Rajalakshmi Engineering College

Name: Seeralan .M
Email: 241501193@rajalakshmi.edu.in
Roll no: 241501193
Phone: 8610861705
Branch: REC
Department: I AIML AE
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

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

2415011193

12
11
1

Output: 36

Answer

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
typedef struct Node{
    int c,e;
    struct Node* next;
}Node;
Node* insert(Node* h,int a,int b){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->c = a;
    newNode->e = b;
    newNode->next == NULL;
    if(h==NULL){
        return newNode;
    }
    Node* t=h;
    while(t->next!=NULL){
        t = t->next;
    }
    t->next=newNode;
    return h;
}
int result(Node* h,int n){
    Node* t=h;
    int s=0;
    while(t!=NULL){
        s+=((t->c) * (pow(n,t->e)));
        t=t->next;
    }
    return s;
}
int main(){
    int n;
    scanf("%d",&n);
    Node* h=NULL;
```

```
    for(int i=n;i>=0;i--){
        int v;
        scanf("%d",&v);
        h = insert(h,v,i);
    }
    int t;
    scanf("%d",&t);
    printf("%d",result(h,t));
    return 0;
}
```

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

2. 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

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
    int d;
    struct Node* next;
}Node;
Node* end(Node* h, int value){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->d = value;
    newNode->next = NULL;
    if(h==NULL) return newNode;
```

```
    Node* t=h;
    while(t->next!=NULL){
        t=t->next;
    }
    t->next = newNode;
    return h;
}
Node* head(Node* h,int value){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->d = value;
    newNode->next = h;
    h = newNode;
    return h;
}
void printList(Node* h1, Node* h2){
    Node* t1 = h1;
    Node* t2 = h2;
    while(t1!=NULL){
        printf("%d ",t1->d);
        t1 = t1->next;
    }
    while(t2!=NULL){
        printf("%d ",t2->d);
        t2 = t2->next;
    }
}
int main(){
    int n;
    scanf("%d",&n);
    Node* even=NULL;
    Node* odd=NULL;
    for(int i=0;i<n;i++){
        int v;
        scanf("%d",&v);
        if(v%2==0){
            even = head(even,v);
        }
        else{
            odd = end(odd,v);
        }
    }
    printList(even,odd);
```

241501193

```
    return 0;
}
```

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

3. 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

```
// You are using GCC
#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;
}
```

2415011193

```
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* deletend(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* deletafter(node* head,int value)
{
    node* temp=head;
    while(temp!=NULL && temp->data!=value){
        temp=temp->next;
```

241501193

```
    }
    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=deletend(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=deletafter(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");
```

241501193

```
    }
  }
  return 0;
}
```

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

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

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
    int data;
    struct Node* next;
}Node;
Node* insert(Node* h,int value){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    if(h==NULL){
        return newNode;
    }
    Node* t=h;
    while(t->next!=NULL){
        t = t->next;
    }
    t->next = newNode;
    return h;
}
void printList(Node* h1,Node* h2){
    Node* t1=h1;
    Node* t2=h2;
    while(t1!=NULL){
        printf("%d",t1->data);
        t1 = t1->next;
    }
    while(t2!=NULL){
        printf("%d",t2->data);
        t2 = t2->next;
    }
}
```

```
int main(){
    int n1;
    scanf("%d",&n1);
    Node* h1=NULL;
    for(int i=0;i<n1;i++){
        int v;
        scanf("%d",&v);
        h1 = insert(h1,v);
    }
    int n2;
    scanf("%d",&n2);
    Node* h2=NULL;
    for(int i=0;i<n2;i++){
        int v;
        scanf("%d",&v);
        h2 = insert(h2,v);
    }
    printList(h1,h2);
    return 0;
}
```

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

5. 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

2415011 93

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;
```

