));
    addstd(&head, createstd("E", 5, (int[]){92, 82, 76, 85, 90}));
    addstd(&head, createstd("F", 6, (int[]){78, 90, 88, 76, 85}));
    addstd(&head, createstd("G", 7, (int[]){85, 99, 78, 92, 88}));
    addstd(&head, createstd("H", 8, (int[]){92, 100, 76, 85, 90}));
    addstd(&head, createstd("I", 9, (int[]){78, 92, 78, 76, 85}));
    addstd(&head, createstd("J", 10, (int[]){85, 60, 78, 92, 88}));
    addstd(&head, createstd("K", 11, (int[]){92, 88, 56, 85, 90}));
    addstd(&head, createstd("L", 12, (int[]){78, 92, 55, 76, 85}));
    addstd(&head, createstd("M", 13, (int[]){85, 90, 23, 92, 88}));
    addstd(&head, createstd("N", 14, (int[]){92, 88, 99, 85, 90}));
    addstd(&head, createstd("O", 15, (int[]){78, 92, 97, 76, 85}));
    addstd(&head, createstd("P", 16, (int[]){85, 91, 78, 92, 88}));
    addstd(&head, createstd("Q", 17, (int[]){92, 88, 34, 85, 90}));
    addstd(&head, createstd("R", 18, (int[]){78, 92, 8, 76, 85}));
    addstd(&head, createstd("S", 19, (int[]){85, 90, 18, 92, 88}));
    addstd(&head, createstd("T", 20, (int[]){92, 88, 16, 85, 90}));
    addstd(&head, createstd("U", 21, (int[]){78, 92, 80, 76, 85}));
    addstd(&head, createstd("V", 22, (int[]){85, 90, 68, 92, 88}));
    addstd(&head, createstd("W", 23, (int[]){92, 88, 66, 85, 90}));
    addstd(&head, createstd("X", 24, (int[]){78, 52, 88, 76, 85}));
    addstd(&head, createstd("Y", 25, (int[]){85, 90, 48, 92, 88}));
    struct std* current = head;
    while (current != NULL)
    {
        displaystd(current);
        current = current->next;
    }
    return 0;
}
```

OUTPUT :

```
Name: A

Roll Number: 1

Marks: 85 90 78 92 88

avg: 86.60

--------------------

Name: B

Roll Number: 2

Marks: 92 88 76 85 90

avg: 86.20

--------------------

Name: C

Roll Number: 3

Marks: 78 92 88 76 85
```

```
avg: 83.80

--------------------

Name: D

Roll Number: 4

Marks: 85 90 76 92 88

avg: 86.20

--------------------

Name: E

Roll Number: 5

Marks: 92 82 76 85 90

avg: 85.00

--------------------

Name: F

Roll Number: 6

Marks: 78 90 88 76 85

avg: 83.40

--------------------

Name: G

Roll Number: 7

Marks: 85 99 78 92 88

avg: 88.40

--------------------

Name: H
```

```
Roll Number: 8

Marks: 92 100 76 85 90

avg: 88.60

--------------------

Name: I

Roll Number: 9

Marks: 78 92 78 76 85

avg: 81.80

--------------------

Name: J

Roll Number: 10

Marks: 85 60 78 92 88

avg: 80.60

--------------------

Name: K

Roll Number: 11

Marks: 92 88 56 85 90

avg: 82.20

--------------------

Name: L

Roll Number: 12

Marks: 78 92 55 76 85

avg: 77.20
```

```
--------------------
Name: M
Roll Number: 13
Marks: 85 90 23 92 88
avg: 75.60
--------------------
Name: N
Roll Number: 14
Marks: 92 88 99 85 90
avg: 90.80
--------------------
Name: O
Roll Number: 15
Marks: 78 92 97 76 85
avg: 85.60
--------------------
Name: P
Roll Number: 16
Marks: 85 91 78 92 88
avg: 86.80
--------------------
Name: Q
Roll Number: 17
```

```
Marks: 92 88 34 85 90

avg: 77.80

--------------------

Name: R

Roll Number: 18

Marks: 78 92 8 76 85

avg: 67.80

--------------------

Name: S

Roll Number: 19

Marks: 85 90 18 92 88

avg: 74.60

--------------------

Name: T

Roll Number: 20

--------------------

Name: X

Roll Number: 24

Marks: 78 52 88 76 85

avg: 75.80

--------------------

Name: Y

Roll Number: 25
```

```
Marks: 85 90 48 92 88

avg: 80.60

--------------------
```

b. Searching and printing the details of students 1, 4, 23, 25

PSEUDOCODE :

```
1.struct Student
   1.1char name[50]
   1.2int roll_number
   1.3int marks[5]
   1.4float average
   1.5struct Student* next
2.function createStudent(name, roll_number, marks)
   2.1allocate memory for newStudent
   2.2if newStudent is NULL
      print "Memory allocation failed."
      exit program
   2.3copy name into newStudent->name
   2.4set newStudent->roll_number to roll_number
   2.5for i = 0 to 4
      set newStudent->marks[i] to marks[i]
   2.6initialize sum to 0
   2.7for i = 0 to 4
```

```
        add marks[i] to sum
    2.8set newStudent->average to sum / 5
    2.9set newStudent->next to NULL
    2.10return newStudent
3.function displayStudent(student)
    3.1print "Name:", student->name
    3.2print "Roll Number:", student->roll_number
    3.3print "Marks:",
    3.4for i = 0 to 4
        print student->marks[i]
    3.5print "Average:", student->average
    3.6print "--------------------"
4.function lengthoflist(head)
    4.1set c to 0
    4.2set ptr to head
    4.3while ptr is not NULL
        increment c by 1
        if c is 1 or 4 or 23 or 25
            call displayStudent(ptr)
        set ptr to ptr->next
5.function appendStudent(head, newStudent)
    5.1if head is NULL
        set head to newStudent
    5.2else
        set current to head
        while current->next is not NULL
            set current to current->next
        set current->next to newStudent
6.function main()
    6.1set head to NULL
    6.2call appendStudent(head, createStudent("A", 1, {85, 90, 78, 92, 88}))
    6.3call appendStudent(head, createStudent("B", 2, {92, 88, 76, 85, 90}))
    ...
    6.4call appendStudent(head, createStudent("Y", 25, {85, 90, 78, 92,
88}))
    6.5set current to head
    6.6call lengthoflist(current)
    6.7return 0
```

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
int c=0;
struct Student
{
    char name[50];
    int roll_number;
    int marks[5];
    float average;
    struct Student* next;
};
struct Student* createStudent(char name[], int roll_number, int marks[])
{
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct
Student));
    if (newStudent == NULL)
    {
        printf("Memory allocation failed.");
        exit(1);
    }
    strcpy(newStudent->name, name);
    newStudent->roll_number = roll_number;
    for (int i = 0; i < 5; i++)
    {
        newStudent->marks[i] = marks[i];
    }
    float sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum += marks[i];
    }
    newStudent->average = sum / 5;
    newStudent->next = NULL;
    return newStudent;
}
void displayStudent(struct Student* student)
{
    printf("Name: %s\n", student->name);
    printf("Roll Number: %d\n", student->roll_number);
    printf("Marks: ");
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", student->marks[i]);
    }
    printf("\n");
    printf("Average: %.2f\n", student->average);
    printf("--------------------\n");
}
void lengthoflist(struct Student*head)
{
    struct Student*ptr=head;
    while(ptr!=NULL)
    {
        c++;
        if(c==1||c==4||c==23||c==25)
        {
            displayStudent(ptr);
        }
        ptr=ptr->next;
    }
}
```

```c
void appendStudent(struct Student** head, struct Student* newStudent)
{
    if (*head == NULL)
    {
        *head = newStudent;
    }
    else
    {
        struct Student* current = *head;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = newStudent;
    }
}
int main()
{
    struct Student* head = NULL;
    appendStudent(&head, createStudent("A", 1, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("B", 2, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("C", 3, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("D", 4, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("E", 5, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("F", 6, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("G", 7, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("H", 8, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("I", 9, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("J", 10, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("K", 11, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("L", 12, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("M", 13, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("N", 14, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("O", 15, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("P", 16, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("Q", 17, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("R", 18, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("S", 19, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("T", 20, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("U", 21, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("V", 22, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("W", 23, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("X", 24, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("Y", 25, (int[]){85, 90, 78, 92, 88}));
    struct Student* current = head;
    lengthoflist(current);
    return 0;
}
```

