

KATHMANDU UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DHULIKHEL, KAVRE



A

LAB REPORT

ON

“STACK AND QUEUE”

Subject Code No: COMP 202

SUBMITTED BY:

Name: Sanjib Dahal

Roll No: 18

Group: CE

Level: II/I

SUBMITTED TO:

Dr. Rajani Chulyadyo

Department of Computer

Science and Engineering

Date of Submission: - 2024/05/22

Lab Report 2

In this lab, we implemented two linear data structures stack and queue using both array and linkedlist. This report aims to demonstrate the functionality and applications of stack and queue using array and linked list through code demonstrations and outputs.

Array

An array is a data structure used to store a collection of items in a contiguous memory space. Each item in the array is identified by its index representing its position in the array.

Linked List

Linked lists are linear data structures consisting of a sequence of elements called nodes, each node holding some information and a pointer to another node in the list.

Stack

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. You can push (add) an item onto the stack, and you can pop (remove) the top item from the stack.

Operations on Stack

The major operations performed in stack are as follows:

1. **push(element)**: Adds an element into the stack
2. **pop()**: Removes an element from the stack
3. **isEmpty()**: Checks if the stack is empty
4. **isFull()**: Checks if the stack is full
5. **top()**: Gives the element at the top

Queue

A queue is another linear data structure that follows the First In, First Out (FIFO) principle. In a queue, elements are added at the rear (enqueue) and removed from the front (dequeue). This means the first item added to the queue is the first one to be removed.

Operations on Queue

The major operations performed in stack are as follows:

1. **enqueue(element)**: Adds an element into the queue
2. **dequeue()**: Removes an element from the queue
3. **isEmpty()**: Checks if the queue is empty
4. **isFull()**: Checks if the queue is full
5. **front()**: Gives the element at the front
6. **rear()**: Gives the element at the rear

GitHub Link to Code

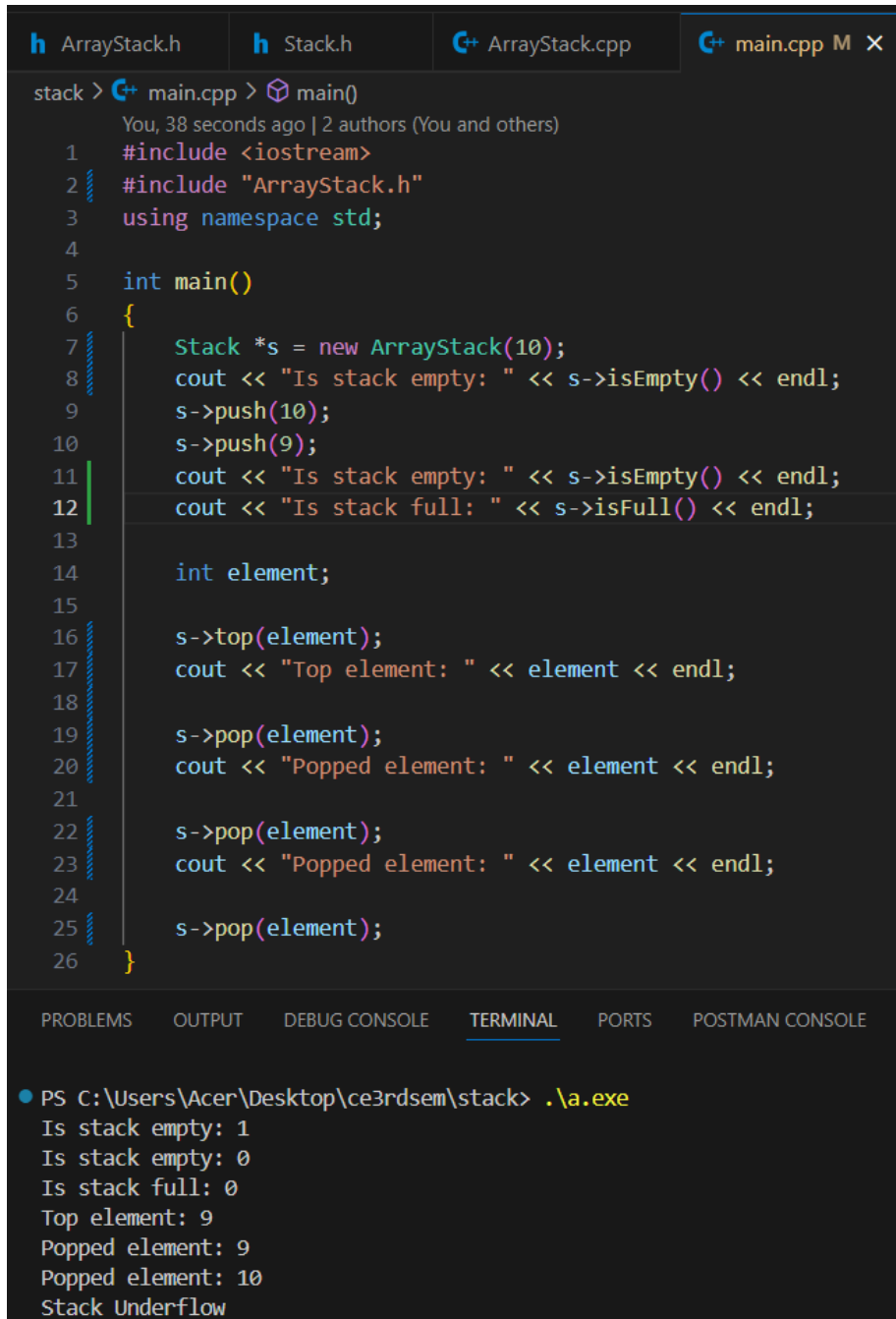
The GitHub link to the repository for the implementation of stack and queue linked list is:

