Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Lab Report
On

"Stack and Queue Implementation Using Linked List and Arrays"

[Code No.: COMP 202]

Submitted by

Hridayanshu Raj Acharya

UNG CE II-I

Roll no:1

Submitted to

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 2024/05/22

Lab Report 2

Code

Github Link: https://github.com/hridayanshu236/lab1

Linear Data Structure

A linear data structure is a type of data structure in which elements are arranged sequentially or linearly, where each element is connected to its previous and next element in a single level. The primary feature of linear data structures is that they traverse the data elements sequentially, meaning you can process each data element one after another.

Stack

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. This means the last element added to the stack will be the first one to be removed.

Operations

1. **Push**: Add an element to the top of the stack.
2. **Pop**: Remove the element from the top of the stack.
3. **Peek/Top**: Retrieve the element at the top of the stack without removing it.
4. **IsEmpty**: Check if the stack is empty.
5. **IsFull**: Check if the stack is full.

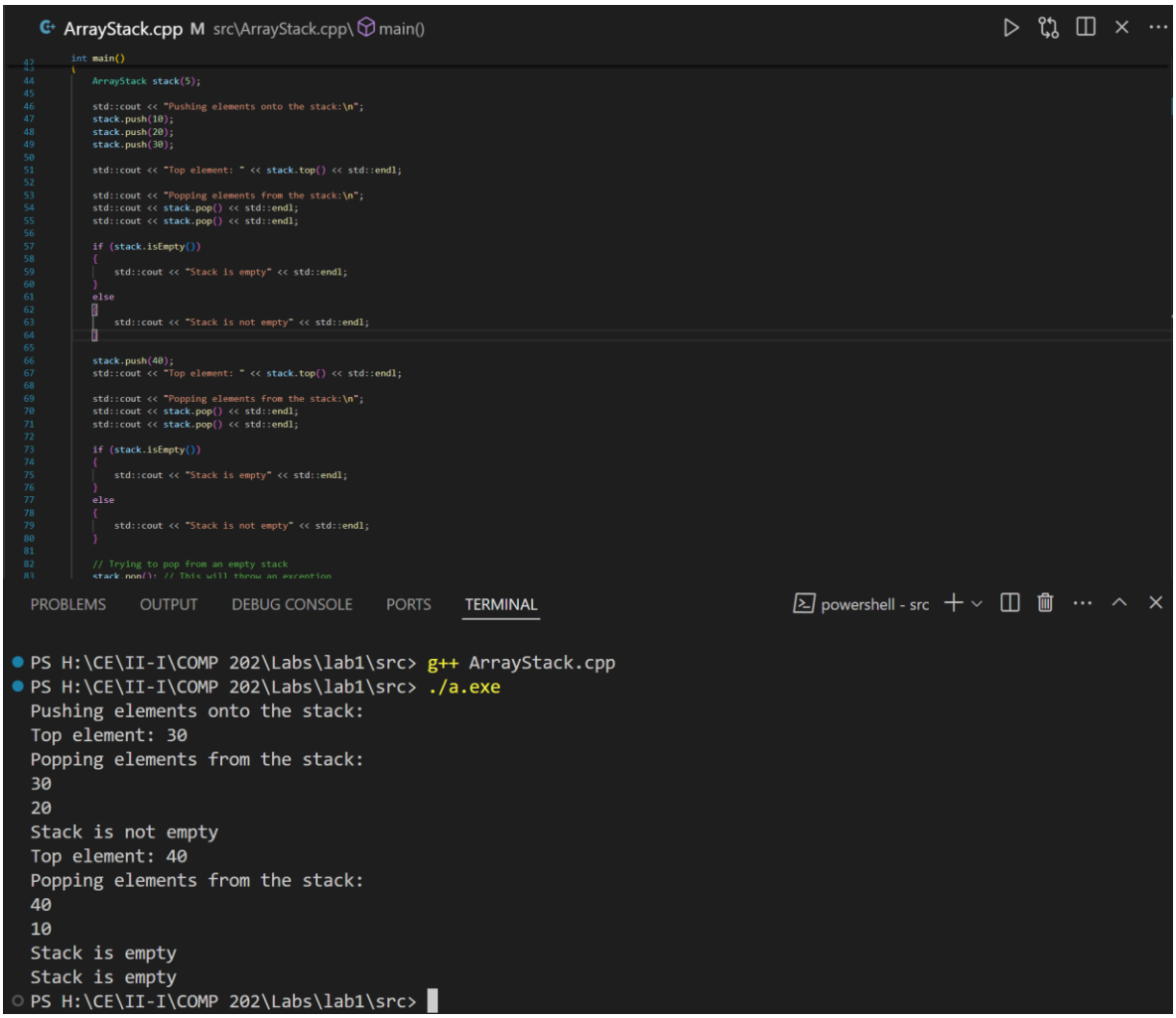
Queue

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. This means the first element added to the queue will be the first one to be removed.

Operations

1. **Enqueue**: Add an element to the end of the queue.
2. **Dequeue**: Remove the element from the front of the queue.
3. **Front/Peek**: Retrieve the element at the front of the queue without removing it.
4. **Rear**: Retrieve the element at the back rear of the queue without removing it.
5. **IsEmpty**: Check if the queue is empty.
6. **IsFull**: Check if the queue is full.

Stack Implementation Using Array:



Operations:

a) **Push**
It is used to add element into the array representing stack after making sure the stack is not full.

b) **Pop**
It is used to remove the element from the top of the stack after making sure the stack is not empty.

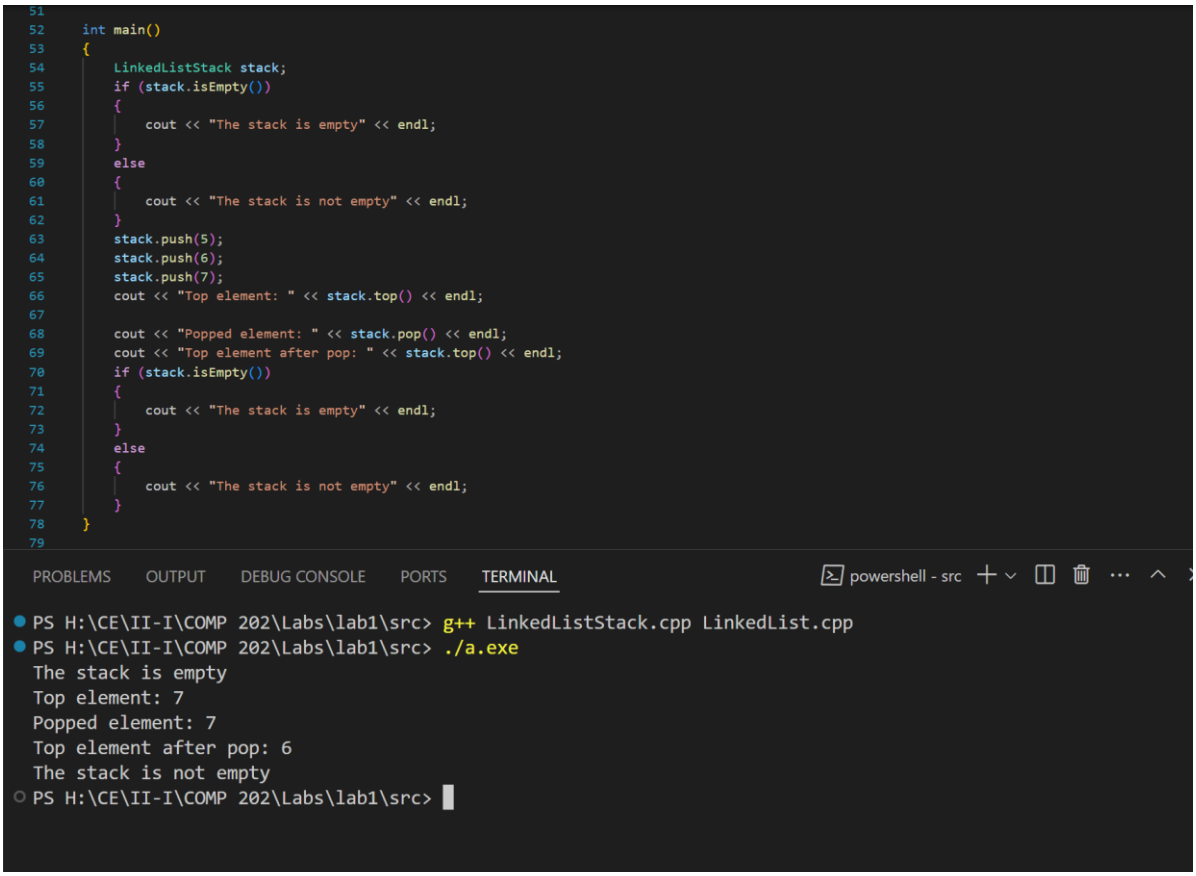
c) **Peek/Top**
It is used to retrieve the element at the top of the stack without removing it.

