Rajalakshmi Engineering College

Name: Niranjana S
Email: 240801222@rajalakshmi.edu.in
Roll no: 240801222
Phone: 9384400179
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 2
Total Mark : 5
Marks Obtained : 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

```
// 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;
}
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;
        }
        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**

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 Num{
  int value;
  struct Num* next;
}Node;
Node* newnode(int value){
  Node* node=(Node*) malloc(sizeof(Node));
  node->value=value;
  node->next=NULL;
  return node;
}

void insertNode(Node**head,int value){

  Node* temp=*head;
  if(temp==NULL){
     *head=newnode(value);
     return;
  }
  while(temp->next!=NULL){
     temp=temp->next;
  }
  temp->next=newnode(value);
}
Node* reverseNode(Node* head){
  Node* reverse=NULL;
  while(head!=NULL){
     Node* node=newnode(head->value);
     node->next=reverse;
     reverse=node;
     head=head->next;
  }
  return reverse;
}
void traverse(Node* head){
  while(head !=NULL){
     printf("%d ", head->value);
     head=head->next;
  }
  printf("\n");
}
void merge(Node* head1,Node* head2){
  while(head1->next!=NULL){
```

```
        head1=head1->next;
    }
    head1->next=head2;
}
int main(){
    int n,value;
    Node* head=NULL;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&value);
        insertNode(&head,value);
    }
    Node*temp=head;

    Node*odd=NULL;
    Node*even=NULL;
    while(temp!=NULL){
        if(temp->value % 2 == 1){
            insertNode(&odd,temp->value);
        }
        else{
            insertNode(&even,temp->value);
        }
        temp=temp->next;
    }

        even=reverseNode(even);
        merge(even,odd);
        traverse(even);
        }
```

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

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;
}
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;
        }
        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**

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 value;
   struct node*next;
}Node;
Node* newnode(int value){
   Node*list=(Node*) malloc (sizeof(Node));
```

```
    list->value=value;
    list->next=NULL;
    return list;
}
void insertNode(Node* *head,int value){
    Node*temp=*head;
    if(*head==NULL){
        *head=newnode(value);
        return;
    }
    while(temp->next!=NULL){
        temp = temp->next;
    }
    temp->next = newnode(value);
}
void traverse(Node* head){
    while(head!=NULL){
        printf("%d ",head->value);
        head = head->next;
    }
}
int main(){
    int n,m,v;
    Node* node1 = NULL;
    Node* node2 = NULL;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&v);
        insertNode(&node1,v);
    }
    scanf("%d",&m);
    for(int i=0; i<m; i++){
        scanf("%d",&v);
        insertNode(&node2,v);
    }
    Node* temp = node1;

    while(temp->next!=NULL){
        temp = temp->next;
    }
    temp->next = node2;
    traverse(node1);
```

}

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

5. 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

```
// You are using GCC
 #include<stdio.h>
 #include<stdlib.h>
 #include<math.h>
 typedef struct poly{
    int x;
    int expon;
    struct poly* next;
 }Node;
 Node* newnode(int x,int expon){

    Node* node=(Node*) malloc(sizeof(Node));
    node->x=x;
```

```
    node->expon=expon;
    node->next=NULL;
    return node;
}
void insertNode(Node** head,int x,int expon){
    Node* temp=*head;
    if(temp==NULL){
        *head=newnode(x,expon);
        return;
    }
    while(temp->next!=NULL){
        temp=temp->next;
}
temp->next=newnode(x,expon);
}
int main(){
    int degree,x;
    scanf("%d",&degree);
    Node* head=NULL;
    for(int i=0;i<=degree;i++){
        scanf("%d",&x);
        insertNode(&head,x,degree-i);
    }
    int value=0;
    int n;
    scanf("%d",&n);
    while(head!=NULL){
        value+=head->x* pow(n,head->expon);
        head=head->next;
    }
    printf("%d",value);
}
```

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

Rajalakshmi Engineering College

Name: Niranjana S
Email: 240801222@rajalakshmi.edu.in
Roll no: 240801222
Phone: 9384400179
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Pranav wants to clockwise rotate a doubly linked list by a specified number of positions. He needs your help to implement a program to achieve this. Given a doubly linked list and an integer representing the number of positions to rotate, write a program to rotate the list clockwise.

Input Format

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

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

The third line consists of an integer k, representing the number of places to rotate the list.

Output Format

The output displays the elements of the doubly linked list after rotating it by k positions.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
1
Output: 5 1 2 3 4

Answer

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

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
```

```
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
struct Node* rotateClockwise(struct Node* head, int k, int n) {
    if (head == NULL || k == 0 || k % n == 0) {
        return head;
    }

    k = k % n;

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

    for (int i = 0; i < k; i++) {
        struct Node* temp = tail;
        tail = tail->prev;

        temp->next = head;
        head->prev = temp;
        head = temp;
        head->prev = NULL;
        tail->next = NULL;
    }
```

```
    return head;
}

int main() {
    int n, data, k;
    struct Node* head = NULL;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        insertAtEnd(&head, data);
    }
    scanf("%d", &k);
    head = rotateClockwise(head, k, n);
    displayList(head);
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

return 0;
}
```