<https://github.com/sanjibdahal/ce3rdsem>

Stack

Implementation using Array:

Output:



```
h ArrayStack.h  h Stack.h  C++ ArrayStack.cpp  C++ main.cpp M X
stack > C++ main.cpp > main()
You, 38 seconds ago | 2 authors (You and others)
1  #include <iostream>
2  #include "ArrayStack.h"
3  using namespace std;
4
5  int main()
6  {
7      Stack *s = new ArrayStack(10);
8      cout << "Is stack empty: " << s->isEmpty() << endl;
9      s->push(10);
10     s->push(9);
11     cout << "Is stack empty: " << s->isEmpty() << endl;
12     cout << "Is stack full: " << s->isFull() << endl;
13
14     int element;
15
16     s->top(element);
17     cout << "Top element: " << element << endl;
18
19     s->pop(element);
20     cout << "Popped element: " << element << endl;
21
22     s->pop(element);
23     cout << "Popped element: " << element << endl;
24
25     s->pop(element);
26 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
● PS C:\Users\Acer\Desktop\ce3rdsem\stack> .\a.exe
Is stack empty: 1
Is stack empty: 0
Is stack full: 0
Top element: 9
Popped element: 9
Popped element: 10
Stack Underflow
```

Operations:

1. **push(element):** This function adds an element to the top of the stack. It first checks if the stack is full using the `isFull()` function. If the stack is not full, it increments the top index and assigns the new element to `arr[top]`.
2. **pop():** This function removes and returns the top element from the stack. It first checks if the stack is empty using the `isEmpty()` function. If the stack is not empty, it stores the top element in a variable `poppedElement`, decrements the top index, and returns `poppedElement`.
3. **isEmpty():** This function checks if the stack is empty by comparing the top index with -1. If top is -1, it means the stack is empty, and the function returns true. Otherwise, it returns false.
4. **isFull():** This function checks if the stack is full by comparing the top index with `maxStackSize - 1`. If top is equal to `maxStackSize - 1`, it means the stack is full, and the function returns true. Otherwise, it returns false.
5. **top():** This function returns the top element of the stack without removing it. It first checks if the stack is empty using the `isEmpty()` function. If the stack is not empty, it returns the element at the top index (`arr[top]`).

Implementation using Linked List:

Output:

```
1  #include <iostream>
2  #include "LinkedListStack.h"
3  using namespace std;
4
5  int main()
6  {
7      Stack *s = new LinkedListStack();
8      cout << "Is stack empty: " << s->isEmpty() << endl;
9      s->push(10);
10     s->push(9);
11     cout << "Is stack empty: " << s->isEmpty() << endl;
12     cout << "Is stack full: " << s->isFull() << endl;
13
14     int element;
15
16     s->top(element);
17     cout << "Top element: " << element << endl;
18
19     s->pop(element);
20     cout << "Popped element: " << element << endl;
21
22     s->pop(element);
23     cout << "Popped element: " << element << endl;
24
25     s->pop(element);
26 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS POSTMAN CONSOLE GITLENS COMMENTS

