VIETNAM NATIONAL UNIVERSITY OF
HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

Report

Exercise 1: Implementing Stack and Queue from scratch

Course name: Data Structures and Algorithms

CSC10004_23CLC09

Students:
Nguyen Le Ho Anh Khoa - 23127211

Teacher:
Bui Duy Dang
Truong Tan Khoa
Nguyen Thanh Tinh

June 15, 2024

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHTN
ĐHQG-HCM

Contents

1 Student Information 2

2 Detailed Experiments 2
2.1 Stack (Array version) . . . 2
2.1.1 Push operators . . . 2
2.1.2 Pop operators . . . 2
2.2 Stack (Linked List version) . . . 3
2.2.1 Push operators . . . 3
2.2.2 Pop operators . . . 3
2.3 Queue (Array version) . . . 4
2.3.1 Enqueue operators . . . 4
2.3.2 Dequeue operators . . . 4
2.4 Queue (Linked List version) . . . 5
2.4.1 Enqueue operators . . . 5
2.4.2 Dequeue operators . . . 5
2.5 Recursion Version . . . 6
2.5.1 Stack . . . 6
2.5.2 Queue . . . 7

3 Self - Evaluation 7

4 Exercise Feedback 8
4.1 What have I learned . . . 8
4.2 What was my difficult . . . 8

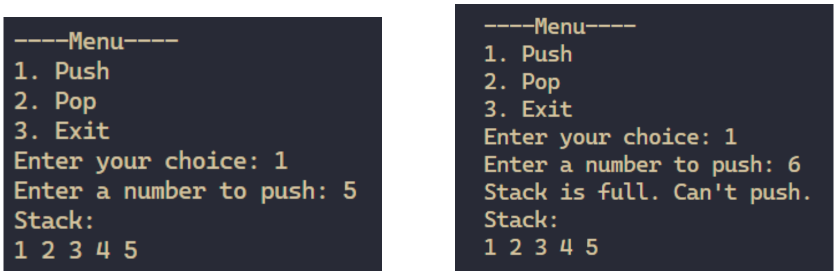
1 Student Information

Class: 23CLC09
Student ID: 23127211
Full name: Nguyen Le Ho Anh Khoa

2 Detailed Experiments

2.1 Stack (Array version)

2.1.1 Push operators



```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter a number to push: 5
Stack:
1 2 3 4 5
```

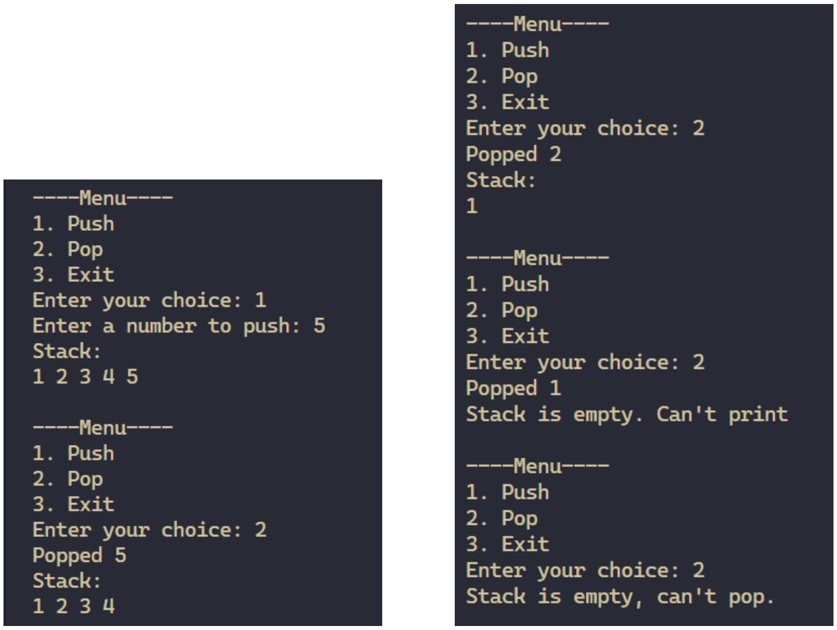
(a) Normal case

```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter a number to push: 6
Stack is full. Can't push.
Stack:
1 2 3 4 5
```

(b) Full case

Figure 1: Push element in Stack by Array (size=5)

