Rajalakshmi Engineering College

Name: ranjani prakash
Email: 240801267@rajalakshmi.edu.in
Roll no: 2116240801267
Phone: 6382555840
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

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

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

int main() {
    node* head = NULL;
    int choice, value, newvalue;
    while(1){
        scanf("%d", &choice);
        switch(choice){
            case 1:
                head = create();
                printf("LINKED LIST CREATED\n");
                break;
            case 2:
                display(head);
                break;
            case 3:
                scanf("%d", &value);
                head = insertbeg(head, value);
                printf("The linked list after insertion at the beginning is:\n");
                display(head);
                break;
            case 4:
                scanf("%d", &value);
                head = insertend(head, value);
                printf("The linked list after insertion at the end is:\n");
                display(head);
                break;
            case 5:
                scanf("%d %d", &value, &newvalue);
                head = insertbefval(head, value, newvalue);
                printf("The linked list after insertion before a value is:\n");
                display(head);
                break;
            case 6:
                scanf("%d %d", &value, &newvalue);
                head = insertaftval(head, value, newvalue);
                printf("The linked list after insertion after a value is:\n");
                display(head);
                break;
            case 7:
                head = deletebeg(head);
                printf("The linked list after deletion from the beginning is:\n");
                display(head);
                break;
            case 8:
                head = 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 = deleteafter(head, value);
                printf("The linked list after deletion after a value is:\n");
                display(head);
                break;
            case 11:
                return 0;
            default:
                printf("Invalid option! Please try again\n");
        }
        }return 0;
}
```

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

2. 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
// 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* deleteafter(node* head, int value){
    node* temp = head;
    while(temp != NULL && temp->data != value){
        temp = temp->next;
    }
    if(temp != NULL && temp->next != NULL){
        node* delnode = temp->next;
        temp->next = delnode->next;
        free(delnode);
    }
    return head;
}

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

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

```
            head = deletebeg(head);
            printf("The linked list after deletion from the beginning is:\n");
            display(head);
            break;
         case 8:
            head = 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 = deleteafter(head, value);
            printf("The linked list after deletion after a value is:\n");
            display(head);
            break;
         case 11:
            return 0;
         default:
            printf("Invalid option! Please try again\n");
      }
   }
return 0;
}
```

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

3. Problem Statement

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

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

Example

Input:

2

13

12

11

1

Output:

36

Explanation:

The degree of the polynomial is 2.

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

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

Calculate the value of x0: 11 * 10 = 11.

Add the values of x2, x1 and x0 together: 13 + 12 + 11 = 36.

Input Format

The first line of input consists of the degree of the polynomial.

The second line consists of the coefficient x2.

The third line consists of the coefficient of x1.

The fourth line consists of the coefficient x0.

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

Output Format

The output is the integer value obtained by evaluating the polynomial at the given value of x.

Refer to the sample output for formatting specifications.

Sample Test Case

```
Input: 2
13
12
11
1
Output: 36
```

Answer

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

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

// Function to create a new node with a given coefficient
Node* createNode(int coefficient) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->coefficient = coefficient;
   newNode->next = NULL;
   return newNode;
}

// Function to evaluate the polynomial at a given value of x
int evaluatePolynomial(Node* head, int x, int degree) {
   int result = 0;
   Node* current = head;
   int power = degree;
```

```
    while (current != NULL) {
        // calculate current->coefficient * x^power
        int termVal = current->coefficient;
        for (int i = 0; i < power; i++) {
            termVal *= x;
        }
        result += termVal;
        current = current->next;
        power--;
    }
    return result;
}

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

    Node* head = NULL;
    Node* tail = NULL;

    // Read coefficients in order of descending powers and create linked list
    for (int i = 0; i <= degree; i++) {
        int coefficient;
        scanf("%d", &coefficient);
        Node* newNode = createNode(coefficient);
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }

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

    // Evaluate polynomial at given x
    int result = evaluatePolynomial(head, x, degree);

    printf("%d\n", result);
    // Free linked list memory
    Node* current = head;
    while (current != NULL) {
        Node* next = current->next;
        free(current);
        current = next;
    }
    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>

// Node structure for singly linked list
typedef struct Node {
   int order_id;
   struct Node* next;
} Node;

// Create a new node with given order_id
Node* createNode(int order_id) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->order_id = order_id;
   newNode->next = NULL;
   return newNode;
}

// Append a node to the end of the list
void append(Node** head_ref, int order_id) {
   Node* newNode = createNode(order_id);
   if (*head_ref == NULL) {
      *head_ref = newNode;
      return;
```

```
    }
    Node* temp = *head_ref;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Merge two lists by linking the end of first to the head of second
Node* mergeLists(Node* morning, Node* evening) {
    if (morning == NULL) return evening;
    if (evening == NULL) return morning;

    Node* temp = morning;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = evening;

    return morning;
}

// Print list elements separated by space with an extra space at the end
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->order_id);
        temp = temp->next;
    }
    printf("\n");
}

// Free the linked list nodes
void freeList(Node* head) {
    Node* curr = head;
    while (curr != NULL) {
        Node* next = curr->next;
        free(curr);
        curr = next;
    }
}

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

    Node* morning = NULL;
    for (int i = 0; i < n; i++) {
        int order;
        scanf("%d", &order);
        append(&morning, order);
    }

    scanf("%d", &m);
    Node* evening = NULL;
    for (int i = 0; i < m; i++) {
        int order;
        scanf("%d", &order);
        append(&evening, order);
    }

    Node* merged = mergeLists(morning, evening);

    printList(merged);

    freeList(merged);

    return 0;
}
```

