Rajalakshmi Engineering College

Name: Sarvesh S
Email: 240701479@rajalakshmi.edu.in
Roll no: 240701479
Phone: 9361488694
Branch: REC
Department: I CSE FE
Batch: 2028
Degree: B.E - CSE

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 2
Total Mark : 5
Marks Obtained : 4.5

Section 1 : Coding

1. 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>
struct node{
   int Element;
   struct node *Next;
};
typedef struct node Node;
void create(Node *);
void InsertBeg(Node *,int );
void InsertLast(Node *,int );
void InsertMidaf(Node *,int ,int );
void InsertMidbf(Node *,int ,int );
void Traverse(Node *);
void DeleteBeg(Node *);
void DeleteEnd(Node *);
void DeleteMidaf(Node *,int );
void DeleteMidbf(Node *,int );
int IsEmpty(Node *);
int IsLast(Node *);
Node *Find(Node *,int );
Node *FindPrevious(Node *,int );
Node *FindPreviousprev(Node *,int );
int main(){
   int choice,e,p;
   Node *List=(Node*)malloc(sizeof(Node));
   List->Next=NULL;
   do{
      if(scanf("%d",&choice)!=1) return 1;
      switch(choice){
```

```
case 1:
   create(List);
   printf("LINKED LIST CREATED\n");
   break;
case 2:
   Traverse(List);
   break;
case 3:
   if(scanf("%d",&e)!=1)return 1;
   InsertBeg(List,e);
   printf("The linked list after insertion at the beginning is:\n");
   Traverse(List);
   break;
case 4:
   if(scanf("%d",&e)!=1)return 1;
   InsertLast(List,e);
   printf("The linked list after insertion at the end is:\n");
   Traverse(List);
   break;
case 5:
   if(scanf("%d",&p)!=1)return 1;
   if(scanf("%d",&e)!=1)return 1;
   InsertMidbf(List,p,e);
   printf("The linked list after insertion before a value is:\n");
   Traverse(List);
   break;
case 6:
   if(scanf("%d",&p)!=1)return 1;
   if(scanf("%d",&e)!=1)return 1;
   InsertMidaf(List,p,e);
   printf("The linked list after insertion after a value is:\n");
   Traverse(List);
   break;
case 7:
   DeleteBeg(List);
   printf("The linked list after deletion from the beginning is:\n");
   Traverse(List);
   break;
case 8:
   DeleteEnd(List);
   printf("The linked list after deletion from the end is:\n");
   Traverse(List);
```

```
            break;
         case 9:
            if(scanf("%d",&p)!=1)return 1;
            DeleteMidbf(List,p);
            printf("The linked list after deletion before a value is:\n");
            Traverse(List);
            break;
         case 10:
            if(scanf("%d",&p)!=1)return 1;
            DeleteMidaf(List,p);
            printf("The linked list after deletion after a value is:\n");
            Traverse(List);
            break;
         case 11:
            return 0;
         default:
            printf("Invalid option! please try again\n");
            break;
      }
   }while(1);
   return 0;
}

int IsEmpty(Node *List){
   if(List->Next==NULL){
      return 1;
   }
   else{
      return 0;
   }
}

int IsLast(Node *position){
   if(position->Next==NULL){
      return 1;
   }
   else{
      return 0;
   }
}

Node *Find(Node *List,int x){
   Node *position=List->Next;
   while(position!=NULL && position->Element!=x){
      position=position->Next;
   }
   if(position==NULL){
      printf("Value not found in the list\n");
   }
   return position;
}

Node *FindPrevious(Node *List,int x){
   Node *position=List;
   while(position->Next!=NULL && position->Next->Element!=x){
      position=position->Next;
   }
   if(position->Next==NULL){
      printf("Value not found in the list\n");
      return NULL;
   }
   return position;
}

