

# Rajalakshmi Engineering College

Name: MOHANAPRIYA N  
Email: 240801203@rajalakshmi.edu.in  
Roll no: 240801203  
Phone: 8667043590  
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 : 1  
Total Mark : 5  
Marks Obtained : 4

#### Section 1 : Coding

##### 1. 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>
struct node{
    int data;
    struct node*next;
};
struct node*head=NULL,*last=NULL,*newnode,*ptr;
void create(int a)
{
    for(int i=0;i<a;i++)
    {
        newnode=(struct node*)malloc(sizeof(struct node));
        scanf("%d",&newnode->data);
        newnode->next=NULL;
```

```

    if(head==NULL)
    {
        head=newnode;
    }
    else{
        last->next=newnode;
    }
    last=newnode;
}

void display()
{
    ptr=head;
    while(ptr!=NULL)
    {
        printf("%d ",ptr->data);
        ptr=ptr->next;
    }
}

int main()
{
    int a,b;
    scanf("%d",&a);
    create(a);
    scanf("%d",&b);
    create(b);
    display();
    return 0;
}

```

**Status :** Correct

**Marks :** 1/1

## 2. 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 List{  
    int value;  
    struct List*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);
}

```

```

void traverse(Node*head)
{
    if(head==NULL)
    {
        printf("The list is empty");
        return;
    }
    while(head!=NULL)
    {
        printf("%d ",head->value);
        head=head->next;
    }
    printf("\n");
}

```

```

void insertNode(Node** head,int value,int pos)
{
    if(pos==0)
    {
        Node*temp=newnode(value);
        temp->next=*head;
        *head=temp;
    }
    else{
        Node*temp=*head;
        while(temp->next!=NULL)
        {
            temp=temp->next;
        }
        temp->next=newnode(value);
    }
}

```

```
}  
}  
}  
void insertBeforeValue(Node**head,int value,int n)
```

```
{  
    Node*temp=*head;  
    if(temp->value==value)  
    {  
        Node*node=newnode(n);  
        node->next=*head;  
        *head=node;  
        return;  
    }  
    while(temp->next->value!=value)  
    {  
        temp=temp->next;  
    }  
    Node*node=newnode(n);  
    node->next=temp->next;  
    temp->next=node;  
}
```

```
void insertAfterValue(Node**head,int value,int n)
```

```
{  
    Node*temp=*head;  
    while(temp->value!=value)  
    {  
        temp=temp->next;  
    }  
    Node*node=newnode(n);  
    node->next=temp->next;  
    temp->next=node;  
}
```

```
void deleteNode(Node**head,int pos)
```

```
{  
    if(pos==0)  
    {  
        Node*temp=*head;  
        *head=temp->next;  
        free(temp);  
    }  
}
```

```

else{
    Node*temp=*head;
    while(temp->next->next!=NULL)
    {
        temp=temp->next;
    }
    Node*temp1=temp->next;
    temp->next=NULL;
    free(temp1);
}
}
int search(Node*head,int value)
{
    while(head!=NULL)
    {
        if(head->value==value)
        {
            return 1;
        }
        head=head->next;
    }
    return 0;
}

void deleteBeforeValue(Node*head,int value)
{
    while(head->next->next->value!=value)
    {
        head=head->next;
    }
    Node*temp=head->next;
    head->next=head->next->next;
    free(temp);
}

void deleteAfterValue(Node* head,int value)
{
    while(head->value!=value)
    {
        head=head->next;
    }
}

```



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

```
int main()
{
    Node*head=NULL;
    while(1)
    {
        int ch;
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
            {
                while(1)
                {
                    int n;
                    scanf("%d",&n);
                    if(n== -1)
                    {
                        break;
                    }
                    insertNode(&head,n);
                }
                printf("LINKED LIST CREATED\n");
                break;
            }
            case 2:
            {
                traverse(head);
                break;
            }
            case 3:
            {
                int n;
                scanf("%d",&n);
                insertNode(&head,n,0);
                printf("The linked list after insertion at the beginning is:\n");
                traverse(head);
                break;
            }
        }
    }
}
```

```
}
```

```
case 4:
```

```
{
```

```
    int n;
```

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

```
    insertNode(&head,n,1);
```

```
    printf("The linked list after insertion at the end is:\n");
```

```
    traverse(head);
```

```
    break;
```

```
}
```

```
case 5:
```

```
{
```

```
    int value,n;
```

