Rajalakshmi Engineering College

Name: Pranav Narayanan
Email: 240701393@rajalakshmi.edu.in
Roll no:
Phone: 9500579427
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

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

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>

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

// 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;
}

// Insert node at the front
void insertFront(Node** head_ref, int data) {
   Node* newNode = createNode(data);
   if (*head_ref == NULL) {
      *head_ref = newNode;
      return;
   }
   newNode->next = *head_ref;
   (*head_ref)->prev = newNode;
   *head_ref = newNode;
}

// Insert node at a given position (1-based)
void insertAtPosition(Node** head_ref, int position, int data) {
   Node* newNode = createNode(data);

   if (position == 1) {
      // Insert at front
      newNode->next = *head_ref;
      if (*head_ref != NULL) {
         (*head_ref)->prev = newNode;
      }
      *head_ref = newNode;
      return;
   }

   Node* temp = *head_ref;
   int count = 1;

   // Traverse to the node after which we want to insert
   while (temp != NULL && count < position - 1) {
      temp = temp->next;
      count++;
   }
```

```
    if (temp == NULL) {
        // Position is out of range (shouldn't happen as per constraints)
        free(newNode);
        return;
    }

    newNode->next = temp->next;
    newNode->prev = temp;

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

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

int main() {
    int N, i, data, position;
    Node* head = NULL;

    scanf("%d", &N);

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

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

    // Print original list
    printList(head);
    printf(" ");

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

    // Print updated list
    printList(head);
    printf("\n");

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

    return 0;
}
```

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

2. 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>

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

// 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;
}

// Insert at the front of the list
void insertFront(Node** head_ref, int data) {
   Node* newNode = createNode(data);
```

```
    newNode->next = *head_ref;
    if (*head_ref != NULL) {
        (*head_ref)->prev = newNode;
    }
    *head_ref = newNode;
}

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

// Delete node at position (1-based)
void deleteAtPosition(Node** head_ref, int position) {
    if (*head_ref == NULL) return;

    Node* temp = *head_ref;
    int count = 1;

    // If head needs to be removed
    if (position == 1) {
        *head_ref = temp->next;
        if (*head_ref != NULL) {
            (*head_ref)->prev = NULL;
        }
        free(temp);
        return;
    }

    // Traverse to the node at position
    while (temp != NULL && count < position) {
        temp = temp->next;
        count++;
    }

    if (temp == NULL) return; // Position out of range (not expected per
constraints)
```

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

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

    free(temp);
}

int main() {
    int N, i, data, X;
    Node* head = NULL;

    scanf("%d", &N);

    for (i = 0; i < N; i++) {
        scanf("%d", &data);
        insertFront(&head, data);
    }

    scanf("%d", &X);

    // Print original list
    printList(head);
    printf(" ");

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

    // Print updated list
    printList(head);
    printf("\n");

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

    return 0;
```

```
}
```

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

3. 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>
```

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

// 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;
}

// Append at the end
void append(Node** head_ref, int data) {
    Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }
    Node* temp = *head_ref;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
}

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

// Rotate clockwise by k positions
void rotateClockwise(Node** head_ref, int k) {
    if (*head_ref == NULL || k == 0)
        return;
    // Find length and last node
    Node* tail = *head_ref;
    int length = 1;
    while (tail->next != NULL) {
        tail = tail->next;
        length++;
    }

    // Effective rotations if k >= length (not needed as per constraints, but good
practice)
    k = k % length;
    if (k == 0)
        return;

    // Move (length - k) steps from head to get new tail
    Node* newTail = *head_ref;
    for (int i = 1; i < length - k; i++) {
        newTail = newTail->next;
    }

    Node* newHead = newTail->next;

    // Break the list and rearrange pointers
    newTail->next = NULL;
    newHead->prev = NULL;

    tail->next = *head_ref;
    (*head_ref)->prev = tail;

    *head_ref = newHead;
}

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

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

    rotateClockwise(&head, k);
    printList(head);
    printf("\n");

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

    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

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

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

// 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;
}

// Append node at end
void append(Node** head_ref, int data) {
   Node* newNode = createNode(data);
   if (*head_ref == NULL) {
      *head_ref = newNode;
      return;
   }
   Node* temp = *head_ref;
```

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

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

// Check if palindrome
int isPalindrome(Node* head) {
    if (head == NULL) return 1;

    // Get tail node
    Node* tail = head;
    while (tail->next != NULL)
        tail = tail->next;

    // Compare from start and end
    while (head != tail && tail->next != head) {
        if (head->data != tail->data)
            return 0;
        head = head->next;
        tail = tail->prev;
    }
    return 1;
}

int main() {
    int N, data;
    Node* head = NULL;

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

    printList(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 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

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

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

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

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

// Append node at the end
void append(Node** head_ref, int data) {
   Node* newNode = createNode(data);
   if (*head_ref == NULL) {
       *head_ref = newNode;
       return;
   }
   Node* temp = *head_ref;
   while (temp->next != NULL)
       temp = temp->next;
   temp->next = newNode;
   newNode->prev = temp;
}

// Print the list elements
void printList(Node* head) {
   Node* temp = head;
   while (temp != NULL) {
       printf("%d ", temp->data);
       temp = temp->next;
   }
}

// Print the middle element(s)
void printMiddle(Node* head, int n) {
   Node* temp = head;
   int mid = n / 2;

   // Move to middle node (for odd), or first middle (for even)
   for (int i = 0; i < (n % 2 == 0 ? mid - 1 : mid); i++) {
       temp = temp->next;
   }

   if (n % 2 == 1) {
       // Odd number of elements - print one middle element
       printf("%d ", temp->data);
   } else {
       // Even number of elements - print two middle elements
       printf("%d %d ", temp->data, temp->next->data);
   }
}

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

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

   printList(head);
   printf(" ");

   printMiddle(head, n);
   printf("\n");

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

   return 0;
}
```

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