Node *FindPreviousprev(Node *List,int x){
   Node *position=List;
   while(position->Next->Next!=NULL && position->Next->Next->Element!=x){
      position=position->Next;
   }
   if(position->Next->Next==NULL){
      printf("Value not found in the list\n");
      return NULL;
   }
   return position;
}

void create(Node *List){
   int e;
   while(scanf("%d",&e)==1 && e!=-1){
      InsertLast(List,e);
   }
}

void InsertBeg(Node *List,int e){
   Node *newnode=(Node*)malloc(sizeof(Node));
   newnode->Element=e;
   if(IsEmpty(List)){
      newnode->Next=NULL;
   }
   else{
      newnode->Next=List->Next;
   }
   List->Next=newnode;
}

void InsertLast(Node *List,int e){
   Node *newnode=(Node*)malloc(sizeof(Node));
   Node *position;
   newnode->Element=e;
   newnode->Next=NULL;
   if(IsEmpty(List)){
      List->Next=newnode;
   }
   else{
      position=List;
      while(position->Next!=NULL){
          position=position->Next;
      }
      position->Next=newnode;
   }
}

void InsertMidaf(Node *List,int p,int e){
   Node *newnode=(Node*)malloc(sizeof(Node));
   Node *position=Find(List,p);
   if(position==NULL) return;
   newnode->Element=e;
   newnode->Next=position->Next;
   position->Next=newnode;
}

void InsertMidbf(Node *List,int p,int e){
   Node *newnode=(Node*)malloc(sizeof(Node));
   Node *position=FindPrevious(List,p);
   if(position==NULL) return;
   newnode->Element=e;
```

```
   newnode->Next=position->Next;
   position->Next=newnode;
}

void Traverse(Node *List){
   if(!IsEmpty(List)){
      Node *position=List;
      while(position->Next!=NULL){
         position=position->Next;
         printf("%d ",position->Element);
      }
      printf("\n");
   }
   else{
      printf("The list is empty\n");
   }
}

void DeleteBeg(Node *List){
   if(!IsEmpty(List)){
      Node *Tempnode=List->Next;
      List->Next=Tempnode->Next;
      free(Tempnode);
   }
   else{
      printf("The list is empty\n");
   }
}

void DeleteEnd(Node *List){
   if(!IsEmpty(List)){
      Node *position=List;
      Node *Tempnode;
      while(position->Next->Next!=NULL){
         position=position->Next;
      }
      Tempnode=position->Next;
      position->Next=NULL;
      free(Tempnode);
   }
   else{
      printf("The list is empty\n");
```

```
    }
}

void DeleteMidaf(Node *List,int e){
    if(!IsEmpty(List)){
        Node *position=Find(List,e);
        if(position==NULL || IsLast(position)) return;
        Node *Tempnode;
        if(!IsEmpty(List)){
            Tempnode=position->Next;
            position->Next=Tempnode->Next;
            free(Tempnode);
        }
    }
    else{
        printf("The list is empty\n");
    }
}

void DeleteMidbf(Node *List,int e){
    if(!IsEmpty(List)){
        Node *position=FindPreviousprev(List,e);
        if(position==NULL || IsLast(position)) return;
        Node *Tempnode;
        if(!IsEmpty(List)){
            Tempnode=position;
            List->Next=Tempnode->Next;
            free(Tempnode);
        }
    }
    else{
        printf("The list is empty\n");
    }
}
```

Status : *Partially correct* ***Marks : 0.75/1***

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>
struct poly{
   int element;
   struct poly *Next;
};
typedef struct poly Poly;
void create(Poly *,int );
void sump(Poly *,int ,int );
int main(){
   int n,e,x;
   Poly *List=(Poly*)malloc(sizeof(Poly));
   if(scanf("%d",&n)!=1) return 1;
   for(int i=0;i<=n;i++){
      if(scanf("%d",&e)!=1) return 1;
      create(List,e);
   }
   scanf("%d",&x);
   sump(List,x,n);
   return 0;
}
```

```
void create(Poly *List,int e){
   Poly *newnode=(Poly*)malloc(sizeof(Poly));
   newnode->element=e;
   newnode->Next=NULL;
   Poly *position=List;
   while(position->Next!=NULL){
      position=position->Next;
   }
   position->Next=newnode;
}

void sump(Poly *List,int x,int n){
   int sum=0,t=n;
   Poly *position=List->Next;
   while(position!=NULL){
      sum+=position->element*pow(x,t);
      position=position->Next;
      t--;
   }
   printf("%d\n",sum);
}
```

