

VIETNAM NATIONAL UNIVERSITY,
HO CHI MINH CITY

UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Implementing Stack and Queue from scratch

CSC10004 – DATA STRUCTURES AND ALGORITHMS

Ngo Nguyen The Khoa – 23127065 – 23CLC09

June 3, 2024

Contents

1	Result Screenshots	2
1.1	Stack (Array version)	2
1.2	Stack (Linked List version)	4
1.3	Queue (Array version)	5
1.4	Queue (Linked List version)	6
2	Recursive versions	7
2.1	Stack	7
2.2	Queue	8
3	Self-evaluation	10
4	Exercise Feedback	10
4.1	What I learned	10
4.2	What I found challenging	10
4.3	What I have used in this exercise	10

1 Result Screenshots

1.1 Stack (Array version)

‘push’ operation

```
-----  
| Stack max size: 3  
| Stack size: 2  
| Stack elements: 1 2  
-----  
  
>>> Stack Lib  
1. Push  
2. Pop  
0. Exit  
  
=: Enter your choice: 1  
  
=: Enter value to be pushed: 3  
  
?: Continue: Yes(1) - No(0)
```

(a) Normal test cases

```
-----  
| Stack max size: 3  
| Stack size: 3  
| Stack elements: 1 2 3  
-----  
  
>>> Stack Lib  
1. Push  
2. Pop  
0. Exit  
  
=: Enter your choice: 1  
  
=: Enter value to be pushed: 4  
!: Stack is full! Please pop before pushing  
  
?: Continue: Yes(1) - No(0)
```

(b) When the stack is full

```
StackArray  
>• cpp main  
-----  
| Stack is not initialized.  
-----  
  
>>> Stack Lib  
1. Push  
2. Pop  
0. Exit  
  
=: Enter your choice: 1  
  
!: Stack is not initialized.  
?: Do you want to initialize the stack? Yes(1) - No(0)  
1  
  
=: Enter stack max size: 3  
  
=: Enter value to be pushed: 1  
  
?: Continue: Yes(1) - No(0)
```

(c) When the stack is uninitialized

Figure 1: Screenshots of Stack (Array version) ‘push’ operation.

‘pop’ operation

```
-----  
| Stack max size: 3  
| Stack size: 0  
| Stack elements:  
  
-----  
>>> Stack Lib  
1. Push  
2. Pop  
0. Exit  
  
=: Enter your choice: 2  
  
!: Stack is empty! Please push before popping  
  
?: Continue: Yes(1) - No(0)
```

(a) When the stack is empty

```
-----  
| Stack max size: 3  
| Stack size: 3  
| Stack elements: 1 2 3  
  
-----  
>>> Stack Lib  
1. Push  
2. Pop  
0. Exit  
  
=: Enter your choice: 2  
  
>>> Popped value: 3  
  
?: Continue: Yes(1) - No(0)
```

(b) Normal test cases

Figure 2: Screenshots of Stack (Array version) ‘pop’ operation.

1.2 Stack (Linked List version)

‘push’ operation

```
StackLinkedList
>• cpp main
-----
| Stack is not initialized.
-----
>>> Stack Lib
1. Push
2. Pop
0. Exit

=: Enter your choice: 1

!: Stack is not initialized.
?: Do you want to initialize the stack? Yes(1) - No(0)
1
=: Enter value to be pushed: 1
?: Continue: Yes(1) - No(0)
```

(a) When the stack is uninitialized

```
-----
| Stack elements: 1
-----
>>> Stack Lib
1. Push
2. Pop
0. Exit

=: Enter your choice: 1

=: Enter value to be pushed: 2
?: Continue: Yes(1) - No(0)
```

(b) Normal test cases

Figure 3: Screenshots of Stack (Linked List version) ‘push’ operation.

‘pop’ operation

```
-----
| Stack elements:
-----
>>> Stack Lib
1. Push
2. Pop
0. Exit

=: Enter your choice: 2

!: Stack is empty! Please push before popping
?: Continue: Yes(1) - No(0)
```

(a) When the stack is empty

```
-----
| Stack elements: 3 2 1
-----
>>> Stack Lib
1. Push
2. Pop
0. Exit

=: Enter your choice: 2

>>> Popped value: 3
?: Continue: Yes(1) - No(0)
```

(b) Normal test cases

Figure 4: Screenshots of Stack (Linked List version) ‘pop’ operation.

