Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_MCQ

Attempt : 1
Total Mark : 10
Marks Obtained : 9

Section 1 : MCQ

1. In a singly linked list, what is the role of the "tail" node?

Answer

It stores the last element of the list

Status : Correct ***Marks : 1/1***

2. Given the linked list: 5 -> 10 -> 15 -> 20 -> 25 -> NULL. What will be the output of traversing the list and printing each node's data?

Answer

5 10 15 20 25

Status : Correct ***Marks : 1/1***

3. Consider the singly linked list: 15 -> 16 -> 6 -> 7 -> 17. You need to delete all nodes from the list which are prime.

What will be the final linked list after the deletion?

Answer

15 -> 16 -> 6

Status : Correct ***Marks : 1/1***

4. Consider the singly linked list: 13 -> 4 -> 16 -> 9 -> 22 -> 45 -> 5 -> 16 -> 6, and an integer K = 10, you need to delete all nodes from the list that are less than the given integer K.

What will be the final linked list after the deletion?

Answer

13 -> 4 -> 16 -> 9 -> 22 -> 45 -> 5 -> 16 -> 6

Status : Wrong ***Marks : 0/1***

5. Linked lists are not suitable for the implementation of?

Answer

Binary search

Status : Correct ***Marks : 1/1***

6. The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list?

```
struct node {
    int data;
    struct node* next;
```

```
};
static void reverse(struct node** head_ref) {
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL) {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   /*ADD A STATEMENT HERE*/
}
```

Answer

```
*head_ref = prev;
```

Status : Correct ***Marks : 1/1***

7. The following function takes a singly linked list of integers as a parameter and rearranges the elements of the lists.

The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

```
struct node {
   int value;
   struct node* next;
};

void rearrange (struct node* list) {
   struct node *p,q;
   int temp;
   if (! List || ! list->next) return;
   p=list; q=list->next;
   while(q) {
      temp=p->value; p->value=q->value;
      q->value=temp;p=q->next;
```

```
    q=p?p->next:0;
  }
}
```

Answer

2, 1, 4, 3, 6, 5, 7

Status : Correct ***Marks : 1/1***

8. Which of the following statements is used to create a new node in a singly linked list?

```
struct node {
  int data;
  struct node * next;
}
typedef struct node NODE;
NODE *ptr;
```

Answer

ptr = (NODE*)malloc(sizeof(NODE));

Status : Correct ***Marks : 1/1***

9. Given a pointer to a node X in a singly linked list. If only one point is given and a pointer to the head node is not given, can we delete node X from the given linked list?

Answer

Possible if X is not last node.

Status : Correct ***Marks : 1/1***

10. Consider an implementation of an unsorted singly linked list. Suppose it has its representation with a head pointer only. Given the representation, which of the following operations can be implemented in O(1) time?

i) Insertion at the front of the linked list

ii) Insertion at the end of the linked list

iii) Deletion of the front node of the linked list

iv) Deletion of the last node of the linked list

Answer

I and III

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Janani is a tech enthusiast who loves working with polynomials. She wants to create a program that can add polynomial coefficients and provide the sum of their coefficients.

The polynomials will be represented as a linked list, where each node of the linked list contains a coefficient and an exponent. The polynomial is represented in the standard form with descending order of exponents.

Input Format

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers each: the coefficient and the exponent of the term in the first polynomial.

The next line of input consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers each: the coefficient and the exponent of the term in the second polynomial.

Output Format

The output prints the sum of the coefficients of the polynomials.

Sample Test Case

Input: 3
2 2
3 1
4 0
3
2 2
3 1
4 0

Output: 18

Answer

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
    int coeff;
    int exp;
    struct Node* next;
} Node;
Node* createNode(int coeff,int exp){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coeff=coeff;
    newNode->exp=exp;
    newNode->next=NULL;
    return newNode;
}
void insert(Node** head,int coeff,int exp){
    Node* newNode=createNode(coeff,exp);
    if(*head==NULL || (*head)->exp < exp){
        newNode->next= *head;
        *head=newNode;
    }else{
```

```
        Node* temp= *head;
        while(temp->next != NULL && temp->next->exp >= exp){
            if(temp->next->exp==exp){
                temp->next->coeff +=coeff;
                free(newNode);
                return;
            }
            temp=temp->next;
        }
        newNode->next=temp->next;
        temp->next=newNode;
    }
}
int sumCoefficients(Node* head){
    int sum=0;
    while(head!=NULL){
        sum+=head->coeff;
        head=head->next;
    }
    return sum;
}
void freeList(Node* head){
    while(head !=NULL){
        Node* temp=head;
        head=head->next;
        free(temp);
    }
}
int main(){
    int n,m,coeff,exp;
    Node* poly1=NULL;
    Node* poly2=NULL;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d %d",&coeff,&exp);
        insert(&poly1,coeff,exp);
    }
    scanf("%d",&m);
    for(int i=0;i<m;i++){
        scanf("%d %d",&coeff,&exp);
        insert(&poly2,coeff,exp);
    }
```

```
    Node* temp=poly1;
    while(temp!=NULL){
        insert(&poly2,temp->coeff,temp->exp);
        temp=temp->next;
    }
    printf("%d\n",sumCoefficients(poly2));
    freeList(poly1);
    freeList(poly2);
    return 0;
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Arun is learning about data structures and algorithms. He needs your help in solving a specific problem related to a singly linked list.