2.1.2 Pop operators



```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter a number to push: 5
Stack:
1 2 3 4 5

-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 2
Popped 5
Stack:
1 2 3 4
```

(a) Normal case

```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 2
Popped 2
Stack:
1

-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 2
Popped 1
Stack is empty. Can't print

-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 2
Stack is empty, can't pop.
```

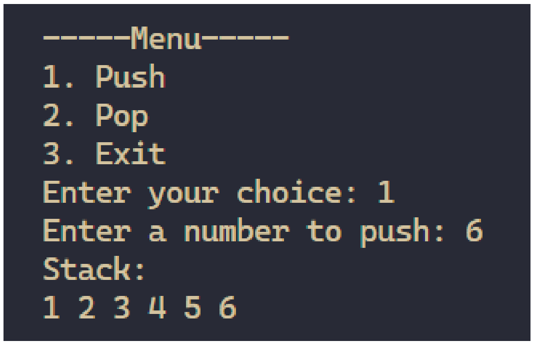
(b) Empty case

Figure 2: Pop element in Stack by Array

2.2 Stack (Linked List version)

2.2.1 Push operators

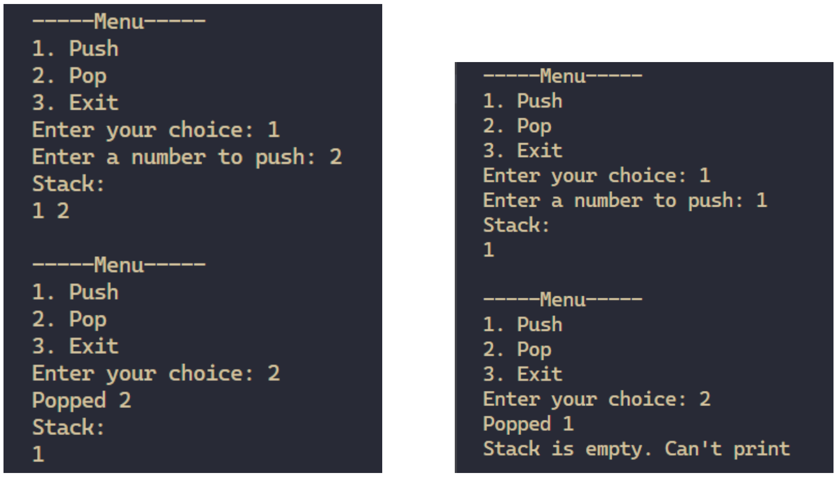
Implement the Stack by singly linked list is created by dynamic allocation, does not require a fixed size when declaring. Therefore, we can add or remove elements easily without changing the original declaration and there is no maximum size when declared as an array.
Therefore, the size of the Stack will depend on the computer's RAM memory.



```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter a number to push: 6
Stack:
1 2 3 4 5 6
```

Figure 3: Push element in Stack by Linked List

2.2.2 Pop operators



```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter a number to push: 2
Stack:
1 2

-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 2
Popped 2
Stack:
1
```

(a) Normal case

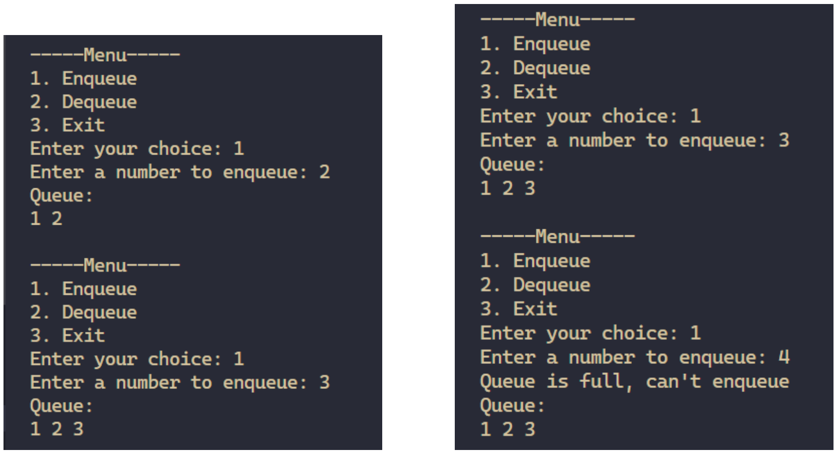
```
-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter a number to push: 1
Stack:
1

-----Menu-----
1. Push
2. Pop
3. Exit
Enter your choice: 2
Popped 1
Stack is empty. Can't print
```