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

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int Element;
   struct node *Next;
};
typedef struct node Node;
void create(Node *);
void InsertBeg(Node *,int );
void InsertLast(Node *,int );
void InsertMidaf(Node *,int ,int );
void InsertMidbf(Node *,int ,int );
void Traverse(Node *);
void DeleteBeg(Node *);
void DeleteEnd(Node *);
void DeleteMidaf(Node *,int );
void DeleteMidbf(Node *,int );
int IsEmpty(Node *);
int IsLast(Node *);
Node *Find(Node *,int );
Node *FindPrevious(Node *,int );
Node *FindPreviousprev(Node *,int );
int main(){
   int choice,e,p;
   Node *List=(Node*)malloc(sizeof(Node));
   List->Next=NULL;
   do{
      scanf("%d",&choice);
      switch(choice){
         case 1:
            create(List);
            printf("LINKED LIST CREATED\n");
```

```
        break;
    case 2:
        Traverse(List);
        break;
    case 3:
        scanf("%d",&e);
        InsertBeg(List,e);
        printf("The linked list after insertion at the beginning is:\n");
        Traverse(List);
        break;
    case 4:
        scanf("%d",&e);
        InsertLast(List,e);
        printf("The linked list after insertion at the end is:\n");
        Traverse(List);
        break;
    case 5:
        scanf("%d",&p);
        scanf("%d",&e);
        InsertMidbf(List,p,e);
        printf("The linked list after insertion before a value is:\n");
        Traverse(List);
        break;
    case 6:
        scanf("%d",&p);
        scanf("%d",&e);
        InsertMidaf(List,p,e);
        printf("The linked list after insertion after a value is:\n");
        Traverse(List);
        break;
    case 7:
        DeleteBeg(List);
        printf("The linked list after deletion from the beginning is:\n");
        Traverse(List);
        break;
    case 8:
        DeleteEnd(List);
        printf("The linked list after deletion from the end is:\n");
        Traverse(List);
        break;
    case 9:
        scanf("%d",&p);
```

```
            DeleteMidbf(List,p);
            printf("The linked list after deletion before a value is:\n");
            Traverse(List);
            break;
        case 10:
            scanf("%d",&p);
            DeleteMidaf(List,p);
            printf("The linked list after deletion after a value is:\n");
            Traverse(List);
            break;
        case 11:
            return 0;
        default:
            printf("Invalid option! Please try again\n");
            break;
        }
    }while(1);
    return 0;
}

int IsEmpty(Node *List){
    if(List->Next==NULL){
        return 1;
    }
    else{
        return 0;
    }
}

int IsLast(Node *position){
    if(position->Next==NULL){
        return 1;
    }
    else{
        return 0;
    }
}

Node *Find(Node *List,int x){
    Node *position=List->Next;
    while(position!=NULL && position->Element!=x){
        position=position->Next;
    }
    if(position==NULL){
        printf("Value not found in the list\n");
        return NULL;
    }
    return position;
}

Node *FindPrevious(Node *List,int x){
    Node *position=List;
    while(position->Next!=NULL && position->Next->Element!=x){
        position=position->Next;
    }
    if(position->Next==NULL){
        printf("Value not found in the list\n");
        return NULL;
    }
    return position;
}