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

```
    if(!search(head,value))
```

```
    {
```

```
        printf("Value not found in the list\n");
```

```
    }
```

```
    else{
```

```
        insertBeforeValue(&head,value,n);
```

```
    }
```

```
    printf("The linked list after insertion before a value is:\n ");
```

```
    traverse(head);
```

```
    break;
```

```
}
```

```
case 6:
```

```
{
```

```
    int value,n;
```

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

```
    if(!search(head,value))
```

```
    {
```

```
        printf("value not found in the list\n");
```

```
    }
```

```
    else{
```

```
        insertAfterValue(&head,value,n);
```

```
    }
```

```
    printf("The linked list after insertion after a value is:\n");
```

```
    traverse(head);
    break;
}
case 7:
{
    deleteNode(&head,0);
    printf("The linked list after deleteion from the beginning is:\n");
    traverse(head);
    break;
}
case 8:
{
    deleteNode(&head,1);
    printf("The linked list after deletion from the end is:\n");
    traverse(head);
    break;
}
case 9:
{
    int n;
    scanf("%d",&n);
    if(!search(head,n))
    {
        printf("Value not found in the list\n");
    }
    else{
        deleteBeforeValue(head,n);
    }
    printf("The linked list after deletion before a value is:\n");
    traverse(head);
    break;
}
case 10:
{
    int n;
    scanf("%d",&n);
    if(!search(head,n))
    {
        printf("Value not found in the list\n");
    }
}
```

```

        else{
            deleteAfterValue(head,n);

        }
        printf("The linked list after deletion after a value is:\n");
        traverse(head);
        break;
    }
    case 11:
    {
        return 0;
        break;
    }
    default:
        printf("Invalid option! Please try again\n");
        break;
    }
}
}

```

**Status :** Partially correct

**Marks :** 0.5/1

### 3. Problem Statement

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

Your task is to help Emily in implementing the same.

#### **Input Format**

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

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.

- For choice 3 to insert a node at the beginning. The next line contains an integer data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

### **Output Format**

For choice 1, print "LINKED LIST CREATED".

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

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

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

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

For choice 11 terminate the program.

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

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 1

5

3

7

-1

2

11

Output: LINKED LIST CREATED

5 3 7

### **Answer**

// You are using GCC

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

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

```
typedef struct List{  
    int value;  
    struct List*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);  
}
```

```
void traverse(Node* head)
{
    if(head==NULL)
    {
        printf("The list is empty");
        return;
    }
    while(head!=NULL)
    {
        printf("%d ",head->value);
        head=head->next;
    }
    printf("\n");
}
void insertNode(Node**head,int value,int pos)
{
    if(pos==0)
    {
        Node*temp=newnode(value);
        temp->next=*head;
        *head=temp;
    }
    else{
        Node*temp=*head;
        while(temp->next !=NULL)
        {
            temp =temp->next;
        }
        temp->next=newnode(value);
    }
}
```

```
void insertBeforeValue(Node**head,int value,int n)
{
    Node*temp =*head;
    if(temp->value==value)
    {
        Node*node=newnode(n);
        node->next=*head;
        *head=node;
        return;
    }
}
```

```

while(temp->next->value!=value)
{
    temp=temp->next;
}
Node*node=newnode(n);
node->next=temp->next;
temp->next=node;
}
void insertAfterValue(Node**head,int value,int n)
{
    Node*temp=*head;
    while(temp->value!=value)
    {
        temp=temp->next;
    }
    Node*node=newnode(n);
    node->next=temp->next;
    temp->next=node;
}
void deleteNode(Node**head,int pos)
{
    if(pos==0)
    {
        Node*temp=*head;
        *head=temp->next;
        free(temp);
    }
    else{
        Node*temp=*head;
        while(temp->next->next!= NULL)
        {
            temp=temp->next;
        }
        Node*temp1=temp->next;
        temp->next=NULL;
        free(temp1);
    }
}
int search(Node*head,int value)
{

