

## Exercise

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

### SOURCE CODE

```
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* next;
};

class Tamia_004 {
    Node* rear;
public:
    Tamia_004(){
        rear = nullptr;
    }

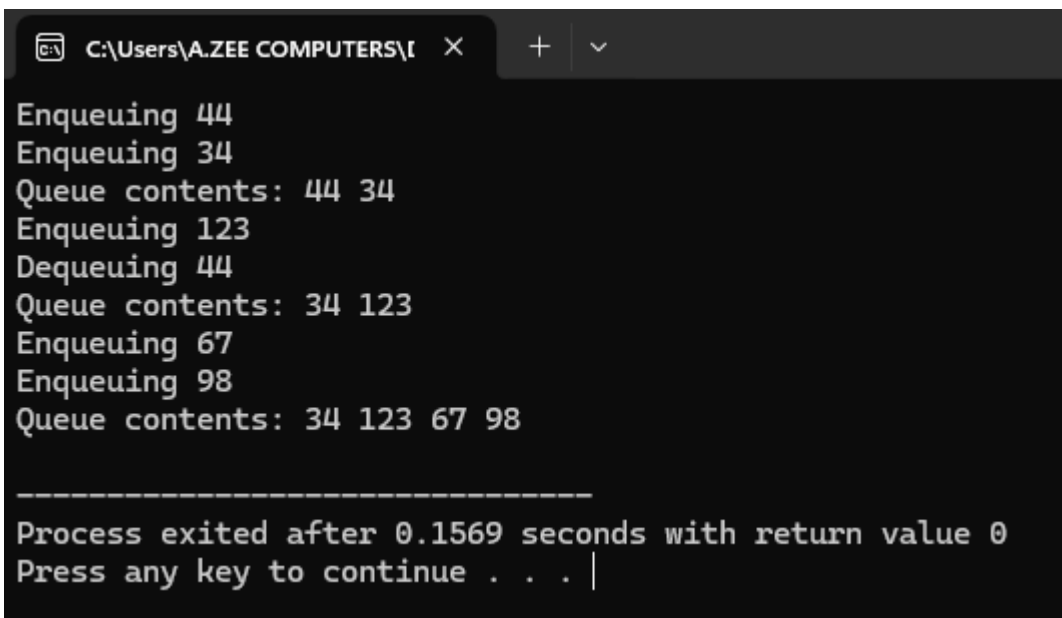
    void enqueue(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        if (rear == nullptr) {
            rear = newNode;
            rear->next = rear;
        } else {
            newNode->next = rear->next;
            rear->next = newNode;
            rear = newNode;
        }
        cout << "Enqueueing " << value << endl;
    }
}
```

```
void dequeue() {  
    if (rear == nullptr) {  
        cout << "Queue is empty. Cannot dequeue." << endl;  
        return;  
    }  
    Node* temp = rear->next;  
    int dequeuedValue = temp->data;  
    if (rear == rear->next) {  
        rear = nullptr;  
    } else {  
        rear->next = temp->next;  
    }  
    delete temp;  
    cout << "Dequeuing " << dequeuedValue << endl;  
}
```

```
void display() {  
    if (rear == nullptr) {  
        cout << "Queue is empty." << endl;  
        return;  
    }  
    Node* temp = rear->next;  
    cout << "Queue contents: ";  
    do {  
        cout << temp->data << " ";  
        temp = temp->next;  
    } while (temp != rear->next);  
    cout << endl;
```

```
    }  
};  
  
int main() {  
    Tamia_004 T;  
    T.enqueue(44);  
    T.enqueue(34);  
    T.display();  
    T.enqueue(123);  
    T.dequeue();  
    T.display();  
    T.enqueue(67);  
    T.enqueue(98);  
    T.display();  
    return 0;  
}
```

## OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I X + v  
Enqueuing 44  
Enqueuing 34  
Queue contents: 44 34  
Enqueuing 123  
Dequeuing 44  
Queue contents: 34 123  
Enqueuing 67  
Enqueuing 98  
Queue contents: 34 123 67 98  
  
-----  
Process exited after 0.1569 seconds with return value 0  
Press any key to continue . . . |
```

2. Implement class of a double ended queue using doubly Linked List.

## SOURCE CODE

```
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class Tamia_004 {
    Node* front;
    Node* rear;

public:
    Tamia_004(){
        front = rear = nullptr;
    }

    void insertFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->prev = nullptr;
        newNode->next = front;
        if (front != nullptr)
            front->prev = newNode;
        else
            rear = newNode;
        front = newNode;
        cout << "Inserting " << value << " at the front." << endl;
    }
}
```