OUTPUT :

```
Name: A

Roll Number: 1

Marks: 85 90 78 92 88

Average: 86.60

--------------------

Name: D

Roll Number: 4

Marks: 85 90 78 92 88

Average: 86.60

--------------------

Name: W

Roll Number: 23

Marks: 92 88 76 85 90

Average: 86.20

--------------------

Name: Y

Roll Number: 25

Marks: 85 90 78 92 88

Average: 86.60

--------------------
```

c. Sort the Linked list of students based on marks without using arrays.

PSEUDOCODE :

```
1.Define structure Student:
   1.1name: string
   1.2roll_number: integer
   1.3marks: array of integers (size 5)
   1.4average: float
   1.5next: pointer to Student
2.Define global variable c as integer and initialize to 0
3.Function createStudent(name, roll_number, marks[]):
   3.1Allocate memory for newStudent
   3.2 If newStudent is NULL, print "Memory allocation failed." and exit
   3.3Copy name to newStudent->name
   3.4Assign roll_number to newStudent->roll_number
   3.5Copy marks to newStudent->marks
   3.6Calculate sum of marks and assign average to newStudent->average
   3.7Set newStudent->next to NULL
   3.8Return newStudent
4.Function freeStudents(head):
   4.1Set current to head
   4.2While current is not NULL:
      Set temp to current
      Set current to current->next
      Free memory of temp
5.Function displayStudent(student):
   5.1Print student's name, roll_number, marks, average
   5.2Print "--------------------"
6.Function lengthoflist(head):
   6.1Set ptr to head
   6.2While ptr is not NULL:
      Increment c
      Display student details using displayStudent(ptr)
      If c is 1, 4, 23, or 25, continue to next iteration
      Set ptr to ptr->next
7.Function appendStudent(head, newStudent):
   7.1If head is NULL:
      Set head to newStudent
   7.2Else:
      Set current to head
      While current->next is not NULL:
         Set current to current->next
      Set current->next to newStudent
8.Function Sort(head):
   8.1Set swapped to 1
   8.2Set lptr to NULL
   8.3While swapped is 1:
      Set swapped to 0
      Set ptr1 to head
      While ptr1->next is not lptr:
         If ptr1->average > ptr1->next->average:
            Swap roll_number, name, marks, and average between ptr1 and ptr1->next
            Set swapped to 1
         Set ptr1 to ptr1->next
      Set lptr to ptr1
9.Function main():
   9.1Set head to NULL
   9.2Append students to the list using appendStudent and createStudent functions
   9.3Sort the list using Sort function
   9.4Set current to head
   9.5While current is not NULL:
      Display student details using displayStudent(current)
      Set current to current->next
   9.6Free memory of students using freeStudents function
   9.7Return 0
```

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int c=0;
struct Student
{
    char name[50];
    int roll_number;
    int marks[5];
    float average;
    struct Student* next;
};
struct Student* createStudent(char name[], int roll_number, int marks[])
{
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct
Student));
```

```c
    if (newStudent == NULL)
    {
        printf("Memory allocation failed.");
        exit(1);
    }
    strcpy(newStudent->name, name);
    newStudent->roll_number = roll_number;
    for (int i = 0; i < 5; i++)
    {
        newStudent->marks[i] = marks[i];
    }
    float sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum += marks[i];
    }
    newStudent->average = sum / 5;
    newStudent->next = NULL;
    return newStudent;
}
void freeStudents(struct Student* head)
{
    struct Student* current = head;
    while (current != NULL)
    {
        struct Student* temp = current;
        current = current->next;
        free(temp);
    }
}
void displayStudent(struct Student* student)
{
    printf("Name: %s\n", student->name);
    printf("Roll Number: %d\n", student->roll_number);
    printf("Marks: ");
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", student->marks[i]);
    }
    printf("\n");
    printf("Average: %.2f\n", student->average);
    printf("--------------------\n");
}
void lengthoflist(struct Student*head)
{
    struct Student*ptr=head;
    while(ptr!=NULL)
    {
        c++;
        displayStudent(ptr);
        if(c==1||c==4||c==23||c==25)
        {
           continue;
        }
        ptr=ptr->next;
    }
}
void appendStudent(struct Student** head, struct Student* newStudent)
{
    if (*head == NULL)
    {
        *head = newStudent;
    }
    else
    {
        struct Student* current = *head;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = newStudent;
    }
}
void Sort(struct Student* head)
{
    int swapped;
    struct Student* ptr1;
    struct Student* lptr = NULL;
    if (head == NULL)
        return;
    do
    {
        swapped = 0;
        ptr1 = head;
        while (ptr1->next != lptr)
        {
            if (ptr1->average > ptr1->next->average)
            {