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

2. Problem Statement

Rohan is a software developer who is working on an application that processes data stored in a Doubly Linked List. He needs to implement a feature that finds and prints the middle element(s) of the list. If the list contains an odd number of elements, the middle element should be printed. If the list contains an even number of elements, the two middle elements should be printed.

Help Rohan by writing a program that reads a list of numbers, prints the list, and then prints the middle element(s) based on the number of elements in the list.

Input Format

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

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

Output Format

The first line prints the elements of the list separated by space. (There is an extra space at the end of this line.)

The second line prints the middle element(s) based on the number of elements.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
20 52 40 16 18

Output: 20 52 40 16 18
40

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
```

```
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void findMiddle(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* slow = head;
    struct Node* fast = head;
    int count = 0;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
        count++;
```

```
    }

    if (fast == NULL) {
        printf("%d %d\n", slow->prev->data, slow->data);
    } else {
        printf("%d\n", slow->data);
    }
}

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

    findMiddle(head);

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

return 0;
}
```

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

3. Problem Statement

Riya is developing a contact management system where recently added contacts should appear first. She decides to use a doubly linked list to store contact IDs in the order they are added. Initially, new contacts are inserted at the front of the list. However, sometimes she needs to insert a new contact at a specific position in the list based on priority.

Help Riya implement this system by performing the following operations:

Insert contact IDs at the front of the list as they are added.Insert a new contact at a given position in the list.

Input Format

The first line of input consists of an integer N, representing the initial size of the linked list.

The second line consists of N space-separated integers, representing the values of the linked list to be inserted at the front.

The third line consists of an integer position, representing the position at which the new value should be inserted (position starts from 1).

The fourth line consists of integer data, representing the new value to be inserted.

Output Format

The first line of output prints the original list after inserting initial elements to the front.

The second line prints the updated linked list after inserting the element at the specified position.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 20 30 40
3
25

Output: 40 30 20 10
40 30 25 20 10

Answer

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

```
#include <stdlib.h>

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    newNode->next = *head;
    (*head)->prev = newNode;
    *head = newNode;
}

void insertAtPosition(struct Node** head, int data, int position) {
    if (position <= 0) {
        printf("Invalid position. Please enter a positive integer.\n");
        return;
    }

    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        if (position == 1) {
            *head = newNode;
        } else {
```

```
            printf("List is empty. Cannot insert at position %d.\n", position);
            free(newNode);
        }
        return;
    }

    if (position == 1) {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
        return;
    }

    struct Node* current = *head;
    int count = 1;
    while (current != NULL && count < position - 1) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        printf("Position is out of bounds.  The list has %d elements.\n", count);
        free(newNode);
        return;
    }

    newNode->next = current->next;
    newNode->prev = current;
    current->next = newNode;
    if (newNode->next != NULL) {
        newNode->next->prev = newNode;
    }
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
```

```
        temp = temp->next;
    }
    printf("\n");
}

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

    scanf("%d", &n);

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

    displayList(head);

    scanf("%d", &position);
    scanf("%d", &data);

    insertAtPosition(&head, data, position);

    displayList(head);

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

return 0;
}
```

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

4. Problem Statement

Bala is a student learning about the doubly linked list and its functionalities. He came across a problem where he wanted to create a

doubly linked list by appending elements to the front of the list.

After populating the list, he wanted to delete the node at the given position from the beginning. Write a suitable code to help Bala.

Input Format

The first line contains an integer N, the number of elements in the doubly linked list.

The second line contains N integers separated by a space, the data values of the nodes in the doubly linked list.

The third line contains an integer X, the position of the node to be deleted from the doubly linked list.

Output Format

The first line of output displays the original elements of the doubly linked list, separated by a space.

The second line prints the updated list after deleting the node at the given position X from the beginning.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50
2
Output: 50 40 30 20 10
50 30 20 10

Answer

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

struct Node {
    int data;
```

```
    struct Node* prev;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    newNode->next = *head;
    (*head)->prev = newNode;
    *head = newNode;
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

struct Node* deleteNodeFromBeginning(struct Node* head, int position) {
    if (head == NULL) {
```

```
        printf("List is empty, nothing to delete\n");
        return NULL;
    }

    if (position <= 0) {
        printf("Invalid position. Position should be >= 1\n");
        return head;
    }

    struct Node* current = head;
    int count = 1;

    while (current != NULL && count < position) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        printf("Position is out of bounds.  List size is %d\n", count-1);
        return head;
    }

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

int main() {
    int n, data, x;
    struct Node* head = NULL;
```