```
void insertRear(int value) {  
    Node* newNode = new Node();  
    newNode->data = value;  
    newNode->next = nullptr;  
    newNode->prev = rear;  
    if (rear != nullptr)  
        rear->next = newNode;  
    else  
        front = newNode;  
    rear = newNode;  
    cout << "Inserting " << value << " at the rear." << endl;  
}
```

```
void deleteFront() {  
    if (front == nullptr) {  
        cout << "Deque is empty. Cannot delete from front." << endl;  
        return;  
    }  
    Node* temp = front;  
    cout << "Deleting " << temp->data << " from front." << endl;  
    front = front->next;  
    if (front != nullptr)  
        front->prev = nullptr;  
    else  
        rear = nullptr;  
    delete temp;  
}
```

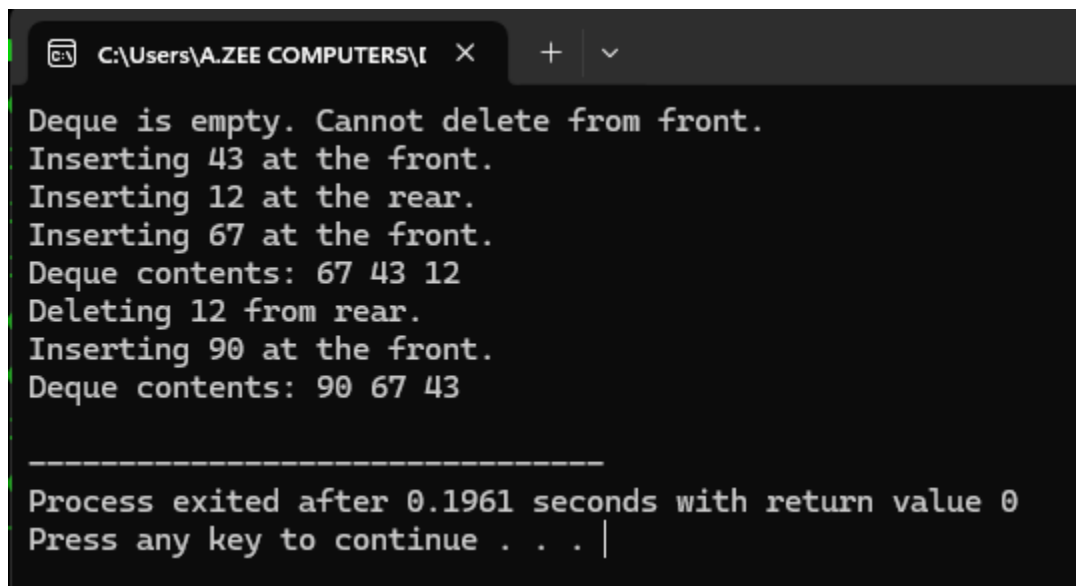
```
void deleteRear() {
    if (rear == nullptr) {
        cout << "Deque is empty. Cannot delete from rear." << endl;
        return;
    }
    Node* temp = rear;
    cout << "Deleting " << temp->data << " from rear." << endl;
    rear = rear->prev;
    if (rear != nullptr)
        rear->next = nullptr;
    else
        front = nullptr;
    delete temp;
}

void display() {
    if (front == nullptr) {
        cout << "Deque is empty." << endl;
        return;
    }
    Node* temp = front;
    cout << "Deque contents: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

};
```

```
int main() {  
    Tamia_004 T;  
    T.deleteFront();  
    T.insertFront(43);  
    T.insertRear(12);  
    T.insertFront(67);  
    T.display();  
    T.deleteRear();  
    T.insertFront(90);  
    T.display();  
    return 0;  
}
```

## OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I X + v  
Deque is empty. Cannot delete from front.  
Inserting 43 at the front.  
Inserting 12 at the rear.  
Inserting 67 at the front.  
Deque contents: 67 43 12  
Deleting 12 from rear.  
Inserting 90 at the front.  
Deque contents: 90 67 43  
  
-----  
Process exited after 0.1961 seconds with return value 0  
Press any key to continue . . . |
```

3. Create two doubly link lists, say L and M . List L should be containing all even elements from 2 to 10 and list M should contain all odd elements from 1 to 9. Create a new list N by concatenating list L and M.