                int temp_roll = ptr1->roll_number;
                ptr1->roll_number = ptr1->next->roll_number;
                ptr1->next->roll_number = temp_roll;
                char temp_name[50];
                strcpy(temp_name, ptr1->name);
                strcpy(ptr1->name, ptr1->next->name);
                strcpy(ptr1->next->name, temp_name);
                for (int i = 0; i < 5; i++)
                {
                    int temp_mark = ptr1->marks[i];
                    ptr1->marks[i] = ptr1->next->marks[i];
                    ptr1->next->marks[i] = temp_mark;
                }
                float temp_avg = ptr1->average;
                ptr1->average = ptr1->next->average;
                ptr1->next->average = temp_avg;
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}
int main()
{
    struct Student* head = NULL;
    appendStudent(&head, createStudent("A", 1, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("B", 2, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("C", 3, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("D", 4, (int[]){85, 90, 76, 92, 88}));
    appendStudent(&head, createStudent("E", 5, (int[]){92, 82, 76, 85, 90}));
    appendStudent(&head, createStudent("F", 6, (int[]){78, 90, 88, 76, 85}));
    appendStudent(&head, createStudent("G", 7, (int[]){85, 99, 78, 92, 88}));
    appendStudent(&head, createStudent("H", 8, (int[]){92, 100, 76, 85, 90}));
    appendStudent(&head, createStudent("I", 9, (int[]){78, 92, 78, 76, 85}));
    appendStudent(&head, createStudent("J", 10, (int[]){85, 60, 78, 92, 88}));
    appendStudent(&head, createStudent("K", 11, (int[]){92, 88, 56, 85, 90}));
    appendStudent(&head, createStudent("L", 12, (int[]){78, 92, 55, 76, 85}));
    appendStudent(&head, createStudent("M", 13, (int[]){85, 90, 23, 92, 88}));
    appendStudent(&head, createStudent("N", 14, (int[]){92, 88, 99, 85, 90}));
    appendStudent(&head, createStudent("O", 15, (int[]){78, 92, 97, 76, 85}));
    appendStudent(&head, createStudent("P", 16, (int[]){85, 91, 78, 92, 88}));
    appendStudent(&head, createStudent("Q", 17, (int[]){92, 88, 34, 85, 90}));
    appendStudent(&head, createStudent("R", 18, (int[]){78, 92, 8, 76, 85}));
    appendStudent(&head, createStudent("S", 19, (int[]){85, 90, 18, 92, 88}));
    appendStudent(&head, createStudent("T", 20, (int[]){92, 88, 16, 85, 90}));
    appendStudent(&head, createStudent("U", 21, (int[]){78, 92, 80, 76, 85}));
    appendStudent(&head, createStudent("V", 22, (int[]){85, 90, 68, 92, 88}));
    appendStudent(&head, createStudent("W", 23, (int[]){92, 88, 66, 85, 90}));
    appendStudent(&head, createStudent("X", 24, (int[]){78, 52, 88, 76, 85}));
    appendStudent(&head, createStudent("Y", 25, (int[]){85, 90, 48, 92, 88}));
    Sort(head);
    struct Student* current = head;
    while (current != NULL)
    {
        displayStudent(current);
        current = current->next;
    }
    freeStudents(head);
    return 0;
}
```

OUTPUT :

```
Name: R

Roll Number: 18

Marks: 78 92 8 76 85

Average: 67.80

--------------------

Name: T

Roll Number: 20

Marks: 92 88 16 85 90

Average: 74.20

--------------------

Name: S

Roll Number: 19

Marks: 85 90 18 92 88

Average: 74.60

--------------------

Name: M

Roll Number: 13

Marks: 85 90 23 92 88

Average: 75.60
```

```
--------------------
Name: X
Roll Number: 24
Marks: 78 52 88 76 85
Average: 75.80
--------------------
Name: L
Roll Number: 12
Marks: 78 92 55 76 85
Average: 77.20
--------------------
Name: Q
Roll Number: 17
Marks: 92 88 34 85 90
Average: 77.80
--------------------
Name: J
Roll Number: 10
Marks: 85 60 78 92 88
Average: 80.60
--------------------
Name: Y
Roll Number: 25
Marks: 85 90 48 92 88
Average: 80.60
```

```
--------------------
Name: I
Roll Number: 9
Marks: 78 92 78 76 85
Average: 81.80
--------------------
Name: K
Roll Number: 11
Marks: 92 88 56 85 90
Average: 82.20
--------------------
Name: U
Roll Number: 21
Marks: 78 92 80 76 85
Average: 82.20
--------------------
Name: F
Roll Number: 6
Marks: 78 90 88 76 85
Average: 83.40
--------------------
Name: C
Roll Number: 3
Marks: 78 92 88 76 85
Average: 83.80
```

```
--------------------
Name: W
Roll Number: 23
Marks: 92 88 66 85 90
Average: 84.20
--------------------
Name: V
Roll Number: 22
Marks: 85 90 68 92 88
Average: 84.60
--------------------
Name: E
Roll Number: 5
Marks: 92 82 76 85 90
Average: 85.00
--------------------
Name: O
Roll Number: 15
Marks: 78 92 97 76 85
Average: 85.60
--------------------
Name: B
Roll Number: 2
Marks: 92 88 76 85 90
Average: 86.20
```

```
--------------------
Name: D
Roll Number: 4
Marks: 85 90 76 92 88
Average: 86.20
--------------------
Name: A
Roll Number: 1
Marks: 85 90 78 92 88
Average: 86.60
--------------------
Name: P
Roll Number: 16
Marks: 85 91 78 92 88
Average: 86.80
--------------------
Name: G
Roll Number: 7
Marks: 85 99 78 92 88
Average: 88.40
--------------------
Name: H
Roll Number: 8
Marks: 92 100 76 85 90
Average: 88.60
```