1.3 Queue (Array version)

‘enqueue’ operation

```

-----
| Queue max size: 3
| Queue size: 1
| Queue elements: 1
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 1

=: Enter value to be pushed: 2

?: Continue: Yes(1) - No(0)

```

(a) Normal test cases

```

-----
| Queue max size: 3
| Queue size: 3
| Queue elements: 1 2 3
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 1

=: Enter value to be pushed: 4
!: Queue is full! Please dequeue before enqueueing

?: Continue: Yes(1) - No(0)

```

(b) When the queue is full

```

QueueArray
>• cpp main
-----
| Queue is not initialized.
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 1

!: Queue is not initialized.
?: Do you want to initialize the queue? Yes(1) - No(0)
1

=: Enter queue max size: 3

=: Enter value to be pushed: 1

?: Continue: Yes(1) - No(0)

```

(c) When the queue is uninitialized

Figure 5: Screenshots of queue (Array version) ‘enqueue’ operation.

‘dequeue’ operation

```

-----
| Queue max size: 3
| Queue size: 0
| Queue elements:
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 2

!: Queue is empty! Please enqueue before dequeuing

?: Continue: Yes(1) - No(0)

```

(a) When the queue is empty

```

-----
| Queue max size: 3
| Queue size: 3
| Queue elements: 1 2 3
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 2

>>> Front value: 1

?: Continue: Yes(1) - No(0)

```

(b) Normal test cases

Figure 6: Screenshots of queue (Array version) 'dequeue' operation.

1.4 Queue (Linked List version)

'enqueue' operation

```

QueueLinkedList
>• cpp main
-----
| Queue is not initialized.
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 1

!: Queue is not initialized.
?: Do you want to initialize the queue? Yes(1) - No(0)
1
=: Enter value to be pushed: 1

?: Continue: Yes(1) - No(0)

```

(a) When the queue is uninitialized

```

-----
| Queue elements: 1
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 1

=: Enter value to be pushed: 2

?: Continue: Yes(1) - No(0)

```

(b) Normal test cases

Figure 7: Screenshots of queue (Linked List version) 'enqueue' operation.

'dequeue' operation

```

-----
| Queue elements:
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 2

!: Queue is empty! Please enqueue before dequeuing

?: Continue: Yes(1) - No(0)

```

(a) When the queue is empty

```

-----
| Queue elements: 1 2 3
-----
>>> Queue Lib
1. Enqueue
2. Dequeue
0. Exit

=: Enter your choice: 2

>>> Front value: 1

?: Continue: Yes(1) - No(0)

```

(b) Normal test cases

Figure 8: Screenshots of queue (Linked List version) ‘dequeue’ operation.

2 Recursive versions

2.1 Stack

Recursion Stack (Array version)

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of $O(1)$ for push and $O(n)$ for copy operations.
- **Practical Performance:**
 - Loop-based implementations are generally faster in practice because they avoid the overhead of function calls and recursion.
 - The recursive version doesn’t work well with large data sets because of stack overflow.

```

StackArray
>> cpp metrics
Push 10000000 elements
    Push with loop version: 158 milliseconds
    Push with recursive version: 161 milliseconds
Copy stack (40000 elements)
    Copy stack with loop version: 115 microseconds
    Copy stack with recursive version: 1299 microseconds

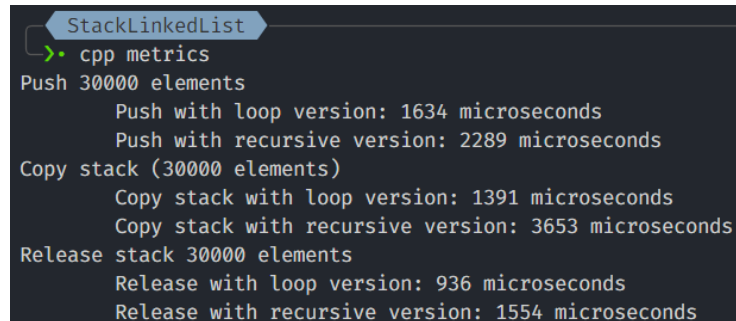
```

Figure 9: Recursive and loop version comparison of Stack (Array version)

Recursion Stack (Linked List version)

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of $O(1)$ for push, $O(n)$ for copy and release operations.
- **Practical Performance:**