(b) Empty case

Figure 4: Pop element in Stack by Linked List

2.3 Queue (Array version)

2.3.1 Enqueue operators



```
------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter a number to enqueue: 2
Queue:
1 2

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter a number to enqueue: 3
Queue:
1 2 3
```

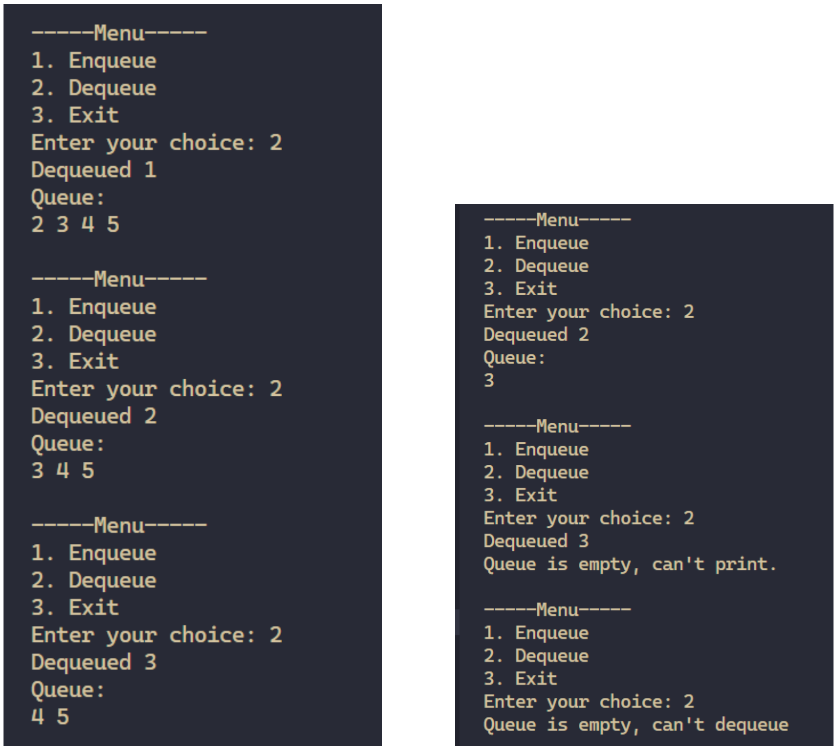
(a) Normal case

```
------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter a number to enqueue: 3
Queue:
1 2 3

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter a number to enqueue: 4
Queue is full, can't enqueue
Queue:
1 2 3
```

(b) Full case

Figure 5: Enqueue element in Queue by Array

2.3.2 Dequeue operators



```
------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 1
Queue:
2 3 4 5

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 2
Queue:
3 4 5

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 3
Queue:
4 5
```

(a) Normal case

```
------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 2
Queue:
3

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 3
Queue is empty, can't print.

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Queue is empty, can't dequeue
```