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

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

    displayList(head);

    scanf("%d", &x);

    head = deleteNodeFromBeginning(head, x);

    displayList(head);

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

return 0;
}
```

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

5. Problem Statement

Tom is a software developer working on a project where he has to check if a doubly linked list is a palindrome. He needs to write a program to solve this problem. Write a program to help Tom check if a given doubly linked list is a palindrome or not.

Input Format

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

The second line consists of N space-separated integers representing the linked

list elements.

Output Format

The first line displays the space-separated integers, representing the doubly linked list.

The second line displays one of the following:

1. If the doubly linked list is a palindrome, print "The doubly linked list is a palindrome".
2. If the doubly linked list is not a palindrome, print "The doubly linked list is not a palindrome".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 2 1

Output: 1 2 3 2 1
The doubly linked list is a palindrome

Answer

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


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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
```

```
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int isPalindrome(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return 1;
    }
    struct Node* front = head;
    struct Node* rear = head;
    while (rear->next != NULL) {
        rear = rear->next;
    }

    while (front != rear && front->prev != rear) {
        if (front->data != rear->data) {
            return 0;
        }
```

```
        front = front->next;
        rear = rear->prev;
    }
    return 1;
}

int main() {
    int n, data;
    struct Node* head = NULL;
    scanf("%d", &n);


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


    displayList(head);
    if (isPalindrome(head)) {
        printf("The doubly linked list is a palindrome\n");
    } else {
        printf("The doubly linked list is not a palindrome\n");
    }
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

return 0;
}
```

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

Rajalakshmi Engineering College

Name: Niranjana S
Email: 240801222@rajalakshmi.edu.in
Roll no: 240801222
Phone: 9384400179
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Guide Harish in developing a simple queue system for a customer service center. The customer service center can handle up to 25 customers at a time. The queue needs to support basic operations such as adding a customer to the queue, serving a customer (removing them from the queue), and displaying the current queue of customers.

Use an array for implementation.

Input Format

The first line of the input consists of an integer N, the number of customers arriving at the service center.

The second line consists of N space-separated integers, representing the customer IDs in the order they arrive.

Output Format

After serving the first customer in the queue, display the remaining customers in the queue.

If a dequeue operation is attempted on an empty queue, display "Underflow".

If the queue is empty, display "Queue is empty".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
101 102 103 104 105
Output: 102 103 104 105

Answer

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

#define MAX_SIZE 25

// Queue structure
struct Queue {
    int front, rear;
    int arr[MAX_SIZE];
};

// Function to initialize the queue
void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

// Function to add a customer to the queue (enqueue)
void enqueue(struct Queue* q, int customer_id) {
    if (q->rear == MAX_SIZE - 1) {
        printf("Queue is full\n");
        return;
```

```
    }
    if (q->front == -1) {
        q->front = 0; // First customer
    }
    q->rear++;
    q->arr[q->rear] = customer_id;
}

// Function to serve a customer (dequeue)
void dequeue(struct Queue* q) {
    if (q->front == -1 || q->front > q->rear) {
        printf("Underflow\n");
        return;
    }
    q->front++;
}

// Function to display the current queue
void displayQueue(struct Queue* q) {
    if (q->front == -1 || q->front > q->rear) {
        printf("Queue is empty\n");
        return;
    }

    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->arr[i]);
    }
    printf("\n");
}

int main() {
    struct Queue q;
    initQueue(&q);

    int N;
    scanf("%d", &N);

    if (N == 0) {
        printf("Underflow\n");
        printf("Queue is empty\n");
        return 0;
    }
```

```
    // Read customer IDs into the queue
    for (int i = 0; i < N; i++) {
        int customer_id;
        scanf("%d", &customer_id);
        enqueue(&q, customer_id);
    }

    // Serve the first customer (dequeue)
    dequeue(&q);

    // Display the remaining queue
    displayQueue(&q);

    return 0;
}
```

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

2. Problem Statement

Amar is working on a project where he needs to implement a special type of queue that allows selective dequeuing based on a given multiple. He wants to efficiently manage a queue of integers such that only elements not divisible by a given multiple are retained in the queue after a selective dequeue operation.

Implement a program to assist Amar in managing his selective queue.

Example

Input:

5

10 2 30 4 50

5

Output:

Original Queue: 10 2 30 4 50

Queue after selective dequeue: 2 4

Explanation:

After selective dequeue with a multiple of 5, the elements that are multiples of 5 should be removed. Therefore, only 10, 30, and 50 should be removed from the queue. The updated Queue is 2 4.

Input Format

The first line contains an integer n, representing the number of elements initially present in the queue.

The second line contains n space-separated integers, representing the elements of the queue.

The third line contains an integer multiple, representing the divisor for selective dequeue operation.

Output Format

The first line of output prints "Original Queue: " followed by the space-separated elements in the queue before the dequeue operation.

The second line prints "Queue after selective dequeue: " followed by the remaining space-separated elements in the queue, after deleting elements that are the multiples of the specified number.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
10 2 30 4 50
5

Output: Original Queue: 10 2 30 4 50
Queue after selective dequeue: 2 4

Answer

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

