Rajalakshmi Engineering College

Name: JEENESHWAAR .S
Email: 241501073@rajalakshmi.edu.in
Roll no: 241501073
Phone: 9884283976
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

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>
struct node{
   int a;
   int b;
   struct node*next;
};
void insertatend(struct node**head,int a,int b){
   struct node*newnode=(struct node*)malloc(sizeof(struct node));
   newnode->a=a;
   newnode->b=b;
   newnode->next=NULL;
   struct node*c=*head;
   if(*head==NULL){
      *head=newnode;
   }
   else{
      while(c->next!=NULL){
         c=c->next;
      }
```

```
        c->next=newnode;
    }
}
void traverse(struct node*head){
    int s=0;
    struct node*c=head;
    while(c!=NULL){
        s+=c->a;
        c=c->next;
        }
    printf("%d",s);
}
int main(){
    int a;
    int b;
    struct node*head=NULL;
    for(int i=0;i<2;i++){
        int n;
        scanf("%d",&n);
        for(int j=0;j<n;j++){
            scanf("%d %d",&a,&b);
            insertatend(&head,a,b);
        }
    }
    traverse(head);

}
```

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

Rajalakshmi Engineering College

Name: JEENESHWAAR .S
Email: 241501073@rajalakshmi.edu.in
Roll no: 241501073
Phone: 9884283976
Branch: REC
Department: I AI & ML FA
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 1
Total Mark : 5
Marks Obtained : 5

Section 1 : Coding

1. 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>
```

```c
#include <stdlib.h>

void rearrange_list(int n, int arr[]) {
    int even_numbers[n], odd_numbers[n];
    int even_count = 0, odd_count = 0;


    for (int i = 0; i < n; i++) {
        if (arr[i] % 2 == 0) {
            even_numbers[even_count++] = arr[i];
        } else {
            odd_numbers[odd_count++] = arr[i];
        }
    }


    for (int i = even_count - 1; i >= 0; i--) {
        printf("%d ", even_numbers[i]);
    }


    for (int i = 0; i < odd_count; i++) {
        printf("%d ", odd_numbers[i]);
    }

    printf("\n");
}

int main() {
    int n;


    scanf("%d", &n);

    int arr[n];


    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
```

```
    rearrange_list(n, arr);

    return 0;
}
```

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

2. 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>

// Define the structure for a linked list node
typedef struct Node {
    int order_id;
    struct Node* next;
} Node;

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

// Function to append a node to the end of a linked list
void append(Node** head, int order_id) {
    Node* newNode = createNode(order_id);
    if (*head == NULL) {
        *head = newNode;  // If list is empty, new node becomes the head
    } else {
        Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;  // Traverse to the last node
        }
        temp->next = newNode;  // Append the new node at the end
    }
}

// Function to merge two linked lists
Node* mergeLists(Node* morningHead, Node* eveningHead) {
```

```
    if (morningHead == NULL) return eveningHead;
    if (eveningHead == NULL) return morningHead;

    // Find the last node of the morning list
    Node* temp = morningHead;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    // Attach the evening list to the last node of the morning list
    temp->next = eveningHead;

    return morningHead;
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->order_id);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int n, m;

    scanf("%d", &n);
    Node* morningHead = NULL;


    for (int i = 0; i < n; i++) {
        int order_id;
        scanf("%d", &order_id);
        append(&morningHead, order_id);
    }


    scanf("%d", &m);
    Node* eveningHead = NULL;
```

```
    for (int i = 0; i < m; i++) {
        int order_id;
        scanf("%d", &order_id);
        append(&eveningHead, order_id);
    }

    Node* mergedHead = mergeLists(morningHead, eveningHead);

    printList(mergedHead);

    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>

typedef struct node{
   int data;
   struct node* next;
}node;

node* create()
{
   node* head=NULL,*temp=NULL,*newnode;
   int value;
   while(1){
      scanf("%d",&value);
      if(value==-1)
      break;

      newnode=(node*)malloc(sizeof(node));
      newnode->data=value;
      newnode->next=NULL;
      if(head==NULL){
         head=newnode;
         temp=head;

