

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN



---

## Bài tập 1

Đề tài: Triển khai ngăn xếp và hàng đợi từ đầu

---

Môn học: Cấu trúc Dữ liệu và Giải thuật

*Sinh viên thực hiện:*

Nguyễn Thái Bảo

Mã số sinh viên: 23120023

*Giáo viên hướng dẫn:*

Thầy Nguyễn Ngọc Đức

Thầy Nguyễn Trần Duy Minh

Thầy Nguyễn Thanh Tình

Ngày 10 tháng 11 năm 2024

# Mục lục

<b>1</b>	<b>Thông tin sinh viên - Student information</b>	<b>6</b>
<b>2</b>	<b>Tự đánh giá - Self-evaluation</b>	<b>6</b>
<b>3</b>	<b>Ngăn xếp (phiên bản Mảng) - Stack (Array version)</b>	<b>7</b>
3.1	Tổng quan về cấu trúc dữ liệu và chương trình . . . . .	7
3.2	Trường hợp bình thường . . . . .	9
3.3	Trường hợp stack đã đầy . . . . .	12
3.4	Trường hợp stack rỗng . . . . .	13
3.5	Kết thúc chương trình . . . . .	14
<b>4</b>	<b>Ngăn xếp (phiên bản Danh sách liên kết) - Stack (Linked List version)</b>	<b>14</b>
4.1	Tổng quan về cấu trúc dữ liệu và chương trình . . . . .	14
4.2	Trường hợp bình thường . . . . .	16
4.3	Trường hợp stack đã đầy . . . . .	18
4.4	Trường hợp stack rỗng . . . . .	19
4.5	Kết thúc chương trình . . . . .	20
<b>5</b>	<b>Hàng đợi (phiên bản Mảng) - Queue (Array version)</b>	<b>21</b>
5.1	Tổng quan về cấu trúc dữ liệu và chương trình . . . . .	21
5.2	Trường hợp bình thường . . . . .	23
5.3	Trường hợp queue đã đầy . . . . .	26
5.4	Trường hợp queue rỗng . . . . .	27
5.5	Kết thúc chương trình . . . . .	28
<b>6</b>	<b>Hàng đợi (phiên bản Danh sách liên kết) - Queue (Linked List version)</b>	<b>28</b>
6.1	Tổng quan về cấu trúc dữ liệu và chương trình . . . . .	28
6.2	Các thao tác trong trường hợp bình thường . . . . .	30
6.3	Trường hợp queue đã đầy . . . . .	32
6.4	Trường hợp queue rỗng . . . . .	33
6.5	Kết thúc chương trình . . . . .	34

<b>7</b>	<b>Các phiên bản đệ quy - Recursive versions</b>	<b>35</b>
7.1	<b>Stack (Array version)</b>	35
7.1.1	Copy Stack	35
7.1.2	Print	37
7.2	<b>Stack (Linked List version)</b>	39
7.2.1	Copy Stack	39
7.2.2	Release	41
7.2.3	Print	43
7.3	<b>Queue (Array version)</b>	45
7.3.1	Copy Queue	45
7.3.2	Dequeue	46
7.3.3	Print	48
7.4	<b>Queue (Linked List version)</b>	50
7.4.1	Release	50
7.4.2	Print	52
7.4.3	Copy Queue	54
<b>8</b>	<b>Nhận xét - Feedback</b>	<b>57</b>

## Danh sách hình vẽ

1	[Stack Array] Giao diện console chương trình . . . . .	9
2	[Stack Array] Push và in các phần tử trong stack . . . . .	10
3	[Stack Array] Pop và lấy giá trị top trong stack . . . . .	10
4	[Stack Array] Copy stack từ một stack khác được tạo ngẫu nhiên và in ra . . . . .	11
5	[Stack Array] Kiểm tra lại giá trị top của stack sau khi sao chép . . . . .	11
6	[Stack Array] Giải phóng và khởi động lại một stack với kích thước mới và kiểm thử . . . . .	12
7	[Stack Array] Không thể push vào stack đã đầy . . . . .	12
8	[Stack Array] Không thể pop và lấy giá trị top từ một stack rỗng . . . . .	13
9	[Stack Array] Khi in một stack rỗng sẽ có cảnh báo . . . . .	13
10	[Stack Array] Nhập số 0 để kết thúc chương trình . . . . .	14
11	[Stack Linked List] Giao diện console chương trình . . . . .	16
12	[Stack Linked List] Push và in các phần tử trong stack . . . . .	16
13	[Stack Linked List] Pop và lấy giá trị top trong stack . . . . .	17
14	[Stack Linked List] Copy stack từ một stack khác được tạo ngẫu nhiên và in ra . . . . .	17
15	[Stack Linked List] Kiểm tra lại giá trị top của stack sau khi sao chép . . . . .	18
16	[Stack Linked List] Giải phóng và khởi động lại một stack với kích thước mới và kiểm thử . . . . .	18
17	[Stack Linked List] Không thể push vào stack đã đầy . . . . .	19
18	[Stack Linked List] Không thể pop và lấy giá trị top từ một stack rỗng . . . . .	19
19	[Stack Linked List] Khi in một stack rỗng sẽ có cảnh báo . . . . .	20
20	[Stack Linked List] Nhập số 0 để kết thúc chương trình . . . . .	20
21	[Queue Array] Giao diện console chương trình . . . . .	23
22	[Queue Array] Enqueue và in các phần tử trong queue . . . . .	24
23	[Queue Array] Dequeue và lấy giá trị front trong queue . . . . .	24
24	[Queue Array] Copy queue từ một queue khác được tạo ngẫu nhiên và in ra . . . . .	25
25	[Queue Array] Kiểm tra lại giá trị front của queue sau khi sao chép . . . . .	25
26	[Queue Array] Giải phóng và khởi động lại một queue với kích thước mới và kiểm thử . . . . .	26
27	[Queue Array] Không thể enqueue vào queue đã đầy . . . . .	26
28	[Queue Array] Không thể dequeue và lấy giá trị front từ một queue rỗng . . . . .	27

29	[Queue Array] Khi in một queue rỗng sẽ có cảnh báo . . . . .	27
30	[Queue Array] Nhập số 0 để kết thúc chương trình . . . . .	28
31	[Queue Linked List] Giao diện console chương trình . . . . .	30
32	[Queue Linked List] Enqueue và in các phần tử trong queue . . . . .	30
33	[Queue Linked List] Dequeue và lấy giá trị front trong queue . . . . .	31
34	[Queue Linked List] Copy queue từ một queue khác được tạo ngẫu nhiên và in ra .	31
35	[Queue Linked List] Kiểm tra lại giá trị front của queue sau khi sao chép . . . . .	32
36	[Queue Linked List] Giải phóng và khởi động lại một queue với kích thước mới và kiểm thử . . . . .	32
37	[Queue Linked List] Không thể enqueue vào queue đã đầy . . . . .	33
38	[Queue Linked List] Không thể dequeue và lấy giá trị front từ một queue rỗng . . .	33
39	[Queue Linked List] Khi in một queue rỗng sẽ có cảnh báo . . . . .	34
40	[Queue Linked List] Nhập số 0 để kết thúc chương trình . . . . .	34
41	[Stack Array] So sánh thời gian thực thi COPY STACK (lần 1) . . . . .	36
42	[Stack Array] So sánh thời gian thực thi COPY STACK (lần 2) . . . . .	37
43	[Stack Array] So sánh thời gian thực thi COPY STACK (lần 1) . . . . .	38
44	[Stack Array] So sánh thời gian thực thi COPY STACK (lần 2) . . . . .	39
45	[Stack Linked List] So sánh thời gian thực thi COPY STACK (lần 1) . . . . .	41
46	[Stack Linked List] So sánh thời gian thực thi COPY STACK (lần 2) . . . . .	41
47	[Stack Linked List] So sánh thời gian thực thi RELEASE (lần 1) . . . . .	42
48	[Stack Linked List] So sánh thời gian thực thi RELEASE (lần 2) . . . . .	43
49	[Stack Linked List] So sánh thời gian thực thi PRINT (lần 1) . . . . .	44
50	[Stack Linked List] So sánh thời gian thực thi PRINT (lần 2) . . . . .	44
51	[Queue Array] So sánh thời gian thực thi COPY QUEUE (lần 1) . . . . .	46
52	[Queue Array] So sánh thời gian thực thi COPY QUEUE (lần 2) . . . . .	46
53	[Queue Array] So sánh thời gian thực thi DEQUEUE (lần 1) . . . . .	48
54	[Queue Array] So sánh thời gian thực thi DEQUEUE (lần 2) . . . . .	48
55	[Queue Array] So sánh thời gian thực thi PRINT (lần 1) . . . . .	50
56	[Queue Array] So sánh thời gian thực thi PRINT (lần 2) . . . . .	50
57	[Queue Linked List] So sánh thời gian thực thi RELEASE (lần 1) . . . . .	52
58	[Queue Linked List] So sánh thời gian thực thi RELEASE (lần 2) . . . . .	52

59	[Queue Linked List] So sánh thời gian thực thi PRINT (lần 1)	53
60	[Queue Linked List] So sánh thời gian thực thi PRINT (lần 2)	54
61	[Queue Linked List] So sánh thời gian thực thi COPY QUEUE (lần 1)	56
62	[Stack Linked List] So sánh thời gian thực thi COPY QUEUE (lần 2)	56

## Danh sách bảng

1	Tự đánh giá mức độ hoàn thành bài tập	6
---	---------------------------------------	---

# 1 Thông tin sinh viên - Student information

- Họ và tên: Nguyễn Thái Bảo
- MSSV: 23120023
- Lớp: 23CTT1
- Môn học: Cấu trúc Dữ liệu và Giải thuật

# 2 Tự đánh giá - Self-evaluation

Đánh giá nội dung bản thân đã làm được so với yêu cầu:

STT	Nội dung	Mức độ hoàn thành	Hệ số điểm
1	Stack (Array version)	100%	15%
2	Stack (Linked List version)	100%	15%
3	Queue (Array version)	100%	15%
4	Queue (Linked List version)	100%	15%
5	Recursive versions	100%	20%
6	Report	100%	20%

Bảng 1: Tự đánh giá mức độ hoàn thành bài tập

Đánh giá: hoàn thành đầy đủ yêu cầu bài tập.