```



```
while(head!=NULL)
{
    if(head->value==value)
    {
        return 1;
    }
    head=head->next;
}
return 0;
}
```

```
void deleteBeforeValue(Node*head,int value)
{
    while(head->next->next->value != value)
    {
        head=head->next;
    }
    Node*temp=head->next;
    head->next=head->next->next;
    free(temp);
}
```

```
void deleteAfterValue(Node*head,int value)
{
    while(head->value!=value)
    {
        head=head->next;
    }
    Node*temp=head->next;
    head->next=head->next->next;
    free(temp);
}
```

```
int main()
{
    Node*head=NULL;
    while(1)
    {
        int ch;
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
```

```

{
    while(1){
        int n;
        scanf("%d",&n);
        if(n==-1){
            break;
        }
        insertNode(&head,n);
    }
    printf("LINKED LIST CREATED\n");
    break;
}
case 2:
    traverse(head);
    break;

case 3:
{
    int n;
    scanf("%d",&n);
    insertNode(&head,n,0);
    printf("The linked list after insertion at the beginning is:\n");
    traverse(head);
    break;
}
case 4:
{
    int n;
    scanf("%d",&n);
    insertNode(&head,n,1);
    printf("The linked list after insertion at the end is:\n");
    traverse(head);
    break;
}
case 5:
{
    int value,n;
    scanf("%d %d",&value,&n);
    if(!search(head,value))
    {
        printf("Value not found in the list\n");
    }
}

```

```

    }
    else{
        insertBeforeValue(&head,value,n);
    }
    printf("The linked list after insertion before a value is:\n");
    traverse((head));
    break;
}
case 6:
{
    int value,n;
    scanf("%d %d",&value,&n);
    if(!search(head,value))
    {
        printf("value not found in the list\n");
    }
    else{
        insertAfterValue(&head,value,n);

    }
    printf("The linked list after insertion after a value is:\n");
    traverse(head);
    break;
}
case 7:
{
    deleteNode(&head,0);
    printf("The linked list after deletion from the beginning is:\n");
    traverse(head);
    break;
}
case 8:
{
    deleteNode(&head,1);
    printf("The linked list after deletion from the end is:\n");
    traverse(head);
    break;
}
case 9:
{
    int n;
    scanf("%d",&n);

```

```

        if(!search(head,n))
        {
            printf("Value not found in the list\n");
        }
        else{
            deleteBeforeValue(head,n);

        }
        printf("The linked list after deletion before a value is:\n");
        traverse(head);
        break;
    }
    case 10:{
        int n;
        scanf("%d",&n);
        if(!search(head,n))
        {
            printf("Value not found in the list\n");
        }
        else{
            deleteAfterValue(head,n);
        }
        printf("The linked list after deletion after a value is:\n");
        traverse(head);
        break;
    }
    case 11:
    {
        return 0;
        break;
    }
    default:
    {
        printf("Invalid option! Please try again\n");
        break;
    }
}
}
}

```

**Status :** Partially correct

**Marks :** 0.5/1

#### 4. Problem Statement

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

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

Example

Input:

2

13

12

11

1

Output:

36

Explanation:

The degree of the polynomial is 2.

Calculate the value of  $x^2$ :  $13 * 12 = 13$ .

Calculate the value of  $x^1$ :  $12 * 11 = 12$ .

Calculate the value of  $x^0$ :  $11 * 10 = 11$ .

Add the values of  $x^2$ ,  $x^1$  and  $x^0$  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  $x^2$ .

The third line consists of the coefficient of  $x^1$ .

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>

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

```
};struct node *head=NULL,*last=NULL,*ptr,*newnode;
```

```
int main()
```

```
{
```

```
    int a,b,s=0;
```

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

```
    for(int i=0;i<a+1;i++)
```

```
    {
```

```
        newnode=(struct node*)malloc(sizeof(struct node));
```

```
        scanf("%d",&newnode->data);
```

```

newnode->next=NULL;
if(head==NULL)
head=newnode;
else
last->next=newnode;
last=newnode;
}
scanf("%d",&b);
int n=a;
ptr=head;
while(ptr!=NULL)
{
s+=(pow(b,n--))*(ptr->data);
ptr=ptr->next;
}
printf("%d",s);
}

```

**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***

```
// 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);

return 0;
}
```

**Status :** Correct

**Marks :** 1/1

# Rajalakshmi Engineering College

Name: MOHANAPRIYA N  
Email: 240801203@rajalakshmi.edu.in  
Roll no: 240801203  
Phone: 8667043590  
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

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>
```

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

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

```
// Function to create a new node
```

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

// Function to append a node to the front of the doubly linked list

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

// Function to delete a node at a given position from the beginning