```
void selectiveDequeue(int queue[], int n, int multiple) {
    printf("Original Queue:");
    for (int i = 0; i < n; i++) {
        printf(" %d", queue[i]);
    }
    printf("\n");

    printf("Queue after selective dequeue:");
    for (int i = 0; i < n; i++) {
        if (queue[i] % multiple != 0) {
            printf(" %d", queue[i]);
        }
    }
    printf("\n");
}

int main() {
    int n, multiple;

    // Read the number of elements
    scanf("%d", &n);
    int queue[n];

    // Read the queue elements
    for (int i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
    }

    // Read the multiple for selective dequeue
    scanf("%d", &multiple);

    // Perform selective dequeue
    selectiveDequeue(queue, n, multiple);

    return 0;
}
```

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

3. Problem Statement

You've been assigned the challenge of developing a queue data structure using a linked list.

The program should allow users to interact with the queue by enqueuing positive integers and subsequently dequeuing and displaying elements.

Input Format

The input consists of a series of integers, one per line. Enter positive integers into the queue.

Enter -1 to terminate input.

Output Format

The output prints the space-separated dequeued elements.

Refer to the sample output for the exact text and format.

Sample Test Case

Input: 1
2
3
4
-1
Output: Dequeued elements: 1 2 3 4

Answer

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

// Define structure for a queue node
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Define queue structure
```

```
typedef struct {
    Node *front, *rear;
} Queue;

// Function to initialize the queue
void initializeQueue(Queue *q) {
    q->front = q->rear = NULL;
}

// Function to enqueue an element
void enqueue(Queue *q, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (q->rear == NULL) {
        q->front = q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}

// Function to dequeue an element
void dequeue(Queue *q) {
    if (q->front == NULL) {
        return;  // Queue is empty
    }
    Node* temp = q->front;
    q->front = q->front->next;

    if (q->front == NULL) {
        q->rear = NULL;
    }

    printf("%d ", temp->data);
    free(temp);
}
```

```
// Function to process input and display dequeued elements
int main() {
    Queue queue;
    initializeQueue(&queue);

    int value;
    while (1) {
        scanf("%d", &value);
        if (value == -1) {
            break;
        }
        enqueue(&queue, value);
    }

    printf("Dequeued elements:");
    while (queue.front != NULL) {
        dequeue(&queue);
    }
    printf("\n");

    return 0;
}
```

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

4. Problem Statement

Sharon is developing a queue using an array. She wants to provide the functionality to find the Kth largest element. The queue should support the addition and retrieval of the Kth largest element effectively. The maximum capacity of the queue is 10.

Assist her in the program.

Input Format

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

The second line consists of N space-separated integers.

The third line consists of an integer K.

Output Format

For each enqueued element, print a message: "Enqueued: " followed by the element.

The last line prints "The [K]th largest element: " followed by the Kth largest element.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
23 45 93 87 25
4

Output: Enqueued: 23
Enqueued: 45
Enqueued: 93
Enqueued: 87
Enqueued: 25
The 4th largest element: 25

Answer

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

// Function to sort the queue in descending order
void sortDescending(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            if (arr[i] < arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
```

```
}
int main() {
    int N, K;

    // Read the number of elements
    scanf("%d", &N);
    if (N > 10) {
        printf("Queue exceeds maximum capacity of 10.\n");
        return 1;
    }

    int queue[N];

    // Read the queue elements
    for (int i = 0; i < N; i++) {
        scanf("%d", &queue[i]);
        printf("Enqueued: %d\n", queue[i]);
    }

    // Read the value of K
    scanf("%d", &K);
    if (K > N || K < 4) {
        printf("Invalid K value.\n");
        return 1;
    }

    // Sort the queue in descending order
    sortDescending(queue, N);