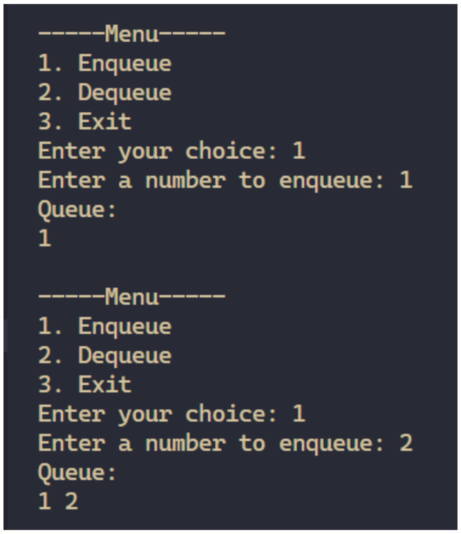
(b) Empty case

Figure 6: Dequeue element in Queue by Array

2.4 Queue (Linked List version)

2.4.1 Enqueue operators

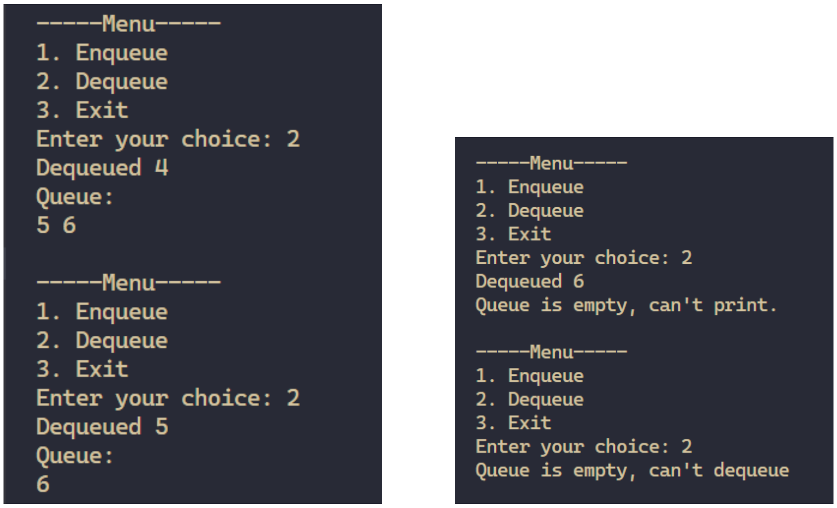
Similar to Stack initialized with Linked List, Queue initialized with Linked List also does not have a maximum size. Therefore, the size of the Queue will depend on the computer's RAM.



```
------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter a number to enqueue: 1
Queue:
1

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter a number to enqueue: 2
Queue:
1 2
```

Figure 7: Enqueue element in Queue by Linked List

2.4.2 Dequeue operators



```
------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 4
Queue:
5 6

------Menu------
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 5
Queue:
6
```

(a) Normal case

```
-----Menu-----
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Dequeued 6
Queue is empty, can't print.

-----Menu-----
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 2
Queue is empty, can't dequeue
```

(b) Empty case

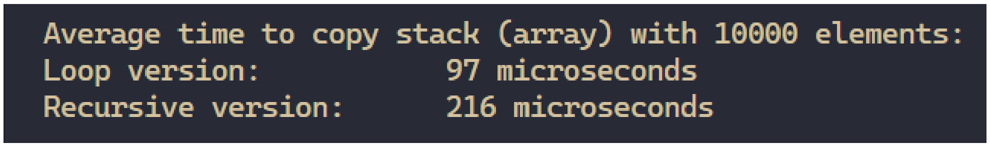
Figure 8: Dequeue element in Queue by Linked List

2.5 Recursion Version

Recursive functions can be slower than loops due to overheads of function calls, more memory usage due to recursive calls added to the stack, and potential repetitive computations. Loops are generally more efficient, but recursion can offer simpler solutions for some problems.
I utilized the function from the Chrono library to measure and compare the running time of the algorithm. [2]

2.5.1 Stack

Array version



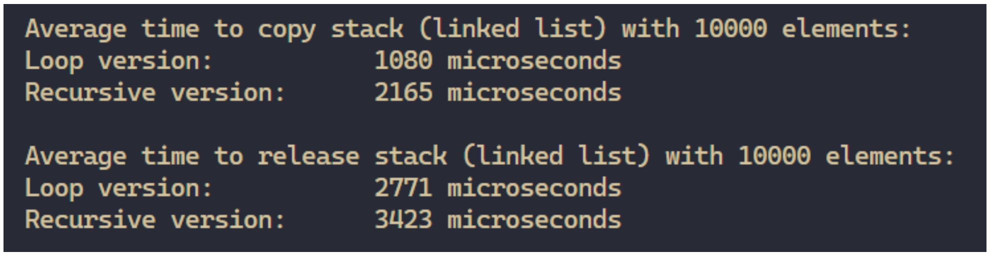
```
Average time to copy stack (array) with 10000 elements:
Loop version:           97 microseconds
Recursive version:      216 microseconds
```

Figure 9: Compare time loop and recursive version of Stack(Array)

- Both implementations function copy Array have a time complexity of **O(n)** and the loop version is faster than recursive version.

The system will have a **Stack Overflow** if using an array with a recursive version with a large number of elements (above 10^8 elements).