Your task is to implement a program to delete a node at a given position. If the position is valid, the program should perform the deletion; otherwise, it should display an appropriate message.

Input Format

The first line of input consists of an integer N, representing the number of elements in the linked list.

The second line consists of N space-separated elements of the linked list.

The third line consists of an integer x, representing the position to delete.

Position starts from 1.

Output Format

The output prints space-separated integers, representing the updated linked list after deleting the element at the given position.

If the position is not valid, print "Invalid position. Deletion not possible."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
8 2 3 1 7
2
Output: 8 3 1 7

Answer

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

void insert(int);
void display_List();
void deleteNode(int);

struct node {
   int data;
   struct node* next;
} *head = NULL, *tail = NULL;


void insert(int value){
   struct node* newn=(struct node*)malloc(sizeof(struct node));
   newn->data=value;
   newn->next=NULL;
   if(head==NULL){
      head=newn;
      tail=newn;
   }else{
      tail->next=newn;
```

```
        tail=newn;
    }
}
void display_List(){
    struct node* temp=head;
    if(temp==NULL){
        printf("List is empty\n");
        return;
    }
    while(temp!=NULL){
        printf("%d ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}
void deleteNode(int pos){
    if(head==NULL){
        printf("Invalid position. Deletion not possible.\n");
        return;
    }
    struct node* temp=head;
    if(pos==1){
        head=head->next;
        free(temp);
        display_List();
        return;
    }
    struct node *prev=NULL;
    int count=1;
    while(temp!=NULL && count<pos){
        prev=temp;
        temp=temp->next;
        count++;
    }
    if(temp==NULL){
        printf("Invalid position. Deletion not possible.\n");
        return;
    }
    prev->next=temp->next;
    if(temp==tail){
        tail=prev;
    }
```

```
    free(temp);
    display_List();
}

int main() {
    int num_elements, element, pos_to_delete;

    scanf("%d", &num_elements);

    for (int i = 0; i < num_elements; i++) {
        scanf("%d", &element);
        insert(element);
    }

    scanf("%d", &pos_to_delete);

    deleteNode(pos_to_delete);

    return 0;
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Imagine you are working on a text processing tool and need to implement a feature that allows users to insert characters at a specific position.

Implement a program that takes user inputs to create a singly linked list of characters and inserts a new character after a given index in the list.

Input Format

The first line of input consists of an integer N, representing the number of characters in the linked list.

The second line consists of a sequence of N characters, representing the linked list.

The third line consists of an integer index, representing the index(0-based) after

which the new character node needs to be inserted.

The fourth line consists of a character value representing the character to be inserted after the given index.

Output Format

If the provided index is out of bounds (larger than the list size):

1. The first line of output prints "Invalid index".
2. The second line prints "Updated list: " followed by the unchanged linked list values.

Otherwise, the output prints "Updated list: " followed by the updated linked list after inserting the new character after the given index.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
a b c d e
2
X
Output: Updated list: a b c X d e

Answer

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   char data;
   struct Node* next;
};
void insertEnd(struct Node** head,char data){
   struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->data=data;
   newNode->next=NULL;
   if(*head==NULL){
      *head=newNode;
```

```
        return;
    }
    struct Node* temp=*head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=newNode;
}
void insertAfter(struct Node** head,int index,char newChar){
    struct Node* temp=*head;
    int count=0;
    while(temp!=NULL && count<index){
        temp=temp->next;
        count++;
    }
    if(temp==NULL){
        printf("Invalid index\n");
        return;
    }
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->data=newChar;
    newNode->next=temp->next;
    temp->next=newNode;
}
void displayList(struct Node* head){
    struct Node* temp=head;
    while(temp!=NULL){
        printf("%c ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}
int main(){
    struct Node* head=NULL;
    int N,index;
    char newChar;
    scanf("%d",&N);
    for(int i=0;i<N;i++){
        char value;
        scanf(" %c",&value);
        insertEnd(&head,value);
    }
```

```
    scanf("%d",&index);
    scanf(" %c",&newChar);
    insertAfter(&head,index,newChar);
    printf("Updated list: ");
    displayList(head);
    return 0;
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

As part of a programming assignment in a data structures course, students are required to create a program to construct a singly linked list by inserting elements at the beginning.

You are an evaluator of the course and guide the students to complete the task.

Input Format

The first line of input consists of an integer N, which is the number of elements.

The second line consists of N space-separated integers.