d) **IsEmpty**
It is used to check if the stack is empty.

e) **IsFull**
It is used to check if the stack is full.

Stack Implementation Using Linked List:

We can use Linked List to implement stack. Linked lists are superior to array while implementing stack because, using this we can dynamically allocate the stack size and is more efficient.



```
int main()
{
    LinkedListStack stack;
    if (stack.isEmpty())
    {
        cout << "The stack is empty" << endl;
    }
    else
    {
        cout << "The stack is not empty" << endl;
    }
    stack.push(5);
    stack.push(6);
    stack.push(7);
    cout << "Top element: " << stack.top() << endl;

    cout << "Popped element: " << stack.pop() << endl;
    cout << "Top element after pop: " << stack.top() << endl;
    if (stack.isEmpty())
    {
        cout << "The stack is empty" << endl;
    }
    else
    {
        cout << "The stack is not empty" << endl;
    }
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL        powershell - src

PS H:\CE\II-I\COMP 202\Labs\lab1\src> g++ LinkedListStack.cpp LinkedList.cpp
PS H:\CE\II-I\COMP 202\Labs\lab1\src> ./a.exe
The stack is empty
Top element: 7
Popped element: 7
Top element after pop: 6
The stack is not empty
PS H:\CE\II-I\COMP 202\Labs\lab1\src>
```

Operations:

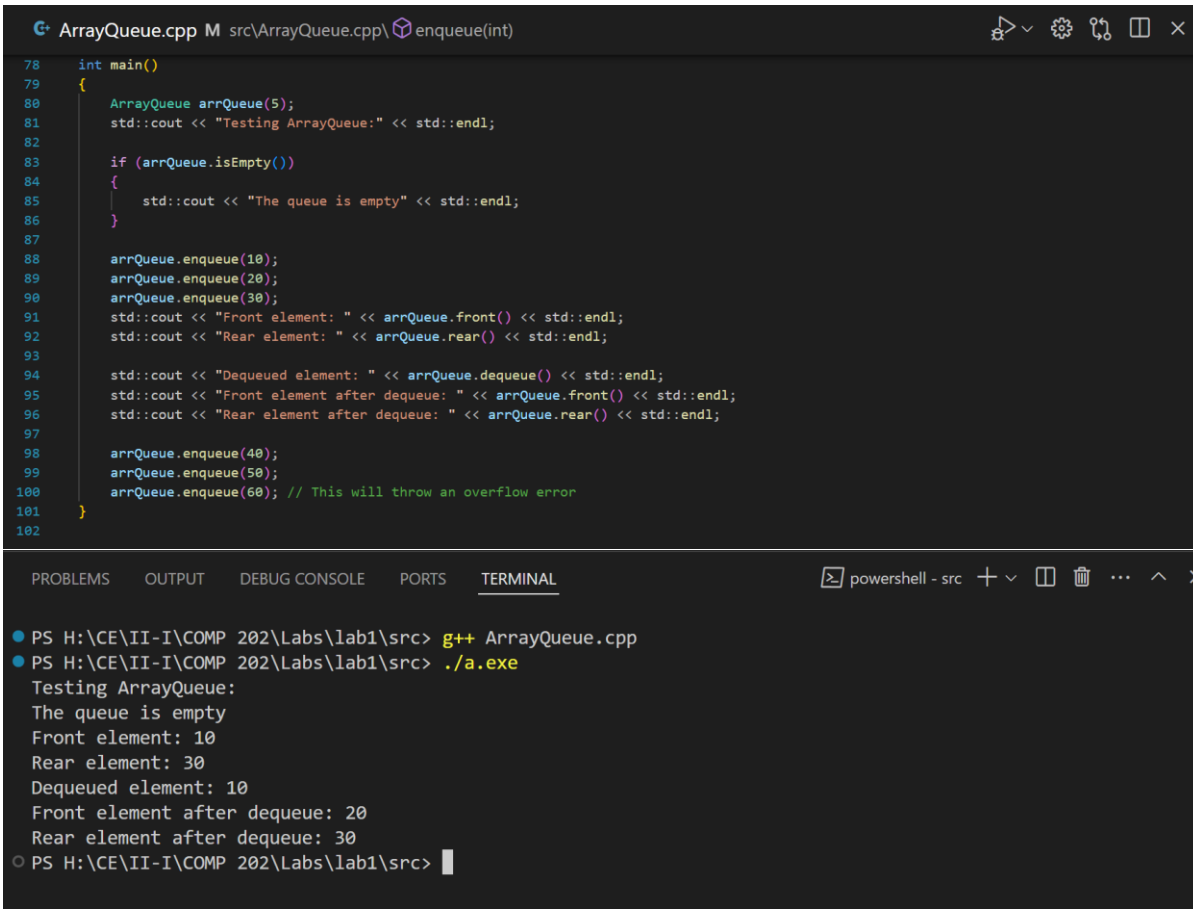
a) **Push**
Add an element to the top of the stack using the addToHead(data) function used in Linked List.

