Rajalakshmi Engineering College

Name: shobbika T
Email: 240701502@rajalakshmi.edu.in
Roll no: 240701502
Phone: 7305423247
Branch: REC
Department: I CSE FE
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

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

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

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

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;

    new_node->data = new_data;
    new_node->next = NULL;
    new_node->prev = NULL;

    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

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

    last->next = new_node;
    new_node->prev = last;
}

// Function to rotate the doubly linked list clockwise by k positions
void rotateListClockwise(struct Node** head_ref, int k) {
    if (*head_ref == NULL || k == 0) {
        return;
    }

    struct Node* last = *head_ref;
    int n = 1; // Start counting from the head

    // Traverse to find the last node and count the total number of nodes
    while (last->next != NULL) {
        last = last->next;
        n++;
    }

    // If k is greater than n, we reduce k to k % n
    k = k % n;

    // If k is 0, no rotation is needed
    if (k == 0) {
        return;
    }

    // Traverse to the node at (n-k)th position
    struct Node* new_tail = *head_ref;
    for (int i = 1; i < (n - k); i++) {
        new_tail = new_tail->next;
    }

    // The new head will be the node after the (n-k)th node
    struct Node* new_head = new_tail->next;

    // Update the pointers to perform the rotation
    new_tail->next = NULL;
    new_head->prev = NULL;
    last->next = *head_ref;
    (*head_ref)->prev = last;

    // Update head pointer
    *head_ref = new_head;
}

// Function to display the doubly linked list
void printList(struct Node* node) {
    if (node == NULL) {
        printf("List is empty\n");
        return;
    }

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

int main() {
    int n, k;

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

    // Create a pointer for the head of the doubly linked list
    struct Node* head = NULL;

    // Read the elements and insert them at the end of the doubly linked list
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertAtEnd(&head, value);
    }

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

    // Rotate the list clockwise by k positions
    rotateListClockwise(&head, k);
```

```
  // Display the list after rotation
  printList(head);

  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

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

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

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last = *head_ref;

   new_node->data = new_data;
   new_node->next = NULL;
   new_node->prev = NULL;

   // If the list is empty, make the new node the head
   if (*head_ref == NULL) {
      *head_ref = new_node;
      return;
   }

   // Traverse to the last node
   while (last->next != NULL) {
      last = last->next;
   }

   // Insert the new node at the end and update the previous pointer
   last->next = new_node;
   new_node->prev = last;
}
```

```
// Function to print the doubly linked list
void printList(struct Node* node) {
   if (node == NULL) {
      printf("List is empty\n");
      return;
   }

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

// Function to find and print the middle element(s)
void printMiddle(struct Node* head, int n) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }

   struct Node* slow = head;
   struct Node* fast = head;

   // Traverse the list with two pointers (slow and fast)
   while (fast != NULL && fast->next != NULL) {
      slow = slow->next;
      fast = fast->next->next;
   }

   // If the number of elements is odd, slow will be the middle element
   if (n % 2 != 0) {
      printf("%d\n", slow->data);
   } else {
      // If the number of elements is even, print two middle elements
      printf("%d %d\n", slow->prev->data, slow->data);
   }
}

int main() {
   int n;
```

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

    // Create a pointer for the head of the doubly linked list
    struct Node* head = NULL;

    // Read the elements and insert them at the end of the doubly linked list
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertAtEnd(&head, value);
    }

    // Print the list
    printList(head);

    // Print the middle element(s)
    printMiddle(head, n);

    return 0;
}
```

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

3. Problem Statement

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

Help Riya implement this system by performing the following operations:

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

Input Format

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

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

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

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

Output Format

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 20 30 40
3
25
Output: 40 30 20 10
40 30 25 20 10