Linked List version



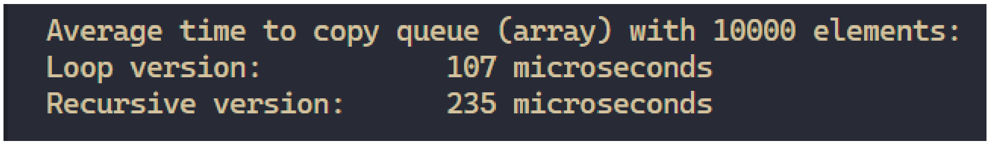
```
Average time to copy stack (linked list) with 10000 elements:
Loop version:           1080 microseconds
Recursive version:      2165 microseconds

Average time to release stack (linked list) with 10000 elements:
Loop version:           2771 microseconds
Recursive version:      3423 microseconds
```

Figure 10: Compare time loop and recursive version of Stack(Linked List)

- Both implementations function copy and release Linked List have a time complexity of **O(n)** and the loop version is faster than recursive version.

2.5.2 Queue

Array version



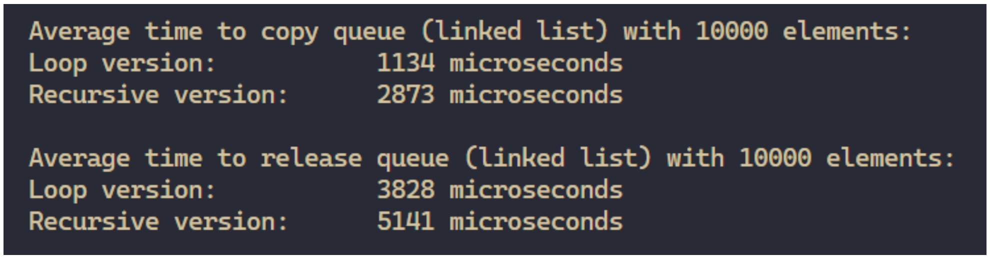
```
Average time to copy queue (array) with 10000 elements:
Loop version:            107 microseconds
Recursive version:       235 microseconds
```

Figure 11: Compare time loop and recursive version of Queue(Array)

- Both implementations function copy Array have a time complexity of **O(n)** and the loop version is faster than recursive version.

The system will have a **Stack Overflow** if using an array with a recursive version with a large number of elements (above 10^8 elements).

Linked List version



```
Average time to copy queue (linked list) with 10000 elements:
Loop version:            1134 microseconds
Recursive version:       2873 microseconds

Average time to release queue (linked list) with 10000 elements:
Loop version:            3828 microseconds
Recursive version:       5141 microseconds
```

Figure 12: Compare time loop and recursive version of Queue(Linked List)

- Both implementations function copy and release Linked List have a time complexity of **O(n)** and the loop version is faster than recursive version.

3 Self - Evaluation

In this exercise, I meticulously implemented a stack and a queue from scratch in C++. My evaluation encompassed several critical aspects.

- I verified the correctness of both implementations, ensuring that they adhered to the fundamental properties of stacks and queues.
- I analyzed efficiency—time and space complexity—for each operation. The stack's push and pop operations achieved the desired O(1) time complexity, while the queue's enqueue and dequeue operations followed the specified requirements.
- I paid close attention to memory usage, avoiding unnecessary wastage.
- I prioritized code readability and maintainability, using descriptive names and adding explanatory comments. Lastly, I thoroughly tested edge cases and implemented robust error handling. The code is modular, well-commented, and meets the specified criteria.

Table 1: Self - Evaluation about my Exercise

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%
	Total	**100%**

4 Exercise Feedback

4.1 What have I learned

- In the past, my approach was sequential programming, but this task has broadened my knowledge to include the fundamentals of object-oriented programming.
- I've acquired skills in creating reports with LaTeX.
- I've employed Github as a vault for my source code and reports, all of which are securely stored in my personal repository.

4.2 What was my difficult

- At the outset, my journey with programming was fraught with difficulties due to my unfamiliarity with object-oriented programming. However, my comprehension has been greatly enhanced after delving into a plethora of resources available on the internet. [1]
- I encountered some difficulties when writing recursive functions. However, after a period of debugging and contemplation, I successfully resolved the issues.
- I inadvertently left my laptop charger behind during the final days of working on this exercise. However, I had backed up my source code on Github. Now, all I need to do is head to the library, log into Overleaf, and complete the report.
- Lastly, my English proficiency isn't quite up to par yet, so there's a possibility that this report contains some grammatical errors.

References

[1] Vankayala Karunakar. Object Oriented Programming in C++. 2024.

[2] sayan mahapatra. Measure execution time of a function in C++. 2023.