      }else{
         temp->next=newnode;
         temp=temp->next;
      }

   }
   return head;
}
void display(node* head)
{
   if(head==NULL){
      printf("The list is empty");
   }
```

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

node* insertbeg(node* head,int value){
    node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=head;
    return newnode;

}

node* insertend(node* head,int value)
{
    node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=NULL;
    if(head==NULL)
    {
        return newnode;
    }
    node* temp=head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=newnode;
    return head;

}

node* insertbefval(node* head,int value,int newdata){
    node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=newdata;

    if(head==NULL) return head;

    if(head->data==value){
```

```
        newnode->next=head;
        return newnode;
    }

    node* temp=head;
    while(temp->next!=NULL && temp->next->data!=value){

        temp=temp->next;
    }
    if(temp->next!=NULL){
        newnode->next=temp->next;
        temp->next=newnode;

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

node* insertaftval(node* head,int value,int newdata)
{

    node* temp=head;
    while(temp!=NULL && temp->data!=value)
    {
        temp=temp->next;

    }
    if(temp!=NULL){
        node* newnode=(struct node*)malloc(sizeof(struct node));
        newnode->data=newdata;
        newnode->next=temp->next;
        temp->next=newnode;
    }
    else{
        printf("Value not found in the list\n");
    }
    return head;
}

node* deletebeg(node* head)
{
```

```
    if(head==NULL)
    {
        return NULL;
    }
    node* temp=head;
    head=head->next;
    free(temp);
    return head;
}

node* deleteend(node* head){
    if(head==NULL){
        return NULL;
    }
    if(head->next==NULL){
        free(head);
        return NULL;
    }

    node* temp=head;
    while(temp->next->next!=NULL)
    {
        temp=temp->next;
    }
    free(temp->next);
    temp->next=NULL;
    return head;

}
node* deletebefore(node* head,int value){
    if(head==NULL || head->next == NULL || head->next->next==NULL){
        return head;
    }

    node* prev2=NULL;
    node* prev=NULL;
    node* curr=head;

    while(curr->next!=NULL){
        if(curr->next->data==value){
            if(prev2!=NULL){
                node* temp=prev2->next;
```

```
            prev2->next=prev->next;
            free (temp);
            return head;
        }
        else{
            node* temp=head;
            head=head->next;
            free(temp);
            return head;
        }
    }
    prev2=prev;
    prev=curr;
    curr=curr->next;

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

node* deleteafter(node* head,int value)
{
  node* temp=head;
  while(temp!=NULL && temp->data!=value){
    temp=temp->next;
  }
  if(temp!=NULL && temp->next!=NULL){
    node* delnode=temp->next;
    temp->next=delnode->next;
    free(delnode);
  }
    return head;
}
void freelist(node* head){
  node* temp;
  while(head!=NULL){
    temp=head;
    head=head->next;
    free(temp);
  }
}
```

```
int main()
{
    node* head=NULL;
    int choice,value,newvalue;

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

        switch(choice){
            case 1:
            head=create();
            printf("LINKED LIST CREATED\n");
            break;
            case 2:
            display(head);
            break;
            case 3:
            scanf("%d",&value);
            head=insertbeg(head,value);
            printf("The linked list after insertion at the beginning is:\n");
            display(head);
            break;
            case 4:
            scanf("%d",&value);
            head=insertend(head,value);
            printf("The linked list after insertion at the end is:\n");
            display(head);
            break;
            case 5:
            scanf("%d %d",&value,&newvalue);
            head=insertbefval(head,value,newvalue);
            printf("The linked list after insertion before a value is:\n");
            display(head);
            break;
            case 6:
            scanf("%d %d",&value,&newvalue);
            head=insertaftval(head,value,newvalue);
            printf("The linked list after insertion after a value is:\n");
            display(head);
            break;
            case 7:
            head=deletebeg(head);
```

```
        printf("The linked list after deletion from the beginning is:\n");
        display(head);
        break;
        case 8:
        head=deleteend(head);
        printf("The linked list after deletion from the end is:\n");
        display(head);
        break;
        case 9:
        scanf("%d",&value);
        head=deletebefore(head,value);
        printf("The linked list after deletion before a value is:\n");
        display(head);
        break;
        case 10:
        scanf("%d",&value);
        head=deleteafter(head,value);
        printf("The linked list after deletion after a value is:\n");
        display(head);
        break;
        case 11:
        return 0;
        freelist(head);
        default:
        printf("Invalid option! Please try again\n");
    }
  }
 return 0;
}
```

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

4. 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>

struct Node {
    int coeff;
    int exp;
    struct Node* next;
};

struct Node* createNode(int coeff, int exp) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->coeff = coeff;
    node->exp = exp;
    node->next = NULL;
    return node;
}

void appendNode(struct Node** head, int coeff, int exp) {
    struct Node* newNode = createNode(coeff, exp);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
```

```
    temp->next = newNode;
}

int evaluatePolynomial(struct Node* head, int x) {
    int result = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        result += temp->coeff * (int)pow(x, temp->exp);
        temp = temp->next;
    }
    return result;
}

