Rajalakshmi Engineering College

Name: Shremathi K
Email: 240701504@rajalakshmi.edu.in
Roll no:
Phone: 8870649491
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 <iostream>
using namespace std;

// Node structure
struct Node {
   int data;
   Node* prev;
   Node* next;
};

// Insert at end
void insertAtEnd(Node*& head, Node*& tail, int value) {
   Node* newNode = new Node{value, nullptr, nullptr};

   if (!head) {
      head = tail = newNode;
   } else {
      tail->next = newNode;
      newNode->prev = tail;
      tail = newNode;
   }
}

// Rotate list clockwise by k positions
void rotateClockwise(Node*& head, Node*& tail, int k, int n) {
   if (k == 0 || k >= n || !head) return;

   // Find new head (n - k)th node from beginning
   int splitPoint = n - k;
   Node* newHead = head;
   for (int i = 0; i < splitPoint; ++i) {
      newHead = newHead->next;
   }

   // Rearranging pointers
   tail->next = head;
   head->prev = tail;

   head = newHead;
   tail = newHead->prev;

   head->prev = nullptr;
   tail->next = nullptr;
}

// Display list
void displayList(Node* head) {
   while (head) {
      cout << head->data << " ";
      head = head->next;
   }
   cout << endl;
}

// Main
int main() {
   int n;
   cin >> n;

   Node* head = nullptr;
   Node* tail = nullptr;

   for (int i = 0; i < n; ++i) {
      int val;
      cin >> val;
      insertAtEnd(head, tail, val);
   }
   int k;
   cin >> k;

   rotateClockwise(head, tail, k, n);
   displayList(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 <iostream>
using namespace std;

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

// Function to insert at end
void insertAtEnd(Node*& head, Node*& tail, int value) {
   Node* newNode = new Node{value, nullptr, nullptr};

   if (!head) {
      head = tail = newNode;
   } else {
      tail->next = newNode;
      newNode->prev = tail;
      tail = newNode;
   }
}

// Display the list
void displayList(Node* head) {
   Node* temp = head;
   while (temp) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}

// Function to print the middle element(s)
void printMiddle(Node* head, int n) {
   Node* temp = head;
   int midIndex = n / 2;

   for (int i = 0; i < midIndex; ++i) {
      temp = temp->next;
   }

   if (n % 2 == 1) {
      // Odd number of elements: print the middle one
      cout << temp->data << endl;
   } else {
      // Even number of elements: print two middle ones
      cout << temp->prev->data << " " << temp->data << endl;
   }
}

// Main function
int main() {
   int n;
   cin >> n;

   Node* head = nullptr;
   Node* tail = nullptr;

   for (int i = 0; i < n; ++i) {
      int val;
      cin >> val;
      insertAtEnd(head, tail, val);
   }

   displayList(head);
   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

```
// You are using GCC
#include <iostream>
using namespace std;

// Node structure
struct Node {
    int data;
    Node* prev;
    Node* next;
};

// Insert at front of the list
void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr, head};

    if (head != nullptr)
        head->prev = newNode;

    head = newNode;
}

// Insert at a given position (1-based index)
void insertAtPosition(Node*& head, int position, int value) {
    Node* newNode = new Node{value, nullptr, nullptr};

    if (position == 1) {
        newNode->next = head;
        if (head) head->prev = newNode;
        head = newNode;
        return;
    }

    Node* current = head;
    int count = 1;

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

    if (!current) return;

    newNode->next = current->next;
    newNode->prev = current;

    if (current->next)
        current->next->prev = newNode;

    current->next = newNode;
}

// Display the list
void displayList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

// Main function
int main() {
    int N;
    cin >> N;

    Node* head = nullptr;
    for (int i = 0; i < N; ++i) {
        int value;
        cin >> value;
        insertAtFront(head, value);
    }

    int position, data;
    cin >> position >> data;

    displayList(head); // Original list
    insertAtPosition(head, position, data);
    displayList(head); // Updated list
    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 <iostream>
using namespace std;

// Node structure
struct Node {
   int data;
   Node* prev;
   Node* next;
};

// Function to insert node at the end
void insertAtEnd(Node*& head, int value) {
   Node* newNode = new Node{value, nullptr, nullptr};
   if (!head) {
      head = newNode;
      return;
   }

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

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

// Function to display the list
void displayList(Node* head) {
   Node* temp = head;
   while (temp) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}

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

```
bool isPalindrome(Node* head) {
    // Find the last node
    Node* tail = head;
    while (tail && tail->next) {
        tail = tail->next;
    }

    // Compare nodes from start and end
    Node* front = head;
    Node* back = tail;

    while (front != back && front->next != back) {
        if (front->data != back->data) {
            return false;
        }
        front = front->next;
        back = back->prev;
    }

    return true;
}

int main() {
    int N;
    cin >> N;

    Node* head = nullptr;
    for (int i = 0; i < N; ++i) {
        int value;
        cin >> value;
        insertAtEnd(head, value);
    }

    // Display the list
    displayList(head);

    // Check if the list is a palindrome
    if (isPalindrome(head)) {
        cout << "The doubly linked list is a palindrome" << endl;
    } else {
        cout << "The doubly linked list is not a palindrome" << endl;
    }
    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 <iostream>
using namespace std;

// Node structure
struct Node {
   int data;
   Node* prev;
   Node* next;
};

// Function to insert a node at the front of the doubly linked list
void insertAtFront(Node*& head, int value) {
   Node* newNode = new Node{value, nullptr, head};  // Create new node
   if (head != nullptr) {
      head->prev = newNode;  // Update the previous pointer of the old head
   }
   head = newNode;  // Move head to the new node
}

// Function to display the list
void displayList(Node* head) {
   Node* temp = head;
   while (temp != nullptr) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}

// Function to delete a node at a specific position
void deleteAtPosition(Node*& head, int position) {
   if (head == nullptr) return;  // If the list is empty

   Node* temp = head;
   // If the position is the first node
   if (position == 1) {
      head = head->next;
```

```
        if (head != nullptr) {
            head->prev = nullptr;  // Set previous of new head to null
        }
        delete temp;
        return;
    }

    // Traverse the list to find the position
    int count = 1;
    while (temp != nullptr && count < position) {
        temp = temp->next;
        count++;
    }

    // If the position is invalid
    if (temp == nullptr) return;

    // Update the links to delete the node
    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }

    delete temp;
}

int main() {
    int N, X;
    cin >> N;  // Number of elements in the list

    Node* head = nullptr;

    // Read the elements and insert them at the front
    for (int i = 0; i < N; ++i) {
        int value;
        cin >> value;
        insertAtFront(head, value);
    }

    // Read the position to delete
    cin >> X;

    // Display the original list
    displayList(head);

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

    // Display the updated list
    displayList(head);

    return 0;
}
```

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