b) **Pop**
Remove the element from the top of the stack using the removeFromHead() function used in Linked List.

c) **Peek/Top**
Retrieve the element at the top of the stack which is at the head of Linked List using returnHead() function used in Linked List

d) **IsEmpty**
Check if the stack is empty using isEmpty() function which is used in Linked List.

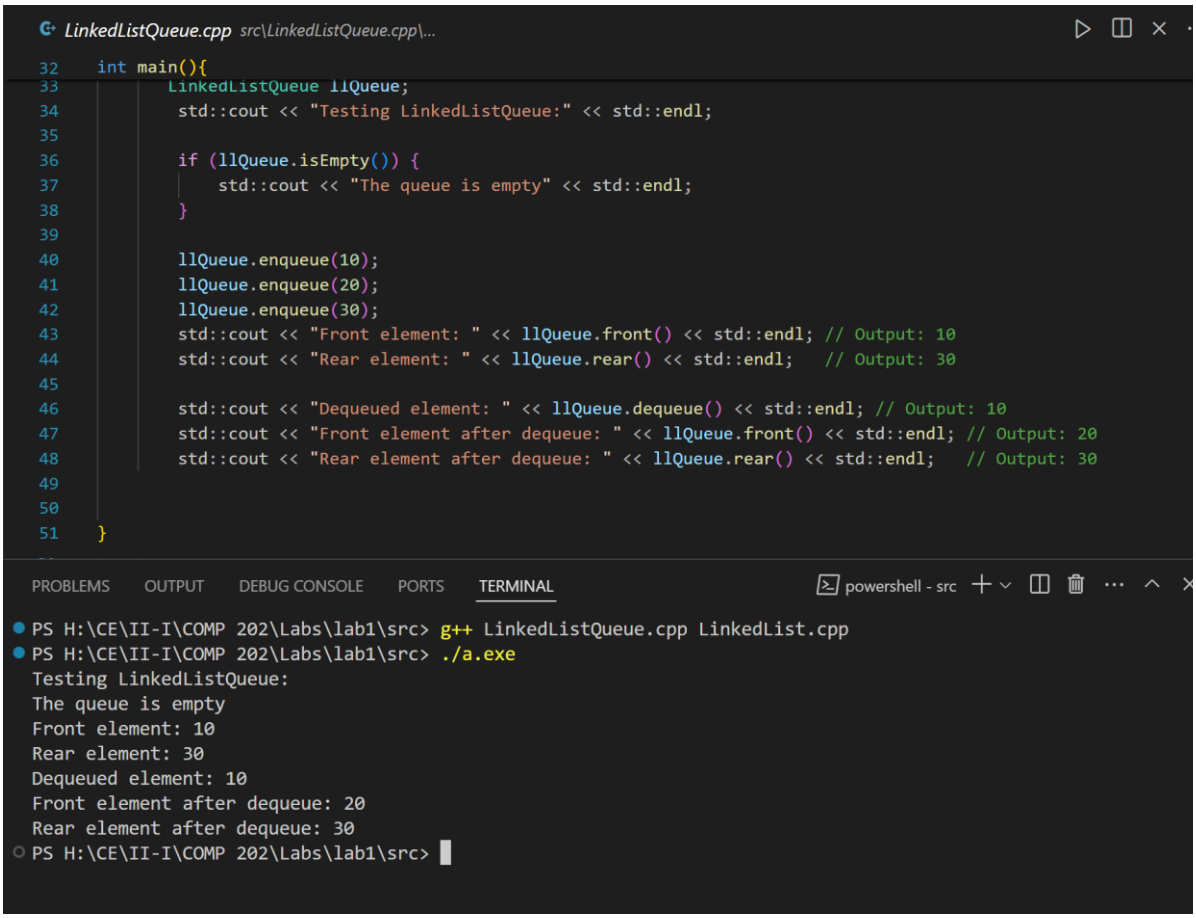
Queue Implementation Using Array



Operations:

a) **Enqueue**
It is used to add an element to the end of the queue after checking whether the queue is full or not.
b) **Dequeue**
It is used to remove the element from the front of the queue after checking if the queue is empty.
c) **Front/Peek**
It is used to retrieve the element at the front of the queue without removing it.
d) **Rear**
It is used to retrieve the element at the back rear of the queue without removing it.
e) **IsEmpty**
f) It checks if the queue/array is empty.
g) **IsFull**
It checks if the queue/array is full.

Queue Implementation Using Linked List



```cpp
int main(){
        LinkedListQueue llQueue;
        std::cout << "Testing LinkedListQueue:" << std::endl;

        if (llQueue.isEmpty()) {
            std::cout << "The queue is empty" << std::endl;
        }

        llQueue.enqueue(10);
        llQueue.enqueue(20);
        llQueue.enqueue(30);
        std::cout << "Front element: " << llQueue.front() << std::endl; // Output: 10
        std::cout << "Rear element: " << llQueue.rear() << std::endl;   // Output: 30