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>

void rearrange_list(int n, int arr[]) {
   // Arrays to store even and odd numbers
   int even_numbers[n];
   int odd_numbers[n];

   int even_count = 0;
   int odd_count = 0;

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

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

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

   printf("\n");
}

int main() {
   int n;

   // Read the size of the list
   scanf("%d", &n);
```

```
    int arr[n];

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

    // Call the function to rearrange and print the result
    rearrange_list(n, arr);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: ranjani prakash
Email: 240801267@rajalakshmi.edu.in
Roll no: 2116240801267
Phone: 6382555840
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

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>

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

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

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

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

// Function to check if doubly linked list is palindrome
int isPalindrome(Node* head, Node* tail) {
   Node* left = head;
   Node* right = tail;

   while (left != NULL && right != NULL && left != right && left->prev != right) {
       if (left->data != right->data) {
           return 0; // Not a palindrome
       }
       left = left->next;
       right = right->prev;
   }
   return 1; // Palindrome
}

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

   Node* head = NULL;
   Node* tail = NULL;
   for (int i = 0; i < N; i++) {
       int val;
       scanf("%d", &val);
       insertEnd(&head, &tail, val);
   }

   printList(head);

   if (isPalindrome(head, tail)) {
       printf("The doubly linked list is a palindrome\n");
   } else {
       printf("The doubly linked list is not a palindrome\n");
   }

   // Free memory
   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>

// Doubly linked list node
typedef struct Node {
   int data;
   struct Node* next;
   struct Node* prev;
} Node;

// Function to create a new node with given data
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
```

```
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

// Function to append node to the end of the list
void append(Node** head_ref, Node** tail_ref, int data) {
    Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        *tail_ref = newNode;
    } else {
        (*tail_ref)->next = newNode;
        newNode->prev = *tail_ref;
        *tail_ref = newNode;
    }
}

// Function to print the list elements separated by space (with an extra space at
end)
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to find length of the list
int length(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    return count;
}
```

```
// Function to print the middle node(s)
void printMiddle(Node* head, int size) {
   if (head == NULL) return;
   Node* slow = head;
   int mid = size / 2;
   if (size % 2 == 1) {
      // Odd number of elements: print middle element
      for (int i = 0; i < mid; i++) {
         slow = slow->next;
      }
      printf("%d\n", slow->data);
   } else {
      // Even number of elements: print two middle elements
      for (int i = 0; i < mid - 1; i++) {
         slow = slow->next;
      }
      printf("%d %d\n", slow->data, slow->next->data);
   }
}

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

   Node* head = NULL;
   Node* tail = NULL;

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

   printList(head);
   printMiddle(head, n);

   // Free list nodes
   Node* curr = head;
   while (curr != NULL) {
      Node* next = curr->next;
      free(curr);
```