## 3 Ngăn xếp (phiên bản Mảng) - Stack (Array version)

### 3.1 Tổng quan về cấu trúc dữ liệu và chương trình

Trong phần này, ta tiến hành tạo một chương trình để kiểm thử các thao tác (**pop** và **push**) đã được triển khai trên ngăn xếp hay Stack (từ đây **stack** và ngăn xếp sẽ được sử dụng để thay thế nhau với cùng ý nghĩa). Chương trình được kiểm thử với các trường hợp bình thường, khi stack rỗng và khi stack đã đầy.

Cấu trúc Stack được cài đặt bằng kiểu dữ liệu cấu trúc **struct** kết hợp với **template** (khuôn mẫu) là một từ khóa trong C++ để tổng quát hóa cho nhiều kiểu dữ liệu khác nhau, giúp mã nguồn trở nên linh hoạt và có thể tái sử dụng. Ở đây, Stack được cài đặt bằng **mảng** có cấu trúc như sau:

- **Biến (variables):**

- **items:** mảng lưu trữ dữ liệu.
- **top:** chỉ số của phần tử đầu stack (có giá trị -1 nếu stack rỗng).
- **maxSize:** kích thước tối đa của stack, là một số nguyên không âm.

- **Hàm (functions):**

- **init:** khởi tạo một stack rỗng với kích thước đã cho.
- **copyStack:** khởi tạo một stack từ một stack khác.
- **release:** dọn dẹp stack khi không còn sử dụng.
- **isEmpty:** kiểm tra stack rỗng hay không.
- **isFull:** kiểm tra stack có đầy chưa.
- **push:** toán tử thêm phần tử mới vào stack, cụ thể là thêm vào cuối của mảng **items** nếu chưa vượt quá **maxSize**.
- **pop:** xóa phần tử ở đầu stack, nếu stack rỗng thì in thông báo ra màn hình.



- **topValue**: lấy giá trị đầu stack, nếu stack rỗng thì in thông báo ra màn hình.
- **print**: in các giá trị trong stack ra màn hình (từ đầu stack tới cuối stack).

Khai báo Stack được đặt trong tệp tin **stack.h**, cài đặt Stack được đặt trong tệp tin **stack.ipp**. Ngoài ra còn có hai tệp tương tự cho cấu trúc Stack sử dụng các hàm đệ quy thay thế cho vòng lặp là **recursive\_stack.h** và **recursive\_stack.ipp**.

Dưới đây là mã nguồn (source code) của Stack:

```

1  template <typename T>
2  struct Stack {
3      T* items;
4      int top;
5      unsigned int maxSize;
6
7      void init(unsigned int stackSize);
8      void copyStack(const Stack<T>& s);
9      void release();
10
11     bool isEmpty();
12     bool isFull();
13     void push(T newItem);
14     T pop();
15     T topValue();
16
17     void print();
18 };

```

Chương trình được xây dựng trong tệp **main.cpp** có giao diện với tiêu đề trên cùng là tên chương trình. Trước tiên, người dùng được yêu cầu nhập kích thước tối đa cho stack. Sau khi nhập một số nguyên không âm hợp lệ, màn hình hiển thị các thao tác trên stack có thể thực hiện ứng với các phương án số (1, 2, 3, ...) để người dùng có thể lựa chọn. Một số thao tác cơ bản trong chương trình là: push, pop, lấy giá trị top, in stack ra màn hình, copy stack, ...

Dưới đây là giao diện cơ bản của chương trình:

```

E:\source\repos\HK3\DSA_Exp X + -
----- Stack (Array version) -----
Enter the size of the stack: 3
Stack initialized with size 3.

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: |
    
```

Hình 1: [Stack Array] Giao diện console chương trình

### 3.2 Trường hợp bình thường

- Trong trường hợp bình thường (stack không rỗng cũng chưa đầy), ta có thể tiến hành push các phần tử vào stack theo nguyên lý LIFO (Last In, First Out) như một ngăn xếp chồng lên nhau, cụ thể ở đây ta thêm các phần tử vào cuối mảng đồng thời cập nhật biến `top`. Lựa chọn 1 trong chương trình cho phép thực hiện thao tác này.
- Khi stack đã có phần tử, ta có thể tiến hành in các phần tử trong stack từ `top` xuống bằng cách chọn lựa chọn 4. Minh họa như hình dưới đây:

```

E:\source\repos\HK3\DSA_Exc >
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 1
Enter the item to push: 3

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 4
3 2 1
-----

```

Hình 2: [Stack Array] Push và in các phần tử trong stack

- Khi stack đã có phần tử, ta có thể xóa phần tử ở trên cùng của stack bằng thao tác **pop**, thực hiện bằng lựa chọn 2. Sau khi xóa, biến **top** cũng sẽ được cập nhật.
- Lựa chọn 3 cho phép người dùng lấy ra giá trị ở đầu stack, cụ thể là phần tử trong mảng ở chỉ số **top**.

```

E:\source\repos\HK3\DSA_Exc >
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 2
Popped item: 3

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

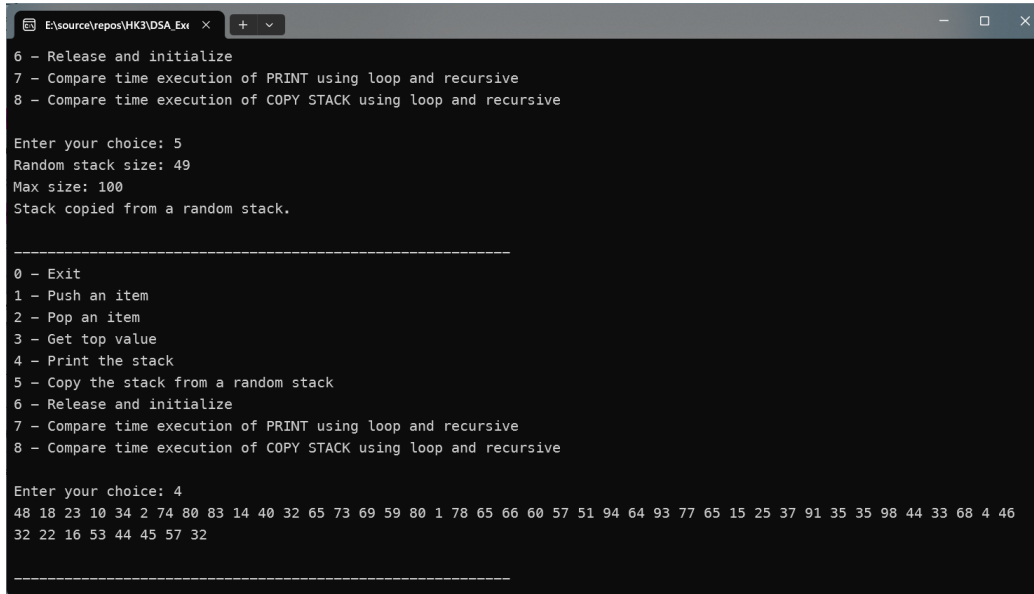
Enter your choice: 3
Top value: 2
-----

```

Hình 3: [Stack Array] Pop và lấy giá trị top trong stack

- Khi chọn lựa chọn 5, chương trình tự động khởi tạo một số nguyên trong khoảng 1 - 100 tương

ứng với số phần tử trong stack tạm, và push vào stack đó các giá trị ngẫu nhiên cũng trong khoảng 1 - 100. Giới hạn của stack tạm là 100 phần tử. Sau cùng, stack hiện tại của chương trình sẽ sao chép stack tạm trên.



```

E:\source\repos\HK3\DSA_Ex x + v
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 5
Random stack size: 49
Max size: 100
Stack copied from a random stack.

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

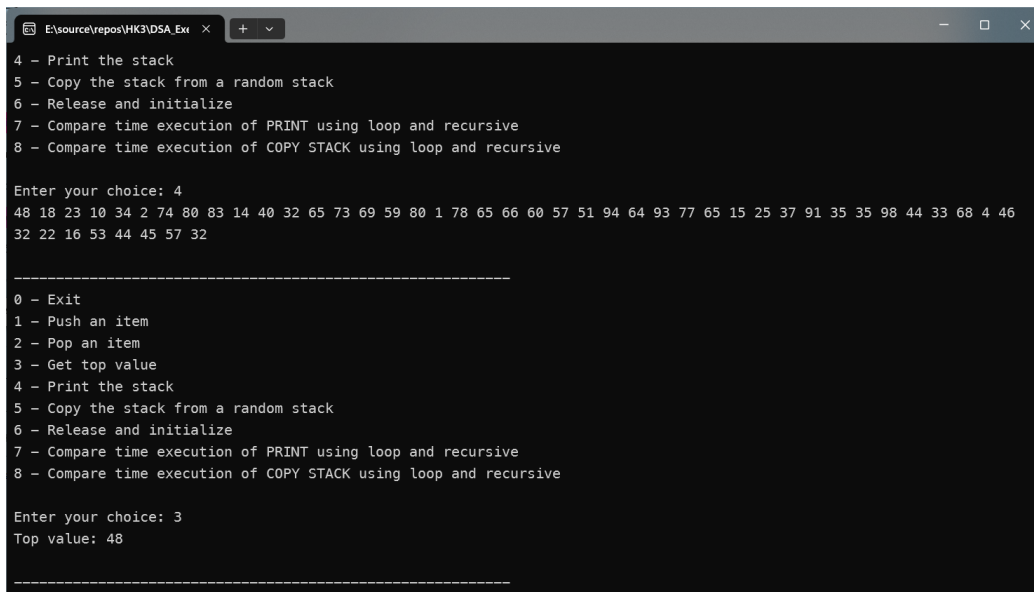
Enter your choice: 4
48 18 23 10 34 2 74 80 83 14 40 32 65 73 69 59 80 1 78 65 66 60 57 51 94 64 93 77 65 15 25 37 91 35 35 98 44 33 68 4 46
32 22 16 53 44 45 57 32

-----

