Exercise

1. Implement class of a Circular Queue using a Linked List.

SOURCE CODE

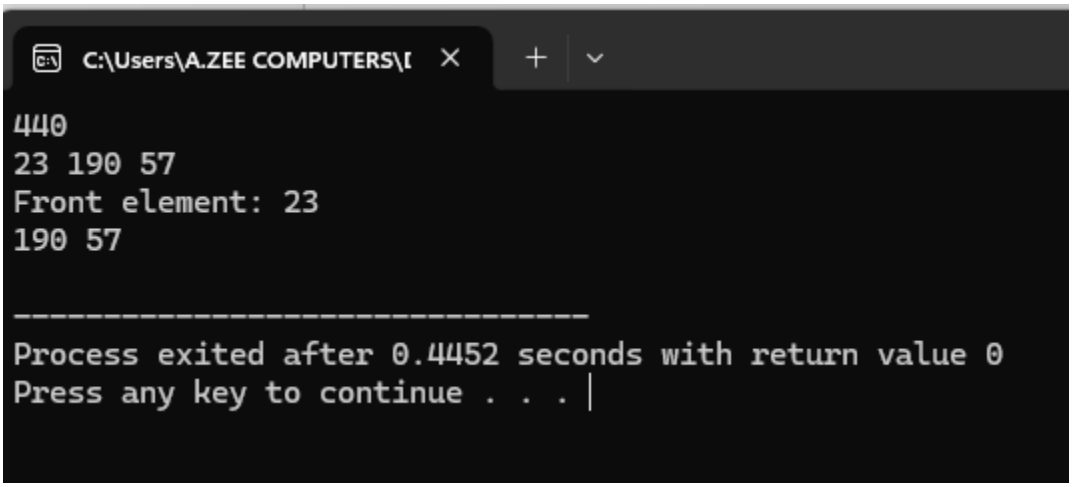
```
#include<iostream>
using namespace std;
class Node{
    public:
        int data;
        Node* next;
        Node(int value){
            data = value;
            next = NULL;
        }
};
class Tamia_004{
private:
   Node* front;
   Node* rear;
public:
    // constructor
   Tamia_004() {
      front = rear = nullptr;
   }
    bool isEmpty() {
      return front == nullptr;
   }
   void enqueue(int value) {
      Node* newNode = new Node(value);
```

```
        newNode->data = value;
        if (isEmpty()) {
            front = rear = newNode;
            newNode->next = front;
        } else {
            rear->next = newNode;
            rear = newNode;
            rear->next = front;
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is underflowing." << endl;
            return;
        }
// agar front or rear ik hi position mn hain to front ko delete kardo
// or dono ko null pe point kardo.
        if (front == rear) {
            delete front;
            front = rear = nullptr;
        } else {
            Node* temp = front;
            front = front->next;
            rear->next = front;
            delete temp;
        }
    }
```

```
    int Front() {
        if (isEmpty()) {
            cout << "Queue is underflowing." << endl;
            return -1;
        }
        return front->data;
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is underflowing." << endl;
            return;
        }

        Node* temp = front;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != front);
        cout << endl;
    }
};

int main() {
    Tamia_004 T;

    T.enqueue(440);
```

```
    T.display();
    T.dequeue();
   T.enqueue(23);
   T.enqueue(190);
   T.enqueue(57);
   T.display();

   cout << "Front element: " << T.Front() << endl;

    T.dequeue();
   T.display();

   return 0;
}
```