```
void deleteNodeAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        return;
    }
```

```
    struct Node* current = *head;
```

// If the node to be deleted is the head node

```
    if (position == 1) {
        *head = current->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(current);
        return;
    }
```

// Traverse to the node at the given position

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

```

// If the position is invalid (out of bounds)
if (current == NULL) {
    return;
}

// Adjust the pointers to remove the node
if (current->prev != NULL) {
    current->prev->next = current->next;
}

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

free(current);
}

// Function to display the doubly linked list
void displayList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

// Function to free the memory allocated for the linked list
void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;
    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}

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

```

```

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

// Read the data values and append to the front
for (int i = 0; i < n; i++) {
    scanf("%d", &data);
    appendFront(&head, data);
}

// Display the original list
displayList(head);

// Read the position to delete
scanf("%d", &x);

// Delete the node at the given position
deleteNodeAtPosition(&head, x);

// Display the updated list
displayList(head);

// Free the allocated memory
freeList(head);

return 0;
}

```

**Status :** Correct

**Marks :** 10/10

## 2. 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>
```

```
// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};
```

```
// Function to create a new node
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->next = NULL;
```

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

// Function to insert a node at the end of the doubly linked list

```
void insertEnd(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;  
}
```

// Function to get the tail of the doubly linked list

```
struct Node* getTail(struct Node* head) {  
    struct Node* temp = head;  
    while (temp != NULL && temp->next != NULL) {  
        temp = temp->next;  
    }  
    return temp;  
}
```

// Function to rotate the doubly linked list clockwise by k positions

```
struct Node* rotateClockwise(struct Node* head, int k, int n) {  
    if (head == NULL || k == 0 || k % n == 0) {  
        return head;  
    }
```

```
    // Adjust k if it's greater than n  
    k = k % n;
```

```
    struct Node* current = head;  
    int count = 1;  
    while (count < n - k) {  
        current = current->next;  
        count++;  
    }
```

```

    }

    struct Node* newHead = current->next;
    struct Node* tail = getTail(head);

    // Make the k-th node from the end the new head
    newHead->prev = NULL;

    // Make the last node point to the original head
    tail->next = head;
    head->prev = tail;

    // Make the (n-k)-th node the new tail
    current->next = NULL;

    return newHead;
}

// Function to display the doubly linked list
void displayList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

// Function to free the memory allocated for the linked list
void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;
    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}

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

```

```

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

// Read the data values and insert at the end
for (int i = 0; i < n; i++) {
    scanf("%d", &data);
    insertEnd(&head, data);
}

// Read the number of positions to rotate
scanf("%d", &k);

// Rotate the doubly linked list clockwise
head = rotateClockwise(head, k, n);

// Display the rotated list
displayList(head);

// Free the allocated memory
freeList(head);

return 0;
}

```

**Status :** Correct

**Marks :** 10/10

### 3. 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>
```

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

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

```
// Function to create a new node
```

```
struct Node* createNode(int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
```

```
// Function to insert a node at the end of the doubly linked list
void append(struct Node** head, struct Node** tail, int data) {
    struct Node* newNode = createNode(data);
    if (!*head) {
        *head = newNode;
        *tail = newNode;
        return;
    }
    newNode->prev = *tail;
    (*tail)->next = newNode;
    *tail = newNode;
}
```

```
// Function to print the doubly linked list
void printList(struct Node* head) {
    struct Node* current = head;
    while (current) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

```
// Function to find and print the middle element(s)
void findMiddle(struct Node* head) {
    struct Node* slowPtr = head;
    struct Node* fastPtr = head;

    while (fastPtr && fastPtr->next) {
        slowPtr = slowPtr->next;
        fastPtr = fastPtr->next->next;
    }

    if (!fastPtr) { // Even number of elements
        printf("%d %d\n", slowPtr->prev->data, slowPtr->data);
    }
}
```

```

    } else { // Odd number of elements
        printf("%d\n", slowPtr->data);
    }
}

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

    scanf("%d", &n);

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

    printList(head);
    findMiddle(head);

    // Free the allocated memory (important for good practice)
    struct Node* current = head;
    struct Node* nextNode;
    while (current) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
    head = NULL;
    tail = NULL;

    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. 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***

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

```
// Node structure for a doubly linked list
struct Node {
    int data;
    struct Node* prev;
```



```

    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        // Handle memory allocation failure
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

```