Output Format

The output prints the singly linked list elements, after inserting them at the beginning.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
78 89 34 51 67

Output: 67 51 34 89 78

Answer

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

struct Node {
    int data;
    struct Node* next;
};

void insertAtFront(struct Node** head,int value){
    struct Node* newn=(struct Node*)malloc(sizeof(struct Node));
    newn->data=value;
    newn->next=*head;
    *head=newn;
}
void printList(struct Node* head){
    struct Node* temp=head;
    while(temp!=NULL){
        printf("%d ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}

int main(){
    struct Node* head = NULL;

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

```
    for (int i = 0; i < n; i++) {
        int activity;
        scanf("%d", &activity);
        insertAtFront(&head, activity);
    }

    printList(head);
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Imagine you are tasked with developing a simple GPA management system using a singly linked list. The system allows users to input student GPA values, insertion should happen at the front of the linked list, delete record by position, and display the updated list of student GPAs.

Input Format

The first line of input contains an integer n, representing the number of students.

The next n lines contain a single floating-point value representing the GPA of each student.

The last line contains an integer position, indicating the position at which a student record should be deleted. Position starts from 1.

Output Format

After deleting the data in the given position, display the output in the format "GPA: " followed by the GPA value, rounded off to one decimal place.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
3.8
3.2
3.5
4.1
2
Output: GPA: 4.1
GPA: 3.2
GPA: 3.8

Answer

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   float gpa;
   struct Node* next;
};
void insertAtFront(struct Node** head,float gpa){
   struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->gpa=gpa;
   newNode->next=*head;
   *head=newNode;
}
void deleteAtPosition(struct Node** head,int position){
   if(*head==NULL){
      return;
   }
   struct Node* temp=*head;
   if(position==1){
      *head=temp->next;
      free(temp);
```

```
        return;
    }
    struct Node* prev=NULL;
    for(int i=1;temp!=NULL && i<position;i++){
        prev=temp;
        temp=temp->next;
    }
    if(temp==NULL){
        return;
    }
    prev->next=temp->next;
    free(temp);
}
void displayList(struct Node* head){
    struct Node* temp=head;
    while(temp!=NULL){
        printf("GPA: %.1f\n",temp->gpa);
        temp=temp->next;
    }
}
int main(){
    struct Node* head=NULL;
    int n,position;
    float gpa;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%f",&gpa);
        insertAtFront(&head,gpa);
    }
    scanf("%d",&position);
    deleteAtPosition(&head,position);
    displayList(head);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 6

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

John is tasked with creating a program to manage student roll numbers using a singly linked list.

Write a program for John that accepts students' roll numbers, inserts them at the end of the linked list, and displays the numbers.

Input Format

The first line of input consists of an integer N, representing the number of students.

The second line consists of N space-separated integers, representing the roll numbers of students.

Output Format

The output prints the space-separated integers singly linked list, after inserting the roll numbers of students at the end.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
23 85 47 62 31

Output: 23 85 47 62 31

Answer

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int rollNumber;
    struct Node* next;
};
void insertAtEnd(struct Node** head,int rollNumber){
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->rollNumber=rollNumber;
    newNode->next=NULL;
    if(*head==NULL){
        *head=newNode;
        return;
    }
    struct Node* temp=*head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=newNode;
}
void displayList(struct Node* head){
    struct Node* temp=head;
    while(temp!=NULL){
        printf("%d ",temp->rollNumber);
        temp=temp->next;
    }
    printf("\n");
}
```

```
int main(){
    struct Node* head=NULL;
    int N,rollNumber;
    scanf("%d",&N);
    for(int i=0;i<N;i++){
        scanf("%d",&rollNumber);
        insertAtEnd(&head,rollNumber);
    }
    displayList(head);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 7

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Dev is tasked with creating a program that efficiently finds the middle element of a linked list. The program should take user input to populate the linked list by inserting each element into the front of the list and then determining the middle element.

Assist Dev, as he needs to ensure that the middle element is accurately identified from the constructed singly linked list:

If it's an odd-length linked list, return the middle element.If it's an even-length linked list, return the second middle element of the two elements.

Input Format

The first line of input consists of an integer n, representing the number of elements in the linked list.

The second line consists of n space-separated integers, representing the elements of the list.

Output Format

The first line of output displays the linked list after inserting elements at the front.

The second line displays "Middle Element: " followed by the middle element of the linked list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50
Output: 50 40 30 20 10
Middle Element: 30

Answer

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

struct Node* push(struct Node* head,int data){
    struct Node* newn=(struct Node*)malloc(sizeof(struct Node));
    newn->data=data;
    newn->next=head;
    return newn;
}
int printMiddle(struct Node* head){
    struct Node* slow=head;
    struct Node* fast=head;
    while(fast!=NULL && fast->next!=NULL){
        slow=slow->next;
```

```
        fast=fast->next->next;
    }
    return slow->data;
}

int main() {
    struct Node* head = NULL;
    int n;

    scanf("%d", &n);
    int value;

    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        head = push(head, value);
    }

    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");


    int middle_element = printMiddle(head);
    printf("Middle Element: %d\n", middle_element);


    current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_week 1_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 15

Section 1 : Coding

1. Problem Statement