OUTPUT



2. Implement class of a Stack using a Linked List.

SOURCE CODE

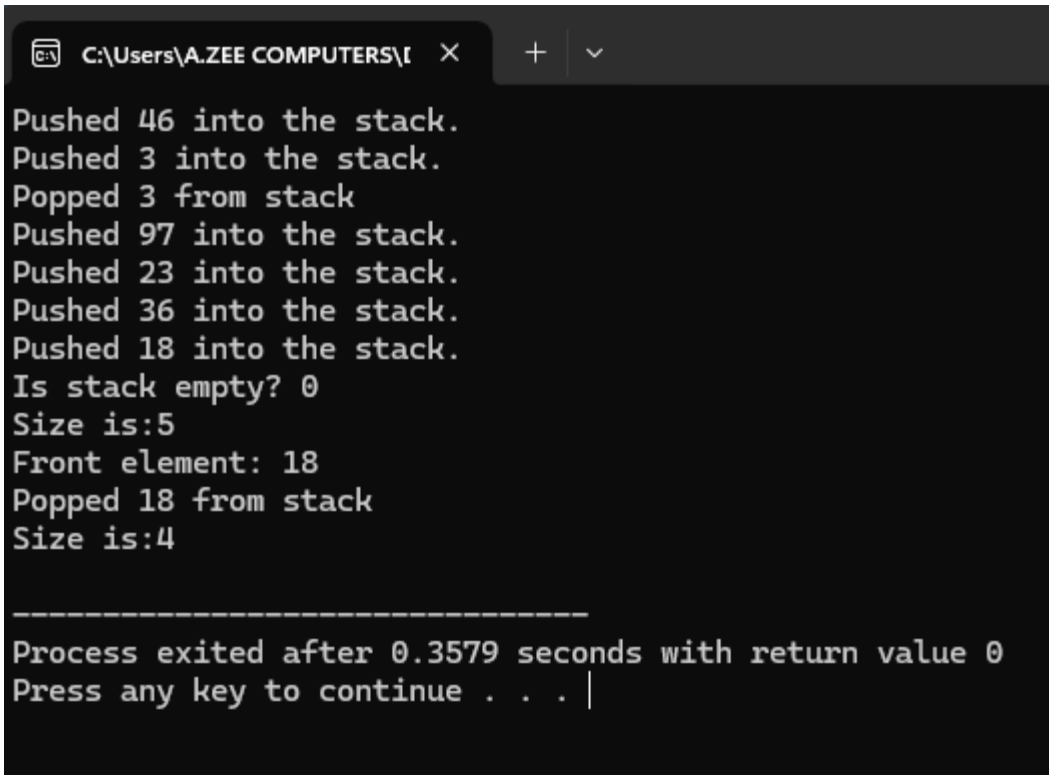
```
 #include<iostream>
using namespace std;
class Node{
```

```
        public:
                int data;
                Node* next;
                Node(int value){
                        data = value;
                        next = NULL;
                }
};
class Tamia_004{
private:
   Node* top;
        int size;
public:
        // constructor
   Tamia_004() {
      top = nullptr;
      size = 0;
   }
        bool isEmpty() {
      return top == nullptr;
   }
   int isSize(){
        return size;
        }
   void Push(int value) {
      Node* temp = new Node(value);
      if (temp == NULL) {
         cout << "Stack is underflowing." << endl;
```

```
            return;
        } else {
            temp->next = top;
            top = temp;
                    size++;
            cout << "Pushed " << value << " into the stack." << endl;
        }
    }

    void Pop() {
        if (isEmpty()) {
            cout << "Stack is underflowing." << endl;
            return;
        }
        else {
            Node* temp = top;
            cout << "Popped "<< top->data << " from stack" << endl;
            top = top->next;
            delete temp;
            size--;
        }
    }

    int Peek() {
        if (isEmpty()) {
            cout << "Stack is underflowing." << endl;
            return -1;
        }
```

```
        return top->data;
    }
};

int main() {
    Tamia_004 T;

    T.Push(46);
    T.Push(3);
            T.Pop();
            T.Push(97);
    T.Push(23);
    T.Push(36);
            T.Push(18);
            cout << "Is stack empty? " << T.isEmpty() << endl;
            cout << "Size is:" << T.isSize() << endl;
    cout << "Front element: " << T.Peek() << endl;
            T.Pop();
            cout << "Size is:" << T.isSize() << endl;

    return 0;
}
```

OUTPUT



3. You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

Example: Input: list1 = [1,2,4], list2 = [1,3,4], Output: [1,1,2,3,4,4]

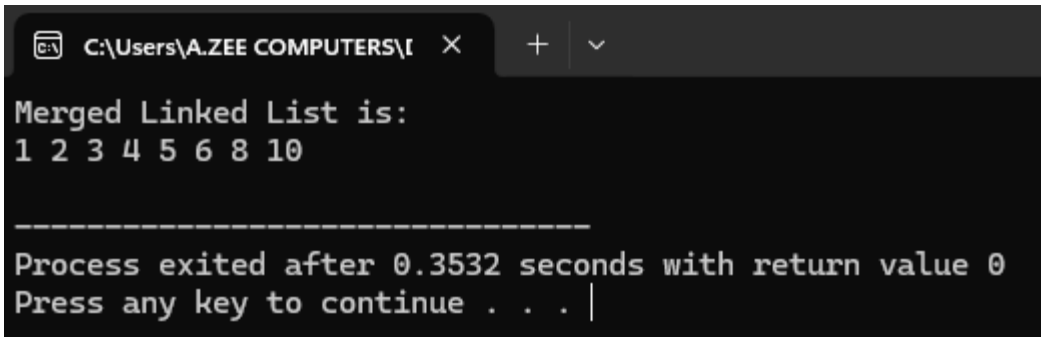
SOURCE CODE

```
#include<iostream>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int x) {
        data = x;
        next = nullptr;
    }
```

```
};

class Tamia_004 {
public:
    Node* mergeSortedList(Node* a, Node* b) {
        Node* result = NULL;

        // Base condition
        if (a == NULL)
            return b;
        else if (b == NULL)
            return a;

        // recursion
        if (a->data <= b->data) {
            result = a;
            result->next = mergeSortedList(a->next, b);
        } else {
            result = b;
            result->next = mergeSortedList(a, b->next);
        }
        return result;
    }

    void display(Node* a, Node* b) {
        Node* res = mergeSortedList(a, b);
        Node* temp = res;
        cout << "Merged Linked List is:" << endl;
```

```
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Node* a = new Node(2);
    a->next = new Node(4);
    a->next->next = new Node(6);
    a->next->next->next = new Node(8);

    Node* b = new Node(1);
    b->next = new Node(3);
    b->next->next = new Node(5);
    b->next->next->next = new Node(10);

    Tamia_004 T;
    T.display(a, b);

    return 0;
}
```