```
--------------------
Name: N
Roll Number: 14
Marks: 92 88 99 85 90
Average: 90.80
--------------------
```

d. Deleting the details of students whose position in the sorted linked list (Note: Not roll number) is 1, 4, 23, 25

PSEUDOCODE :

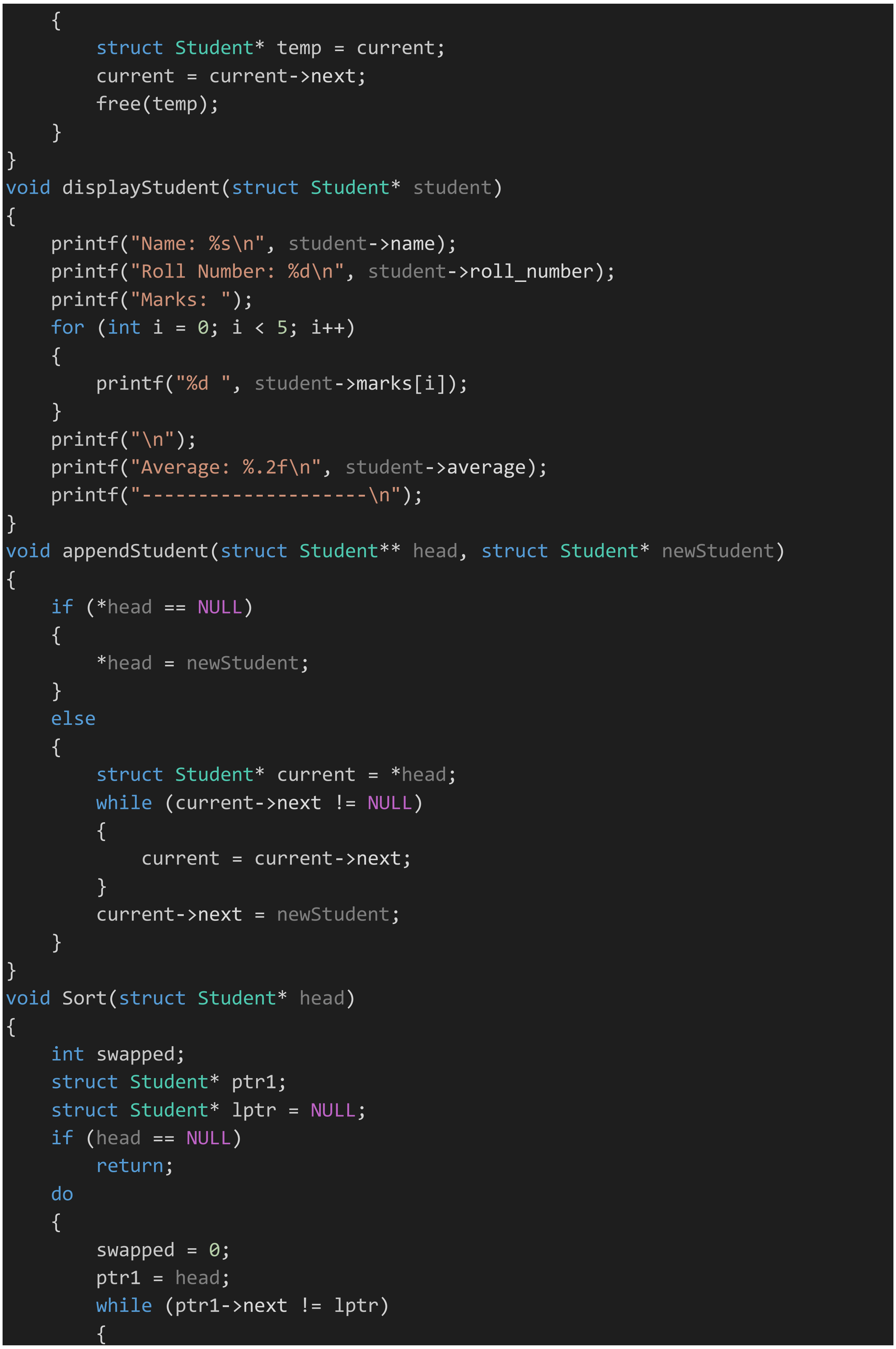
```
1.Structure Student:
   1.1String name[50]
   1.2Integer roll_number
   1.3Integer marks[5]
   1.4Float average
   1.5Pointer to Student next
2.Function createStudent(name, roll_number, marks):
   2.1Allocate memory for newStudent
   2.2If newStudent is NULL
      Print "Memory allocation failed."
      Exit the program
   2.3Copy name into newStudent->name
   2.4Assign roll_number to newStudent->roll_number
   2.5For i in range 0 to 4:
      Assign marks[i] to newStudent->marks[i]
   2.6Initialize sum to 0
   2.7For i in range 0 to 4:
      Add marks[i] to sum
   2.8Calculate average as sum divided by 5
   2.9Set newStudent->average to average
   2.10Set newStudent->next to NULL
   2.11Return newStudent
3.Function freeStudents(head):
   3.1Set current to head
   3.2While current is not NULL:
      Set temp to current
      Set current to current->next
      Free memory for temp
4.Function displayStudent(student):
   4.1Print "Name:", student->name
   4.2Print "Roll Number:", student->roll_number
   4.3Print "Marks:", student->marks[0], student->marks[1], ..., student-
>marks[4]
   4.4Print "Average:", student->average
   4.5Print "--------------------"
5.Function appendStudent(head, newStudent):
   5.1If head is NULL:
      Set head to newStudent
   5.2Else:
      Set current to head
```

```
        While current->next is not NULL:
            Set current to current->next
        Set current->next to newStudent
6.Function Sort(head):
    6.1Set swapped to True
    6.2Set lptr to NULL
    6.3While swapped is True:
        Set swapped to False
        Set ptr1 to head
        While ptr1->next is not lptr:
            If ptr1->average > ptr1->next->average:
                Swap roll_number, name, marks, and average of ptr1 and ptr1-
>next
                Set swapped to True
            Set ptr1 to ptr1->next
        Set lptr to ptr1
7.Function deleteStudentsByPosition(head, positions, numPositions):
    7.1If head is NULL or positions is NULL or numPositions is 0:
        Return
    7.2Set current to head
    7.3Set prev to NULL
    7.4Set currentPosition to 1
    7.5While current is not NULL:
        For i in range 0 to numPositions - 1:
            If positions[i] is equal to currentPosition:
```