241501193

```
        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* deletend(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* deletafter(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=deletend(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=deletafter(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***

Rajalakshmi Engineering College

Name: Seeralan .M
Email: 241501193@rajalakshmi.edu.in
Roll no: 241501193
Phone: 8610861705
Branch: REC
Department: I AIML AE
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_MCQ

Attempt : 1
Total Mark : 10
Marks Obtained : 10

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. 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***

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. 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***

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

Answer

Binary search

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

6. 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 -> 16 -> 22 -> 45 -> 16

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

7. 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**

8. 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**

9. 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***

10. 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***

241501193

Rajalakshmi Engineering College

Name: Seeralan .M
Email: 241501193@rajalakshmi.edu.in
Roll no: 241501193
Phone: 8610861705
Branch: REC
Department: I AIML AE
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_week 1_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 30

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

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

struct PolynomialTerm {
    int coefficient;
    int exponent;
```

```
    struct PolynomialTerm* next;
};

typedef struct PolynomialTerm PolynomialTerm;

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

void insertTerm(PolynomialTerm** poly, int coefficient, int exponent) {
    PolynomialTerm* newNode = createTermNode(coefficient, exponent);
    if (*poly == NULL) {
        *poly = newNode;
        return;
    }

    if (exponent < (*poly)->exponent) {
        newNode->next = *poly;
        *poly = newNode;
        return;
    }

    PolynomialTerm* current = *poly;
    PolynomialTerm* prev = NULL;
    while (current && current->exponent <= exponent) {
        prev = current;
        current = current->next;
    }
    prev->next = newNode;
    newNode->next = current;
}

PolynomialTerm* addPolynomials(PolynomialTerm* poly1, PolynomialTerm*
poly2) {
    PolynomialTerm* result = NULL;
    PolynomialTerm* tail = NULL;
```

```
    while (poly1 || poly2) {
        int coeff = 0;
        int exp = 0;

        if (poly1 && poly2) {
            if (poly1->exponent == poly2->exponent) {
                coeff = poly1->coefficient + poly2->coefficient;
                exp = poly1->exponent;
                poly1 = poly1->next;
                poly2 = poly2->next;
            } else if (poly1->exponent > poly2->exponent) {
                coeff = poly1->coefficient;
                exp = poly1->exponent;
                poly1 = poly1->next;
            } else {
                coeff = poly2->coefficient;
                exp = poly2->exponent;
                poly2 = poly2->next;
            }
        } else if (poly1) {
            coeff = poly1->coefficient;
            exp = poly1->exponent;
            poly1 = poly1->next;
        } else {
            coeff = poly2->coefficient;
            exp = poly2->exponent;
            poly2 = poly2->next;
        }

        if (coeff != 0) {
            insertTerm(&result, coeff, exp);
        }
    }

    return result;
}

void printPolynomial(PolynomialTerm* polynomial) {
    if (polynomial == NULL) {
        printf("0");
        return;
    }
```

```
    while (polynomial != NULL) {
        if (polynomial->coefficient != 0) {
            printf("%dx^%d", polynomial->coefficient, polynomial->exponent);
            if (polynomial->next != NULL) {
                printf(" + ");
            }
        }
        polynomial = polynomial->next;
    }
    printf("\n");
}

void freePolynomial(PolynomialTerm* polynomial) {
    while (polynomial != NULL) {
        PolynomialTerm* temp = polynomial;
        polynomial = polynomial->next;
        free(temp);
    }
}

int main() {
    int coefficient, exponent;
    PolynomialTerm* poly1 = NULL;
    PolynomialTerm* poly2 = NULL;

    while (1) {
        scanf("%d %d", &coefficient, &exponent);
        if (coefficient == 0 && exponent == 0) {
            break;
        }
        insertTerm(&poly1, coefficient, exponent);
    }

    while (1) {
        scanf("%d %d", &coefficient, &exponent);
        if (coefficient == 0 && exponent == 0) {
            break;
        }
        insertTerm(&poly2, coefficient, exponent);
    }
```

```
    printPolynomial(poly1);
    printPolynomial(poly2);

    PolynomialTerm* result = addPolynomials(poly1, poly2);
    printPolynomial(result);

    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(result);

    return 0;
}
```

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

2. Problem Statement

Keerthi is a tech enthusiast and is fascinated by polynomial expressions. She loves to perform various operations on polynomials.