Akila is a tech enthusiast and wants to write a program to add two polynomials. Each polynomial is represented as a linked list, where each node in the list represents a term in the polynomial.

A term in the polynomial is represented in the format ax^b, where a is the coefficient and b is the exponent.

Akila needs your help to implement a program that takes two polynomials as input, adds them, and stores the result in ascending order in a new polynomial-linked list. Write a program to help her.

Input Format

The input consists of lines containing pairs of integers representing the

coefficients and exponents of polynomial terms.

Each line represents a single term, with the coefficient and exponent separated by a space.

The input for each polynomial ends with a line containing "0 0".

Output Format

The output consists of three lines representing the first, second, and resulting polynomial after the addition operation, with terms sorted in ascending order of exponents.

Each line contains terms of the polynomial in the format "coefficientx^exponent", separated by " + ".

If the resulting polynomial is zero, the output is "0".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3 4
2 3
1 2
0 0
1 2
2 3
3 4
0 0
Output: 1x^2 + 2x^3 + 3x^4
1x^2 + 2x^3 + 3x^4
2x^2 + 4x^3 + 6x^4

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int coefficient;
```

```
    int exponent;
    struct Node* next;
} Node;

typedef struct Polynomial {
    Node* head;
} Polynomial;

Node* createNode(int coefficient, int exponent) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}

void insert(Polynomial* poly, int coefficient, int exponent) {
    if (coefficient == 0) return;
    Node* newNode = createNode(coefficient, exponent);
    if (!poly->head || poly->head->exponent > exponent) {
        newNode->next = poly->head;
        poly->head = newNode;
        return;
    }
    Node* current = poly->head;
    while (current->next && current->next->exponent < exponent) {
        current = current->next;
    }
    if (current->next && current->next->exponent == exponent) {
        current->next->coefficient += coefficient;
        if (current->next->coefficient == 0) {
            Node* temp = current->next;
            current->next = current->next->next;
            free(temp);
        }
    } else {
        newNode->next = current->next;
        current->next = newNode;
    }
}

Polynomial add(Polynomial* poly1, Polynomial* poly2) {
```

```
    Polynomial result;
    result.head = NULL;
    Node* p1 = poly1->head;
    Node* p2 = poly2->head;
    while (p1 || p2) {
        if (p1 && (!p2 || p1->exponent < p2->exponent)) {
            insert(&result, p1->coefficient, p1->exponent);
            p1 = p1->next;
        } else if (p2 && (!p1 || p2->exponent < p1->exponent)) {
            insert(&result, p2->coefficient, p2->exponent);
            p2 = p2->next;
        } else {
            insert(&result, p1->coefficient + p2->coefficient, p1->exponent);
            p1 = p1->next;
            p2 = p2->next;
        }
    }
    return result;
}

void printPolynomial(Polynomial* poly) {
    if (!poly->head) {
        printf("0\n");
        return;
    }
    Node* current = poly->head;
    while (current) {
        printf("%dx^%d", current->coefficient, current->exponent);
        if (current->next) printf(" + ");
        current = current->next;
    }
    printf("\n");
}

void readPolynomial(Polynomial* poly) {
    poly->head = NULL;
    int coefficient, exponent;
    while (scanf("%d %d", &coefficient, &exponent) && !(coefficient == 0 &&
exponent == 0)) {
        insert(poly, coefficient, exponent);
    }
}
```

```
int main() {
    Polynomial poly1, poly2, result;
    readPolynomial(&poly1);
    readPolynomial(&poly2);
    result = add(&poly1, &poly2);
    printPolynomial(&poly1);
    printPolynomial(&poly2);
    printPolynomial(&result);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

2. Problem Statement

John is working on a math processing application, and his task is to simplify polynomials entered by users. The polynomial is represented as a linked list, where each node contains two properties:

Coefficient of the term.

Exponent of the term.

John's goal is to combine all the terms that have the same exponent, effectively simplifying the polynomial.

Input Format

The first line of input consists of an integer representing the number of terms in the polynomial.

The next n lines of input consist of two integers, representing the coefficient and exponent of the polynomial in each line separated by space.

Output Format

The first line of output prints the original polynomial in the format 'cx^e + cx^e + ...' (where c is the coefficient and e is the exponent of each term).

The second line of output displays the simplified polynomial in the same format as the original polynomial.

If the polynomial is 0, then only '0' will be printed.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
5 2
3 1
6 2

Output: Original polynomial: 5x^2 + 3x^1 + 6x^2
Simplified polynomial: 11x^2 + 3x^1

Answer

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

// Structure for a node in the linked list
typedef struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int coefficient, int exponent) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node into the linked list at the end
void insert(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
```

```
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to print a polynomial
void printPolynomial(Node* head) {
    if (!head) {
        printf("0\n");
        return;
    }
    while (head) {
        printf("%dx^%d", head->coefficient, head->exponent);
        if (head->next)
            printf(" + ");
        head = head->next;
    }
    printf("\n");
}

// Function to simplify the polynomial by combining like terms
Node* simplifyPolynomial(Node* head) {
    int terms[101] = {0}; // To store coefficients for each exponent
    Node* temp = head;

    while (temp) {
        terms[temp->exponent] += temp->coefficient;
        temp = temp->next;
    }

    Node* simplified = NULL;
    for (int i = 100; i >= 0; i--) {
        if (terms[i] != 0) {
            insert(&simplified, terms[i], i);
        }
    }

    if (!simplified) insert(&simplified, 0, 0); // If all terms canceled out
    return simplified;
}
```

```
int main() {
    int n, coefficient, exponent;
    scanf("%d", &n);
    Node* polynomial = NULL;

    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coefficient, &exponent);
        insert(&polynomial, coefficient, exponent);
    }

    printf("Original polynomial: ");
    printPolynomial(polynomial);

    Node* simplified = simplifyPolynomial(polynomial);
    printf("Simplified polynomial: ");
    printPolynomial(simplified);

    return 0;
}
```