```
            If prev is NULL:
               Set head to current->next
            Else:
               Set prev->next to current->next
            Free memory for current
            Break
      7.6Set prev to current
      7.7Set current to current->next
      7.8Increment currentPosition
8.Function main():
   8.1Set head to NULL
   8.2Append student information using appendStudent and
createStudent
   8.3Sort students using Sort
   8.4Print "Sorted Students (Before Deletion):"
   8.5Set current to head
   8.6While current is not NULL:
      Call displayStudent with current
      Set current to current->next
   8.7Define positionsToDelete array
   8.8Set numPositions to length of positionsToDelete
   8.9Delete students using deleteStudentsByPosition
   8.10Print "Remaining Students (After Deletion):"
   8.11Set current to head
   8.12While current is not NULL:
        Call displayStudent with current
        Set current to current->next
    8.13Free memory using freeStudents
    8.14Return 0
```

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Student
{
    char name[50];
    int roll_number;
    int marks[5];
    float average;
    struct Student* next;
};
struct Student* createStudent(char name[], int roll_number, int marks[])
{
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct 
Student));
    if (newStudent == NULL)
    {
        printf("Memory allocation failed.");
        exit(1);
    }
    strcpy(newStudent->name, name);
    newStudent->roll_number = roll_number;
    for (int i = 0; i < 5; i++)
    {
        newStudent->marks[i] = marks[i];
    }
    float sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum += marks[i];
    }
    newStudent->average = sum / 5;
    newStudent->next = NULL;
    return newStudent;
}
void freeStudents(struct Student* head)
{
    struct Student* current = head;
    while (current != NULL)
```



```c
    {
        struct Student* temp = current;
        current = current->next;
        free(temp);
    }
}
void displayStudent(struct Student* student)
{
    printf("Name: %s\n", student->name);
    printf("Roll Number: %d\n", student->roll_number);
    printf("Marks: ");
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", student->marks[i]);
    }
    printf("\n");
    printf("Average: %.2f\n", student->average);
    printf("--------------------\n");
}
void appendStudent(struct Student** head, struct Student* newStudent)
{
    if (*head == NULL)
    {
        *head = newStudent;
    }
    else
    {
        struct Student* current = *head;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = newStudent;
    }
}
void Sort(struct Student* head)
{
    int swapped;
    struct Student* ptr1;
    struct Student* lptr = NULL;
    if (head == NULL)
        return;
    do
    {
        swapped = 0;
        ptr1 = head;
        while (ptr1->next != lptr)
        {
            if (ptr1->average > ptr1->next->average)
            {

                int temp_roll = ptr1->roll_number;
                ptr1->roll_number = ptr1->next->roll_number;
                ptr1->next->roll_number = temp_roll;
                char temp_name[50];
                strcpy(temp_name, ptr1->name);
                strcpy(ptr1->name, ptr1->next->name);
                strcpy(ptr1->next->name, temp_name);
                for (int i = 0; i < 5; i++)
                {
                    int temp_mark = ptr1->marks[i];
                    ptr1->marks[i] = ptr1->next->marks[i];
                    ptr1->next->marks[i] = temp_mark;
                }
                float temp_avg = ptr1->average;
                ptr1->average = ptr1->next->average;
                ptr1->next->average = temp_avg;
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}
void deleteStudentsByPosition(struct Student** head, int positions[], int
numPositions)
{
    if (*head == NULL || positions == NULL || numPositions == 0)
    {
        return;
    }
    struct Student* current = *head;
    struct Student* prev = NULL;
    int currentPosition = 1;
    while (current != NULL)
    {
        for (int i = 0; i < numPositions; i++)
        {
            if (positions[i] == currentPosition)
            {
                if (prev == NULL)
                {
                    *head = current->next;
                }
                else
                {
```

```c
                prev->next = current->next;
            }
            free(current);
            break;
        }
    }
    prev = current;
    current = current->next;
    currentPosition++;
  }
}
int main()
{
    struct Student* head = NULL;
    appendStudent(&head, createStudent("A", 1, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("B", 2, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("C", 3, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("D", 4, (int[]){85, 90, 76, 92, 88}));
    appendStudent(&head, createStudent("E", 5, (int[]){92, 82, 76, 85, 90}));
    appendStudent(&head, createStudent("F", 6, (int[]){78, 90, 88, 76, 85}));
    appendStudent(&head, createStudent("G", 7, (int[]){85, 99, 78, 92, 88}));
    appendStudent(&head, createStudent("H", 8, (int[]){92, 100, 76, 85, 90}));
    appendStudent(&head, createStudent("I", 9, (int[]){78, 92, 78, 76, 85}));
    appendStudent(&head, createStudent("J", 10, (int[]){85, 60, 78, 92, 88}));
    appendStudent(&head, createStudent("K", 11, (int[]){92, 88, 56, 85, 90}));
    appendStudent(&head, createStudent("L", 12, (int[]){78, 92, 55, 76, 85}));
    appendStudent(&head, createStudent("M", 13, (int[]){85, 90, 23, 92, 88}));
    appendStudent(&head, createStudent("N", 14, (int[]){92, 88, 99, 85, 90}));
    appendStudent(&head, createStudent("O", 15, (int[]){78, 92, 97, 76, 85}));
    appendStudent(&head, createStudent("P", 16, (int[]){85, 91, 78, 92, 88}));
    appendStudent(&head, createStudent("Q", 17, (int[]){92, 88, 34, 85, 90}));
    appendStudent(&head, createStudent("R", 18, (int[]){78, 92, 8, 76, 85}));
    appendStudent(&head, createStudent("S", 19, (int[]){85, 90, 18, 92, 88}));
    appendStudent(&head, createStudent("T", 20, (int[]){92, 88, 16, 85, 90}));
    appendStudent(&head, createStudent("U", 21, (int[]){78, 92, 80, 76, 85}));
    appendStudent(&head, createStudent("V", 22, (int[]){85, 90, 68, 92, 88}));
    appendStudent(&head, createStudent("W", 23, (int[]){92, 88, 66, 85, 90}));
    appendStudent(&head, createStudent("X", 24, (int[]){78, 52, 88, 76, 85}));
    appendStudent(&head, createStudent("Y", 25, (int[]){85, 90, 48, 92, 88}));
    Sort(head);
    printf("Sorted Students (Before Deletion):\n");
    struct Student* current = head;
    while (current != NULL)
    {
        displayStudent(current);
        current = current->next;
    }
    int positionsToDelete[] = {1, 4, 23, 25};
    int numPositions = sizeof(positionsToDelete) /
sizeof(positionsToDelete[0]);
    deleteStudentsByPosition(&head, positionsToDelete, numPositions);
    printf("Remaining Students (After Deletion):\n");
    current = head;
    while (current != NULL)
    {
        displayStudent(current);
        current = current->next;
    }
    freeStudents(head);
    return 0;
}
```

OUTPUT :

```
Sorted Students (Before Deletion):

Name: R

Roll Number: 18

Marks: 78 92 8 76 85

Average: 67.80

--------------------

Name: T

Roll Number: 20

Marks: 92 88 16 85 90

Average: 74.20

--------------------

Name: S

Roll Number: 19

Marks: 85 90 18 92 88

Average: 74.60

--------------------
```

```
Name: M
Roll Number: 13
Marks: 85 90 23 92 88
Average: 75.60
--------------------
Name: X
Roll Number: 24
Marks: 78 52 88 76 85
Average: 75.80
--------------------
Name: L
Roll Number: 12
Marks: 78 92 55 76 85
Average: 77.20
--------------------
Name: Q
Roll Number: 17
Marks: 92 88 34 85 90
Average: 77.80
--------------------
Name: J
Roll Number: 10
Marks: 85 60 78 92 88
Average: 80.60
--------------------
```

```
Name: Y

Roll Number: 25

Marks: 85 90 48 92 88

Average: 80.60

--------------------

Name: I

Roll Number: 9

Marks: 78 92 78 76 85

Average: 81.80

--------------------

Name: K

Roll Number: 11

Marks: 92 88 56 85 90

Average: 82.20

--------------------

Name: U

Roll Number: 21

Marks: 78 92 80 76 85

Average: 82.20

--------------------

Name: F

Roll Number: 6

Marks: 78 90 88 76 85

Average: 83.40

--------------------
```

```
Name: C
Roll Number: 3
Marks: 78 92 88 76 85
Average: 83.80
--------------------
Name: W
Roll Number: 23
Marks: 92 88 66 85 90
Average: 84.20
--------------------
Name: V
Roll Number: 22
Marks: 85 90 68 92 88
Average: 84.60
--------------------
Name: E
Roll Number: 5
Marks: 92 82 76 85 90
Average: 85.00
--------------------
Name: O
Roll Number: 15
Marks: 78 92 97 76 85
Average: 85.60
--------------------
```

```
Name: B
Roll Number: 2
Marks: 92 88 76 85 90
Average: 86.20
--------------------
Name: D
Roll Number: 4
Marks: 85 90 76 92 88
Average: 86.20
--------------------
Name: A
Roll Number: 1
Marks: 85 90 78 92 88
Average: 86.60
--------------------
Name: P
Roll Number: 16
Marks: 85 91 78 92 88
Average: 86.80
--------------------
Name: G
Roll Number: 7
Marks: 85 99 78 92 88
Average: 88.40
--------------------
```

```
Name: H
Roll Number: 8
Marks: 92 100 76 85 90
Average: 88.60
--------------------
Name: N
Roll Number: 14
Marks: 92 88 99 85 90
Average: 90.80
--------------------
Remaining Students (After Deletion):
Name: T
Roll Number: 20
Marks: 92 88 16 85 90
Average: 74.20
--------------------
Name: S
Roll Number: 19
Marks: 85 90 18 92 88
Average: 74.60
--------------------
Name: X
Roll Number: 24
Marks: 78 52 88 76 85
Average: 75.80
```

```
--------------------
Name: L
Roll Number: 12
Marks: 78 92 55 76 85
Average: 77.20
--------------------
Name: Q
Roll Number: 17
Marks: 92 88 34 85 90
Average: 77.80
--------------------
Name: J
Roll Number: 10
Marks: 85 60 78 92 88
Average: 80.60
--------------------
Name: Y
Roll Number: 25
Marks: 85 90 48 92 88
Average: 80.60
--------------------
Name: I
Roll Number: 9
Marks: 78 92 78 76 85
Average: 81.80
```

```
--------------------
Name: K
Roll Number: 11
Marks: 92 88 56 85 90
Average: 82.20
--------------------
Name: U
Roll Number: 21
Marks: 78 92 80 76 85
Average: 82.20
--------------------
Name: F
Roll Number: 6
Marks: 78 90 88 76 85
Average: 83.40
--------------------
Name: C
Roll Number: 3
Marks: 78 92 88 76 85
Average: 83.80
--------------------
Name: W
Roll Number: 23
Marks: 92 88 66 85 90
Average: 84.20
```

```
--------------------
Name: V
Roll Number: 22
Marks: 85 90 68 92 88
Average: 84.60
--------------------
Name: E
Roll Number: 5
Marks: 92 82 76 85 90
Average: 85.00
--------------------
Name: O
Roll Number: 15
Marks: 78 92 97 76 85
Average: 85.60
--------------------
Name: B
Roll Number: 2
Marks: 92 88 76 85 90
Average: 86.20
--------------------
Name: D
Roll Number: 4
Marks: 85 90 76 92 88
Average: 86.20
```

```
--------------------
Name: A
Roll Number: 1
Marks: 85 90 78 92 88
Average: 86.60
--------------------
Name: P
Roll Number: 16
Marks: 85 91 78 92 88
Average: 86.80
--------------------
Name: H
Roll Number: 8
Marks: 92 100 76 85 90
Average: 88.60
--------------------
```