```

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

```

```

// Function to display the doubly linked list
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

```

```

// Function to check if the doubly linked list is a palindrome
bool isPalindrome(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return true; // An empty list or a list with one element is a palindrome
    }

    struct Node* front = head;
    struct Node* rear = head;
    while (rear->next != NULL) {
        rear = rear->next; // Move rear to the last node
    }

    while (front != rear && front->prev != rear) {
        if (front->data != rear->data) {
            return false; // If data doesn't match, it's not a palindrome
        }
        front = front->next;
        rear = rear->prev;
    }
    return true; // If the loop finishes without finding a mismatch, it's a palindrome
}

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

    struct Node* head = NULL;
    for (int i = 0; i < n; i++) {
        int data;
        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");
    }

    // Free the allocated memory

```

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

return 0;
}
```

**Status :** Correct

**Marks :** 10/10

## 5. 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;  
};
```

```
// Function to create a new node
```

```
struct Node* createNode(int data) {  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    newNode->data = data;  
    newNode->prev = NULL;  
    newNode->next = NULL;  
    return newNode;  
}
```

```
// Insert at front
```

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

// Insert at specific position (1-based indexing)

```
void insertAtPosition(struct Node** head, int position, int data) {
    if (position == 1) {
        insertAtFront(head, data);
        return;
    }
```

```
    struct Node* temp = *head;
    int count = 1;
```

```
    while (temp != NULL && count < position - 1) {
        temp = temp->next;
        count++;
    }
```

```
    if (temp == NULL)
        return; // Position out of bounds (safe-guard)
```

```
    struct Node* newNode = createNode(data);
    newNode->next = temp->next;
    newNode->prev = temp;
```

```
    if (temp->next != NULL)
        temp->next->prev = newNode;
```

```
    temp->next = newNode;
}
```

// Function to print the list

```
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
```

```
    }  
    printf("\n");  
}  
  
// Main function  
int main() {  
    int N, position, data;  
    scanf("%d", &N);  
  
    int i, value;  
    struct Node* head = NULL;  
  
    for (i = 0; i < N; i++) {  
        scanf("%d", &value);  
        insertAtFront(&head, value);  
    }  
  
    scanf("%d", &position);  
    scanf("%d", &data);  
  
    printList(head);  
    insertAtPosition(&head, position, data);  
    printList(head);  
  
    return 0;  
}
```

**Status :** Correct

**Marks : 10/10**

# Rajalakshmi Engineering College

Name: MOHANAPRIYA N  
Email: 240801203@rajalakshmi.edu.in  
Roll no: 240801203  
Phone: 8667043590  
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

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 20
```

```
int main() {
```

```
    int queue[MAX]; // Array to represent the queue
```

```
    int front = 0, rear = -1; // Queue pointers
```

```
    int n;
```



```
// Input: number of tickets
scanf("%d", &n);

// Constraints check (optional)
if (n < 2 || n > MAX) {
    printf("Invalid number of tickets.\n");
    return 1;
}

// Input: ticket IDs
for (int i = 0; i < n; i++) {
    int ticket;
    scanf("%d", &ticket);

    // Constraints check (optional)
    if (ticket < 0 || ticket > 100) {
        printf("Invalid ticket identifier.\n");
        return 1;
    }

    rear++;
    queue[rear] = ticket;
}

// Display initial queue
printf("Queue: ");
for (int i = front; i <= rear; i++) {
    printf("%d ", queue[i]);
}

// Dequeue operation
front++;

// Display queue after dequeue
printf(" Queue After Dequeue: ");
for (int i = front; i <= rear; i++) {
    printf("%d ", queue[i]);
}

printf("\n");

return 0;
```

}

**Status :** Correct

**Marks :** 10/10

## 2. 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 array in descending order
void sortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                // swap
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

// Main function
int main() {
    int queue[10]; // fixed size queue
    int N, K;

    // Read N
    scanf("%d", &N);

    if (N > 10) {
        printf("Queue size exceeds maximum limit of 10.\n");
        return 1;
    }