```
    curr = next;
  }
  return 0;
}
```

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

3. 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

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

// Define the doubly linked list node structure
typedef struct Node {
   int data;
   struct Node* prev;
   struct Node* next;
} Node;

// Function to create a new node with given data
Node* createNode(int data) {
   Node* newNode = (Node*) malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}

// Function to insert node at the front of the list
void insertAtFront(Node** head_ref, int data) {
   Node* newNode = createNode(data);
   newNode->next = *head_ref;
   if (*head_ref != NULL) {
      (*head_ref)->prev = newNode;
   }
   *head_ref = newNode;
}

// Function to print the elements of the list separated by space
void printList(Node* head) {
   Node* current = head;
```

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

// Function to delete node at given position (1-indexed) from the beginning
void deleteNodeAtPosition(Node** head_ref, int position) {
    if (*head_ref == NULL || position < 1) {
        return;
    }

    Node* current = *head_ref;
    // Move to the node to be deleted
    for (int i=1; current != NULL && i < position; i++) {
        current = current->next;
    }

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

    // If node to be deleted is head
    if (current == *head_ref) {
        *head_ref = current->next;
        if (*head_ref != NULL) {
            (*head_ref)->prev = NULL;
        }
        free(current);
        return;
    }

    // Adjust pointers of prev and next nodes
    if (current->prev != NULL) {
        current->prev->next = current->next;
    }
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    free(current);
```

```
}

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

    Node* head = NULL;

    // Read elements and insert at front
    for (int i = 0; i < N; i++) {
        int val;
        scanf("%d", &val);
        insertAtFront(&head, val);
    }

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

    // Print original list
    printList(head);

    // Delete node at position X
    deleteNodeAtPosition(&head, X);

    // Print updated list
    printList(head);

    // Free allocated memory
    Node* curr = head;
    while (curr != NULL) {
        Node* next = curr->next;
        free(curr);
        curr = next;
    }

    return 0;
}
```

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

4. 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>

// Node structure for doubly linked list
typedef struct Node {
   int data;
   struct Node* prev;
   struct Node* next;
} Node;
```

```
// Function to create a new node with given data
Node* createNode(int data) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert node at the end
void insertAtEnd(Node** head_ref, int data) {
    Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }
    Node* last = *head_ref;
    while (last->next != NULL)
        last = last->next;
    last->next = newNode;
    newNode->prev = last;
}

// Function to print list elements separated by space with an extra space at end
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to rotate the list clockwise by k positions
void rotateClockwise(Node** head_ref, int k) {
    if (*head_ref == NULL || k == 0)
        return;
    // Find the length of the list
    Node* tail = *head_ref;
    int length = 1;
    while (tail->next != NULL) {
        tail = tail->next;
        length++;
    }

    // Adjust k to be within the length
    k = k % length;
    if (k == 0)
        return;

    // Find the new tail: (length - k)-th node
    Node* newTail = *head_ref;
    for (int i = 1; i < length - k; i++) {
        newTail = newTail->next;
    }

    // New head is next of newTail
    Node* newHead = newTail->next;

    // Break the link
    newTail->next = NULL;
    newHead->prev = NULL;

    // Old tail next points to old head
    tail->next = *head_ref;
    (*head_ref)->prev = tail;

    // Update head
    *head_ref = newHead;
}

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

    rotateClockwise(&head, k);

    printList(head);

    // Free memory
    Node* current = head;
    while (current != NULL) {
        Node* next = current->next;
        free(current);
        current = next;
    }

    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>

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

// Function to create a new node with given data
Node* createNode(int data) {
   Node* newNode = (Node*) malloc(sizeof(Node));
```

```
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the front of the list
void insertAtFront(Node** head_ref, int data) {
    Node* newNode = createNode(data);
    newNode->next = *head_ref;
    if (*head_ref != NULL) {
        (*head_ref)->prev = newNode;
    }
    *head_ref = newNode;
}

// Function to insert a node at a specific position
void insertAtPosition(Node** head_ref, int position, int data) {
    if (position < 1) return; // Invalid position

    Node* newNode = createNode(data);
    if (position == 1) {
        insertAtFront(head_ref, data);
        return;
    }

    Node* current = *head_ref;
    for (int i = 1; current != NULL && i < position - 1; i++) {
        current = current->next;
    }

    if (current == NULL) {
        return; // Position is greater than the length of the list
    }

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

// Function to print the elements of the list separated by space
void printList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

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

    Node* head = NULL;

    // Read initial elements and insert at front
    for (int i = 0; i < N; i++) {
        int val;
        scanf("%d", &val);
        insertAtFront(&head, val);
    }

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

    // Print original list
    printList(head);

    // Insert new value at the specified position
    insertAtPosition(&head, position, data);

    // Print updated list
    printList(head);

    // Free allocated memory
    Node* current = head;
    while (current != NULL) {
        Node* next = current->next;
        free(current);
        current = next;
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: ranjani prakash
Email: 240801267@rajalakshmi.edu.in
Roll no: 2116240801267
Phone: 6382555840
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

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

#define MAX_TICKETS 20

// Define the structure for the queue
struct Queue {
   int tickets[MAX_TICKETS];
   int front;
   int rear;
```

```
};

// Function to initialize the queue
void initializeQueue(struct Queue* q) {
    q->front = 0;
    q->rear = 0;
}

// Function to enqueue a ticket
void enqueue(struct Queue* q, int ticket) {
    if (q->rear < MAX_TICKETS) {
        q->tickets[q->rear] = ticket;
        q->rear++;
    } else {
        printf("Queue is full. Cannot enqueue more tickets.\n");
    }
}

// Function to dequeue a ticket
int dequeue(struct Queue* q) {
    if (q->front < q->rear) {
        return q->tickets[q->front++];
    } else {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1; // Return -1 if the queue is empty
    }
}

// Function to display the current state of the queue
void displayQueue(struct Queue* q) {
    if (q->front < q->rear) {
        for (int i = q->front; i < q->rear; i++) {
            printf("%d", q->tickets[i]);
            if (i < q->rear - 1) {
                printf(" ");
            }
        }
    }
}

int main() {
    struct Queue ticketQueue;
    initializeQueue(&ticketQueue);

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

    // Read the ticket identifiers and enqueue them
    for (int i = 0; i < n; i++) {
        int ticket;
        scanf("%d", &ticket);
        enqueue(&ticketQueue, ticket);
    }

    // Display the current state of the queue
    printf("Queue: ");
    displayQueue(&ticketQueue);
    printf("\n");

    // Dequeue the first ticket
    dequeue(&ticketQueue);

    // Display the queue after dequeue
    printf("Queue After Dequeue: ");
    displayQueue(&ticketQueue);
    printf("\n");

    return 0;
}
```