```

Hình 4: [Stack Array] Copy stack từ một stack khác được tạo ngẫu nhiên và in ra

- Để kiểm tra việc sao chép có chính xác hay không, ta có thể kiểm tra lại giá trị **top** của stack sau khi sao chép.



```

E:\source\repos\HK3\DSA_Ex x + v
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 4
48 18 23 10 34 2 74 80 83 14 40 32 65 73 69 59 80 1 78 65 66 60 57 51 94 64 93 77 65 15 25 37 91 35 35 98 44 33 68 4 46
32 22 16 53 44 45 57 32

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 3
Top value: 48

-----

```

Hình 5: [Stack Array] Kiểm tra lại giá trị top của stack sau khi sao chép

- Lựa chọn 6 cho phép giải phóng stack hiện có và khởi tạo lại stack với kích thước mới.

```

E:\source\repos\HK3\DSA_Ext x + v
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 6
Enter the new size of the stack: 0
Stack initialized with size 0.

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 2
Warning: Stack is empty!
-----
    
```

Hình 6: [Stack Array] Giải phóng và khởi động lại một stack với kích thước mới và kiểm thử

### 3.3 Trường hợp stack đã đầy

Ở ví dụ sau, ban đầu ta khởi tạo stack với giá trị `maxSize` là 3. Sau khi in, ta thấy stack hiện có đủ 3 phần tử, do đó khi tiến hành thao tác push tiếp theo, chương trình sẽ thông báo stack đã đầy và không thể thực hiện thao tác này.

```

E:\source\repos\HK3\DSA_Ext x + v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 4
3 2 1

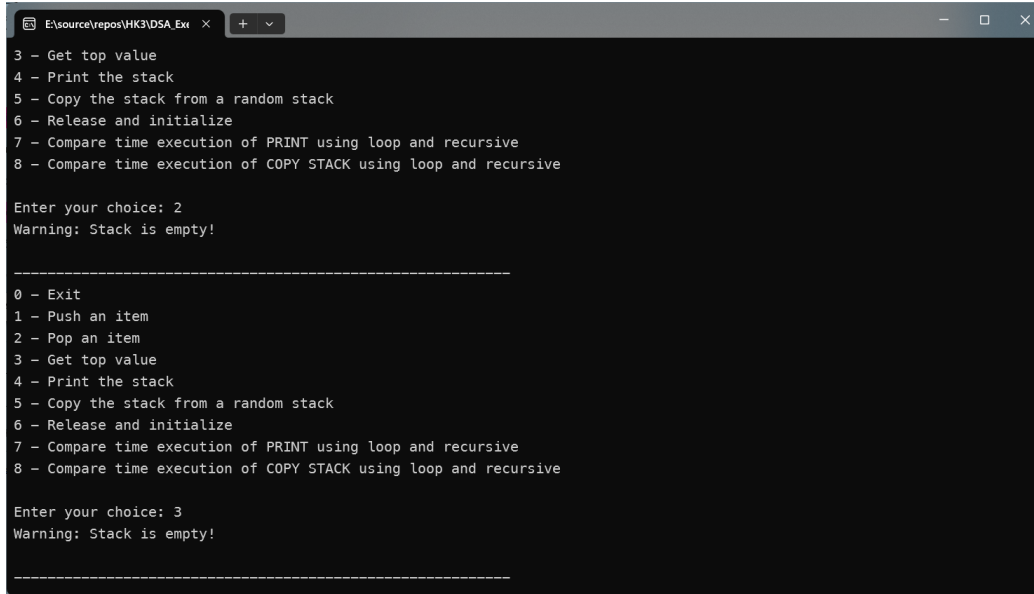
-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 1
Warning: Stack is full!
-----
    
```

Hình 7: [Stack Array] Không thể push vào stack đã đầy

### 3.4 Trường hợp stack rỗng

- Khi stack rỗng, tức là trong stack không có bất kì phần tử nào, việc pop và lấy giá trị đầu stack là không hợp lệ và chương trình sẽ thông báo rằng stack đang rỗng.



```

E:\source\repos\HK3\DSA_Ext x + v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 2
Warning: Stack is empty!

-----

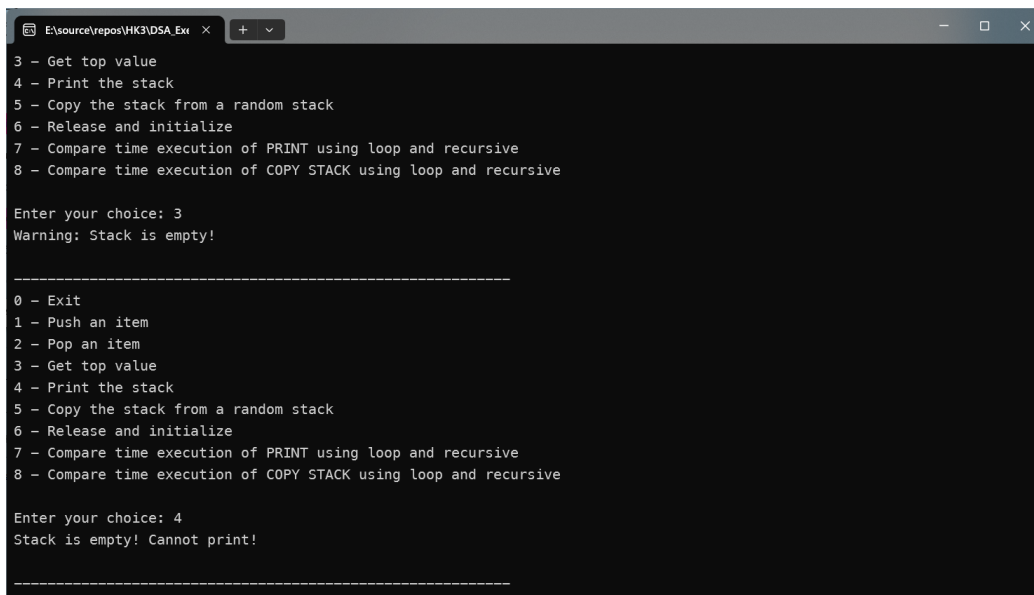
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 3
Warning: Stack is empty!

-----
    
```

Hình 8: [Stack Array] Không thể pop và lấy giá trị top từ một stack rỗng

- Tương tự, việc in ra một stack rỗng cũng không có ý nghĩa, do đó chương trình cũng sẽ hiển thị thông báo.



```

E:\source\repos\HK3\DSA_Ext x + v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 3
Warning: Stack is empty!

-----

0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 4
Stack is empty! Cannot print!

-----
    
```

Hình 9: [Stack Array] Khi in một stack rỗng sẽ có cảnh báo

### 3.5 Kết thúc chương trình

Sau khi đã thực hiện các thao tác và muốn dừng chương trình, người dùng có thể đơn giản nhập số 0 để kết thúc.

```

Microsoft Visual Studio Debug
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 4
Stack is empty! Cannot print!

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 0

E:\source\repos\HK3\DSA_Exercise_1\Debug\Stack Array.exe (process 24536) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
    
```

Hình 10: [Stack Array] Nhập số 0 để kết thúc chương trình

## 4 Ngăn xếp (phiên bản Danh sách liên kết) - Stack (Linked List version)

### 4.1 Tổng quan về cấu trúc dữ liệu và chương trình

Trong phần này, ta tiến hành tạo một chương trình để kiểm thử các thao tác đã được triển khai trên stack (pop và push). Chương trình được kiểm thử với các trường hợp bình thường, khi stack rỗng và khi stack đã đầy.

Cấu trúc Stack được cài đặt bằng kiểu dữ liệu cấu trúc **struct** kết hợp với **template** (khuôn mẫu) là một từ khóa trong C++ để tổng quát hóa cho nhiều kiểu dữ liệu khác nhau, giúp mã nguồn trở nên linh hoạt và có thể tái sử dụng. Ở đây, Stack được cài đặt bằng **danh sách liên kết** có cấu trúc như sau:

- **Biến (variables):**

- **top**: một con trỏ trỏ đến node trên cùng của stack. Ở đây, node cũng có kiểu dữ liệu cấu trúc với hai biến cơ bản là dữ liệu và một con trỏ trỏ đến node tiếp theo.
- **maxSize**: kích thước tối đa của stack, là một số nguyên không âm.
- **count**: đếm số phần tử hiện có trong stack, là một số nguyên không âm.

- **Hàm (functions)**: tương tự như hàm của struct Stack tại 3.1 (trừ hàm `isFull`).

Khai báo Stack được đặt trong tệp tin `stack.h`, cài đặt Stack được đặt trong tệp tin `stack.ipp`. Ngoài ra còn có hai tệp tương tự cho cấu trúc Stack sử dụng các hàm đệ quy thay thế cho vòng lặp là `recursive_stack.h` và `recursive_stack.ipp`.