OUTPUT



4. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

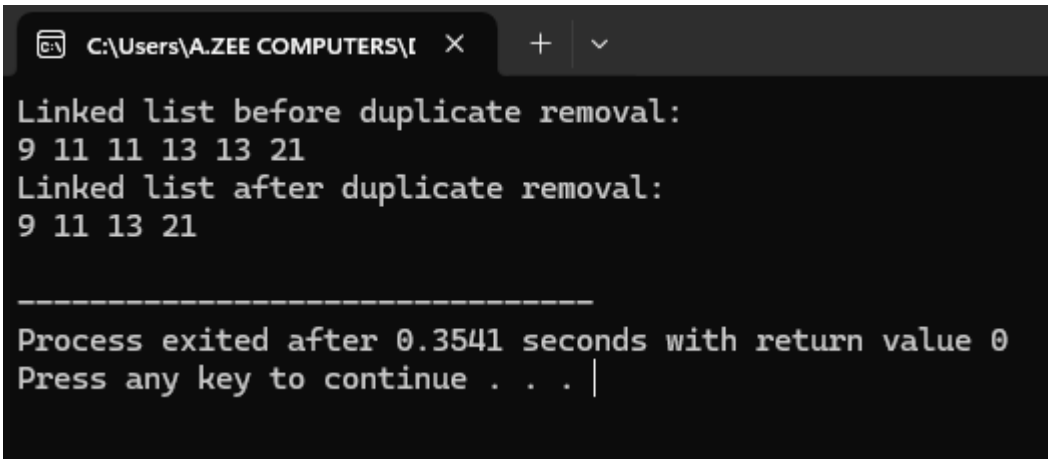
SOURCE CODE

```
#include<iostream>
using namespace std;
class Node {
  public:
    int data;
    Node *next;
    Node(int x) {
        data = x;
        next = nullptr;
    }
};

class Tamia_004{
     public:
            Node *deleteDuplicates(Node *head) {
     Node *current = head;

    while (current != NULL && current->next != NULL) {
```

```
            if (current->data == current->next->data) {
                Node *next_next = current->next->next;
                current->next = next_next;
            }
            else
                current = current->next;
        }
        return head;
    }

    void display(Node *node) {
        while (node != NULL) {
            cout << node->data << " ";
            node = node->next;
        }
        cout << endl;
    }
    };
    int main() {
        Tamia_004 T;

        Node *head = new Node(9);
        head->next = new Node(11);
        head->next->next = new Node(11);
        head->next->next->next = new Node(13);
        head->next->next->next->next = new Node(13);
        head->next->next->next->next->next = new Node(21);
```

```
    cout << "Linked list before duplicate removal:" << endl;
    T.display(head);

    head = T.deleteDuplicates(head);

    cout << "Linked list after duplicate removal:" << endl;
    T.display(head);

    return 0;
}
```

OUTPUT



5. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

SOURCE CODE

```
#include <iostream>
#include <stack>
using namespace std;
class Node {
public:
    int data;
```

```
    Node* next;
    Node(int d) {
        data = d;
        next = nullptr;
    }
};
class Tamia_004{
    public:
            bool Palindrome(Node* head) {
    Node* currentNode = head;

    stack<int> s;
    while (currentNode != nullptr) {
        s.push(currentNode->data);
        currentNode = currentNode->next;
    }
    while (head != nullptr) {
        int c = s.top();
        s.pop();
        if (head->data != c) {
            return false;
        }
        head = head->next;
    }
    return true;
    }
};
int main() {
```

```
    Tamia_004 T;
    Node* head = new Node(11);
    head->next = new Node(22);
    head->next->next = new Node(55);
    head->next->next->next = new Node(22);
    head->next->next->next->next = new Node(11);

    bool result = T.Palindrome(head);
    if (result)
        cout << "True\n";
    else
        cout << "False\n";
    return 0;
}
```

OUTPUT