## SOURCE CODE

```
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class Tamia_004 {
    Node* front;
    Node* rear;

public:
    Tamia_004(){
        front = rear = nullptr;
    }

    void insertRear(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
        newNode->prev = rear;
        if (rear != nullptr)
            rear->next = newNode;
        else
            front = newNode;
        rear = newNode;
    }

    static void concatenate(Tamia_004 &L, Tamia_004 &M) {
        if (L.front == nullptr) {
            L.front = M.front;
            L.rear = M.rear;
            return;
        }
    }
}
```



```
    }
    if (M.front == nullptr)
        return;
    // Link the end of list L to the start of list M
    L.rear->next = M.front;
    M.front->prev = L.rear;
    // Update the rear of list L to be the rear of list M
    L.rear = M.rear;
}

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

};

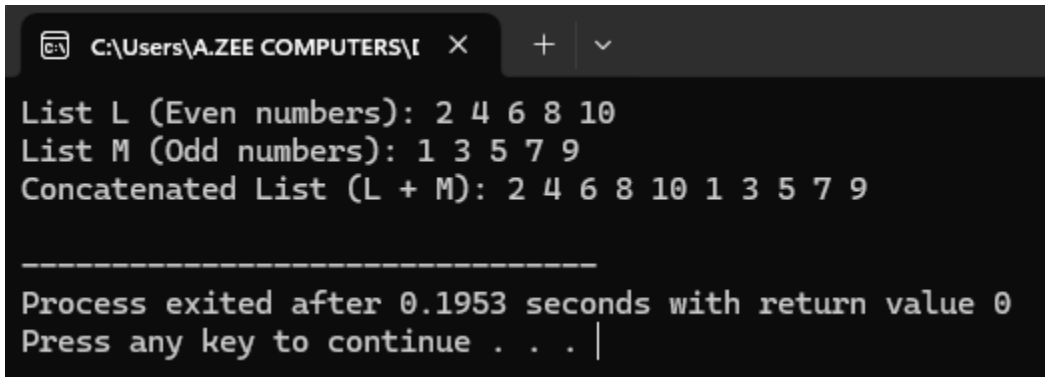
int main() {
    Tamia_004 L, M;
    for (int i = 2; i <= 10; i += 2) L.insertRear(i);
    for (int i = 1; i <= 9; i += 2) M.insertRear(i);

    cout << "List L (Even numbers): ";
    L.display();

    cout << "List M (Odd numbers): ";
    M.display();
    Tamia_004::concatenate(L, M);
    cout << "Concatenated List (L + M): ";
```

```
L.display();  
  
return 0;  
}
```

## OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I X + v  
List L (Even numbers): 2 4 6 8 10  
List M (Odd numbers): 1 3 5 7 9  
Concatenated List (L + M): 2 4 6 8 10 1 3 5 7 9  
  
-----  
Process exited after 0.1953 seconds with return value 0  
Press any key to continue . . . |
```

4. Sort the contents of list N created in Q4 in descending order.

## SOURCE CODE

```
#include <iostream>  
using namespace std;  
class Node {  
public:  
    int data;  
    Node* prev;  
    Node* next;  
};  
class Tamia_004 {  
    Node* front;  
    Node* rear;  
  
public:  
    Tamia_004() {  
        front = rear = nullptr;  
    }  
}
```

```
void insertRear(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    newNode->prev = rear;
    if (rear != nullptr)
        rear->next = newNode;
    else
        front = newNode;
    rear = newNode;
}

static void concatenate(Tamia_004 &L, Tamia_004 &M) {
    if (L.front == nullptr) {
        L.front = M.front;
        L.rear = M.rear;
        return;
    }
    if (M.front == nullptr)
        return;

    // Link the end of list L to the start of list M
    L.rear->next = M.front;
    M.front->prev = L.rear;

    // Update the rear of list L to be the rear of list M
    L.rear = M.rear;
}

void sortDescending() {
    for (Node* i = front; i != nullptr; i = i->next) {
        for (Node* j = i->next; j != nullptr; j = j->next) {
```

```
        if (i->data < j->data) {
            swap(i->data, j->data);
        }
    }
}

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

};

int main() {
    Tamia_004 L, M;
    for (int i = 2; i <= 10; i += 2) L.insertRear(i);
    for (int i = 1; i <= 9; i += 2) M.insertRear(i);
    cout << "List L (Even numbers): ";
    L.display();