Status : Wrong ***Marks : 0/10***

3. Problem Statement

Rani is studying polynomials in her class. She has learned about polynomial multiplication and is eager to try it out on her own. However, she finds the process of manually multiplying polynomials quite tedious. To make her task easier, she decides to write a program to multiply two polynomials represented as linked lists.

Help Rani by designing a program that takes two polynomials as input and outputs their product polynomial. Each polynomial is represented by a linked list of terms, where each term has a coefficient and an exponent. The terms are entered in descending order of exponents.

Input Format

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers each: the coefficient and the exponent of the term in the first polynomial.

The next line of input consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers each: the coefficient and the exponent of the term in the second polynomial.

Output Format

The first line of output prints the first polynomial.

The second line of output prints the second polynomial.

The third line of output prints the resulting polynomial after multiplying the given polynomials.

The polynomials should be displayed in the format, where each term is represented as ax^b, where a is the coefficient and b is the exponent.

Refer to the sample output for the exact format.

Sample Test Case

Input: 2
2 3
3 2
2
3 2
2 1

Output: 2x^3 + 3x^2
3x^2 + 2x
6x^5 + 13x^4 + 6x^3

Answer

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

// Structure for a node in the linked list
```

```
typedef struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int coefficient, int exponent) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node into the linked list at the end
void insert(Node** head, int coefficient, int exponent) {
    Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to print a polynomial
void printPolynomial(Node* head) {
    if (!head) {
        printf("0\n");
        return;
    }
    while (head) {
        if (head->exponent==1){
            printf("%dx", head->coefficient);
        }else if(head->exponent==0){
            printf("%d", head->coefficient);
        }
        else{
```

```
        printf("%dx^%d", head->coefficient, head->exponent);}
        if (head->next)
            printf(" + ");
        head = head->next;
    }
    printf("\n");
}

// Function to multiply two polynomials
Node* multiplyPolynomials(Node* poly1, Node* poly2) {
    if (!poly1 || !poly2) return NULL;

    int terms[200] = {0}; // To store coefficients for each exponent (max exponent
100+100)
    Node* temp1 = poly1;
    while (temp1) {
        Node* temp2 = poly2;
        while (temp2) {
            terms[temp1->exponent + temp2->exponent] += temp1->coefficient *
temp2->coefficient;
            temp2 = temp2->next;
        }
        temp1 = temp1->next;
    }

    Node* result = NULL;
    for (int i = 199; i >= 0; i--) {
        if (terms[i] != 0) {
            insert(&result, terms[i], i);
        }
    }

    if (!result) insert(&result, 0, 0); // If result is zero
    return result;
}

int main() {
    int n, m, coefficient, exponent;

    // Read first polynomial
    scanf("%d", &n);
    Node* poly1 = NULL;
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &coefficient, &exponent);
        insert(&poly1, coefficient, exponent);
    }

    // Read second polynomial
    scanf("%d", &m);
    Node* poly2 = NULL;
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &coefficient, &exponent);
        insert(&poly2, coefficient, exponent);
    }

    //printf("First polynomial: ");
    printPolynomial(poly1);
    //printf("Second polynomial: ");
    printPolynomial(poly2);

    Node* result = multiplyPolynomials(poly1, poly2);
    //printf("Resultant polynomial: ");
    printPolynomial(result);