Answer

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

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

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

```
void insertAtFront(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = *head_ref;
   new_node->prev = NULL;

   if (*head_ref != NULL) {
      (*head_ref)->prev = new_node;
   }

   *head_ref = new_node;
}

// Function to insert a node at a specific position
void insertAtPosition(struct Node** head_ref, int position, int new_data) {
   if (position < 1) {
      printf("Invalid position\n");
      return;
   }

   // Create the new node
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   new_node->data = new_data;

   // If inserting at the front (position = 1)
   if (position == 1) {
      insertAtFront(head_ref, new_data);
      return;
   }

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

   // Traverse to the node just before the specified position
   while (current != NULL && count < position - 1) {
      current = current->next;
      count++;
   }

   // If the position is beyond the end of the list, return
   if (current == NULL) {
      printf("Invalid position\n");
```

```
        return;
    }

    // Insert the new node at the given position
    new_node->next = current->next;
    new_node->prev = current;

    // If the new node is not inserted at the last position, update the next node's
prev
    if (current->next != NULL) {
        current->next->prev = new_node;
    }

    current->next = new_node;
}

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

int main() {
    int n;

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

    struct Node* head = NULL;

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

    // Print the original list
    printList(head);
    // Read the position and the new value to be inserted
    int position, data;
    scanf("%d", &position);
    scanf("%d", &data);

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

    // Print the updated list
    printList(head);

    return 0;
}
```

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

4. Problem Statement

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

Input Format

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

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

Output Format

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

The second line displays one of the following:

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

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 2 1

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

Answer

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

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

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
   struct Node* last = *head_ref;

   new_node->data = new_data;
   new_node->next = NULL;

   // If the list is empty, make the new node the head
   if (*head_ref == NULL) {
      new_node->prev = NULL;
      *head_ref = new_node;
      return;
   }

   // Traverse to the last node
   while (last->next != NULL) {
      last = last->next;
   }
```

```
    // Update the pointers
    last->next = new_node;
    new_node->prev = last;
}

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

// Function to check if the doubly linked list is a palindrome
int isPalindrome(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return 1; // Empty or single-element list is a palindrome
    }

    struct Node* left = head;
    struct Node* right = head;

    // Move right to the end of the list
    while (right->next != NULL) {
        right = right->next;
    }

    // Compare elements from left and right towards the center
    while (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;

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

    struct Node* head = NULL;

    // Read the elements and insert them into the doubly linked list
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertAtEnd(&head, value);
    }

    // Print the doubly linked list
    printList(head);

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

    return 0;
}
```

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

5. 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 structure for a doubly linked list node
struct Node {
   int data;
   struct Node* next;
   struct Node* prev;
};

// Function to insert a new node at the front of the doubly linked list
void insertAtFront(struct Node** head_ref, int new_data) {
   // Create a new node
   struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
```

```
    // Assign data to the new node
    new_node->data = new_data;

    // The new node's next is the current head
    new_node->next = *head_ref;

    // The new node's prev is NULL because it is the first node
    new_node->prev = NULL;

    // If the list is not empty, update the previous head's prev pointer
    if (*head_ref != NULL) {
        (*head_ref)->prev = new_node;
    }

    // Update the head to be the new node
    *head_ref = new_node;
}

// Function to delete the node at a given position
void deleteNodeAtPosition(struct Node** head_ref, int position) {
    if (*head_ref == NULL) return; // List is empty

    struct Node* temp = *head_ref;

    // If the position is 1 (delete the head node)
    if (position == 1) {
        *head_ref = temp->next;
        if (*head_ref != NULL) {
            (*head_ref)->prev = NULL;
        }
        free(temp);
        return;
    }

    // Traverse to the node at the given position
    for (int i = 1; temp != NULL && i < position; i++) {
        temp = temp->next;
    }

    // If the position is out of bounds
    if (temp == NULL) return;
    // Update the pointers of the previous and next nodes
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }

    // Free the memory of the node to be deleted
    free(temp);
}

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

int main() {
    int N, X;

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

    struct Node* head = NULL;

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

    // Read the position of the node to delete
    scanf("%d", &X);

    // Print the original list
    printList(head);
    // Delete the node at the given position
    deleteNodeAtPosition(&head, X);

    // Print the updated list
    printList(head);

    return 0;
}
```

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