e. In the resulting new list, swap the following elements which are denoted by following positions:

i. 1 and 2

ii. 4 and 17

iii. 21 and 1.

PSEUDOCODE :

```
1.Structure Student:
   1.1String name[50]
```

```
  1.2Integer roll_number
  1.3Integer marks[5]
  1.4Float average
  1.5Student* next
2.Function createStudent(name, roll_number, marks):
  2.1newStudent = Allocate memory for Student
  2.2If newStudent is NULL:
    Print "Memory allocation failed."
    Exit
  2.3Copy name to newStudent->name
  2.4Set newStudent->roll_number to roll_number
  2.5For i from 0 to 4:
    Set newStudent->marks[i] to marks[i]
  2.6sum = 0
  2.7For i from 0 to 4:
    sum += marks[i]
  2.8Set newStudent->average to sum / 5
  2.9Set newStudent->next to NULL
  2.10Return newStudent
3.Function freeStudents(head):
  3.1current = head
  3.2While current is not NULL:
    temp = current
    current = current->next
    Free memory for temp
```

```
4.Function displayStudent(student):
   4.1Print "Name:", student->name
   4.2Print "Roll Number:", student->roll_number
   4.3Print "Marks:",
   4.4For i from 0 to 4:
      Print student->marks[i], " "
   4.4Print newline
   4.5Print "Average:", student->average with 2 decimal places
   4.6Print "--------------------"
5.Function appendStudent(head, newStudent):
   5.1If head is NULL:
      Set head to newStudent
   5.2Else:
      current = head
      While current->next is not NULL:
         Set current to current->next
      Set current->next to newStudent
6.Function bubbleSort(head):
   6.1swapped = 1
   6.2lptr = NULL
   6.3While swapped is 1:
      swapped = 0
      ptr1 = head
      While ptr1->next is not lptr:
         If ptr1->average > ptr1->next->average:
           Swap ptr1 and ptr1->next (name, roll_number, marks, average)
           swapped = 1
         Set ptr1 to ptr1->next
      6.4Set lptr to ptr1
7.Function deleteStudents(head, positions, numPositions):
   7.1If head is NULL or positions is NULL or numPositions is 0:
      Return
   7.2current = head
   7.3prev = NULL
   7.4currentPosition = 1
   7.5While current is not NULL:
      For i from 0 to numPositions - 1:
         If positions[i] is currentPosition:
           If prev is NULL:
             Set head to current->next
           Else:
             Set prev->next to current->next
           Free memory for current
           Break loop
      Set prev to current
      Set current to current->next
      Increment currentPosition
8.Function swapStudents(head, pos1, pos2):
   8.1student1 = head
   8.2student2 = head
```

```
8.3prev1 = NULL
8.4prev2 = NULL
8.5currentPosition = 1
8.6While student1 is not NULL and currentPosition is not pos1:
   Set prev1 to student1
   Set student1 to student1->next
   Increment currentPosition
8.7Reset currentPosition to 1
8.8While student2 is not NULL and currentPosition is not pos2:
   Set prev2 to student2
   Set student2 to student2->next
   Increment currentPosition
8.9If student1 is NULL or student2 is NULL:
   Print "Invalid positions for swapping."
   Return
8.10If prev1 is not NULL:
   Set prev1->next to student2
8.11Else:
   Set head to student2
8.12If prev2 is not NULL:
   Set prev2->next to student1
8.13Else:
   Set head to student1
8.14temp = student1->next
8.15Set student1->next to student2->next
```

```
 8.16Set student2->next to temp
9.Function main:
 9.1head = NULL
 9.2AppendStudent(&head, createStudent("A", 1, {85, 90, 78, 92, 88}))
 9.3AppendStudent(&head, createStudent("B", 2, {92, 88, 76, 85, 90}))
 ... (Append other students)
 9.4 BubbleSort(head)
 9.5Print "Sorted Students (Before Deletion):"
 9.6current = head
 9.7While current is not NULL:
    displayStudent(current)
    Set current to current->next
 9.8positionsToDelete[] = {1, 4, 23, 25}
 9.9numPositions = Length of positionsToDelete
 9.10DeleteStudents(&head, positionsToDelete, numPositions)
 9.11Print "Remaining Students (After Deletion):"
 9.12current = head
 9.13While current is not NULL:
    displayStudent(current)
    Set current to current->next
 9.14SwapStudents(head, 1, 2)
 9.15SwapStudents(head, 4, 17)
 9.16SwapStudents(head, 21, 1)
 9.17Print "Students After Swapping:"
 9.18current = head
```

```
9.19While current is not NULL:
    displayStudent(current)
    Set current to current->next
9.20FreeStudents(head)
9.21Return 0
```