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

```

// Read K
scanf("%d", &K);

// Validate K
if (K > N) {
    printf("\nInvalid input: K cannot be greater than number of elements in the
queue.\n");
    return 1;
}

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

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

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 enqueueing 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>
```

```
// Node structure
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
// Queue front and rear pointers
```

```
struct Node* front = NULL;
```

```
struct Node* rear = NULL;
```

```
// Enqueue operation
```

```
void enqueue(int value) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->data = value;
```

```
    newNode->next = NULL;
```

```
    if (rear == NULL) {
```

```
        front = rear = newNode;
```

```
    } else {
```

```
        rear->next = newNode;
```

```
        rear = newNode;
```

```
    }
```

```
}
```

```
// Dequeue operation and print
```

```

void dequeueAndPrint() {
    printf("Dequeued elements:");
    while (front != NULL) {
        printf(" %d", front->data);
        struct Node* temp = front;
        front = front->next;
        free(temp);
    }
    printf("\n");
}

```

```

int main() {
    int num;
    while (1) {
        scanf("%d", &num);
        if (num == -1)
            break;
        if (num >= 0)
            enqueue(num);
    }

    dequeueAndPrint();
    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. 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>

```
int main() {
    int n, i, multiple;
    int queue[50], result[50]; // As per constraints, max 50 elements
    int resultIndex = 0;

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

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

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

    // Print original queue
    printf("Original Queue:");
    for (i = 0; i < n; i++) {
        printf(" %d", queue[i]);
    }

    // Perform selective dequeue: store elements not divisible by 'multiple'
    for (i = 0; i < n; i++) {
        if (queue[i] % multiple != 0) {
            result[resultIndex++] = queue[i];
        }
    }

    // Print the result queue
    printf("\nQueue after selective dequeue:");
    for (i = 0; i < resultIndex; i++) {
```



```
        printf(" %d", result[i]);  
    }  
  
    printf("\n");  
    return 0;  
}
```

**Status :** Correct

**Marks :** 10/10

## 5. 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 25

int queue[MAX];

int front = -1, rear = -1;

// Enqueue operation

void enqueue(int customerID) {

if (rear == MAX - 1) {

// Queue is full (though this is not required per constraints)

return;

}

if (front == -1)

front = 0;

rear++;

queue[rear] = customerID;

}

// Dequeue operation

void dequeue() {

if (front == -1 || front > rear) {

printf("Underflow\nQueue is empty\n");

} else {

front++; // Serve one customer

if (front > rear) {

// Queue has become empty

printf("Queue is empty\n");

} else {

// Display the current queue

for (int i = front; i <= rear; i++) {

printf("%d ", queue[i]);

}

printf("\n");

}

}

```

}

// Main function
int main() {
    int N;
    scanf("%d", &N);

    if (N == 0) {
        // Read dummy zero for empty list as per input format
        int dummy;
        scanf("%d", &dummy);
        printf("Underflow\nQueue is empty\n");
        return 0;
    }

    // Enqueue customer IDs
    for (int i = 0; i < N; i++) {
        int customerID;
        scanf("%d", &customerID);
        enqueue(customerID);
    }

    // Serve (dequeue) one customer and show remaining queue
    dequeue();

    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: MOHANAPRIYA N  
Email: 240801203@rajalakshmi.edu.in  
Roll no: 240801203  
Phone: 8667043590  
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

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>

// Structure for a tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* 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 {
        root->right = insert(root->right, data);
    }
    return root;
}

```

```

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

```

```

// Main function
int main() {
    int N;
    scanf("%d", &N);

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

    postOrder(root);
    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

## 2. 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>
```

```
// Define a node structure for the BST
struct Node {
    int data;
    struct Node* left;
    struct Node* 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 data 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 search for a target value in the BST
int search(struct Node* root, int target) {
    if (root == NULL) {
        return 0; // not found
    }
    if (root->data == target) {
        return 1; // found
    } else if (target < root->data) {
        return search(root->left, target);
    } else {
        return search(root->right, target);
    }
}
```

```
int main() {
    struct Node* root = NULL;
    int value;

    // Read BST elements until -1
    while (scanf("%d", &value) && value != -1) {
        root = insert(root, value);
    }
}
```

```
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

### 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 a node in the BST
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
```

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

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

```
// Function to trim the BST by removing nodes outside the [min, max] range
struct Node* trimBST(struct Node* root, int min, int max) {
    if (root == NULL) {
        return NULL;
    }
```

```

    }
    // Recursively trim left and right subtrees
    root->left = trimBST(root->left, min, max);
    root->right = trimBST(root->right, min, max);