    return 0;
}
```

Status : Partially correct **Marks : 5/10**

Rajalakshmi Engineering College

Name: Reshma AL
Email: 241901089@rajalakshmi.edu.in
Roll no: 241901089
Phone: 9884453399
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 1
Total Mark : 5
Marks Obtained : 4

Section 1 : Coding

1. Problem Statement

Write a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Input Format

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer

data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

Output Format

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
5

3
7
-1
2
11
Output: LINKED LIST CREATED
5 3 7

Answer

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

// Define the structure of a linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert at the end of the list
struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (!head) return newNode;

    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    return head;
}

// Function to insert at the beginning
struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = head;
    return newNode;
```

```
}

// Function to insert before a specific value
struct Node* insertBefore(struct Node* head, int value, int data) {
    if (!head) {
        printf("Value not found in the list\n");
        return head;
    }

    if (head->data == value) return insertAtBeginning(head, data);

    struct Node* temp = head;
    while (temp->next && temp->next->data != value) temp = temp->next;

    if (!temp->next) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}

// Function to insert after a specific value
struct Node* insertAfter(struct Node* head, int value, int data) {
    struct Node* temp = head;
    while (temp && temp->data != value) temp = temp->next;

    if (!temp) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}

// Function to delete from the beginning
```

```
struct Node* deleteFromBeginning(struct Node* head) {
    if (!head) {
        printf("The list is empty\n");
        return NULL;
    }

    struct Node* temp = head;
    head = head->next;
    free(temp);
    return head;
}

// Function to delete from the end
struct Node* deleteFromEnd(struct Node* head) {
    if (!head) {
        printf("The list is empty\n");
        return NULL;
    }

    if (!head->next) {
        free(head);
        return NULL;
    }

    struct Node* temp = head;
    while (temp->next->next) temp = temp->next;

    free(temp->next);
    temp->next = NULL;
    return head;
}

// Function to delete before a specific value
struct Node* deleteBefore(struct Node* head, int value) {
    if (!head || head->data == value || !head->next) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* prev = NULL, *current = head;
    while (current->next && current->next->data != value) {
        prev = current;
```

```
        current = current->next;
    }

    if (!current->next) {
        printf("Value not found in the list\n");
        return head;
    }

    if (!prev) {
        head = current->next;
        free(current);
    } else {
        prev->next = current->next;
        free(current);
    }

    return head;
}

// Function to delete after a specific value
struct Node* deleteAfter(struct Node* head, int value) {
    struct Node* temp = head;
    while (temp && temp->data != value) temp = temp->next;

    if (!temp || !temp->next) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
    return head;
}

// Function to print the linked list
void printList(struct Node* head) {
    if (!head) {
        printf("The list is empty\n");
        return;
    }
```

```
    struct Node* temp = head;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main function to handle user input
int main() {
    struct Node* head = NULL;
    int choice, data, value;

    while (1) {
        scanf("%d", &choice);

        switch (choice) {
            case 1: // Create linked list
                while (1) {
                    scanf("%d", &data);
                    if (data == -1) break;
                    head = insertAtEnd(head, data);
                }
                printf("LINKED LIST CREATED\n");
                break;

            case 2: // Display linked list
                printList(head);
                break;

            case 3: // Insert at beginning
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                printf("The linked list after insertion at the beginning is:\n");
                printList(head);
                break;

            case 4: // Insert at end
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                printf("The linked list after insertion at the end is:\n");
                printList(head);
```

```
            break;

        case 5: // Insert before a specific value
            scanf("%d %d", &value, &data);
            head = insertBefore(head, value, data);
            printf("The linked list after insertion before a value is:\n");
            printList(head);
            break;

        case 6: // Insert after a specific value
            scanf("%d %d", &value, &data);
            head = insertAfter(head, value, data);
            printf("The linked list after insertion after a value is:\n");
            printList(head);
            break;

        case 7: // Delete from beginning
            head = deleteFromBeginning(head);
            printf("The linked list after deletion from the beginning is:\n");
            printList(head);
            break;

        case 8: // Delete from end
            head = deleteFromEnd(head);
            printf("The linked list after deletion from the end is:\n");
            printList(head);
            break;

        case 9: // Delete before a specific value
            scanf("%d", &value);
            head = deleteBefore(head, value);
            printf("The linked list after deletion before a value is:\n");
            printList(head);
            break;

        case 10: // Delete after a specific value
            scanf("%d", &value);
            head = deleteAfter(head, value);
            printf("The linked list after deletion after a value is:\n");
            printList(head);
            break;
```

```
            case 11: // Exit the program
                return 0;

            default:
                printf("Invalid option! Please try again\n");
        }
    }

    return 0;
}
```

Status : Partially correct **Marks : 0.5/1**

2. Problem Statement

John is working on evaluating polynomials for his math project. He needs to compute the value of a polynomial at a specific point using a singly linked list representation.

Help John by writing a program that takes a polynomial and a value of x as input, and then outputs the computed value of the polynomial.

Example

Input:

2

13

12

11

1

Output:

36

Explanation:

The degree of the polynomial is 2.

Calculate the value of x2: 13 * 12 = 13.

Calculate the value of x1: 12 * 11 = 12.

Calculate the value of x0: 11 * 10 = 11.

Add the values of x2, x1 and x0 together: 13 + 12 + 11 = 36.

Input Format

The first line of input consists of the degree of the polynomial.

The second line consists of the coefficient x2.

The third line consists of the coefficient of x1.

The fourth line consists of the coefficient x0.

The fifth line consists of the value of x, at which the polynomial should be evaluated.

