Rajalakshmi Engineering College

Name: Sudharsan K
Email: 240801341@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

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

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

Input Format

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

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

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

Output Format

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

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

Answer

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

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

// Function to insert a node at the front of the doubly linked list
void insertFront(struct Node** head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = *head;

   if (*head != NULL) {
        (*head)->prev = newNode;
    }

    *head = newNode;
}

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

// Function to delete a node at the given position
void deleteNode(struct Node** head, int position) {
    if (*head == NULL) return;

    struct Node* temp = *head;

    // If head node itself holds the position to be deleted
    if (position == 1) {
        *head = temp->next;  // Move head to the next node
        if (*head != NULL) {
            (*head)->prev = NULL;  // Set the previous of new head to NULL
        }
        free(temp);  // Free the old head
        return;
    }

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

    // If position is greater than the number of nodes
    if (temp == NULL) return;

    // Change the next of the previous node
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    // Change the prev of the next node
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }

    // Free memory
    free(temp);
}

int main() {
    int N, X;

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

    struct Node* head = NULL;

    // Reading the data values and inserting them at the front
    for (int i = 0; i < N; i++) {
        int value;
        scanf("%d", &value);
        insertFront(&head, value);
    }

    // Reading the position of the node to be deleted
    scanf("%d", &X);

    // Printing the original doubly linked list
    printList(head);

    // Deleting the node at position X
    deleteNode(&head, X);

    // Printing the updated doubly linked list after deletion
    printList(head);

    return 0;
}
```

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

2. 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>

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

// Function to insert a node at the end of the doubly linked list
void insertEnd(struct Node** head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (*head == NULL) {
      newNode->prev = NULL;
      *head = newNode;
      return;
   }

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

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

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

// Function to check if the doubly linked list is a palindrome
```

```
int isPalindrome(struct Node* head) {
   if (head == NULL) return 1; // Empty list is trivially a palindrome

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

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

   // Compare left and right pointers until they meet in the middle
   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;

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

   struct Node* head = NULL;

   // Reading the data values and inserting them at the end of the list
   for (int i = 0; i < N; i++) {
      int value;
      scanf("%d", &value);
      insertEnd(&head, value);
   }

   // Printing the original doubly linked list
   printList(head);

   // Checking if the 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***

3. Problem Statement

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

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

Input Format

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

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

Output Format

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
20 52 40 16 18

Output: 20 52 40 16 18
40

Answer

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

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

// Function to insert a node at the end of the doubly linked list
void insertEnd(struct Node** head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = *head;
   newNode->data = data;
   newNode->next = NULL;

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

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

   // Add the new node at the end
   temp->next = newNode;
```

```
    newNode->prev = temp;
}

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

// Function to find and print the middle element(s) of the list
void printMiddle(struct Node* head, int n) {
    struct Node* temp = head;
    int middleIndex = n / 2;  // Middle index for odd/even number of nodes

    if (n % 2 == 1) {
        // If the list has an odd number of elements, print the middle element
        for (int i = 0; i < middleIndex; i++) {
            temp = temp->next;
        }
        printf("%d\n", temp->data);
    } else {
        // If the list has an even number of elements, print the two middle elements
        for (int i = 0; i < middleIndex - 1; i++) {
            temp = temp->next;
        }
        printf("%d %d\n", temp->data, temp->next->data);
    }
}

int main() {
    int n;

    // Reading the number of elements in the doubly linked list
    scanf("%d", &n);

    struct Node* head = NULL;

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

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

    // Print the middle element(s) based on the number of elements
    printMiddle(head, n);

    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>

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

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

// Function to insert a node at the end of the list
void append(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   struct Node* temp = *head;

   if (*head == NULL) {
      *head = newNode;
      return;
   }

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

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

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

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

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

    // Last node before rotation
    struct Node* newHead = temp->next;
    if (newHead == NULL) {
        return;
    }

    newHead->prev = NULL;
    temp->next = NULL;

    // Find the last node of the original list
    struct Node* last = newHead;
    while (last->next != NULL) {
        last = last->next;
    }

    // Attach the original head to the last node
    last->next = *head;
    (*head)->prev = last;

    // Update the head pointer
    *head = newHead;
}
```

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

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

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

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

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

   // Perform rotation
   rotate(&head, n - k);

   // Print rotated list
   printList(head);

   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>

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

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

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

// Function to insert a node at a specific position
void insertAtPosition(struct Node** head, int position, int data) {
    struct Node* newNode = createNode(data);
    struct Node* temp = *head;

    // If inserting at the front
    if (position == 1) {
        newNode->next = *head;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
        return;
    }

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

    if (temp == NULL) {
        return;
    }

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

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

    temp->next = newNode;
}

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

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

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

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

    // Print the original list
    printList(head);

    // Read position and new data
    scanf("%d", &position);
    scanf("%d", &data);

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

    // Print updated list
    printList(head);

    return 0;
}
```

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