    // If the current node is out of range, remove it
    if (root->data < min) {
        struct Node* rightChild = root->right;
        free(root);
        return rightChild;
    }
    if (root->data > max) {
        struct Node* leftChild = root->left;
        free(root);
        return leftChild;
    }
    return root;
}

```

```

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

```

```

// Main function
int main() {
    int N, min, max;

    // Read input
    scanf("%d", &N);
    int elements[N];
    for (int i = 0; i < N; i++) {
        scanf("%d", &elements[i]);
    }
    scanf("%d %d", &min, &max);

    // Create the BST

```

```

    struct Node* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, elements[i]);
    }

    // Trim the BST by removing nodes outside the range
    root = trimBST(root, min, max);

    // Print the remaining elements in in-order traversal
    inorder(root);
    printf("\n");

    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. 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>
```

```
typedef struct Node {  
    int key;  
    struct Node* left;  
    struct Node* right;  
} Node;
```

```
typedef struct Queue {  
    Node* data[100];  
    int front, rear;  
} Queue;
```

```
void enqueue(Queue* q, Node* node) {
    q->data[q->rear++] = node;
}
```

```
Node* dequeue(Queue* q) {
    return q->data[q->front++];
}
```

```
int isEmpty(Queue* q) {
    return q->front == q->rear;
}
```

```
Node* newNode(int key) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->key = key;
    node->left = node->right = NULL;
    return node;
}
```

```
Node* insert(Node* root, int key) {
    if (!root) return newNode(key);
    if (key < root->key)
        root->left = insert(root->left, key);
    else if (key > root->key)
        root->right = insert(root->right, key);
    return root;
}
```

```
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
```

```
Node* deleteNode(Node* root, int key) {
    if (!root) return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
```

```

else {
    if (!root->left) {
        Node* temp = root->right;
        free(root);
        return temp;
    }
    else if (!root->right) {
        Node* temp = root->left;
        free(root);
        return temp;
    }
    Node* temp = minValueNode(root->right);
    root->key = temp->key;
    root->right = deleteNode(root->right, temp->key);
}
return root;
}

```

```

void printLevelOrder(Node* root) {
    if (!root) return;
    Queue q = { .front = 0, .rear = 0 };
    enqueue(&q, root);
    while (!isEmpty(&q)) {
        Node* current = dequeue(&q);
        printf("%d ", current->key);
        if (current->left) enqueue(&q, current->left);
        if (current->right) enqueue(&q, current->right);
    }
    printf("\n");
}

```

```

int main() {
    int N, x, y;
    scanf("%d", &N);
    int arr[N];

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

    scanf("%d", &x);
    scanf("%d", &y);
}

```

```

Node* root = NULL;

// Insert in the order given (not sorted, to match exact output)
for (int i = 0; i < N; i++) {
    root = insert(root, arr[i]);
}

printf("Initial BST:\n");
printLevelOrder(root);

root = insert(root, x);
printf("BST after inserting a new node %d:\n", x);
printLevelOrder(root);

root = deleteNode(root, y);
printf("BST after deleting node %d:\n", y);
printLevelOrder(root);

return 0;
}

```

**Status :** Correct

**Marks :** 10/10

## 5. 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>
```

```
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 node into the BST based on preorder traversal
```

```
struct Node* insertBST(struct Node* root, int data, int* prev_value) {  
    // If root is null, create a new node  
    if (root == NULL) {  
        return newNode(data);  
    }
```

```
    // If the data is less than the previous value, insert it into the left subtree
```

```

    if (data < root->data) {
        root->left = insertBST(root->left, data, prev_value);
    } else {
        root->right = insertBST(root->right, data, prev_value);
    }
    return root;
}

```

// Function to perform inorder traversal and print the result

```

void inorder(struct Node* root) {
    if (root == NULL) {
        return;
    }
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

```

// Function to build the BST from preorder traversal array

```

struct Node* buildBSTFromPreorder(int preorder[], int n) {
    struct Node* root = NULL;
    int prev_value = 0;
    for (int i = 0; i < n; i++) {
        root = insertBST(root, preorder[i], &prev_value);
    }
    return root;
}

```

```

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

```

// Read the preorder traversal

```

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

```

// Build the BST from the preorder traversal

```

struct Node* root = buildBSTFromPreorder(preorder, n);

```

// Perform in-order traversal and print the result

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

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: MOHANAPRIYA N  
Email: 240801203@rajalakshmi.edu.in  
Roll no: 240801203  
Phone: 8667043590  
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