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

2. 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

```
#include <stdio.h>

#define MAX_CUSTOMERS 25

void displayQueue(int queue[], int size) {
    if (size == 0) {
        printf("Queue is empty\n");
    } else {
        for (int i = 0; i < size; i++) {
            printf("%d", queue[i]);
            if (i < size - 1) {
                printf(" ");
            }
        }
        printf("\n");
    }
}

void serveCustomer(int queue[], int *size) {
    if (*size == 0) {
        printf("Underflow\n");
    } else {
        // Shift customers in the queue
        for (int i = 0; i < *size - 1; i++) {
            queue[i] = queue[i + 1];
        }
        (*size)--; // Decrease the size of the queue
    }
}

int main() {
    int n;
    int queue[MAX_CUSTOMERS];

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

    // If no customers, handle underflow case
    if (n == 0) {
        printf("Underflow\n");
        printf("Queue is empty\n");
        return 0;
    }

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

    // Serve the first customer
    serveCustomer(queue, &n);

    // Display the remaining customers in the queue
    displayQueue(queue, n);

    return 0;
}
```

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

3. 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

```
#include <stdio.h>

#define MAX_SIZE 50

void displayQueue(int queue[], int size) {
   for (int i = 0; i < size; i++) {
      printf("%d", queue[i]);
      if (i < size - 1) {
         printf(" ");
      }
   }
   printf("\n");
}

void selectiveDequeue(int queue[], int *size, int multiple) {
   int newQueue[MAX_SIZE];
   int newSize = 0;

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

```
            if (queue[i] % multiple != 0) {
                newQueue[newSize++] = queue[i];
            }
        }

        // Update the original queue and its size
        for (int i = 0; i < newSize; i++) {
            queue[i] = newQueue[i];
        }
        *size = newSize;
    }

    int main() {
        int n;
        int queue[MAX_SIZE];

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

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

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

        // Display the original queue
        printf("Original Queue: ");
        displayQueue(queue, n);