- Loop-based implementations are generally faster in practice because they avoid the overhead of function calls and recursion.
- The recursive version doesn't work well with large data sets because of stack overflow.



```

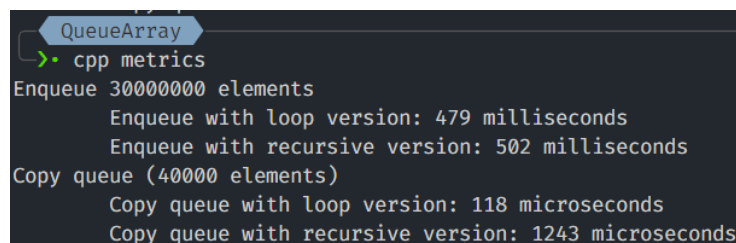
StackLinkedList
>• cpp metrics
Push 30000 elements
    Push with loop version: 1634 microseconds
    Push with recursive version: 2289 microseconds
Copy stack (30000 elements)
    Copy stack with loop version: 1391 microseconds
    Copy stack with recursive version: 3653 microseconds
Release stack 30000 elements
    Release with loop version: 936 microseconds
    Release with recursive version: 1554 microseconds
  
```

Figure 10: Recursive and loop version comparison of Stack (Linked List version)

2.2 Queue

Recursion Queue (Array version)

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of $O(1)$ for **enqueue** and $O(n)$ for **copy** operations.
- **Practical Performance:**
 - Loop-based implementations are generally faster in practice because they avoid the overhead of function calls and recursion.
 - The recursive version doesn't work well with large data sets because of stack overflow.



```

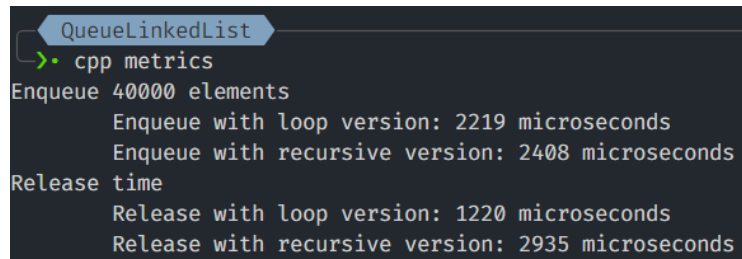
QueueArray
>• cpp metrics
Enqueue 30000000 elements
    Enqueue with loop version: 479 milliseconds
    Enqueue with recursive version: 502 milliseconds
Copy queue (40000 elements)
    Copy queue with loop version: 118 microseconds
    Copy queue with recursive version: 1243 microseconds
  
```

Figure 11: Recursive and loop version comparison of Queue (Array version)

Recursion Queue (Linked List version)

- **Theoretical Time Complexity:** Both implementations have the same theoretical time complexity of $O(1)$ for **enqueue** and $O(n)$ for **release** operations.
- **Practical Performance:**
 - Loop-based implementations are generally faster in practice because they avoid the overhead of function calls and recursion.

- The recursive version doesn't work well with large data sets because of stack overflow.



A terminal window with a dark background and light blue text. At the top, a light blue header bar contains the text 'QueueLinkedList'. Below the header, the prompt '>' is followed by 'cpp metrics'. The output shows 'Enqueue 40000 elements' with two sub-entries: 'Enqueue with loop version: 2219 microseconds' and 'Enqueue with recursive version: 2408 microseconds'. Below this, 'Release time' is shown with two sub-entries: 'Release with loop version: 1220 microseconds' and 'Release with recursive version: 2935 microseconds'.

```
QueueLinkedList
> cpp metrics
Enqueue 40000 elements
    Enqueue with loop version: 2219 microseconds
    Enqueue with recursive version: 2408 microseconds
Release time
    Release with loop version: 1220 microseconds
    Release with recursive version: 2935 microseconds
```

Figure 12: Recursive and loop version comparison of Queue (Linked List version)

3 Self-evaluation

No.	Details	Score
1	Stack (Array version)	100%
2	Stack (Linked List version)	100%
3	Queue (Array version)	100%
4	Queue (Linked List version)	100%
5	Recursive versions	100%
6	Report	100%

4 Exercise Feedback

4.1 What I learned

Because almost the things could be done easily, I have learned nothing new from this exercise.

4.2 What I found challenging

This exercise is quite simple and easy, so I have no difficulties to finish this task.

4.3 What I have used in this exercise

- I used C++ to implement the data structures and the test cases.
- I also used \LaTeX to write this report.
- All the source code and the report are available in [my github repo](#)