Node *FindPreviousprev(Node *List,int x){
    Node *position=List;
    while(position->Next->Next!=NULL && position->Next->Next->Element!=x){
        position=position->Next;
    }
    if(position->Next->Next==NULL){
        printf("Value not found in the list\n");
    }
    return position;
}

void create(Node *List){
    int e;
    while(scanf("%d",&e)==1 && e!=-1){
        InsertLast(List,e);
    }
}

void InsertBeg(Node *List,int e){
    Node *newnode=(Node*)malloc(sizeof(Node));
    newnode->Element=e;
    if(IsEmpty(List)){
        newnode->Next=NULL;
    }
    else{
        newnode->Next=List->Next;
    }
    List->Next=newnode;
}

void InsertLast(Node *List,int e){
    Node *newnode=(Node*)malloc(sizeof(Node));
    newnode->Element=e;
    Node *position;
    newnode->Next=NULL;
    if(IsEmpty(List)){
        List->Next=newnode;
    }
    else{
        position=List;
        while(position->Next!=NULL){
            position=position->Next;
        }
        position->Next=newnode;
    }
}

void InsertMidaf(Node *List,int p,int e){
    Node *newnode=(Node*)malloc(sizeof(Node));
    Node *position=Find(List,p);
    if(position==NULL) return;
    newnode->Element=e;
    newnode->Next=position->Next;
    position->Next=newnode;
}

void InsertMidbf(Node *List,int p,int e){
    Node *newnode=(Node*)malloc(sizeof(Node));
    Node *position=FindPrevious(List,p);
    if(position==NULL) return;
    newnode->Element=e;
    newnode->Next=position->Next;
    position->Next=newnode;
}
```

```
void Traverse(Node *List){
   if(!IsEmpty(List)){
      Node *position=List;
      while(position->Next!=NULL){
         position=position->Next;
         printf("%d ",position->Element);
      }
      printf("\n");
   }
   else{
      printf("The list is empty\n");
   }
}

void DeleteBeg(Node *List){
   if(!IsEmpty(List)){
      Node *Tempnode=List->Next;
      List->Next=Tempnode->Next;
      free(Tempnode);
   }
   else{
      printf("The list is empty\n");
   }
}

void DeleteEnd(Node *List){
   if(!IsEmpty(List)){
      Node *position=List;
      Node *Tempnode;
      while(position->Next->Next!=NULL){
         position=position->Next;
      }
      Tempnode=position->Next;
      position->Next=NULL;
      free(Tempnode);
   }
   else{
      printf("The list is empty\n");
   }
}
```

```
void DeleteMidaf(Node *List,int e){
   if(!IsEmpty(List)){
      Node *position=Find(List,e);
      if(position==NULL || IsLast(position)) return;
      Node *Tempnode;
      if(!IsLast(position)){
         Tempnode=position->Next;
         position->Next=Tempnode->Next;
         free(Tempnode);
      }
   }
   else{
      printf("The list is empty\n");
   }
}