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***

// You are using GCC

#include <stdio.h>

int main() {

int n, i, j, key, swapCount = 0;

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

int arr[n];

// Reading array elements

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

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

}

// Insertion Sort with swap counting

```

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

    // Move elements of arr[0..i-1], that are greater than key,
    // to one position ahead of their current position
    while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j--;
        swapCount++; // Count each shift as a swap
    }
    arr[j + 1] = key;
}

printf("%d\n", swapCount);
return 0;
}

```

**Status :** Correct

**Marks :** 10/10

## 2. 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**

```
// You are using GCC
```

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

```
// Merge function for merge sort
```

```
void merge(int arr[], int l, int m, int r) {
```

```
    int i, j, k;
```

```
    int n1 = m - l + 1;
```

```
    int n2 = r - m;
```

```
    int L[20], R[20]; // Max size needed = 10+10=20
```

```
    for (i = 0; i < n1; i++)
```

```
        L[i] = arr[l + i];
```

```
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[m + 1 + j];
```

```
    i = 0; j = 0; k = l;
```

```
    while (i < n1 && j < n2) {
```

```
        if (L[i] <= R[j])
```

```
            arr[k++] = L[i++];
```

```
        else
```



```

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

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

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

// Merge sort implementation
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

```

```

// Function to remove duplicates from a sorted array
int removeDuplicates(int arr[], int n, int result[]) {
    int i, j = 0;
    for (i = 0; i < n; i++) {
        if (i == 0 || arr[i] != arr[i - 1]) {
            result[j++] = arr[i];
        }
    }
    return j; // New length of array
}

```

```

int main() {
    int N, M, i;
    int arr1[10], arr2[10], merged[20], unique[20];

    // Input first array
    scanf("%d", &N);
    for (i = 0; i < N; i++)
        scanf("%d", &arr1[i]);

    // Input second array
    scanf("%d", &M);

```

```

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

// Merge the two arrays
for (i = 0; i < N; i++)
    merged[i] = arr1[i];
for (i = 0; i < M; i++)
    merged[N + i] = arr2[i];

int total = N + M;

// Sort the merged array
mergeSort(merged, 0, total - 1);

// Remove duplicates
int newSize = removeDuplicates(merged, total, unique);

// Print the result
for (i = 0; i < newSize; i++) {
    printf("%d ", unique[i]);
}
printf("\n");

return 0;
}

```

**Status :** Correct

**Marks :** 10/10

### 3. 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***

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

```
int iteration = 1;
```

```
void printArray(int arr[], int start, int end) {
    printf("Iteration %d: ", iteration++);
    for (int i = start; i <= end; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```

// Partition function for descending order
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choosing last element as pivot
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] >= pivot) { // For descending order
            i++;
            // Swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // Swap pivot with arr[i+1]
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    printArray(arr, low, high); // Print iteration
    return i + 1;
}

```

```

// QuickSort recursive function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high); // pi is partition index
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

```

```

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

    quickSort(scores, 0, n - 1);
    printf("Sorted Order: ");
}

```

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

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

```

int main() {
    int n, i, j, key;
    int times[20]; // Max constraint is 20

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

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

    // Insertion Sort Algorithm
    for (i = 1; i < n; i++) {
        key = times[i];
        j = i - 1;

        // Move elements greater than key one position ahead
        while (j >= 0 && times[j] > key) {
            times[j + 1] = times[j];
            j--;
        }
        times[j + 1] = key;
    }

    // Print sorted finishing times
    for (i = 0; i < n; i++) {
        printf("%d ", times[i]);
    }

    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***

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

```
// Function to calculate digit sum
```

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

```
// Merge function for merge sort
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
```

```
    int L[10], R[10]; // Since N <= 10
```

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

```
    int i = 0, j = 0, k = left;
```

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

```
    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}
```

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

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



```

int arr[10];
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(" ");
}

// Find the integer with highest digit sum
int maxDigitSum = -1;
int result = arr[0];
for (int i = 0; i < N; i++) {
    int dsum = digitSum(arr[i]);
    if (dsum > maxDigitSum) {
        maxDigitSum = dsum;
        result = arr[i];
    }
}

printf(" The integer with the highest digit sum is: %d", result);
return 0;
}

```

**Status :** Correct

**Marks :** 10/10