Dưới đây là mã nguồn (source code) của Stack:

```

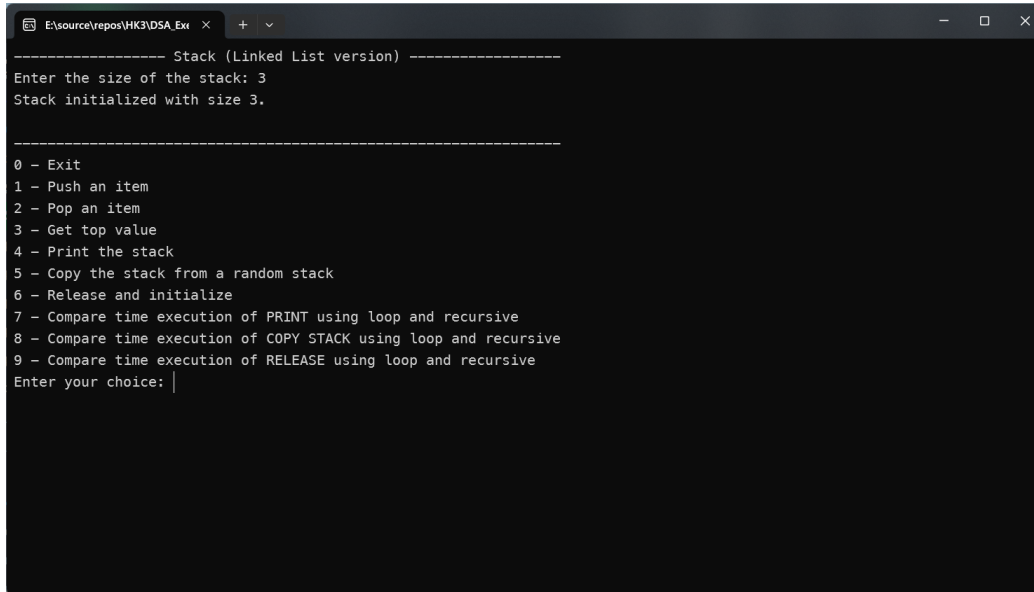
1  template <typename T>
2  struct Node {
3      T data;
4      Node* next;
5  };
6
7  template <typename T>
8  struct Stack {
9      Node<T>* top;
10     unsigned int count;
11     unsigned int maxSize;
12
13     void init(unsigned int size);
14     void copyStack(const Stack<T>& s);
15     void release();
16     bool isEmpty();
17     void push(T newItem);
18     T pop();
19     T topValue();
20     void print();
21 };

```

Chương trình được xây dựng tương tự như tại phần 3.1. Dưới đây là giao diện cơ bản của chương



trình:



```

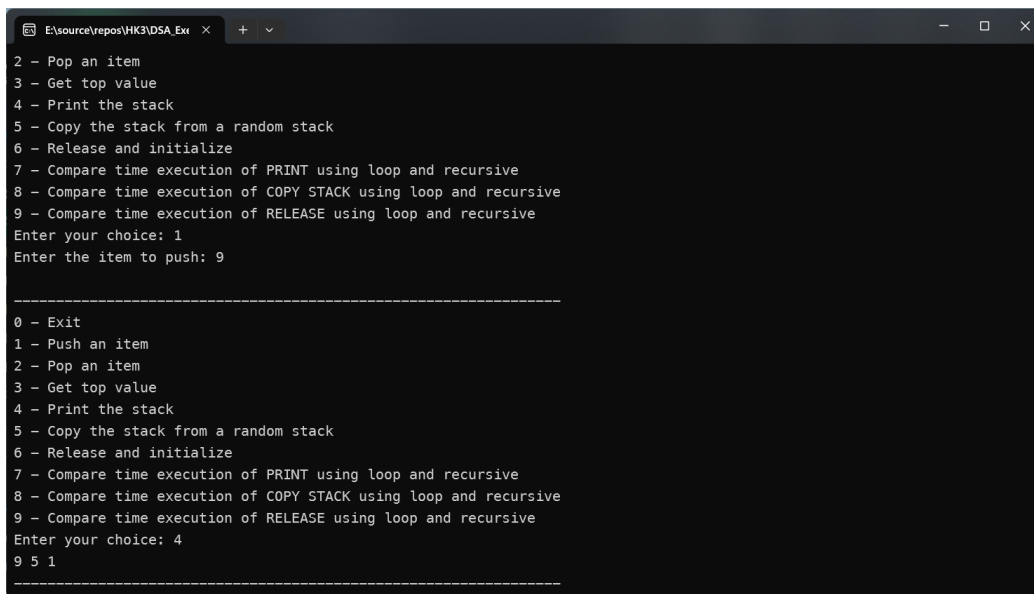
E:\source\repos\HK3\DSA_Ext x + v
----- Stack (Linked List version) -----
Enter the size of the stack: 3
Stack initialized with size 3.

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: |
    
```

Hình 11: [Stack Linked List] Giao diện console chương trình

## 4.2 Trường hợp bình thường

Các thao tác tương tự như 3.2. Sau đây là các kết quả từ chương trình:



```

E:\source\repos\HK3\DSA_Ext x + v
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 1
Enter the item to push: 9

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 4
9 5 1
-----
    
```

Hình 12: [Stack Linked List] Push và in các phần tử trong stack

```

E:\source\repos\HK3\DSA_Ex x + v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 2
Popped item: 9

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 3
Top value: 5

-----

```

Hình 13: [Stack Linked List] Pop và lấy giá trị top trong stack

```

E:\source\repos\HK3\DSA_Ex x + v
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 5
Random stack size: 73
Max size: 100
Stack copied from a random stack.

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 4
29 34 48 19 8 19 47 29 87 15 41 49 74 17 92 54 91 52 64 14 69 49 48 31 49 3 54 14 66 55 14 37 78 22 53 10 41 16 9 48 29
25 7 2 62 37 20 9 43 60 46 19 61 93 75 65 96 88 66 21 79 36 80 13 83 13 81 54 91 29 96 72 27

-----

```

Hình 14: [Stack Linked List] Copy stack từ một stack khác được tạo ngẫu nhiên và in ra

```

E:\source\repos\HK3\DSA_Ex  X  +  v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 4
29 34 48 19 8 19 47 29 87 15 41 49 74 17 92 54 91 52 64 14 69 49 48 31 49 3 54 14 66 55 14 37 78 22 53 10 41 16 9 48 29
25 7 2 62 37 20 9 43 60 46 19 61 93 75 65 96 88 66 21 79 36 80 13 83 13 81 54 91 29 96 72 27
-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 3
Top value: 29
-----

```

Hình 15: [Stack Linked List] Kiểm tra lại giá trị top của stack sau khi sao chép

```

E:\source\repos\HK3\DSA_Ex  X  +  v
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 6
Enter the new size of the stack: 0
Stack initialized with size 0.
-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 2
Stack is empty!
-----

```

Hình 16: [Stack Linked List] Giải phóng và khởi động lại một stack với kích thước mới và kiểm thử

### 4.3 Trường hợp stack đã đầy

Các thao tác tương tự như 3.3. Sau đây là kết quả từ chương trình:

```

E:\source\repos\HK3\DSA_Ex  X  +  v
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 4
7 5 2
-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 1
Stack is full!
-----

```

Hình 17: [Stack Linked List] Không thể push vào stack đã đầy

## 4.4 Trường hợp stack rỗng

Các thao tác tương tự như 3.4. Sau đây là các kết quả từ chương trình:

```

E:\source\repos\HK3\DSA_Ex  X  +  v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 2
Stack is empty!
-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 3
Stack is empty!
-----

```

Hình 18: [Stack Linked List] Không thể pop và lấy giá trị top từ một stack rỗng

```

E:\source\repos\HK3\DSA_Exi  X + v
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 3
Stack is empty!

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 4
Stack is empty! Cannot print!

-----

```

Hình 19: [Stack Linked List] Khi in một stack rỗng sẽ có cảnh báo

## 4.5 Kết thúc chương trình

Sau khi đã thực hiện các thao tác và muốn dừng chương trình, người dùng có thể đơn giản nhập số 0 để kết thúc.

```

Microsoft Visual Studio Debug  X + v
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 4
Stack is empty! Cannot print!

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive
9 - Compare time execution of RELEASE using loop and recursive
Enter your choice: 0
Exiting...

E:\source\repos\HK3\DSA_Exercise_1\64\Debug\Stack Linked List.exe (process 25124) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

Hình 20: [Stack Linked List] Nhập số 0 để kết thúc chương trình

## 5 Hàng đợi (phiên bản Mảng) - Queue (Array version)

### 5.1 Tổng quan về cấu trúc dữ liệu và chương trình

Trong phần này, ta tiến hành tạo một chương trình để kiểm thử các thao tác (**enqueue** và **dequeue**) đã được triển khai trên hàng đợi hay Queue (kể từ đây **queue** và hàng đợi sẽ được sử dụng để thay thế nhau với cùng ý nghĩa) . Chương trình được kiểm thử với các trường hợp bình thường, khi hàng đợi rỗng và khi hàng đợi đã đầy.

Cấu trúc Queue được cài đặt bằng kiểu dữ liệu cấu trúc **struct** kết hợp với **template** (khuôn mẫu) là một từ khóa trong C++ để tổng quát hóa cho nhiều kiểu dữ liệu khác nhau, giúp mã nguồn trở nên linh hoạt và có thể tái sử dụng. Ở đây, Queue được cài đặt bằng **mảng** có cấu trúc như sau:

- **Biến (variables):**

- **items:** mảng lưu trữ dữ liệu.
- **front:** chỉ số của phần tử đầu hàng đợi (có giá trị -1 nếu queue rỗng).
- **rear:** chỉ số của phần tử cuối hàng đợi (có giá trị -1 nếu queue rỗng).
- **count:** cập nhật số phần tử trong hàng đợi, là một số nguyên không âm.
- **maxSize:** kích thước tối đa của hàng đợi, là một số nguyên không âm.

- **Hàm (functions):**

- **init:** khởi tạo một queue rỗng với kích thước đã cho.
- **copyQueue:** khởi tạo một queue từ một queue khác.
- **release:** dọn dẹp queue khi không còn sử dụng.
- **isEmpty:** kiểm tra queue rỗng hay không.
- **enqueue:** toán tử thêm phần tử mới vào queue, cụ thể là thêm vào cuối của mảng **items** nếu chưa vượt quá **maxSize** (chức năng như **push** trong queue).

- **dequeue**: xóa phần tử ở đầu queue, tức là ở đầu mảng **items**, nếu queue rỗng thì in thông báo ra màn hình (chức năng như **pop** trong queue).
- **frontValue**: lấy giá trị đầu queue, nếu queue rỗng thì in thông báo ra màn hình.
- **print**: in các giá trị trong queue ra màn hình (từ đầu tới cuối queue).

Khai báo Queue được đặt trong tệp tin **queue.h**, cài đặt Queue được đặt trong tệp tin **queue.ipp**. Ngoài ra còn có hai tệp tương tự cho cấu trúc Queue sử dụng các hàm đệ quy thay thế cho vòng lặp là **recursive\_queue.h** và **recursive\_queue.ipp**.

Dưới đây là mã nguồn (source code) của Queue:

```

1  template <typename T>
2  struct Queue {
3      T* items;
4      int front;
5      int rear;
6      unsigned int count;
7      unsigned int maxSize;
8
9      void init(unsigned int queueSize);
10     void copyQueue(const Queue<T>& q);
11     void release();
12
13     bool isEmpty();
14     void enqueue(T newItem);
15     T dequeue();
16     T frontValue();
17
18     void print();
19 };

```

Chương trình được xây dựng trong **main.cpp** có giao diện với tiêu đề trên cùng là tên chương trình. Trước tiên, người dùng được yêu cầu nhập kích thước tối đa cho queue. Sau khi nhập một số nguyên không âm hợp lệ, màn hình hiển thị các thao tác trên queue có thể thực hiện ứng với các phương án số để người dùng có thể lựa chọn. Một số thao tác cơ bản trong chương trình là: enqueue, dequeue,

lấy giá trị front, in ra màn hình, copy queue, ...

Dưới đây là giao diện cơ bản của chương trình:

```

E:\source\repos\HK3\DSA_Ext
----- Queue (Array version) -----
Enter the size of the Queue: 5
Queue initialized with size 5.