Today, she is working on a program to multiply two polynomials and delete a specific term from the result.

Keerthi needs your help to implement this program. She wants to take the coefficients and exponents of the terms of the two polynomials as input, perform the multiplication, and then allow the user to specify an exponent for deletion from the resulting polynomial, and display the result.

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 representing the coefficient and the exponent of the term in the first polynomial.

The next line 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 representing the coefficient and the exponent of the term in the second polynomial.

The last line consists of an integer, representing the exponent of the term that Keerthi wants to delete from the multiplied polynomial.

Output Format

The first line of output displays the resulting polynomial after multiplication.

The second line displays the resulting polynomial after deleting the specified term.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
2 2
3 1
4 0
2
1 2
2 1
2
Output: Result of the multiplication: 2x^4 + 7x^3 + 10x^2 + 8x
Result after deleting the term: 2x^4 + 7x^3 + 8x

Answer

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

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

struct Node* createNode(int coefficient, int exponent) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}

void insertTerm(struct Node** head, int coefficient, int exponent) {
    if (coefficient == 0) {
        return; // Skip inserting terms with coefficient 0
    }

    if (*head == NULL) {
        *head = createNode(coefficient, exponent);
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = createNode(coefficient, exponent);
    }
}

void displayPolynomial(struct Node* head) {
    struct Node* temp = head;
    int isFirst = 1;
    while (temp != NULL) {
        if (temp->coefficient != 0) {
            if (!isFirst && temp->coefficient > 0) {
                printf(" + ");
            }
            if (temp->exponent == 0) {
                printf("%d", temp->coefficient);
            } else if (temp->exponent == 1) {
                printf("%dx", temp->coefficient);
            } else {
                printf("%dx^%d", temp->coefficient, temp->exponent);
            }
            isFirst = 0;
        }
        temp = temp->next;
    }
    printf("\n");
```

241501193

```
}

void removeDuplicates(struct Node* head) {
    struct Node* ptr1, *ptr2, *dup;
    ptr1 = head;

    while (ptr1 != NULL && ptr1->next != NULL) {
        ptr2 = ptr1;

        while (ptr2->next != NULL) {
            if (ptr1->exponent == ptr2->next->exponent) {
                ptr1->coefficient = ptr1->coefficient + ptr2->next->coefficient;
                dup = ptr2->next;
                ptr2->next = ptr2->next->next;
                free(dup);
            } else {
                ptr2 = ptr2->next;
            }
        }
        ptr1 = ptr1->next;
    }
}

struct Node* multiplyPolynomials(struct Node* poly1, struct Node* poly2) {
    struct Node* result = NULL;
    struct Node* temp1 = poly1;

    while (temp1 != NULL) {
        struct Node* temp2 = poly2;
        while (temp2 != NULL) {
            int coeff = temp1->coefficient * temp2->coefficient;
            int exp = temp1->exponent + temp2->exponent;
            insertTerm(&result, coeff, exp);
            temp2 = temp2->next;
        }
        temp1 = temp1->next;
    }

    removeDuplicates(result);
    return result;
}
```

2415011193

```
void deleteTerm(struct Node** head, int exponent) {
    struct Node* prev = NULL;
    struct Node* curr = *head;

    while (curr != NULL) {
        if (curr->exponent == exponent) {
            if (prev == NULL) {
                *head = curr->next;
            } else {
                prev->next = curr->next;
            }
            free(curr);
            return;
        }
        prev = curr;
        curr = curr->next;
    }
}

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

void addSamePowerCoefficients(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        struct Node* current = temp->next;
        while (current != NULL) {
            if (temp->exponent == current->exponent) {
                temp->coefficient += current->coefficient;
                deleteTerm(&head, current->exponent);
                current = temp->next; // Start again from the next node
            } else {
                current = current->next;
            }
        }
        temp = temp->next;
    }
```

241501193