    // Output the Kth largest element
    printf("The %dth largest element: %d\n", K, queue[K - 1]);

    return 0;
}
```

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

5. Problem Statement

You are tasked with developing a simple ticket management system for a

customer support department. In this system, customers submit support tickets, which are processed in a First-In-First-Out (FIFO) order. The system needs to handle the following operations:

Ticket Submission (Enqueue Operation): New tickets are submitted by customers. Each ticket is assigned a unique identifier (represented by an integer). When a new ticket arrives, it should be added to the end of the queue.

Ticket Processing (Dequeue Operation): The support team processes tickets in the order they are received. The ticket at the front of the queue is processed first. After processing, the ticket is removed from the queue.

Display Ticket Queue: The system should be able to display the current state of the ticket queue, showing the sequence of ticket identifiers from front to rear.

Input Format

The first input line contains an integer n, the number of tickets submitted by customers.

The second line consists of a single integer, representing the unique identifier of each submitted ticket, separated by a space.

Output Format

The first line displays the "Queue: " followed by the ticket identifiers in the queue after all tickets have been submitted.

The second line displays the "Queue After Dequeue: " followed by the ticket identifiers in the queue after processing (removing) the ticket at the front.

Refer to the sample output for the exact text and format.

Sample Test Case

Input: 6
14 52 63 95 68 49
Output: Queue: 14 52 63 95 68 49

Queue After Dequeue: 52 63 95 68 49

Answer

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

#define MAX_SIZE 20

// Function to display the queue
void displayQueue(int queue[], int size) {
    printf("Queue:");
    for (int i = 0; i < size; i++) {
        printf(" %d", queue[i]);
    }
    printf("\n");
}

int main() {
    int n;

    // Read the number of tickets
    scanf("%d", &n);
    if (n < 2 || n > MAX_SIZE) {
        printf("Invalid number of tickets.\n");
        return 1;
    }

    int queue[MAX_SIZE];

    // Read the ticket identifiers
    for (int i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
    }

    // Display the original queue
    displayQueue(queue, n);

    // Remove the front ticket (FIFO queue behavior)
    printf("Queue After Dequeue:");
    for (int i = 1; i < n; i++) {
        printf(" %d", queue[i]);
    }
```

```
    printf("\n");
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Niranjana S
Email: 240801222@rajalakshmi.edu.in
Roll no: 240801222
Phone: 9384400179
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Viha, a software developer, is working on a project to automate searching for a target value in a Binary Search Tree (BST). She needs to create a program that takes an integer target value as input and determines if that value is present in the BST or not.

Write a program to assist Viha.

Input Format

The first line of input consists of integers separated by spaces, which represent the elements to be inserted into the BST. The input is terminated by entering -1.

The second line consists of an integer target, which represents the target value to be searched in the BST.

Output Format

If the target value is found in the BST, print "[target] is found in the BST".

Else, print "[target] is not found in the BST"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5 3 7 1 4 6 8 -1
4
Output: 4 is found in the BST

Answer

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

struct Node {
    int data;
    struct Node *left, *right;
};


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}


struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
}
int search(struct Node* root, int target) {
```

```
    if (root == NULL) return 0;
    if (root->data == target) return 1;
    if (target < root->data) return search(root->left, target);
    return search(root->right, target);
}

int main() {
    struct Node* root = NULL;
    int num;
    while (1) {
        scanf("%d", &num);
        if (num == -1) break;
        root = insert(root, num);
    }
    int target;
    scanf("%d", &target);
    if (search(root, target))
        printf("%d is found in the BST\n", target);
    else
        printf("%d is not found in the BST\n", target);

    return 0;
}
```

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

2. Problem Statement

Joseph, a computer science student, is interested in understanding binary search trees (BST) and their node arrangements. He wants to create a program to explore BSTs by inserting elements into a tree and displaying the nodes using post-order traversal of the tree.

Write a program to help Joseph implement the program.

Input Format

The first line of input consists of an integer N, representing the number of elements to insert into the BST.

The second line consists of N space-separated integers data, which is the data

to be inserted into the BST.

Output Format

The output prints N space-separated integer values after the post-order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 15 5 3
Output: 3 5 15 10

Answer

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

// Node structure for the BST
struct Node {
    int data;
    struct Node *left, *right;
};

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

// Function to insert a new node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
}
```

```
// Function for post-order traversal of the BST
void postOrderTraversal(struct Node* root) {
    if (root == NULL) return;
    postOrderTraversal(root->left);
    postOrderTraversal(root->right);
    printf("%d ", root->data);
}

int main() {
    int n, i, value;
    struct Node* root = NULL;

    // Read the number of elements
    scanf("%d", &n);