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: |
    
```

Hình 21: [Queue Array] Giao diện console chương trình

## 5.2 Trường hợp bình thường

- Trong trường hợp bình thường (hàng đợi không rỗng cũng chưa đầy), ta có thể tiến hành enqueue các phần tử vào queue theo nguyên lý FIFO (First In, First Out) như một hàng đợi, cụ thể ở đây ta thêm các phần tử vào cuối mảng đồng thời cập nhật biến **front** là 0 để chỉ vào đầu mảng (phần tử đầu queue) và cập nhật biến **rear** sau mỗi lần thêm phần tử. Lựa chọn 1 trong chương trình cho phép thực hiện thao tác này.
- Khi queue đã có phần tử, ta có thể tiến hành in các phần tử trong queue bằng cách chọn lựa chọn 4. Minh họa như hình dưới đây:



```

E:\source\repos\HK3\DSA_Ex x + v - □ x
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 1
Enter the item to enqueue: 9

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 4
1 3 5 7 9
-----

```

Hình 22: [Queue Array] Enqueue và in các phần tử trong queue

- Khi queue đã có phần tử, ta có thể xóa phần tử ở đầu queue bằng thao tác **dequeue**, thực hiện bằng lựa chọn 2. Sau khi xóa, biến **rear** cũng sẽ được cập nhật, biến **front** đặt bằng -1 nếu sau khi lấy phần tử vừa rồi ra ngoài, hàng đợi trở thành rỗng.
- Lựa chọn 3 cho phép người dùng lấy ra giá trị ở đầu queue, cụ thể là phần tử trong mảng ở chỉ số **front**.

```

E:\source\repos\HK3\DSA_Ex x + v - □ x
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 2
Dequeued item: 1

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 3
Front value: 3
-----

```

Hình 23: [Queue Array] Dequeue và lấy giá trị front trong queue

- Khi chọn lựa chọn 5, chương trình tự động khởi tạo một số nguyên trong khoảng 1 - 100 tương ứng với số phần tử trong queue tạm, và enqueue vào queue đó các giá trị ngẫu nhiên cũng trong khoảng 1 - 100. Giới hạn của queue tạm là 100 phần tử. Sau cùng, queue hiện tại của chương trình sẽ sao chép queue tạm trên.

```

E:\source\repos\HK3\DSA_Ext x + v
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 5
Random queue size: 66
Max size: 100
Queue copied from a random queue.

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 4
23 40 73 94 17 86 75 33 70 4 59 17 71 44 28 43 64 81 12 72 68 69 72 43 72 32 39 24 10 92 83 68 80 74 99 46 90 98 88 41 8
9 97 70 91 12 70 55 50 46 71 69 73 33 86 23 5 53 92 22 0 13 51 21 42 7 42
-----

```

Hình 24: [Queue Array] Copy queue từ một queue khác được tạo ngẫu nhiên và in ra

- Để kiểm tra việc sao chép có chính xác hay không, ta có thể kiểm tra lại giá trị **front** của queue sau khi sao chép.

```

E:\source\repos\HK3\DSA_Ext x + v
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 4
23 40 73 94 17 86 75 33 70 4 59 17 71 44 28 43 64 81 12 72 68 69 72 43 72 32 39 24 10 92 83 68 80 74 99 46 90 98 88 41 8
9 97 70 91 12 70 55 50 46 71 69 73 33 86 23 5 53 92 22 0 13 51 21 42 7 42

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 3
Front value: 23
-----

```

Hình 25: [Queue Array] Kiểm tra lại giá trị front của queue sau khi sao chép

- Lựa chọn 6 cho phép giải phóng queue hiện có và khởi tạo lại queue với kích thước mới.

```

E:\source\repos\HK3\DSA_Ext x + v
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 6
Enter the new size of the Queue: 0
Queue initialized with size 0.

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 2
Warning: Queue is empty!
-----
    
```

Hình 26: [Queue Array] Giải phóng và khởi động lại một queue với kích thước mới và kiểm thử

### 5.3 Trường hợp queue đã đầy

Ở ví dụ sau, ban đầu ta khởi tạo queue với giá trị `maxSize` là 5. Sau khi in, ta thấy queue hiện có đủ 5 phần tử là: 1, 3, 5, 7, 9, do đó khi tiến hành thao tác enqueue tiếp theo, chương trình sẽ thông báo queue đã đầy và không thể thực hiện thao tác này.

```

E:\source\repos\HK3\DSA_Ext x + v
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 4
1 3 5 7 9

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 1
Warning: Queue is full!
-----
    
```

Hình 27: [Queue Array] Không thể enqueue vào queue đã đầy

## 5.4 Trường hợp queue rỗng

- Khi queue rỗng, tức là trong queue không có bất kì phần tử nào, việc pop và lấy giá trị đầu queue là không hợp lệ và chương trình sẽ thông báo rằng queue đang rỗng.

```

E:\source\repos\HK3\DSA_Ext x + v
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 2
Warning: Queue is empty!

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 3
Warning: Queue is empty!

-----

```

Hình 28: [Queue Array] Không thể dequeue và lấy giá trị front từ một queue rỗng

- Tương tự, việc in ra một queue rỗng cũng không có ý nghĩa, do đó chương trình cũng sẽ hiển thị thông báo.

```

E:\source\repos\HK3\DSA_Ext x + v
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 3
Warning: Queue is empty!

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 4
Queue is empty! Cannot print!

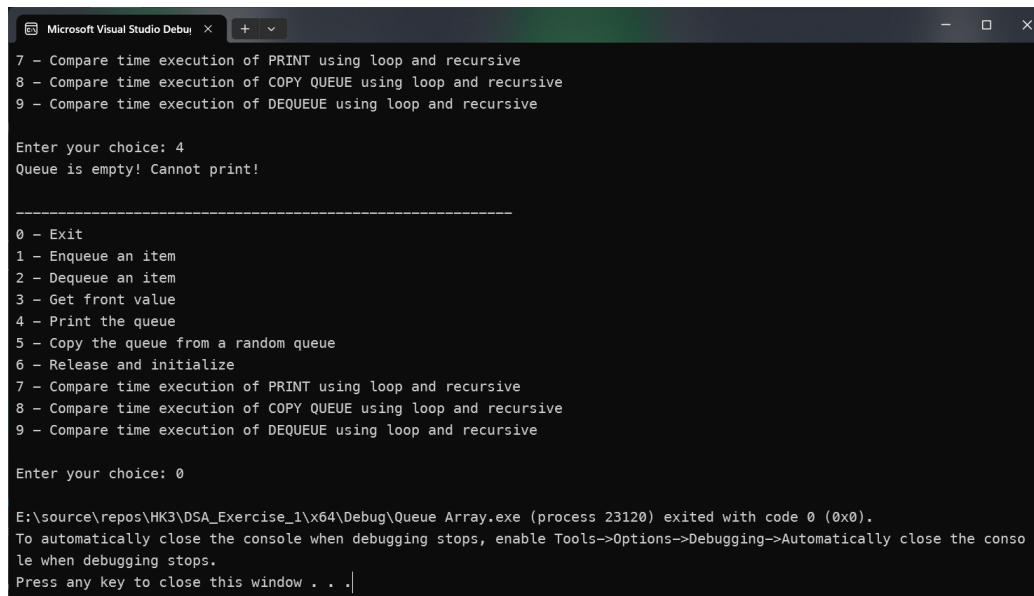
-----

```

Hình 29: [Queue Array] Khi in một queue rỗng sẽ có cảnh báo

## 5.5 Kết thúc chương trình

Sau khi đã thực hiện các thao tác và muốn dừng chương trình, người dùng có thể đơn giản nhập số 0 để kết thúc.



```

Microsoft Visual Studio Debug
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 4
Queue is empty! Cannot print!

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY QUEUE using loop and recursive
9 - Compare time execution of DEQUEUE using loop and recursive

Enter your choice: 0

E:\source\repos\HK3\DSA_Exercise_1\Debug\Queue Array.exe (process 23120) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
    
```

Hình 30: [Queue Array] Nhập số 0 để kết thúc chương trình

## 6 Hàng đợi (phiên bản Danh sách liên kết) - Queue (Linked List version)

### 6.1 Tổng quan về cấu trúc dữ liệu và chương trình

Tương tự ở phần 5.1, điểm khác biệt là ở phần này, chúng ta sẽ cài đặt cấu trúc Queue bằng **Danh sách liên kết** thay cho **mảng**.

Trước hết, tạo một **struct** Node có kiểu dữ liệu cấu trúc với hai biến cơ bản là dữ liệu và một con trỏ trỏ đến node tiếp theo.

Cấu trúc Queue có đặc điểm như sau:

- **Biến (variables):**
  - **front:** con trỏ trỏ đến phần tử đầu hàng đợi.

- **rear**: con trỏ trỏ đến phần tử cuối hàng đợi.
- **count**: cập nhật số phần tử trong hàng đợi.
- **maxSize**: kích thước tối đa của hàng đợi, là một số nguyên không âm.

- **Hàm (functions)**: hoàn toàn tương tự các hàm ở **struct Queue** tại [5.1](#).

Khai báo Queue được đặt trong tệp tin `queue.h`, cài đặt Queue được đặt trong tệp tin `queue.ipp`. Ngoài ra còn có hai tệp tương tự cho cấu trúc Queue sử dụng các hàm đệ quy thay thế cho vòng lặp là `recursive_queue.h` và `recursive_queue.ipp`.

Dưới đây là mã nguồn (source code) của Queue:

```

1  template <typename T>
2  struct Node {
3      T data;
4      Node* next;
5  };
6
7  template <typename T>
8  struct Queue {
9      Node<T>* front;
10     Node<T>* rear;
11     unsigned int count;
12     unsigned int maxSize;
13
14     void init(unsigned int size);
15     void copyQueue(const Queue<T>& q);
16     void release();
17     bool isEmpty();
18     void enqueue(T newItem);
19     T dequeue();
20     T frontValue();
21     void print();
22 };

```

Chương trình được xây dựng tương tự như phần [5.1](#). Dưới đây là giao diện cơ bản của chương trình:

```

E:\source\repos\HK3\DSA_Ex  X  +  v
----- Queue (Linked List version) -----
Enter the size of the Queue: 4
Queue initialized with size 4.