void DeleteMidbf(Node *List,int e){
   if(!IsEmpty(List)){
      Node *position=FindPreviousprev(List,e);
      if(position==NULL || IsLast(position)) return;
      Node *Tempnode;
      if(!IsLast(position)){
         Tempnode=position;
         List->Next=Tempnode->Next;
         free(Tempnode);
      }
   }
   else{
      printf("The list is empty\n");
   }
}
```

Status : *Partially correct* ***Marks : 0.75/1***

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>
struct node{
   int Element;
   struct node *Next;
```

```
};
typedef struct node Node;
void InsertLast(Node *,int );
void Traverse(Node *);
void create(Node *List1,Node *List2,Node *result);
int main(){
   int n,m,e;
   Node *List1=(Node*)malloc(sizeof(Node));
   Node *List2=(Node*)malloc(sizeof(Node));
   Node *result=(Node*)malloc(sizeof(Node));
   List1->Next=NULL;
   List2->Next=NULL;
   if(scanf("%d",&n)!=1) return 1;
   for(int i=0;i<n;i++){
      if(scanf("%d",&e)!=1) return 1;
      InsertLast(List1,e);
   }
   if(scanf("%d",&m)!=1) return 1;
   for(int i=0;i<m;i++){
      if(scanf("%d",&e)!=1) return 1;
      InsertLast(List2,e);
   }
   create(List1,List2,result);
   Traverse(result);
   return 0;
}

void Traverse(Node *List){
   if(List->Next!=NULL){
      Node *position=List;
      while(position->Next!=NULL){
         position=position->Next;
         printf("%d ",position->Element);
      }
      printf("\n");
   }
}

void InsertLast(Node *List,int e){
   Node *newnode=(Node*)malloc(sizeof(Node));
   Node *position;
   newnode->Element=e;
```

```
    newnode->Next=NULL;
    if(List->Next==NULL){
        List->Next=newnode;
    }
    else{
        position=List;
        while(position->Next!=NULL){
            position=position->Next;
        }
        position->Next=newnode;
    }
}

void create(Node *List1,Node *List2,Node *result){
    if(!List1->Next || !List2->Next) return;
    for(Node *ptr1=List1->Next;ptr1;ptr1=ptr1->Next){
        int newelement=ptr1->Element;
        Node *respos=result;
        while(respos->Next!=NULL){
            respos=respos->Next;
        }
        Node *newnode=(Node*)malloc(sizeof(Node));
        newnode->Element=newelement;
        newnode->Next=respos->Next;
        respos->Next=newnode;
    }
    for(Node *ptr2=List2->Next;ptr2;ptr2=ptr2->Next){
        int newelement=ptr2->Element;
        Node *respos=result;
        while(respos->Next!=NULL){
            respos=respos->Next;
        }
        Node *newnode=(Node*)malloc(sizeof(Node));
        newnode->Element=newelement;
        newnode->Next=respos->Next;
        respos->Next=newnode;
    }
}
```

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>

struct node{
   int Element;
   struct node *Next;
};
typedef struct node Node;
void InsertBeg(Node *,int );
void InsertLast(Node *,int );
void Traverse(Node *);
Node *rearr(Node*);
int main(){
   int n,e;
   Node *List=(Node*)malloc(sizeof(Node));
   List->Next=NULL;
   if(scanf("%d",&n)!=1) return 1;
   for(int i=0;i<n;i++){
      if(scanf("%d",&e)!=1) return 1;
      InsertLast(List,e);
   }
   List->Next=rearr(List->Next);
   Traverse(List);
   return 0;
}
```

```
Node *rearr(Node *head){
   Node *even=NULL,*odd=NULL,*oddtail=NULL,*eventail=NULL;
   Node *pos=head;
   while(pos!=NULL){
      Node *newnode=(Node*)malloc(sizeof(Node));
      newnode->Element=pos->Element;
      newnode->Next=NULL;
      if(pos->Element%2==0){
         if(even==NULL){
            even=newnode;
            eventail=newnode;
         }
         else{
            newnode->Next=even;
            even=newnode;
         }
      }
      else{
         if(odd==NULL){
            odd=newnode;
            oddtail=newnode;
         }
         else{
            oddtail->Next=newnode;
            oddtail=newnode;
         }
      }
      pos=pos->Next;
   }
   if(even==NULL) return odd;
   if(eventail!=NULL) eventail->Next=odd;
   return even;
}

void Traverse(Node *List){
   if(List->Next!=NULL){
      Node *position=List;
      while(position->Next!=NULL){
         position=position->Next;
         printf("%d ",position->Element);
      }
      printf("\n");
```

```
    }
}

void InsertLast(Node *List,int e){
    Node *newnode=(Node*)malloc(sizeof(Node));
    Node *position;
    newnode->Element=e;
    newnode->Next=NULL;
    if(List->Next==NULL){
        List->Next=newnode;
    }
    else{
        position=List;
        while(position->Next!=NULL){
            position=position->Next;
        }
        position->Next=newnode;
    }
}

void InsertBeg(Node *List,int e){
    Node *newnode=(Node*)malloc(sizeof(Node));
    newnode->Element=e;
    if(List->Next==NULL){
        newnode->Next=NULL;
    }
    else{
        newnode->Next=List->Next;
    }
    List->Next=newnode;
}
```

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