int main() {
    int degree, coeff, x;
    scanf("%d", &degree);
    struct Node* poly = NULL;
    for (int i = degree; i >= 0; i--) {
        scanf("%d", &coeff);
        appendNode(&poly, coeff, i);
    }
    scanf("%d", &x);
    int result = evaluatePolynomial(poly, x);
    printf("%d\n", result);
    return 0;
}
```

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

5. 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>

typedef struct node{
   int data;
   struct node* next;
}node;

node* create()
{
   node* head=NULL,*temp=NULL,*newnode;
   int value;
   while(1){
      scanf("%d",&value);
      if(value==-1)
      break;

      newnode=(node*)malloc(sizeof(node));
      newnode->data=value;
      newnode->next=NULL;
      if(head==NULL){
         head=newnode;
         temp=head;

      }else{
         temp->next=newnode;
         temp=temp->next;
      }
```

```
    }
    return head;
}
void display(node* head)
{
    if(head==NULL){
        printf("The list is empty");
    }
    node* temp=head;
    while(temp!=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}

node* insertbeg(node* head,int value){
    node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=head;
    return newnode;

}

node* insertend(node* head,int value)
{
    node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=NULL;
    if(head==NULL)
    {
        return newnode;
    }
    node* temp=head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=newnode;
    return head;
```

```
}
node* insertbefval(node* head,int value,int newdata){
    node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=newdata;

    if(head==NULL) return head;

    if(head->data==value){
        newnode->next=head;
        return newnode;
    }

    node* temp=head;
    while(temp->next!=NULL && temp->next->data!=value){

        temp=temp->next;
    }
    if(temp->next!=NULL){
        newnode->next=temp->next;
        temp->next=newnode;

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

node* insertaftval(node* head,int value,int newdata)
{

    node* temp=head;
    while(temp!=NULL && temp->data!=value)
    {
        temp=temp->next;

    }
    if(temp!=NULL){
        node* newnode=(struct node*)malloc(sizeof(struct node));
        newnode->data=newdata;
        newnode->next=temp->next;
        temp->next=newnode;
```

```
    }
    else{
        printf("Value not found in the list\n");
    }
    return head;
}

node* deletebeg(node* head)
{
    if(head==NULL)
    {
        return NULL;
    }
    node* temp=head;
    head=head->next;
    free(temp);
    return head;
}

node* deleteend(node* head){
    if(head==NULL){
        return NULL;
    }
    if(head->next==NULL){
        free(head);
        return NULL;
    }

    node* temp=head;
    while(temp->next->next!=NULL)
    {
        temp=temp->next;
    }
    free(temp->next);
    temp->next=NULL;
    return head;

}
node* deletebefore(node* head,int value){
    if(head==NULL || head->next == NULL || head->next->next==NULL){
        return head;
    }
```

```
    node* prev2=NULL;
    node* prev=NULL;
    node* curr=head;

    while(curr->next!=NULL){
        if(curr->next->data==value){
            if(prev2!=NULL){
                node* temp=prev2->next;
                prev2->next=prev->next;
                free (temp);
                return head;
            }
            else{
                node* temp=head;
                head=head->next;
                free(temp);
                return head;
            }
        }
        prev2=prev;
        prev=curr;
        curr=curr->next;

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

node* deleteafter(node* head,int value)
{
    node* temp=head;
    while(temp!=NULL && temp->data!=value){
        temp=temp->next;
    }
    if(temp!=NULL && temp->next!=NULL){
        node* delnode=temp->next;
        temp->next=delnode->next;
        free(delnode);
    }
        return head;
}
```

```
void freelist(node* head){
   node* temp;
   while(head!=NULL){
      temp=head;
      head=head->next;
      free(temp);
   }
}

int main()
{
   node* head=NULL;
   int choice,value,newvalue;

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

      switch(choice){
         case 1:
         head=create();
         printf("LINKED LIST CREATED\n");
         break;
         case 2:
         display(head);
         break;
         case 3:
         scanf("%d",&value);
         head=insertbeg(head,value);
         printf("The linked list after insertion at the beginning is:\n");
         display(head);
         break;
         case 4:
         scanf("%d",&value);
         head=insertend(head,value);
         printf("The linked list after insertion at the end is:\n");
         display(head);
         break;
         case 5:
         scanf("%d %d",&value,&newvalue);
         head=insertbefval(head,value,newvalue);
         printf("The linked list after insertion before a value is:\n");
         display(head);
```

```
            break;
            case 6:
            scanf("%d %d",&value,&newvalue);
            head=insertaftval(head,value,newvalue);
            printf("The linked list after insertion after a value is:\n");
            display(head);
            break;
            case 7:
            head=deletebeg(head);
            printf("The linked list after deletion from the beginning is:\n");
            display(head);
            break;
            case 8:
            head=deleteend(head);
            printf("The linked list after deletion from the end is:\n");
            display(head);
            break;
            case 9:
            scanf("%d",&value);
            head=deletebefore(head,value);
            printf("The linked list after deletion before a value is:\n");
            display(head);
            break;
            case 10:
            scanf("%d",&value);
            head=deleteafter(head,value);
            printf("The linked list after deletion after a value is:\n");
            display(head);
            break;
            case 11:
            return 0;
            freelist(head);
            default:
            printf("Invalid option! Please try again\n");
        }
    }
    return 0;
}
```

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