```
}

int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;
    struct Node* result = NULL;

    int coefficient, exponent;
    int numTerms1, numTerms2;

    scanf("%d", &numTerms1);

    for (int i = 0; i < numTerms1; i++) {
        scanf("%d", &coefficient);
        scanf("%d", &exponent);
        insertTerm(&poly1, coefficient, exponent);
    }
    scanf("%d", &numTerms2);

    for (int i = 0; i < numTerms2; i++) {
        scanf("%d", &coefficient);
        scanf("%d", &exponent);
        insertTerm(&poly2, coefficient, exponent);
    }

    result = multiplyPolynomials(poly1, poly2);

    printf("Result of the multiplication: ");
    displayPolynomial(result);



    scanf("%d", &exponent);
    deleteTerm(&result, exponent);

    printf("Result after deleting the term: ");
    displayPolynomial(result);

    // Free memory used by the linked lists
    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(result);
```

241501193

```
    return 0;
}
```

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

3. Problem Statement

Timothy wants to evaluate polynomial expressions for his mathematics homework. He needs a program that allows him to input the coefficients of a polynomial based on its degree and compute the polynomial's value for a given input of x. Implement a function that takes the degree, coefficients, and the value of x, and returns the evaluated result of the polynomial.

Example

Input:

degree of the polynomial = 2

coefficient of x2 = 13

coefficient of x1 = 12

coefficient of x0 = 11

x = 1

Output:

36

Explanation:

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

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

Calculate the value of 11x0: 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 an integer representing the degree of the polynomial.

The second line consists of an integer representing the coefficient of x2.

The third line consists of an integer representing the coefficient of x1.

The fourth line consists of an integer representing the coefficient of x0.

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

Output Format

The output is an 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

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

int evaluatePolynomial(int degree, int coefficients[], int x) {
    int result = 0;
    for (int i = 0; i <= degree; i++) {
        result += coefficients[i] * pow(x, degree - i);
    }
    return result;
}
```

2415011 93

```
int main() {
    int degree, x;

    // Taking input for the degree of the polynomial

    scanf("%d", &degree);

    int coefficients[degree + 1];

    // Taking input for coefficients
    for (int i = 0; i <= degree; i++) {

        scanf("%d", &coefficients[i]);
    }

    // Taking input for x

    scanf("%d", &x);

    // Evaluating the polynomial
    int result = evaluatePolynomial(degree, coefficients, x);

    // Displaying the result
    printf(" %d\n", result);

    return 0;
}
```

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

2415011193

Rajalakshmi Engineering College

Name: Seeralan .M
Email: 241501193@rajalakshmi.edu.in
Roll no: 241501193
Phone: 8610861705
Branch: REC
Department: I AIML AE
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

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
typedef struct poly{
    int coeff;
    int expon;
    struct poly* next;
}Node;
Node* newnode(int coeff,int expon){
    Node* new_node=(Node*)malloc(sizeof(Node));
    new_node->coeff = coeff;
    new_node->expon = expon;
    new_node->next = NULL;
    return new_node;
}
void insertNode(Node** head,int coeff,int expon){
    Node* temp = *head;
    if(temp == NULL){
        *head = newnode(coeff,expon);
        return;
```

```
    }
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = newnode(coeff,expon);
}
int main()
{
    int n,coeff,expon;
    scanf("%d",&n);
    Node* poly1;
    Node* poly2;
    for(int i=0;i<n;i++)
    {
        scanf("%d%d",&coeff,&expon);
        insertNode(&poly1, coeff, expon);
    }
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d%d",&coeff,&expon);
        insertNode(&poly2, coeff, expon);
    }
    int sum=0;
    while(poly1 != NULL)
    {
        sum += poly1->coeff;
        poly1 = poly1->next;
    }
    while(poly2 != NULL)
    {
        sum += poly2->coeff;
        poly2 = poly2->next;
    }
    printf("%d",sum);
}
```

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