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: |

```

Hình 31: [Queue Linked List] Giao diện console chương trình

## 6.2 Các thao tác trong trường hợp bình thường

Các thao tác tương tự như 5.2. Sau đây là các kết quả từ chương trình:

```

E:\source\repos\HK3\DSA_Ex  X  +  v
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 1
Enter the item to enqueue: 8

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 4
2 4 6 8
-----

```

Hình 32: [Queue Linked List] Enqueue và in các phần tử trong queue

```

E:\source\repos\HK3\DSA_Ex x + v
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 2
Dequeued item: 2

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 3
Front value: 4

-----

```

Hình 33: [Queue Linked List] Dequeue và lấy giá trị front trong queue

```

E:\source\repos\HK3\DSA_Ex x + v
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 5
Random queue size: 8
Max size: 100
Queue copied from a random queue.

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 4
79 75 89 74 49 84 11 12

-----

```

Hình 34: [Queue Linked List] Copy queue từ một queue khác được tạo ngẫu nhiên và in ra



```

E:\source\repos\HK3\DSA_Ex x + v
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 4
79 75 89 74 49 84 11 12

-----

0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 3
Front value: 79

-----

```

Hình 35: [Queue Linked List] Kiểm tra lại giá trị front của queue sau khi sao chép

```

E:\source\repos\HK3\DSA_Ex x + v
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 6
Enter the new size of the Queue: 0
Queue initialized with size 0.

-----

0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 2
Warning: Queue is empty!

-----

```

Hình 36: [Queue Linked List] Giải phóng và khởi động lại một queue với kích thước mới và kiểm thử

## 6.3 Trường hợp queue đã đầy

Các thao tác tương tự như 5.3. Sau đây là kết quả từ chương trình:

```

E:\source\repos\HK3\DSA_Ex x + v
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 4
2 4 6 8
-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 1
Warning: Queue is full!
-----

```

Hình 37: [Queue Linked List] Không thể enqueue vào queue đã đầy

## 6.4 Trường hợp queue rỗng

Các thao tác tương tự như 5.4. Sau đây là các kết quả từ chương trình:

```

E:\source\repos\HK3\DSA_Ex x + v
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 2
Warning: Queue is empty!
-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 3
Warning: Queue is empty!
-----

```

Hình 38: [Queue Linked List] Không thể dequeue và lấy giá trị front từ một queue rỗng

```

E:\source\repos\HK3\DSA_Exi X + v
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 3
Warning: Queue is empty!

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 4
Queue is empty! Cannot print!

-----

```

Hình 39: [Queue Linked List] Khi in một queue rỗng sẽ có cảnh báo

## 6.5 Kết thúc chương trình

Sau khi đã thực hiện các thao tác và muốn dừng chương trình, người dùng có thể đơn giản nhập số 0 để kết thúc.

```

Microsoft Visual Studio Debu X + v
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 4
Queue is empty! Cannot print!

-----
0 - Exit
1 - Enqueue an item
2 - Dequeue an item
3 - Get front value
4 - Print the queue
5 - Copy the queue from a random queue
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of RELEASE using loop and recursive
9 - Compare time execution of COPY QUEUE using loop and recursive

Enter your choice: 0

E:\source\repos\HK3\DSA_Exercise_1\x64\Debug\Queue Linked List.exe (process 19872) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

Hình 40: [Queue Linked List] Nhập số 0 để kết thúc chương trình

## 7 Các phiên bản đệ quy - Recursive versions

### 7.1 Stack (Array version)

#### 7.1.1 Copy Stack

Phiên bản vòng lặp:

- Mã nguồn:

```

1 template <typename T>
2 void Stack<T>::copyStack(const Stack<T>& s) {
3     if (s.top == -1) return;
4     init(s.maxSize);
5     top = s.top;
6     for (int i = 0; i <= top; i++) {
7         items[i] = s.items[i];
8     }
9 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp for duyệt qua mỗi phần tử một lần, n phần tử tương ứng với n lần, n vòng lặp. Mỗi vòng lặp thực hiện hữu hạn phép so sánh và gán với độ phức tạp  $O(1)$ . Các thao tác ngoài vòng lặp trong hàm cũng chỉ có độ phức tạp  $O(1)$ .

Phiên bản đệ quy

- Mã nguồn:

```

1 template <typename T>
2 void Stack<T>::copyStackRecursive(const Stack<T>& s) {
3     if (s.top == -1) return;
4     init(s.maxSize);
5     copyStackHelper(s, s.top);
6 }
7
8 template <typename T>
9 void Stack<T>::copyStackHelper(const Stack<T>& s, int index) {
10     if (index == -1) return;

```

```

11     copyStackHelper(s, index - 1);
12     push(s.items[index]);
13 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Mỗi lần gọi hàm đệ quy sẽ xử lý trên một phần tử của stack dựa vào chỉ số index, chỉ số đầu tiên là **top** và gọi cho đến trường hợp cơ sở là **index = -1** , do đó có  $n + 1$  lần gọi đệ quy với  $n$  là số phần tử hiện có trong stack và trong mỗi hàm đệ quy có các thao tác hữu hạn với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một stack chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 8

Time execution for COPY STACK using loop: 19 microseconds
Time execution for COPY STACK using recursive: 452 microseconds
-----

```

Hình 41: [Stack Array] So sánh thời gian thực thi COPY STACK (lần 1)

```

-----
0 - Exit
1 - Push an item
2 - Pop an item
3 - Get top value
4 - Print the stack
5 - Copy the stack from a random stack
6 - Release and initialize
7 - Compare time execution of PRINT using loop and recursive
8 - Compare time execution of COPY STACK using loop and recursive

Enter your choice: 8

Time execution for COPY STACK using loop: 9 microseconds
Time execution for COPY STACK using recursive: 124 microseconds
-----

```

Hình 42: [Stack Array] So sánh thời gian thực thi COPY STACK (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (tính bằng microseconds). Tuy nhiên, so sánh giữa hai hàm thì hàm COPY STACK sử dụng vòng lặp có thời gian nhanh hơn ở cả hai lần.

### 7.1.2 Print

**Phiên bản vòng lặp:**

- Mã nguồn:

```

1 template <typename T>
2 void Stack<T>::print() {
3     if (isEmpty()) {
4         cout << "Stack is empty! Cannot print!" << endl;
5         return;
6     }
7     for (int i = top; i >= 0; i--) {
8         cout << items[i] << " ";
9     }
10    cout << endl;
11 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp for duyệt qua mỗi phần tử một lần,  $n$  phần tử tương ứng với  $n$  lần. Mỗi vòng lặp thực hiện in ra màn hình với độ phức tạp  $O(1)$ .

### Phiên bản đệ quy

- Mã nguồn:

```

1  template <typename T>
2  void RecursiveStack<T>::printRecursive() {
3      if (isEmpty()) {
4          cout << "Stack is empty! Cannot print!" << endl;
5          return;
6      }
7      printHelper(top);
8  }
9
10 template <typename T>
11 void RecursiveStack<T>::printHelper(int index) {
12     if (index == -1) return;
13     cout << items[index] << " ";
14     printHelper(index - 1);
15 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Mỗi lần gọi hàm đệ quy sẽ xử lý trên một phần tử của stack dựa vào chỉ số index, chỉ số đầu tiên là top và gọi cho đến trường hợp cơ sở là  $index = -1$ , do đó có  $n + 1$  lần gọi đệ quy và trong mỗi hàm đệ quy có các thao tác hữu hạn với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một stack chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 8
6 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 4
6 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
4 3 2 1 0
Time execution for PRINT using loop: 113611 microseconds
Time execution for PRINT using recursive: 113464 microseconds

```

Hình 43: [Stack Array] So sánh thời gian thực thi COPY STACK (lần 1)

```

149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133 132 131 130 129 128 127 126 125 124 123 122 121 120
119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 8
6 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 4
6 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
4 3 2 1 0
Time execution for PRINT using loop: 122013 microseconds
Time execution for PRINT using recursive: 120768 microseconds
=====

```

Hình 44: [Stack Array] So sánh thời gian thực thi COPY STACK (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm khá nhanh (hơn 100 miliseconds) và tương đồng nhau, có thể chênh lệch nhưng không đáng kể.