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

        // Display the queue after selective dequeue
        printf("Queue after selective dequeue: ");
        displayQueue(queue, 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

```
#include <stdio.h>

#define MAX_SIZE 10

void enqueue(int queue[], int *size, int element) {
   if (*size < MAX_SIZE) {
      queue[*size] = element;
      (*size)++;
      printf("Enqueued: %d\n", element);
   } else {
      printf("Queue is full. Cannot enqueue more elements.\n");
   }
}

int kthLargest(int queue[], int size, int k) {
   // Sort the queue to find the Kth largest element
   for (int i = 0; i < size - 1; i++) {
      for (int j = 0; j < size - i - 1; j++) {
         if (queue[j] < queue[j + 1]) {
            // Swap
            int temp = queue[j];
            queue[j] = queue[j + 1];
            queue[j + 1] = temp;
         }
      }
   }
   return queue[k - 1]; // Kth largest element
}

int main() {
   int n;
   int queue[MAX_SIZE];
   int size = 0;

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

   // Read the elements and enqueue them
```

```
    for (int i = 0; i < n; i++) {
        int element;
        scanf("%d", &element);
        enqueue(queue, &size, element);
    }

    // Read the value of K
    int k;
    scanf("%d", &k);

    // Find and print the Kth largest element
    int kth_largest = kthLargest(queue, size, k);
    printf("The %dth largest element: %d\n", k, kth_largest);

    return 0;
}
```

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

5. 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

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

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

// Define the structure for the queue
struct Queue {
   struct Node* front;
   struct Node* rear;
};

// Function to create a new queue
struct Queue* createQueue() {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = queue->rear = NULL;
   return queue;
}

// Function to enqueue an element
void enqueue(struct Queue* queue, int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;

   if (queue->rear == NULL) {
      // If the queue is empty, both front and rear are the new node
      queue->front = queue->rear = newNode;
      return;
```

```
    }

    // Add the new node at the end of the queue and update the rear
    queue->rear->next = newNode;
    queue->rear = newNode;
}

// Function to dequeue elements and return them as a string
void dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        return; // Queue is empty
    }

    struct Node* temp = queue->front;
    queue->front = queue->front->next;

    // If the front becomes NULL, then change rear also as NULL
    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    // Print the dequeued element
    printf("%d ", temp->data);
    free(temp); // Free the memory of the dequeued node
}

// Function to free the queue
void freeQueue(struct Queue* queue) {
    struct Node* current = queue->front;
    struct Node* next;

    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }

    free(queue);
}

int main() {
    struct Queue* queue = createQueue();
    int value;

    // Read integers until -1 is entered
    while (1) {
        scanf("%d", &value);
        if (value == -1) {
            break;
        }
        enqueue(queue, value);
    }

    // Print the dequeued elements
    printf("Dequeued elements: ");
    while (queue->front != NULL) {
        dequeue(queue);
    }
    printf("\n");

    // Free the queue
    freeQueue(queue);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: ranjani prakash
Email: 240801267@rajalakshmi.edu.in
Roll no: 2116240801267
Phone: 6382555840
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

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

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

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

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

// Function to insert a new node in the BST
Node* insert(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 to perform in-order traversal of the BST
void inorderTraversal(Node* root) {
    if (root == NULL) {
        return;
    }
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

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

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

    Node* root = NULL;

    // Construct the BST from preorder traversal
    for (int i = 0; i < n; i++) {
        root = insert(root, preorder[i]);
    }

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

    // Free the allocated memory (not shown for brevity)
    // You should implement a function to free the BST nodes

    return 0;
}
```

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

2. 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

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

```
// Define the structure for the BST node
struct Node {
   int data;
   struct Node *left, *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 into the BST
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 the 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 [min, max]
struct Node* removeOutsideRange(struct Node* root, int min, int max) {
   if (root == NULL) {
        return NULL;
    }

    // Remove nodes that are outside the range [min, max]
    if (root->data < min) {
        struct Node* temp = root->right;
        free(root);
        return removeOutsideRange(temp, min, max);
    }
    if (root->data > max) {
        struct Node* temp = root->left;
        free(root);
        return removeOutsideRange(temp, min, max);
    }

    // Recursively process the left and right subtrees
    root->left = removeOutsideRange(root->left, min, max);
    root->right = removeOutsideRange(root->right, min, max);

    return root;
}

int main() {
    int N, min, max;

    // Input the number of elements to be inserted into the BST
    scanf("%d", &N);

    // Input the elements to be inserted into the BST
    int elements[N];
    for (int i = 0; i < N; i++) {
        scanf("%d", &elements[i]);
    }

    // Input the min and max range
    scanf("%d %d", &min, &max);