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Student
{
    char name[50];
    int roll_number;
    int marks[5];
    float average;
    struct Student* next;
};
struct Student* createStudent(char name[], int roll_number, int marks[])
{
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct
Student));
    if (newStudent == NULL)
    {
        printf("Memory allocation failed.");
        exit(1);
    }
    strcpy(newStudent->name, name);
    newStudent->roll_number = roll_number;
    for (int i = 0; i < 5; i++)
    {
        newStudent->marks[i] = marks[i];
    }
    float sum = 0;
    for (int i = 0; i < 5; i++)
    {
        sum += marks[i];
    }
    newStudent->average = sum / 5;
    newStudent->next = NULL;
    return newStudent;
}
void freeStudents(struct Student* head)
{
    struct Student* current = head;
    while (current != NULL)
    {
        struct Student* temp = current;
        current = current->next;
        free(temp);
    }
}
void displayStudent(struct Student* student)
{
    printf("Name: %s\n", student->name);
    printf("Roll Number: %d\n", student->roll_number);
    printf("Marks: ");
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", student->marks[i]);
    }
    printf("\n");
    printf("Average: %.2f\n", student->average);
    printf("--------------------\n");
}
void appendStudent(struct Student** head, struct Student* newStudent)
{
    if (*head == NULL)
    {
        *head = newStudent;
    }
    else
    {
        struct Student* current = *head;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = newStudent;
    }
}
void bubbleSort(struct Student* head)
{
    int swapped;
    struct Student* ptr1;
    struct Student* lptr = NULL;
    if (head == NULL)
        return;
    do
    {
        swapped = 0;
        ptr1 = head;
        while (ptr1->next != lptr)
        {
            if (ptr1->average > ptr1->next->average)
            {

                int temp_roll = ptr1->roll_number;
                ptr1->roll_number = ptr1->next->roll_number;
                ptr1->next->roll_number = temp_roll;
                char temp_name[50];
                strcpy(temp_name, ptr1->name);
                strcpy(ptr1->name, ptr1->next->name);
                strcpy(ptr1->next->name, temp_name);
                for (int i = 0; i < 5; i++)
                {
                    int temp_mark = ptr1->marks[i];
                    ptr1->marks[i] = ptr1->next->marks[i];
                    ptr1->next->marks[i] = temp_mark;
                }
                float temp_avg = ptr1->average;
                ptr1->average = ptr1->next->average;
                ptr1->next->average = temp_avg;
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}
void deleteStudents(struct Student** head, int positions[], int numPositions)
{
    if (*head == NULL || positions == NULL || numPositions == 0)
    {
        return;
    }
    struct Student* current = *head;
    struct Student* prev = NULL;
    int currentPosition = 1;
    while (current != NULL)
    {

        for (int i = 0; i < numPositions; i++)
        {
            if (positions[i] == currentPosition)
            {

                if (prev == NULL)
                {
                    *head = current->next;
                }
                else
                {
                    prev->next = current->next;
                }
                free(current);
                break;
            }
        }
        prev = current;
        current = current->next;
        currentPosition++;
    }
}
void swapStudents(struct Student* head, int pos1, int pos2)
{
    struct Student* student1 = head;
    struct Student* student2 = head;
    struct Student* prev1 = NULL;
    struct Student* prev2 = NULL;
    int currentPosition = 1;
    while (student1 != NULL && currentPosition != pos1)
    {
        prev1 = student1;
        student1 = student1->next;
        currentPosition++;
    }
    currentPosition = 1;
    while (student2 != NULL && currentPosition != pos2)
    {
        prev2 = student2;
        student2 = student2->next;
        currentPosition++;
    }
    if (student1 == NULL || student2 == NULL)
    {
        printf("Invalid positions for swapping.\n");
        return;
    }
    if (prev1 != NULL)
    {
        prev1->next = student2;
    }
    else
    {
        head = student2;
    }
    if (prev2 != NULL)
```

```c
    {
        prev2->next = student1;
    }
    else
    {
        head = student1;
    }
    struct Student* temp = student1->next;
    student1->next = student2->next;
    student2->next = temp;
}
int main()
{
    struct Student* head = NULL;
    appendStudent(&head, createStudent("A", 1, (int[]){85, 90, 78, 92, 88}));
    appendStudent(&head, createStudent("B", 2, (int[]){92, 88, 76, 85, 90}));
    appendStudent(&head, createStudent("C", 3, (int[]){78, 92, 88, 76, 85}));
    appendStudent(&head, createStudent("D", 4, (int[]){85, 90, 76, 92, 88}));
    appendStudent(&head, createStudent("E", 5, (int[]){92, 82, 76, 85, 90}));
    appendStudent(&head, createStudent("F", 6, (int[]){78, 90, 88, 76, 85}));
    appendStudent(&head, createStudent("G", 7, (int[]){85, 99, 78, 92, 88}));
    appendStudent(&head, createStudent("H", 8, (int[]){92, 100, 76, 85, 90}));
    appendStudent(&head, createStudent("I", 9, (int[]){78, 92, 78, 76, 85}));
    appendStudent(&head, createStudent("J", 10, (int[]){85, 60, 78, 92, 88}));
    appendStudent(&head, createStudent("K", 11, (int[]){92, 88, 56, 85, 90}));
    appendStudent(&head, createStudent("L", 12, (int[]){78, 92, 55, 76, 85}));
    appendStudent(&head, createStudent("M", 13, (int[]){85, 90, 23, 92, 88}));
    appendStudent(&head, createStudent("N", 14, (int[]){92, 88, 99, 85, 90}));
    appendStudent(&head, createStudent("O", 15, (int[]){78, 92, 97, 76, 85}));
    appendStudent(&head, createStudent("P", 16, (int[]){85, 91, 78, 92, 88}));
    appendStudent(&head, createStudent("Q", 17, (int[]){92, 88, 34, 85, 90}));
    appendStudent(&head, createStudent("R", 18, (int[]){78, 92, 8, 76, 85}));
    appendStudent(&head, createStudent("S", 19, (int[]){85, 90, 18, 92, 88}));
    appendStudent(&head, createStudent("T", 20, (int[]){92, 88, 16, 85, 90}));
    appendStudent(&head, createStudent("U", 21, (int[]){78, 92, 80, 76, 85}));
    appendStudent(&head, createStudent("V", 22, (int[]){85, 90, 68, 92, 88}));
    appendStudent(&head, createStudent("W", 23, (int[]){92, 88, 66, 85, 90}));
    appendStudent(&head, createStudent("X", 24, (int[]){78, 52, 88, 76, 85}));
    appendStudent(&head, createStudent("Y", 25, (int[]){85, 90, 48, 92, 88}));
    bubbleSort(head);
    printf("Sorted Students (Before Deletion):\n");
    struct Student* current = head;
    while (current != NULL)
    {
        displayStudent(current);
        current = current->next;
    }
    int positionsToDelete[] = {1, 4, 23, 25};
    int numPositions = sizeof(positionsToDelete) /
sizeof(positionsToDelete[0]);
    deleteStudents(&head, positionsToDelete, numPositions);
    printf("Remaining Students (After Deletion):\n");
    current = head;
    while (current != NULL)
    {
        displayStudent(current);
        current = current->next;
    }
    swapStudents(head, 1, 2);
    swapStudents(head, 4, 17);
    swapStudents(head, 21, 1);
    printf("Students After Swapping:\n");
    current = head;
    while (current != NULL)
    {
        displayStudent(current);
        current = current->next;
    }
    freeStudents(head);
    return 0;
}
```

OUTPUT :