Output Format

The output is the integer value obtained by evaluating the polynomial at the given value of x.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
13
12
11
1

Output: 36

Answer

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

```
typedef struct Term{
    int coeff;
    int exp;
    struct Term* next;
} Term;
Term* createTerm(int coeff,int exp){
    Term* newTerm=(Term*)malloc(sizeof(Term));
    newTerm->coeff=coeff;
    newTerm->exp=exp;
    newTerm->next=NULL;
    return newTerm;
}
void insertTerm(Term** poly,int coeff,int exp){
    Term* newTerm=createTerm(coeff,exp);
    if(*poly==NULL || exp> (*poly)->exp){
        newTerm->next=*poly;
        *poly=newTerm;
    }else{
        Term* current=*poly;
        while(current->next!=NULL && exp <= current->next->exp){
            current=current->next;
        }
        newTerm->next=current->next;
        current->next=newTerm;
    }
}
int evaluatePolynomial(Term* poly,int x){
    int result=0;
    Term* current=poly;
    while(current!=NULL){
        result+=current->coeff * pow(x,current->exp);
        current=current->next;
    }
    return result;
}
int main(){
    int degree,coeff,x;
    Term* poly=NULL;
    scanf("%d",&degree);
    for(int i=degree;i>=0;i--){
        scanf("%d",&coeff);
        insertTerm(&poly,coeff,i);
```

```
    }
    scanf("%d",&x);
    int result=evaluatePolynomial(poly,x);
    printf("%d\n",result);
    Term* current=poly;
    while(current!=NULL){
        Term* temp=current;
        current=current->next;
        free(temp);
    }
    return 0;
}
```

Status : Correct **Marks : 1/1**

3. Problem Statement

Emily is developing a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Your task is to help Emily in implementing the same.

Input Format

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).

- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

Output Format

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
5
3
7
-1
2
11
Output: LINKED LIST CREATED
5 3 7

Answer

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

// Define the structure of a linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert at the end of the list
struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (!head) return newNode;

    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    return head;
}

// Function to insert at the beginning
struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = head;
    return newNode;
}

// Function to insert before a specific value
struct Node* insertBefore(struct Node* head, int value, int data) {
    if (!head) {
        printf("Value not found in the list\n");
        return head;
    }
```

```
    if (head->data == value) return insertAtBeginning(head, data);

    struct Node* temp = head;
    while (temp->next && temp->next->data != value) temp = temp->next;

    if (!temp->next) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}

// Function to insert after a specific value
struct Node* insertAfter(struct Node* head, int value, int data) {
    struct Node* temp = head;
    while (temp && temp->data != value) temp = temp->next;

    if (!temp) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}

// Function to delete from the beginning
struct Node* deleteFromBeginning(struct Node* head) {
    if (!head) {
        printf("The list is empty\n");
        return NULL;
    }

    struct Node* temp = head;
    head = head->next;
```

```
    free(temp);
    return head;
}

// Function to delete from the end
struct Node* deleteFromEnd(struct Node* head) {
    if (!head) {
        printf("The list is empty\n");
        return NULL;
    }

    if (!head->next) {
        free(head);
        return NULL;
    }

    struct Node* temp = head;
    while (temp->next->next) temp = temp->next;

    free(temp->next);
    temp->next = NULL;
    return head;
}

// Function to delete before a specific value
struct Node* deleteBefore(struct Node* head, int value) {
    if (!head || head->data == value || !head->next) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* prev = NULL, *current = head;
    while (current->next && current->next->data != value) {
        prev = current;
        current = current->next;
    }

    if (!current->next) {
        printf("Value not found in the list\n");
        return head;
    }
```

```
    if (!prev) {
        head = current->next;
        free(current);
    } else {
        prev->next = current->next;
        free(current);
    }

    return head;
}

// Function to delete after a specific value
struct Node* deleteAfter(struct Node* head, int value) {
    struct Node* temp = head;
    while (temp && temp->data != value) temp = temp->next;

    if (!temp || !temp->next) {
        printf("Value not found in the list\n");
        return head;
    }

    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
    return head;
}

// Function to print the linked list
void printList(struct Node* head) {
    if (!head) {
        printf("The list is empty\n");
        return;
    }

    struct Node* temp = head;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
// Main function to handle user input
int main() {
    struct Node* head = NULL;
    int choice, data, value;

    while (1) {
        scanf("%d", &choice);

        switch (choice) {
            case 1: // Create linked list
                while (1) {
                    scanf("%d", &data);
                    if (data == -1) break;
                    head = insertAtEnd(head, data);
                }
                printf("LINKED LIST CREATED\n");
                break;

            case 2: // Display linked list
                printList(head);
                break;

            case 3: // Insert at beginning
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                printf("The linked list after insertion at the beginning is:\n");
                printList(head);
                break;

            case 4: // Insert at end
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                printf("The linked list after insertion at the end is:\n");
                printList(head);
                break;

            case 5: // Insert before a specific value
                scanf("%d %d", &value, &data);
                head = insertBefore(head, value, data);
                printf("The linked list after insertion before a value is:\n");
                printList(head);
                break;
```