    // Initialize the root of the BST
    struct Node* root = NULL;

    // Insert elements into the BST
    for (int i = 0; i < N; i++) {
        root = insert(root, elements[i]);
    }

    // Remove nodes outside the specified range
    root = removeOutsideRange(root, min, max);

    // Perform an in-order traversal to print the remaining elements
    inorder(root);
    printf("\n");

    return 0;
}
```

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

3. 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

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

// Define the structure for the BST 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 into the BST
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 find the minimum value node in the BST
struct Node* minValueNode(struct Node* root) {
    struct Node* current = root;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}

// Function to delete a node from the BST
struct Node* deleteNode(struct Node* root, int data) {
    if (root == NULL) {
        return root;
    }

    // Recur down the tree
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        // Node to be deleted found

        // Node with only 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: Get the inorder successor (smallest in the right
subtree)
        struct Node* temp = minValueNode(root->right);

        // Copy the inorder successor's content to this node
        root->data = temp->data;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Function to perform level-order traversal of the BST
void levelOrder(struct Node* root) {
    if (root == NULL) {
        return;
    }

    struct Node* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;

    while (front < rear) {
        struct Node* current = queue[front++];
        printf("%d ", current->data);

        if (current->left != NULL) {
            queue[rear++] = current->left;
        }

        if (current->right != NULL) {
            queue[rear++] = current->right;
        }
    }
    printf("\n");
}

int main() {
    int N, X, Y;

    // Input the number of initial elements in the BST
    scanf("%d", &N);

    // Create the BST by inserting the initial keys
    int elements[N];
    struct Node* root = NULL;

    // Input the initial keys
    for (int i = 0; i < N; i++) {
        scanf("%d", &elements[i]);
        root = insert(root, elements[i]);
    }

    // Input the new key to be inserted into the BST
    scanf("%d", &X);

    // Input the key to be deleted from the BST
    scanf("%d", &Y);

    // Output the initial BST in level-order
    printf("Initial BST: ");
    levelOrder(root);

    // Insert the new key and print the BST after insertion
    root = insert(root, X);
    printf("BST after inserting a new node %d: ", X);
    levelOrder(root);

    // Delete the key and print the BST after deletion
    root = deleteNode(root, Y);
    printf("BST after deleting node %d: ", Y);
    levelOrder(root);
    return 0;
}
```

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

4. 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

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

```
// Define the structure for the BST 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 into the BST
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 post-order traversal of the BST
void postOrder(struct Node* root) {
    if (root == NULL) {
        return;
    }

    postOrder(root->left);
    postOrder(root->right);
    printf("%d ", root->data);
}
```

```
int main() {
    int N;

    // Input the number of elements to insert into the BST
    scanf("%d", &N);

    int elements[N];
    struct Node* root = NULL;

    // Input the elements to insert into the BST
    for (int i = 0; i < N; i++) {
        scanf("%d", &elements[i]);
        root = insert(root, elements[i]);
    }

    // Perform a post-order traversal and print the result
    postOrder(root);
    printf("\n");

    return 0;
}
```

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

5. 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

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

// Define the structure for the BST 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 into the BST
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 search for a target value in the BST
int search(struct Node* root, int target) {
    // Base case: root is null or target is found
    if (root == NULL) {
        return 0;  // Target not found
    }
    if (root->data == target) {
        return 1;  // Target found
    }

    // If the target is smaller, search in the left subtree
    if (target < root->data) {
        return search(root->left, target);
    }

    // If the target is larger, search in the right subtree
    return search(root->right, target);
}

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

    // Input the elements for the BST (terminate with -1)
    while (scanf("%d", &element) && element != -1) {
        root = insert(root, element);
    }

    int target;
    // Input the target value to search for
    scanf("%d", &target);

    // Search for the target value in the BST
    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***

Rajalakshmi Engineering College

Name: ranjani prakash
Email: 240801267@rajalakshmi.edu.in
Roll no: 2116240801267
Phone: 6382555840
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 : 47.5

Section 1 : Coding

1. 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>
#include <stdlib.h>

// Function to calculate the digit sum of a number
int digitSum(int num) {
   int sum = 0;
   while (num > 0) {
      sum += num % 10;
      num /= 10;
   }
   return sum;
}

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

   // Create temporary arrays
   int* L = (int*)malloc(n1 * sizeof(int));
   int* R = (int*)malloc(n2 * sizeof(int));