    // Read and insert elements into the BST
    for (i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    // Perform post-order traversal
    postOrderTraversal(root);
    printf("\n");

    return 0;
}
```

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

3. Problem Statement

Yogi is working on a program to manage a binary search tree (BST) containing integer values. He wants to implement a function that removes nodes from the tree that fall outside a specified range defined by a minimum and maximum value.

Help Yogi by writing a function that achieves this.

Input Format

The first line of input consists of an integer N, representing the number of elements to be inserted into the BST.

The second line consists of N space-separated integers, representing the elements to be inserted into the BST.

The third line consists of two space-separated integers min and max, representing the minimum value and the maximum value of the range.

Output Format

The output prints the remaining elements of the BST in an in-order traversal, after removing nodes that fall outside the specified range.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 20 12
5 15

Output: 5 10 12 15

Answer

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

// Structure for tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
```

```
    node->left = node->right = NULL;
    return node;
}

// Function to insert a new node with given data
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

// Function to perform in-order traversal of BST
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Function to remove nodes that are outside the given range
struct Node* removeOutsideRange(struct Node* root, int min, int max) {
    if (root == NULL) return NULL;

    // Recur for left and right subtrees
    root->left = removeOutsideRange(root->left, min, max);
    root->right = removeOutsideRange(root->right, min, max);

    // If current node's data is outside the range, delete it
    if (root->data < min) {
        struct Node* temp = root->right;
        free(root);
        return temp;
    }
    if (root->data > max) {
        struct Node* temp = root->left;
        free(root);
        return temp;
    }
```

```
    return root;
}

int main() {
    int N, min, max;
    scanf("%d", &N);

    struct Node* root = NULL;
    int data;
    for (int i = 0; i < N; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }

    scanf("%d %d", &min, &max);

    root = removeOutsideRange(root, min, max);

    inorder(root);
    printf("\n");

    return 0;
}
```

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

4. Problem Statement

Aishu is participating in a coding challenge where she needs to reconstruct a Binary Search Tree (BST) from given preorder traversal data and then print the in-order traversal of the reconstructed BST.

Since Aishu is just learning about tree data structures, she needs your help to write a program that does this efficiently.

Input Format

The first line consists of an integer n, representing the number of nodes in the BST.

The second line of input contains n integers separated by spaces, which represent the preorder traversal of the BST.

Output Format

The output displays n space-separated integers, representing the in-order traversal of the reconstructed BST.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 6
10 5 1 7 40 50
Output: 1 5 7 10 40 50

Answer

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

// Node structure for the BST
struct Node {
    int data;
    struct Node *left, *right;
};

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

// Function to insert a node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
```

```
}

// Function to build the BST from the given preorder traversal
struct Node* buildBST(int preorder[], int n) {
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        root = insert(root, preorder[i]);
    }
    return root;
}

// Function for in-order traversal of the BST
void inOrderTraversal(struct Node* root) {
    if (root == NULL) return;
    inOrderTraversal(root->left);
    printf("%d ", root->data);
    inOrderTraversal(root->right);
}

int main() {
    int n;

    // Read the number of nodes
    scanf("%d", &n);
    int preorder[n];

    // Read the preorder traversal data
    for (int i = 0; i < n; i++) {
        scanf("%d", &preorder[i]);
    }

    // Reconstruct the BST
    struct Node* root = buildBST(preorder, n);

    // Print the in-order traversal of the reconstructed BST
    inOrderTraversal(root);
    printf("\n");