## 7.2 Stack (Linked List version)

### 7.2.1 Copy Stack

Phiên bản vòng lặp:

- Mã nguồn:

```

1  template <typename T>
2  void Stack<T>::copyStack(const Stack<T>& s) {
3      if (s.top == NULL) return;
4      init(s.maxSize);
5      Node<T>* temp = s.top;
6      while (temp != NULL) {
7          push(temp->data);
8          temp = temp->next;
9      }
10 }
11
12 template <typename T>
13 void Stack<T>::push(T newItem) {
14     if (count == maxSize) {
15         cout << "Stack is full!\n";
16         return;
17     }
18     Node<T>* newNode = new Node<T>;
19     newNode->data = newItem;

```



```

20     newNode->next = top;
21     top = newNode;
22     count++;
23 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp while duyệt qua  $n$  phần tử trong stack từ node `top` cho đến khi gặp node `NULL` ứng với  $n$  vòng lặp, với mỗi thao tác trong vòng lặp như so sánh, gán, `push` có độ phức tạp  $O(1)$ .

### Phiên bản đệ quy

- Mã nguồn:

```

1  template <typename T>
2  void RecursiveStack<T>::copyStackRecursive(const RecursiveStack<T>& s) {
3      init(s.maxSize);
4      copyStackHelper(top, s.top);
5  }
6
7  template <typename T>
8  void RecursiveStack<T>::copyStackHelper(Node<T>*& top, Node<T>* sTop) {
9      if (sTop == NULL) {
10         top = NULL;
11         return;
12     }
13     top = new Node<T>;
14     top->data = sTop->data;
15     count++;
16     copyStackHelper(top->next, sTop->next);
17 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ `top` xuống dưới đáy stack và đến trường hợp cơ bản là tới node `NULL`, với  $n$  là số phần tử trong stack. Trong mỗi hàm đệ quy thực hiện các thao tác so sánh, gán với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một stack chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```
Time execution for COPY STACK using loop: 366 microseconds
Time execution for COPY STACK using recursive: 752 microseconds
-----
```

Hình 45: [Stack Linked List] So sánh thời gian thực thi COPY STACK (lần 1)

```
Time execution for COPY STACK using loop: 350 microseconds
Time execution for COPY STACK using recursive: 287 microseconds
-----
```

Hình 46: [Stack Linked List] So sánh thời gian thực thi COPY STACK (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (vài trăm microseconds) và tương đồng nhau, không cố định hàm nào nhanh hơn, như thể hiện ở hai hình trên.

### 7.2.2 Release

**Phiên bản vòng lặp:**

- Mã nguồn:

```
1 template <typename T>
2 void Stack<T>::release() {
3     while (top != NULL) {
4         Node<T>* temp = top;
5         top = top->next;
6         delete temp;
7     }
8     maxSize = 0;
9     count = 0;
10 }
```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp while duyệt qua n phần tử trong stack cho đến khi gặp node NULL ứng với n vòng

lặp, với các thao tác trong vòng lặp như gán và xóa có độ phức tạp  $O(1)$ . Các phép gán trong hàm ngoài vòng lặp cũng chỉ có độ phức tạp  $O(1)$ .

### Phiên bản đệ quy

- Mã nguồn:

```

1 template <typename T>
2 void RecursiveStack<T>::releaseRecursive() {
3     releaseHelper(top);
4     maxSize = 0;
5 }
6
7 template <typename T>
8 void RecursiveStack<T>::releaseHelper(Node<T>*& top) {
9     if (top == NULL) return;
10    releaseHelper(top->next);
11    delete top;
12    count--;
13    top = NULL;
14 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ **top** xuống dưới đáy stack cho đến trường hợp cơ bản là tối node NULL, với  $n$  là số phần tử có trong stack. Trong mỗi hàm đệ quy thực hiện các thao tác so sánh, xóa, trừ, gán với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một stack chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

Time execution for RELEASE using loop: 367 microseconds
Time execution for RELEASE using recursive: 467 microseconds

```

Hình 47: [Stack Linked List] So sánh thời gian thực thi RELEASE (lần 1)

```
Time execution for RELEASE using loop: 151 microseconds
Time execution for RELEASE using recursive: 274 microseconds
```

Hình 48: [Stack Linked List] So sánh thời gian thực thi RELEASE (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm khá nhỏ (khoảng vài trăm microseconds). Theo hai mẫu trên thì hàm RELEASE dùng đệ quy có chút lợi thế, nhưng không đáng kể trong thang thời gian microseconds.

### 7.2.3 Print

**Phiên bản vòng lặp:**

- Mã nguồn:

```
1 template <typename T>
2 void Stack<T>::print() {
3     if (isEmpty()) {
4         cout << "Stack is empty! Cannot print!\n";
5         return;
6     }
7     Node<T>* temp = top;
8     while (temp != NULL) {
9         cout << temp->data << " ";
10        temp = temp->next;
11    }
12 }
```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp while duyệt qua  $n$  phần tử trong stack ứng với  $n$  vòng lặp, với mỗi thao tác trong vòng lặp như in, gán có độ phức tạp  $O(1)$ . Các thao tác bên ngoài vòng lặp như so sánh và gán cũng có độ phức tạp  $O(1)$ .

**Phiên bản đệ quy**

- Mã nguồn:

```

1 void RecursiveStack<T>::printRecursive() {
2     if (isEmpty()) {
3         cout << "Stack is empty! Cannot print!" << endl;
4         return;
5     }
6     printHelper(top);
7 }
8
9 template <typename T>
10 void RecursiveStack<T>::printHelper(Node<T>* top) {
11     if (top == NULL) return;
12     cout << top->data << " ";
13     printHelper(top->next);
14 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ `top` xuống dưới đáy stack tới trường hợp cơ bản là gặp node `NULL`, với  $n$  là số phần tử trong stack. Trong mỗi hàm đệ quy thực hiện các thao tác như so sánh, in ra màn hình với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một stack chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50
49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
9 8 7 6 5 4 3 2 1 0
Time execution for PRINT using loop: 158358 microseconds
Time execution for PRINT using recursive: 119162 microseconds

```

Hình 49: [Stack Linked List] So sánh thời gian thực thi PRINT (lần 1)

```

89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50
49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
9 8 7 6 5 4 3 2 1 0
Time execution for PRINT using loop: 116044 microseconds
Time execution for PRINT using recursive: 150341 microseconds

```

Hình 50: [Stack Linked List] So sánh thời gian thực thi PRINT (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (khoảng hơn 100

milliseconds), không cố định hàm nào nhanh hơn hàm nào, khá tương đồng nhau.

## 7.3 Queue (Array version)

### 7.3.1 Copy Queue

#### Phiên bản vòng lặp

- Mã nguồn:

```

1 template <typename T>
2 void Queue<T>::copyQueue(const Queue<T>& q) {
3     if (q.count == 0) return;
4     init(q.maxSize);
5     front = q.front;
6     rear = q.rear;
7     count = q.count;
8     for (unsigned int i = 0; i < count; i++) {
9         items[i] = q.items[i];
10    }
11 }
```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp for duyệt qua mỗi phần tử một lần,  $n$  phần tử tương ứng với  $n$  lần,  $n$  vòng lặp. Mỗi vòng lặp thực hiện hữu hạn phép gán với độ phức tạp  $O(1)$ . Các thao tác ngoài vòng lặp cũng có độ phức tạp chỉ  $O(1)$ .

#### Phiên bản đệ quy

- Mã nguồn:

```

1 template <typename T>
2 void RecursiveQueue<T>::copyQueueRecursive(const RecursiveQueue<T>& q) {
3     if (q.count == 0) return;
4     init(q.maxSize);
5     copyQueueHelper(q, 0);
6 }
7
8 template <typename T>
```

```

9 void RecursiveQueue<T>::copyQueueHelper(const RecursiveQueue<T>& q, int
    index) {
10     if (index == q.count) return;
11     if (front == -1) front = 0;
12     items[index] = q.items[index];
13     count++;
14     rear++;
15     copyQueueHelper(q, index + 1);
16 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ `index = 0` xuống cuối hàng đợi khi `index = q.count`, với  $n$  là số phần tử trong hàng đợi. Trong mỗi hàm đệ quy thực hiện các thao tác so sánh, gán với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một queue chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

Time execution for COPY QUEUE using loop: 10 microseconds
Time execution for COPY QUEUE using recursive: 413 microseconds

```

Hình 51: [Queue Array] So sánh thời gian thực thi COPY QUEUE (lần 1)

```

Time execution for COPY QUEUE using loop: 6 microseconds
Time execution for COPY QUEUE using recursive: 66 microseconds

```

Hình 52: [Queue Array] So sánh thời gian thực thi COPY QUEUE (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm rất nhỏ (vài đến vài chục microseconds). Tuy nhiên, so sánh giữa hai hàm thì hàm COPY QUEUE sử dụng vòng lặp có thời gian tốt hơn tương đối ở cả hai lần.

### 7.3.2 Dequeue

#### Phiên bản vòng lặp

- Mã nguồn:

```

1 template <typename T>
2 T Queue<T>::dequeue() {
3     if (isEmpty()) {
4         cout << "Queue is empty\n";
5         return T();
6     }
7     T data = items[front];
8     for (unsigned int i = 0; i < count - 1; i++) {
9         items[i] = items[i + 1];
10    }
11    count--;
12    rear--;
13    if (count == 0) front = -1;
14    return data;
15 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp for duyệt qua mỗi phần tử một lần,  $n - 1$  phần tử tương ứng với  $n - 1$  lần (trừ phần tử cuối). Mỗi vòng lặp thực hiện hữu hạn gắn với độ phức tạp  $O(1)$ . Các thao tác ngoài vòng lặp cũng có độ phức tạp  $O(1)$ .

## Phiên bản đệ quy

- Mã nguồn:

```

1 template <typename T>
2 T RecursiveQueue<T>::dequeueRecursive() {
3     if (isEmpty()) {
4         cout << "Queue is empty\n";
5         return T();
6     }
7     T data = items[front];
8     dequeueHelper(0);
9     if (count == 0) front = -1;
10    return data;
11 }

```



```

12
13 template <typename T>
14 void RecursiveQueue<T>::dequeueHelper(int index) {
15     if (index == count - 1) {
16         count--;
17         rear--;
18         return;
19     }
20     items[index] = items[index + 1];
21     dequeueHelper(index + 1);
22 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n$  lần gọi hàm đệ quy từ `index = 0` xuống trước cuối hàng đợi, với  $n$  là số phần tử trong queue. Trong mỗi hàm đệ quy thực hiện các thao tác so sánh, gán với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một queue chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

Time execution for DEQUEUE using loop: 4623 microseconds
Time execution for DEQUEUE using recursive: 23315 microseconds

```

Hình 53: [Queue Array] So sánh thời gian thực thi DEQUEUE (lần 1)

```

Time execution for DEQUEUE using loop: 5526 microseconds
Time execution for DEQUEUE using recursive: 28726 microseconds

```

Hình 54: [Queue Array] So sánh thời gian thực thi DEQUEUE (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (khoảng vài nghìn đến chục nghìn microseconds). Tuy nhiên, so sánh giữa hai hàm thì hàm DEQUEUE sử dụng vòng lặp có thời gian nhanh hơn khá nhiều ở cả hai lần.