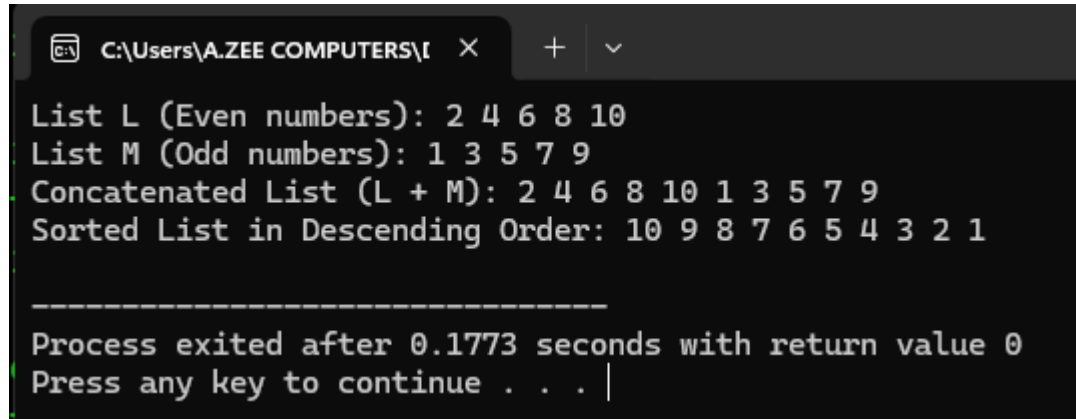
    cout << "List M (Odd numbers): ";
    M.display();

    Tamia_004::concatenate(L, M);
    cout << "Concatenated List (L + M): ";
    L.display();

    L.sortDescending();
    cout << "Sorted List in Descending Order: ";
    L.display();
}
```

```
    return 0;  
}
```

## OUTPUT



```
C:\Users\A.ZEE COMPUTERS\I >
List L (Even numbers): 2 4 6 8 10
List M (Odd numbers): 1 3 5 7 9
Concatenated List (L + M): 2 4 6 8 10 1 3 5 7 9
Sorted List in Descending Order: 10 9 8 7 6 5 4 3 2 1

-----
Process exited after 0.1773 seconds with return value 0
Press any key to continue . . . |
```

5. You have a browser of one tab where you start on the homepage and you can visit another url, get back in the history number of steps or move forward in the history number of steps.

- Implement the BrowserHistory class: BrowserHistory(string homepage) Initializes the object with the homepage of the browser.

- void visit(string url) Visits url from the current page. It clears up all the forward history.

- string back(int steps) Move steps back in history. If you can only return x steps in the history and steps > x, you will return only x steps. Return the current url after moving back in history at most steps.

- string forward(int steps) Move steps forward in history. If you can only forward x steps in the history and steps > x, you will forward only x steps. Return the current url after forwarding in history at most steps.

## SOURCE CODE

```
#include <iostream>
#include <string>
using namespace std;
```

```
class Node {
public:
    string url;
    Node* prev;
    Node* next;

    Node(string URL = "") {
        url = URL;
        prev = next = nullptr;
    }
};

class Tamia_004 {
    Node* current;

public:
    Tamia_004(string homepage) {
        current = new Node(homepage);
    }

    void visit(string url) {
        Node* newNode = new Node(url);
        current->next = newNode;
        newNode->prev = current;
        current = newNode;
        cout << "Visited: " << url << endl;
    }

    string back(int steps) {
        while (steps-- > 0 && current->prev != nullptr) {
            current = current->prev;
        }
    }
};
```

```
        cout << "After going back, current page is: " << current->url << endl;
        return current->url;
    }

    string forward(int steps) {
        while (steps-- > 0 && current->next != nullptr) {
            current = current->next;
        }
        cout << "After going forward, current page is: " << current->url << endl;
        return current->url;
    }
};

int main() {
    Tamia_004 browserHistory("Haroon.com");

    browserHistory.visit("AliBaBa.com");
    browserHistory.visit("Temu.com");
    browserHistory.visit("Daraz.com");

    browserHistory.back(1);
    browserHistory.back(1);
    browserHistory.forward(1);

    browserHistory.visit("Yango.com");

    browserHistory.back(2);
    browserHistory.forward(2);
    return 0;
}
```

## OUTPUT

```
C:\Users\A.ZEE COMPUTERS\I  X  +  v
Visited: AliBaBa.com
Visited: Temu.com
Visited: Daraz.com
After going back, current page is: Temu.com
After going back, current page is: AliBaBa.com
After going forward, current page is: Temu.com
Visited: Yango.com
After going back, current page is: AliBaBa.com
After going forward, current page is: Yango.com

-----
Process exited after 0.2399 seconds with return value 0
Press any key to continue . . .
```