    return 0;
}
```

Status : Correct *Marks : 10/10*

5. Problem Statement

Arun is exploring operations on binary search trees (BST). He wants to write a program with an unsorted distinct integer array that represents the BST keys and construct a height-balanced BST from it.

After constructing, he wants to perform the following operations that can alter the structure of the tree and traverse them using a level-order traversal:

InsertionDeletion

Your task is to assist Arun in completing the program without any errors.

Input Format

The first line of input consists of an integer N, representing the number of initial keys in the BST.

The second line consists of N space-separated integers, representing the initial keys.

The third line consists of an integer X, representing the new key to be inserted into the BST.

The fourth line consists of an integer Y, representing the key to be deleted from the BST.

Output Format

The first line of output prints "Initial BST: " followed by a space-separated list of keys in the initial BST after constructing it in level order traversal.

The second line prints "BST after inserting a new node X: " followed by a space-separated list of keys in the BST after inserting X n level order traversal.

The third line prints "BST after deleting node Y: " followed by a space-separated list of keys in the BST after deleting Y n level order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
25 14 56 28 12
34
12
Output: Initial BST: 25 14 56 12 28
BST after inserting a new node 34: 25 14 56 12 28 34
BST after deleting node 12: 25 14 56 28 34

Answer

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

// Node structure for the BST
struct Node {
    int data;
    struct Node *left, *right;
};

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

// Function to insert a node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
}

// Function to find the minimum value node in a BST (for deletion)
struct Node* findMin(struct Node* root) {
    while (root && root->left != NULL) root = root->left;
    return root;
}
```

```
// Function to delete a node from the BST
struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data) root->left = deleteNode(root->left, key);
    else if (key > root->data) root->right = deleteNode(root->right, key);
    else {
        // Node with one child or no child
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        // Node with two children
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Function to perform level order traversal
void levelOrderTraversal(struct Node* root) {
    if (root == NULL) return;
    struct Node* queue[100] = {0};
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        struct Node* current = queue[front++];
        printf("%d ", current->data);
        if (current->left) queue[rear++] = current->left;
        if (current->right) queue[rear++] = current->right;
    }
    printf("\n");
}

int main() {
    int n, i, value, x, y;
```

```
    struct Node* root = NULL;

    // Read the number of initial keys
    scanf("%d", &n);

    // Read and insert the initial keys into the BST
    for (i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    // Print the initial BST
    printf("Initial BST: ");
    levelOrderTraversal(root);

    // Read the key to be inserted
    scanf("%d", &x);
    root = insert(root, x);
    printf("BST after inserting a new node %d: ", x);
    levelOrderTraversal(root);

    // Read the key to be deleted
    scanf("%d", &y);
    root = deleteNode(root, y);
    printf("BST after deleting node %d: ", y);
    levelOrderTraversal(root);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Niranjana S
Email: 240801222@rajalakshmi.edu.in
Roll no: 240801222
Phone: 9384400179
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 6_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Alex is working on a project that involves merging and sorting two arrays. He wants to write a program that merges two arrays, sorts the merged array in ascending order, removes duplicates, and prints the sorted array without duplicates.

Help Alex to implement the program using the merge sort algorithm.

Input Format

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

The second line consists of N integers, separated by spaces, representing the elements of the first array.

The third line consists of an integer M, representing the number of elements in the second array.

The fourth line consists of M integers, separated by spaces, representing the elements of the second array.

Output Format

The output prints space-separated integers, representing the merged and sorted array in ascending order, with duplicate elements removed.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
1 2 3 4
3
3 4 5

Output: 1 2 3 4 5

Answer

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

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2) {
```

```
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

void removeDuplicates(int arr[], int *n) {
    int temp[*n];
    int j = 0;
    for (int i = 0; i < *n; i++) {
        if (i == 0 || arr[i] != arr[i - 1]) {
            temp[j++] = arr[i];
        }
    }
    for (int i = 0; i < j; i++) {
        arr[i] = temp[i];
```

```
    }
    *n = j;
}

int main() {
    int N, M;

    scanf("%d", &N);
    int arr1[N];
    for (int i = 0; i < N; i++)
        scanf("%d", &arr1[i]);

    scanf("%d", &M);
    int arr2[M];
    for (int i = 0; i < M; i++)
        scanf("%d", &arr2[i]);

    int mergedArr[N + M];

    for (int i = 0; i < N; i++)
        mergedArr[i] = arr1[i];

    for (int i = 0; i < M; i++)
        mergedArr[N + i] = arr2[i];

    int size = N + M;

    mergeSort(mergedArr, 0, size - 1);
    removeDuplicates(mergedArr, &size);

    for (int i = 0; i < size; i++)
        printf("%d ", mergedArr[i]);

    return 0;
}
```

Status : Correct *Marks : 10/10*

2. Problem Statement

You are working as a programmer at a sports academy, and the academy holds various sports competitions regularly.

As part of the academy's system, you need to sort the scores of the participants in descending order using the Quick Sort algorithm.

Write a program that takes the scores of n participants as input and uses the Quick Sort algorithm to sort the scores in descending order. Your program should display the sorted scores after the sorting process.

Input Format

The first line of input consists of an integer n, which represents the number of scores.

The second line of input consists of n integers, which represent scores separated by spaces.

Output Format

Each line of output represents an iteration of the Quick Sort algorithm, displaying the elements of the array at that iteration.

After the iterations are complete, the last line of output prints the sorted scores in descending order separated by space.

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 5
78 54 96 32 53

Output: Iteration 1: 78 54 96 53 32
Iteration 2: 96 54 78
Iteration 3: 78 54
Sorted Order: 96 78 54 53 32

Answer

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void printArray(int arr[], int start, int end) {
    for (int i = start; i <= end; i++) {
        printf("%d ", arr[i]);
    }
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] >= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high, int *iteration) {
    if (low < high) {
        int pi = partition(arr, low, high);
        (*iteration)++;

        printf("Iteration %d: ", *iteration);
        printArray(arr, low, high);
        printf("  ");

        quickSort(arr, low, pi - 1, iteration);
        quickSort(arr, pi + 1, high, iteration);
    }
}
```

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

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

    int iteration = 0;
    quickSort(scores, 0, n - 1, &iteration);

    printf("Sorted Order: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", scores[i]);
    }
    printf("\n");
    return 0;
}
```

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

3. Problem Statement

You are working on an optimization task for a sorting algorithm that uses insertion sort. Your goal is to determine the efficiency of the algorithm by counting the number of swaps needed to sort an array of integers.