### 7.3.3 Print

#### Phiên bản vòng lặp

- Mã nguồn:

```

1 template <typename T>
2 void Queue<T>::print() {
3     if (isEmpty()) {
4         cout << "Queue is empty\n";
5         return;
6     }
7     for (unsigned int i = 0; i < count; i++) {
8         cout << items[i] << " ";
9     }
10 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp for duyệt qua mỗi phần tử một lần,  $n$  phần tử tương ứng với  $n$  lần duyệt. Mỗi vòng lặp thực hiện hữu hạn phép in ra màn hình với độ phức tạp  $O(1)$ .

## Phiên bản đệ quy

- Mã nguồn:

```

1 template <typename T>
2 void RecursiveQueue<T>::printRecursive() {
3     if (isEmpty()) {
4         cout << "Queue is empty! Cannot print!\n";
5         return;
6     }
7     printHelper(0);
8 }
9
10 template <typename T>
11 void RecursiveQueue<T>::printHelper(int index) {
12     if (index == count) return;
13     cout << items[index] << " ";
14     printHelper(index + 1);
15 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ `index = 0` xuống cuối hàng đợi khi `index = count`, với  $n$  là số phần tử trong queue. Trong mỗi hàm đệ quy thực hiện các thao tác so sánh, in ra màn hình với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một queue chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```
1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
1996 1997 1998 1999
Time execution for PRINT using loop: 114769 microseconds
Time execution for PRINT using recursive: 117345 microseconds
```

Hình 55: [Queue Array] So sánh thời gian thực thi PRINT (lần 1)

```
1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
1996 1997 1998 1999
Time execution for PRINT using loop: 128255 microseconds
Time execution for PRINT using recursive: 117790 microseconds
```

Hình 56: [Queue Array] So sánh thời gian thực thi PRINT (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (khoảng hơn 100 miliseconds), khá tương đồng nhau và khó xác định hàm nào nhanh hơn hàm nào.

## 7.4 Queue (Linked List version)

### 7.4.1 Release

#### Phiên bản vòng lặp

- Mã nguồn:

```
1 template <typename T>
2 void Queue<T>::release() {
3     while (front != NULL) {
4         Node<T>* temp = front;
5         front = front->next;
6         delete temp;
7     }
8     rear = NULL;
9     maxSize = 0;
```

```
10     count = 0;
11 }
```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp while duyệt qua  $n$  phần tử trong queue tương ứng  $n$  vòng lặp, với mỗi thao tác trong vòng lặp như gán, xóa có độ phức tạp  $O(1)$ . Các thao tác ngoài vòng lặp có độ phức tạp  $O(1)$ .

### Phiên bản đệ quy

- Mã nguồn:

```
1 template <typename T>
2 void RecursiveQueue<T>::releaseRecursive() {
3     releaseHelper(front);
4     rear = NULL;
5     maxSize = 0;
6     count = 0;
7 }
8
9 template <typename T>
10 void RecursiveQueue<T>::releaseHelper(Node<T>* node) {
11     if (node == NULL) return;
12     releaseHelper(node->next);
13     delete node;
14 }
```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ  $index = 0$  xuống cuối hàng đợi đến node NULL, với  $n$  là số phần tử trong queue. Trong mỗi hàm đệ quy thực hiện thao tác xóa với độ phức tạp  $O(1)$ .

Hữu hạn các thao tác gán cũng có độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một queue chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```
Time execution for RELEASE using loop: 488 microseconds
Time execution for RELEASE using recursive: 1006 microseconds
```

Hình 57: [Queue Linked List] So sánh thời gian thực thi RELEASE (lần 1)

```
Time execution for RELEASE using loop: 223 microseconds
Time execution for RELEASE using recursive: 352 microseconds
```

Hình 58: [Queue Linked List] So sánh thời gian thực thi RELEASE (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (khoảng vài trăm đến 1000 microseconds). Tuy nhiên, so sánh giữa hai hàm thì hàm RELEASE sử dụng vòng lặp có thời gian tốt hơn ở cả hai lần.

#### 7.4.2 Print

##### Phiên bản vòng lặp

- Mã nguồn:

```
1 template <typename T>
2 void Queue<T>::print() {
3     if (isEmpty()) {
4         cout << "Queue is empty! Cannot print!\n";
5         return;
6     }
7     Node<T>* temp = front;
8     while (temp != NULL) {
9         cout << temp->data << " ";
10        temp = temp->next;
11    }
12 }
```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp while duyệt qua  $n$  phần tử trong queue tương ứng với  $n$  vòng lặp, với mỗi thao tác trong vòng lặp như gán, in có độ phức tạp  $O(1)$ . Thao tác gán và kiểm tra queue rỗng cũng

chỉ có độ phức tạp  $O(1)$ .

### Phiên bản đệ quy

- Mã nguồn:

```

1 template <typename T>
2 void RecursiveQueue<T>::printRecursive() {
3     if (isEmpty()) {
4         cout << "Queue is empty! Cannot print!" << endl;
5         return;
6     }
7     printHelper(front);
8 }
9
10 template <typename T>
11 void RecursiveQueue<T>::printHelper(Node<T>* node) {
12     if (node == NULL) return;
13     cout << node->data << " ";
14     printHelper(node->next);
15 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ  $\text{index} = 0$  xuống cuối hàng đợi đến node NULL, với  $n$  là số phần tử trong queue. Trong mỗi hàm đệ quy thực hiện so sánh, in ra màn hình với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một queue chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:

```

1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
1996 1997 1998 1999
Time execution for PRINT using loop: 112202 microseconds
Time execution for PRINT using recursive: 115906 microseconds

```

Hình 59: [Queue Linked List] So sánh thời gian thực thi PRINT (lần 1)

```
1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
1996 1997 1998 1999
Time execution for PRINT using loop: 117665 microseconds
Time execution for PRINT using recursive: 152399 microseconds
```

Hình 60: [Queue Linked List] So sánh thời gian thực thi PRINT (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (khoảng hơn 100 miliseconds). Tuy nhiên, so sánh giữa hai hàm thì hàm PRINT sử dụng vòng lặp có thời gian nhanh tốt hơn ở cả hai lần.

### 7.4.3 Copy Queue

#### Phiên bản vòng lặp

- Mã nguồn:

```
1  template <typename T>
2  void Queue<T>::copyQueue(const Queue<T>& q) {
3      init(q.maxSize);
4      Node<T>* temp = q.front;
5      while (temp != NULL) {
6          enqueue(temp->data);
7          temp = temp->next;
8      }
9  }
10
11 template <typename T>
12 void Queue<T>::enqueue(T newItem) {
13     if (count == maxSize) {
14         cout << "Queue is full!\n";
15         return;
16     }
17     Node<T>* newNode = new Node<T>;
18     newNode->data = newItem;
19     newNode->next = NULL;
20     if (front == NULL) {
21         front = rear = newNode;
22     }
```

```

23     else rear->next = newNode;
24     rear = newNode;
25     count++;
26 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Vòng lặp while duyệt qua  $n$  phần tử trong queue tương ứng với  $n$  vòng lặp, với mỗi thao tác trong vòng lặp như so sánh, gán, **enqueue** có độ phức tạp  $O(1)$ . Hàm **enqueue** hay **init** có độ phức tạp  $O(1)$  do có một số hữu hạn các thao tác như gán, so sánh.

### Phiên bản đệ quy

- Mã nguồn:

```

1  template <typename T>
2  void RecursiveQueue<T>::copyQueueRecursive(const RecursiveQueue<T>& q) {
3      init(q.maxSize);
4      copyQueueHelper(q.front);
5  }
6
7  template <typename T>
8  void RecursiveQueue<T>::copyQueueHelper(Node<T>* node) {
9      if (node == NULL) return;
10     enqueue(node->data);
11     copyQueueHelper(node->next);
12 }

```

- Độ phức tạp thuật toán:  $O(n)$ .

Có  $n + 1$  lần gọi hàm đệ quy từ **index** = 0 xuống cuối hàng đợi đến node NULL, với  $n$  là số phần tử trong queue. Trong mỗi hàm đệ quy thực hiện các thao tác so sánh, **enqueue** với độ phức tạp  $O(1)$ .

### So sánh thời gian thực thi:

Thực hiện copy một queue chứa 2000 phần tử từ 0 đến 1999. Kết quả thu được như sau:



```
Time execution for COPY QUEUE using loop: 227 microseconds
Time execution for COPY QUEUE using recursive: 509 microseconds
```

Hình 61: [Queue Linked List] So sánh thời gian thực thi COPY QUEUE (lần 1)

```
Time execution for COPY QUEUE using loop: 348 microseconds
Time execution for COPY QUEUE using recursive: 331 microseconds
```

Hình 62: [Stack Linked List] So sánh thời gian thực thi COPY QUEUE (lần 2)

**Nhận xét chung:** Nhìn chung, thời gian thực hiện của hai hàm tương đối nhỏ (khoảng vài trăm microseconds), khá tương đồng nhau và khó xác định cụ thể hàm nào nhanh hơn.

## 8 Nhận xét - Feedback

### Những khó khăn gặp phải:

- Triển khai nội dung các hàm trong file `.cpp` bị lỗi khi xây dựng chương trình trong Visual Studio. Giải pháp khắc phục là đặt các cài đặt hàm (`implementation`) trong file `.ipp` hoặc `.tpp`.
- Viết chương trình dễ sử dụng, đủ chức năng, thực hiện trơn tru các thao tác trên Stack và Queue.
- Hiểu và vận dụng đệ quy để viết các hàm thay thế cho các hàm sử dụng vòng lặp một cách chính xác.
- Sửa các lỗi cú pháp, logic xảy ra khi viết và chạy chương trình, tốn nhiều thời gian để xác định, hiểu và sửa.

### Những điều đã học được:

- Hiểu rõ bản chất của Stack và Queue
- Biết cách cài đặt Stack và Queue bằng các cấu trúc dữ liệu cơ bản như mảng và danh sách liên kết.
- Cải thiện khả năng tư duy đệ quy, viết các hàm tương đương giữa vòng lặp và đệ quy.
- Cải thiện kĩ năng đọc và sửa lỗi.
- Trau dồi kĩ năng viết báo cáo bằng Latex.