```
@Dahal on ~/Desktop/ce3rdsem/stack ~/main
# g++ -I include main.cpp src/LinkedListStack.cpp src/LinkedList.cpp
@Dahal on ~/Desktop/ce3rdsem/stack ~/main
# ./a.exe
Is stack empty: 1
Is stack empty: 0
Is stack full: 0
Top element: 9
Popped element: 9
Popped element: 10
Stack Underflow
```

Operations:

1. **push(element):** This function creates a new Node with the given element as its data and sets its next pointer to the current top node. It adds an element using addToHead() function of Linked List.
2. **pop():** This function first checks if the stack is empty using the isEmpty() function. If the stack is not empty, it stores the data of the top node in a variable poppedElement, calls removeFromHead() function of Linked List.
3. **isEmpty():** This function checks if the stack is empty by calling isEmpty() function of Linked List.
4. **top():** This function returns the data of the top node without removing it. It first checks if the stack is empty using the isEmpty() function. If the stack is not empty, it returns the data of the HEAD node (HEAD->data).

Queue

Implementation using Array:

Output:

```
1  #include <iostream>
2  #include "ArrayQueue.h"
3  using namespace std;
4
5  int main() {
6      Queue *queue = new ArrayQueue(3); // size = 3
7      int element, rear, front;
8      queue->enqueue(103);
9      queue->enqueue(201);
10     queue->enqueue(302);
11     queue->enqueue(403);
12     // rear and front element
13     cout << "Front: " << queue->front(front) << " " << front << endl;
14     cout << "Rear: " << queue->rear(rear) << " " << rear << endl;
15
16     if(queue->dequeue(element))
17         cout << "Dequeued: " << element << endl;
18     if(queue->dequeue(element))
19         cout << "Dequeued: " << element << endl;
20     if(queue->dequeue(element))
21         cout << "Dequeued: " << element << endl;
22     if(queue->dequeue(element))
23         cout << "Dequeued: " << element << endl;
24
25     return 0;
26 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE GITLENS COMMENTS

```
@Dahal on ~ /Desktop/ce3rdsem/queue %main
# g++ -I include main.cpp src/ArrayQueue.cpp
@Dahal on ~ /Desktop/ce3rdsem/queue %main
# ./a.exe
Queue is full
Front: 1 103
Rear: 1 302
Dequeued: 103
Dequeued: 201
Dequeued: 302
Queue is empty
```

Operations:

1. **enqueue(element):** It adds a new element to the rear of the queue if the queue is not full.
2. **dequeue():** It removes the front element from the queue and updates the front and rear index accordingly after checking queue is not empty.
3. **Front():** Returns the data of the front element if the queue is not empty.
4. **Rear():** Returns the data of the rear element if the queue is not empty.
5. **isEmpty():** Returns true if the queue is empty, false otherwise.
6. **isFull():** Returns true if the queue is full, false otherwise.

Implementation using Linked List

Output:

```
1  #include <iostream>
2  #include "LinkedListQueue.h"
3  using namespace std;
4
5  int main() {
6      Queue *queue = new LinkedListQueue();
7      int element, rear, front;
8      queue->enqueue(103);
9      queue->enqueue(201);
10     queue->enqueue(302);
11     // rear and front element
12     cout << "Front: " << queue->front(front) << " " << front << endl;
13     cout << "Rear: " << queue->rear(rear) << " " << rear << endl;
14
15     if(queue->dequeue(element))
16         cout << "Dequeued: " << element << endl;
17     if(queue->dequeue(element))
18         cout << "Dequeued: " << element << endl;
19     if(queue->dequeue(element))
20         cout << "Dequeued: " << element << endl;
21     if(queue->dequeue(element))
22         cout << "Dequeued: " << element << endl;
23
24     return 0;
25 }
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE GITLENS COMMENTS

```
@Dahal on ~/Desktop/ce3rdsem/queue main
# g++ -I include main.cpp src/LinkedListQueue.cpp src/LinkedList.cpp
# ./a.exe
Front: 1 103
Rear: 1 302
Dequeued: 103
Dequeued: 201
Dequeued: 302
Queue is empty
```

Operations:

1. **enqueue(element):** This function creates a new Node with the given element as its data and sets its next pointer to the current top node. It adds an element using addToTail() function of Linked List.
2. **dequeue():** This function first checks if the queue is empty using the isEmpty() function. If the queue is not empty, it stores the data of the front node in a variable poppedElement, calls removeFromHead() function of Linked List.
3. **front():** This function returns the data of the front node without removing it. It first checks if the queue is empty using the isEmpty() function. If the queue is not empty, it returns the data of the HEAD node (HEAD->data).
4. **rear():** This function returns the data of the rear node without removing it. It first checks if the queue is empty using the isEmpty() function. If the queue is not empty, it returns the data of the TAIL node (TAIL->data).
5. **isEmpty():** This function checks if the queue is empty by calling isEmpty() function of Linked List.

Conclusion:

Overall, stack and queue are efficient and flexible linear data structures with different operations to insert, delete in it. In this way, we can implement stack and queue using array and linked list.