Write a program that takes an array as input and calculates the number of swaps performed during the insertion sort process.

Example 1:

Input:

5

2 1 3 1 2

Output:

4

Explanation:

Step 1: [2, 1, 3, 1, 2] (No swaps)

Step 2: [1, 2, 3, 1, 2] (1 swap, element 1 shifts 1 place to the left)

Step 3: [1, 2, 3, 1, 2] (No swaps)

Step 4: [1, 1, 2, 3, 2] (2 swaps; element 1 shifts 2 places to the left)

Step 5: [1, 1, 2, 2, 3] (1 swap, element 2 shifts 1 place to the left)

Total number of swaps: 1 + 2 + 1 = 4

Example 2:

Input:

7

12 15 1 5 6 14 11

Output:

10

Explanation:

Step 1: [12, 15, 1, 5, 6, 14, 11] (No swaps)

Step 2: [12, 15, 1, 5, 6, 14, 11] (1 swap, element 15 shifts 1 place to the left)

Step 3: [12, 15, 1, 5, 6, 14, 11] (No swaps)

Step 4: [1, 12, 15, 5, 6, 14, 11] (2 swaps, element 1 shifts 2 places to the left)

Step 5: [1, 5, 12, 15, 6, 14, 11] (1 swap, element 5 shifts 1 place to the left)

Step 6: [1, 5, 6, 12, 15, 14, 11] (2 swaps, element 6 shifts 2 places to the left)

Step 7: [1, 5, 6, 12, 14, 15, 11] (1 swap, element 14 shifts 1 place to the left)

Step 8: [1, 5, 6, 11, 12, 14, 15] (3 swaps, element 11 shifts 3 places to the left)

Total number of swaps: 1 + 2 + 1 + 2 + 1 + 3 = 10

Input Format

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

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

Output Format

The output prints the number of swaps performed during the insertion sort process.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
2 1 3 1 2
Output: 4

Answer

```
#include <stdio.h>

int insertionSortAndCountSwaps(int arr[], int n) {
    int swapCount = 0;

    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            swapCount++;
            j--;
        }
        arr[j + 1] = key;
    }

    return swapCount;
```

```
}
int main() {
    int n;
    scanf("%d", &n);
    int arr[n];


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

    int swapCount = insertionSortAndCountSwaps(arr, n);

    printf("%d\n", swapCount);

    return 0;
}
```

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

4. Problem Statement

You're a coach managing a list of finishing times for athletes in a race. The times are stored in an array, and you need to sort this array in ascending order to determine the rankings.

You'll use the insertion sort algorithm to accomplish this.

Input Format

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

The second line contains n space-separated integers, each representing the finishing time of an athlete in seconds.

Output Format

The output prints the sorted finishing times of the athletes in ascending order.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
75 89 65 90 70

Output: 65 70 75 89 90

Answer

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n;


    scanf("%d", &n);
    int times[n];


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


    insertionSort(times, n);


    for (int i = 0; i < n; i++) {
```

```
            printf("%d", times[i]);
            if (i < n - 1) {
                printf(" ");
            }
        }
        printf("\n");

        return 0;
}
```

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

5. Problem Statement

Vishnu, a math enthusiast, is given a task to explore the magic of numbers. He has an array of positive integers, and his goal is to find the integer with the highest digit sum in the sorted array using the merge sort algorithm.

You have to assist Vishnu in implementing the merge sort algorithm.

Input Format

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

The second line consists of N space-separated integers, representing the array elements.

Output Format

The first line of output prints "The sorted array is: " followed by the sorted array, separated by a space.

The second line prints "The integer with the highest digit sum is: " followed by an integer representing the highest-digit sum.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
123 456 789 321 654
Output: The sorted array is: 123 321 456 654 789
The integer with the highest digit sum is: 789

Answer

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];


    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
```

```
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int digitSum(int num) {
    int sum = 0;
    while (num > 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

int main() {
    int N;


    scanf("%d", &N);
    int arr[N];


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


    mergeSort(arr, 0, N - 1);


    printf("The sorted array is: ");
```

```
    for (int i = 0; i < N; i++) {
        printf("%d", arr[i]);
        if (i < N - 1) {
            printf(" ");
        }
    }
    printf("\n");


    int maxDigitSum = 0;
    int maxDigitSumNum = arr[0];

    for (int i = 0; i < N; i++) {
        int currentDigitSum = digitSum(arr[i]);
        if (currentDigitSum > maxDigitSum) {
            maxDigitSum = currentDigitSum;
            maxDigitSumNum = arr[i];
        }
    }


    printf("The integer with the highest digit sum is: %d\n", maxDigitSumNum);

    return 0;
}
```

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