```
        case 6: // Insert after a specific value
            scanf("%d %d", &value, &data);
            head = insertAfter(head, value, data);
            printf("The linked list after insertion after a value is:\n");
            printList(head);
            break;

        case 7: // Delete from beginning
            head = deleteFromBeginning(head);
            printf("The linked list after deletion from the beginning is:\n");
            printList(head);
            break;

        case 8: // Delete from end
            head = deleteFromEnd(head);
            printf("The linked list after deletion from the end is:\n");
            printList(head);
            break;

        case 9: // Delete before a specific value
            scanf("%d", &value);
            head = deleteBefore(head, value);
            printf("The linked list after deletion before a value is:\n");
            printList(head);
            break;

        case 10: // Delete after a specific value
            scanf("%d", &value);
            head = deleteAfter(head, value);
            printf("The linked list after deletion after a value is:\n");
            printList(head);
            break;

        case 11: // Exit the program
            return 0;

        default:
            printf("Invalid option! Please try again\n");
    }
}
```

```
    return 0;
}
```

Status : Partially correct ***Marks : 0.5/1***

4. Problem Statement

Imagine you are managing the backend of an e-commerce platform. Customers place orders at different times, and the orders are stored in two separate linked lists. The first list holds the orders from morning, and the second list holds the orders from the evening.

Your task is to merge the two lists so that the final list holds all orders in sequence from the morning list followed by the evening orders, in the same order

Input Format

The first line contains an integer n , representing the number of orders in the morning list.

The second line contains n space-separated integers representing the morning orders.

The third line contains an integer m , representing the number of orders in the evening list.

The fourth line contains m space-separated integers representing the evening orders.

Output Format

The output should be a single line containing space-separated integers representing the merged order list, with morning orders followed by evening orders.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
101 102 103
2
104 105

Output: 101 102 103 104 105

Answer

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

// Node structure for linked list
struct Node {
    int order_id;
    struct Node* next;
};

// Function to insert at the end of the list
struct Node* insertAtEnd(struct Node* head, int order_id) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->order_id = order_id;
    newNode->next = NULL;

    if (!head) return newNode; // If list is empty, new node becomes head

    struct Node* temp = head;
    while (temp->next) temp = temp->next; // Traverse to last node
    temp->next = newNode;

    return head;
}

// Function to merge two lists (morning -> evening)
struct Node* mergeLists(struct Node* morning, struct Node* evening) {
    if (!morning) return evening; // If morning list is empty, return evening
    struct Node* temp = morning;
    while (temp->next) temp = temp->next; // Traverse to last node of morning
    temp->next = evening; // Connect morning list to evening list
    return morning;
}

// Function to print the merged list
void printList(struct Node* head) {
```

```
    struct Node* temp = head;
    while (temp) {
        printf("%d ", temp->order_id);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int n, m, order_id;
    struct Node *morning = NULL, *evening = NULL;

    // Read morning orders
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &order_id);
        morning = insertAtEnd(morning, order_id);
    }

    // Read evening orders
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &order_id);
        evening = insertAtEnd(evening, order_id);
    }

    // Merge the two lists
    struct Node* mergedList = mergeLists(morning, evening);

    // Print the merged order list
    printList(mergedList);

    return 0;
}
```

Status : Correct **Marks : 1/1**

5. Problem Statement

Bharath is very good at numbers. As he is piled up with many works, he decides to develop programs for a few concepts to simplify his work. As a

first step, he tries to arrange even and odd numbers using a linked list. He stores his values in a singly-linked list.

Now he has to write a program such that all the even numbers appear before the odd numbers. Finally, the list is printed in such a way that all even numbers come before odd numbers. Additionally, the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Example

Input:

6

3 1 0 4 30 12

Output:

12 30 4 0 3 1

Explanation:

Even elements: 0 4 30 12

Reversed Even elements: 12 30 4 0

Odd elements: 3 1

So the final list becomes: 12 30 4 0 3 1

Input Format

The first line consists of an integer n representing the size of the linked list.

The second line consists of n integers representing the elements separated by space.

Output Format

The output prints the rearranged list separated by a space.

The list is printed in such a way that all even numbers come before odd numbers and the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 6
3 1 0 4 30 12

Output: 12 30 4 0 3 1

Answer

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Num{
   int value;
   struct Num* next;
}Node;
Node* newnode(int value){
   Node* node=(Node*) malloc(sizeof(Node));
   node->value=value;
   node->next=NULL;
   return node;
}
void insertNode(Node** head,int value){
   Node* temp=*head;
   if(temp==NULL){
      *head=newnode(value);
   return;
   }
   while(temp->next!=NULL){
      temp=temp->next;
   }
   temp->next=newnode(value);
}
Node* reverseNode(Node* head){
   Node* reverse=NULL;
   while(head!=NULL){
      Node* node=newnode(head->value);
      node->next=reverse;
      reverse=node;
      head=head->next;
```

```
    }
    return reverse;
}
void traverse(Node* head){
    while(head!=NULL){
        printf("%d ",head->value);
        head=head->next;
    }
    printf("\n");
}
void merge(Node* head1,Node* head2){
    while(head1->next!=NULL){
        head1=head1->next;
    }
    head1->next=head2;
}
int main(){
    int n,value;
    Node* head=NULL;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&value);
        insertNode(&head,value);
    }
    Node* temp=head;
    Node* odd=NULL;
    Node* even=NULL;
    while(temp!=NULL){
        if(temp->value%2==1){
            insertNode(&odd,temp->value);
        }else{
            insertNode(&even,temp->value);
        }
        temp=temp->next;
    }
    even=reverseNode(even);
    merge(even,odd);
    traverse(even);
}
```

Status : Correct *Marks : 1/1*