```
Sorted Students (Before Deletion):

Name: R

Roll Number: 18

Marks: 78 92 8 76 85

Average: 67.80

--------------------

Name: T

Roll Number: 20

Marks: 92 88 16 85 90

Average: 74.20

--------------------
```

```
Name: S
Roll Number: 19
Marks: 85 90 18 92 88
Average: 74.60
--------------------
Name: M
Roll Number: 13
Marks: 85 90 23 92 88
Average: 75.60
--------------------
Name: X
Roll Number: 24
Marks: 78 52 88 76 85
Average: 75.80
--------------------
Name: L
Roll Number: 12
Marks: 78 92 55 76 85
Average: 77.20
--------------------
Name: Q
Roll Number: 17
Marks: 92 88 34 85 90
Average: 77.80
--------------------
```

```
Name: J
Roll Number: 10
Marks: 85 60 78 92 88
Average: 80.60
--------------------
Name: Y
Roll Number: 25
Marks: 85 90 48 92 88
Average: 80.60
--------------------
Name: I
Roll Number: 9
Marks: 78 92 78 76 85
Average: 81.80
--------------------
Name: K
Roll Number: 11
Marks: 92 88 56 85 90
Average: 82.20
--------------------
Name: U
Roll Number: 21
Marks: 78 92 80 76 85
Average: 82.20
--------------------
```

```
Name: F
Roll Number: 6
Marks: 78 90 88 76 85
Average: 83.40
--------------------
Name: C
Roll Number: 3
Marks: 78 92 88 76 85
Average: 83.80
--------------------
Name: W
Roll Number: 23
Marks: 92 88 66 85 90
Average: 84.20
--------------------
Name: V
Roll Number: 22
Marks: 85 90 68 92 88
Average: 84.60
--------------------
Name: E
Roll Number: 5
Marks: 92 82 76 85 90
Average: 85.00
--------------------
```

```
Name: O

Roll Number: 15

Marks: 78 92 97 76 85

Average: 85.60

--------------------

Name: B

Roll Number: 2

Marks: 92 88 76 85 90

Average: 86.20

--------------------

Name: D

Roll Number: 4

Marks: 85 90 76 92 88

Average: 86.20

--------------------

Name: A

Roll Number: 1

Marks: 85 90 78 92 88

Average: 86.60

--------------------

Name: P

Roll Number: 16

Marks: 85 91 78 92 88

Average: 86.80

--------------------
```

```
Name: G
Roll Number: 7
Marks: 85 99 78 92 88
Average: 88.40
--------------------
Name: H
Roll Number: 8
Marks: 92 100 76 85 90
Average: 88.60
--------------------
Name: N
Roll Number: 14
Marks: 92 88 99 85 90
Average: 90.80
--------------------
Remaining Students (After Deletion):
Name: T
Roll Number: 20
Marks: 92 88 16 85 90
Average: 74.20
--------------------
Name: S
Roll Number: 19
Marks: 85 90 18 92 88
Average: 74.60
```

```
--------------------
Name: X
Roll Number: 24
Marks: 78 52 88 76 85
Average: 75.80
--------------------
Name: L
Roll Number: 12
Marks: 78 92 55 76 85
Average: 77.20
--------------------
Name: Q
Roll Number: 17
Marks: 92 88 34 85 90
Average: 77.80
--------------------
Name: J
Roll Number: 10
Marks: 85 60 78 92 88
Average: 80.60
--------------------
Name: Y
Roll Number: 25
Marks: 85 90 48 92 88
Average: 80.60
```

```
--------------------
Name: I
Roll Number: 9
Marks: 78 92 78 76 85
Average: 81.80
--------------------
Name: K
Roll Number: 11
Marks: 92 88 56 85 90
Average: 82.20
--------------------
Name: U
Roll Number: 21
Marks: 78 92 80 76 85
Average: 82.20
--------------------
Name: F
Roll Number: 6
Marks: 78 90 88 76 85
Average: 83.40
--------------------
Name: C
Roll Number: 3
Marks: 78 92 88 76 85
Average: 83.80
```

```
--------------------
Name: W
Roll Number: 23
Marks: 92 88 66 85 90
Average: 84.20
--------------------
Name: V
Roll Number: 22
Marks: 85 90 68 92 88
Average: 84.60
--------------------
Name: E
Roll Number: 5
Marks: 92 82 76 85 90
Average: 85.00
--------------------
Name: O
Roll Number: 15
Marks: 78 92 97 76 85
Average: 85.60
--------------------
Name: B
Roll Number: 2
Marks: 92 88 76 85 90
Average: 86.20
```

```
--------------------
Name: D
Roll Number: 4
Marks: 85 90 76 92 88
Average: 86.20
--------------------
Name: A
Roll Number: 1
Marks: 85 90 78 92 88
Average: 86.60
--------------------
Name: P
Roll Number: 16
Marks: 85 91 78 92 88
Average: 86.80
--------------------
Name: H
Roll Number: 8
Marks: 92 100 76 85 90
Average: 88.60
--------------------
Invalid positions for swapping.
Students After Swapping:
Name: T
Roll Number: 20
```

```
Marks: 92 88 16 85 90

Average: 74.20

--------------------

Name: X

Roll Number: 24

Marks: 78 52 88 76 85

Average: 75.80

--------------------

Name: L

Roll Number: 12

Marks: 78 92 55 76 85

Average: 77.20

--------------------

Name: D

Roll Number: 4

Marks: 85 90 76 92 88

Average: 86.20

--------------------

Name: J

Roll Number: 10

Marks: 85 60 78 92 88

Average: 80.60

--------------------

Name: Y

Roll Number: 25
```

```
Marks: 85 90 48 92 88

Average: 80.60

--------------------

Name: I

Roll Number: 9

Marks: 78 92 78 76 85

Average: 81.80

--------------------

Name: K

Roll Number: 11

Marks: 92 88 56 85 90

Average: 82.20

--------------------

Name: U

Roll Number: 21

Marks: 78 92 80 76 85

Average: 82.20

--------------------

Name: F

Roll Number: 6

Marks: 78 90 88 76 85

Average: 83.40

--------------------

Name: C

Roll Number: 3
```

```
Marks: 78 92 88 76 85

Average: 83.80

--------------------

Name: W

Roll Number: 23

Marks: 92 88 66 85 90

Average: 84.20

--------------------

Name: V

Roll Number: 22

--------------------

Name: H

Roll Number: 8

Marks: 92 100 76 85 90

Average: 88.60

--------------------
```