   // Copy data to temporary arrays
```

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

    // Merge the temporary arrays back into arr[left..right]
    i = 0; // Initial index of first sub-array
    j = 0; // Initial index of second sub-array
    k = left; // Initial index of merged sub-array
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    // Free temporary arrays
    free(L);
    free(R);
}

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

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

int main() {
    int N;

    // Read the number of elements
    scanf("%d", &N);
    int* arr = (int*)malloc(N * sizeof(int));

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

    // Perform merge sort
    mergeSort(arr, 0, N - 1);

    // Print the sorted array
    printf("The sorted array is: ");
    for (int i = 0; i < N; i++) {
        printf("%d", arr[i]);
        if (i < N - 1) {
            printf(" ");
        }
    }
    printf("\n");

    // Find the integer with the highest digit sum
    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];
        }
    }

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

    // Free allocated memory
    free(arr);

    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

```
#include <stdio.h>

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

int partition(int arr[], int low, int high, int iteration) {
   int pivot = arr[high]; // Choosing the last element as pivot
   int i = low - 1; // Index of smaller element

   for (int j = low; j < high; j++) {
      if (arr[j] > pivot) { // Change to '>' for descending order
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);

   // Print the current state of the array after partitioning
   printf("Iteration %d: ", iteration);
   for (int k = low; k <= high; k++) {
      printf("%d", arr[k]);
      if (k < high) {
            printf(" ");
        }
    }
    printf("\n");

    return i + 1; // Return the partitioning index
}

void quickSort(int arr[], int low, int high, int* iteration) {
    if (low < high) {
        int pi = partition(arr, low, high, ++(*iteration)); // Partitioning index
        quickSort(arr, low, pi - 1, iteration); // Recursively sort elements before
partition
        quickSort(arr, pi + 1, high, iteration); // Recursively sort elements after
partition
    }
}

int main() {
    int n;

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

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

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

    // Print the sorted order
    printf("Sorted Order: ");
    for (int i = 0; i < n; i++) {
        printf("%d", scores[i]);
        if (i < n - 1) {
            printf(" ");
        }
    }
    printf("\n");
    return 0;
}
```

Status : Partially correct **Marks : 7.5/10**

3. 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;

        // 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--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n;

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

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

    // Sort the finishing times using insertion sort
    insertionSort(times, n);

    // Print the sorted finishing times
    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***

4. 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

```c
#include <stdio.h>

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

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

    return swapCount;
}

int main() {
    int n;

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

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

    // Count the number of swaps during insertion sort
    int totalSwaps = insertionSortCountSwaps(arr, n);

    // Print the total number of swaps
    printf("%d\n", totalSwaps);

    return 0;
}
```

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

5. 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;

    // Create temporary arrays
    int* L = (int*)malloc(n1 * sizeof(int));
    int* R = (int*)malloc(n2 * sizeof(int));

    // Copy data to temporary arrays
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    // Merge the temporary arrays back into arr[left..right]
    i = 0; // Initial index of first sub-array
    j = 0; // Initial index of second sub-array
    k = left; // Initial index of merged sub-array
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    free(L);
    free(R);
}

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

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int N, M;

    // Read the first array
    scanf("%d", &N);
    int* arr1 = (int*)malloc(N * sizeof(int));
    for (int i = 0; i < N; i++) {
        scanf("%d", &arr1[i]);
    }

    // Read the second array
    scanf("%d", &M);
    int* arr2 = (int*)malloc(M * sizeof(int));
    for (int i = 0; i < M; i++) {
        scanf("%d", &arr2[i]);
    }

    // Merge the two arrays
    int totalSize = N + M;
    int* mergedArray = (int*)malloc(totalSize * sizeof(int));
    for (int i = 0; i < N; i++) {
        mergedArray[i] = arr1[i];
    }
    for (int i = 0; i < M; i++) {
        mergedArray[N + i] = arr2[i];
    }

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

    // Remove duplicates and print the result
    int lastPrinted = -1; // To track the last printed number
    for (int i = 0; i < totalSize; i++) {
        if (mergedArray[i] != lastPrinted) {
            printf("%d ", mergedArray[i]);
            lastPrinted = mergedArray[i];
        }
    }
    printf("\n");

    // Free allocated memory
    free(arr1);
    free(arr2);
    free(mergedArray);

    return 0;
}
```

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