        std::cout << "Dequeued element: " << llQueue.dequeue() << std::endl; // Output: 10
        std::cout << "Front element after dequeue: " << llQueue.front() << std::endl; // Output: 20
        std::cout << "Rear element after dequeue: " << llQueue.rear() << std::endl;   // Output: 30


}
```

```
PS H:\CE\II-I\COMP 202\Labs\lab1\src> g++ LinkedListQueue.cpp LinkedList.cpp
PS H:\CE\II-I\COMP 202\Labs\lab1\src> ./a.exe
Testing LinkedListQueue:
The queue is empty
Front element: 10
Rear element: 30
Dequeued element: 10
Front element after dequeue: 20
Rear element after dequeue: 30
PS H:\CE\II-I\COMP 202\Labs\lab1\src>
```

Operations:

a) **Enqueue**
It is used to add an element to the end of the queue after checking whether the queue is full or not using addToTail(data) function from Linked List.

b) **Dequeue**
It is used to remove the element from the front of the queue after checking if the queue is empty using removeFromHead() function from Linked List.

c) **Front/Peek**
It is used to retrieve the element at the front of the queue without removing it using returnHead() function from Linked List.

d) **Rear**
It is used to retrieve the element at the back rear of the queue without removing it using returnTail() function from Linked List.

e) **IsEmpty**

f) It checks if the queue/Linked List is empty using isEmpty() function from Linked List to check whether the linked list is empty.

Conclusion:

In this way, Stack and Queue can be implemented using both Arrays and Linked List.