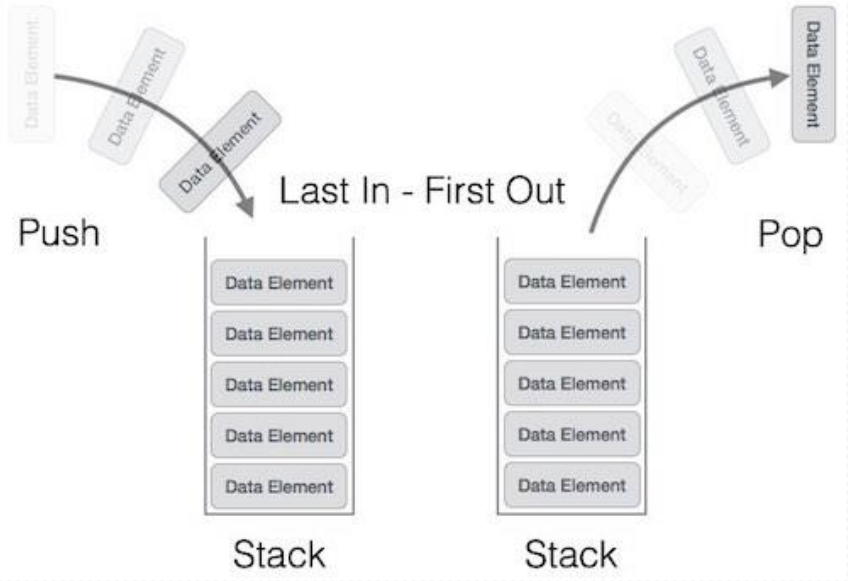
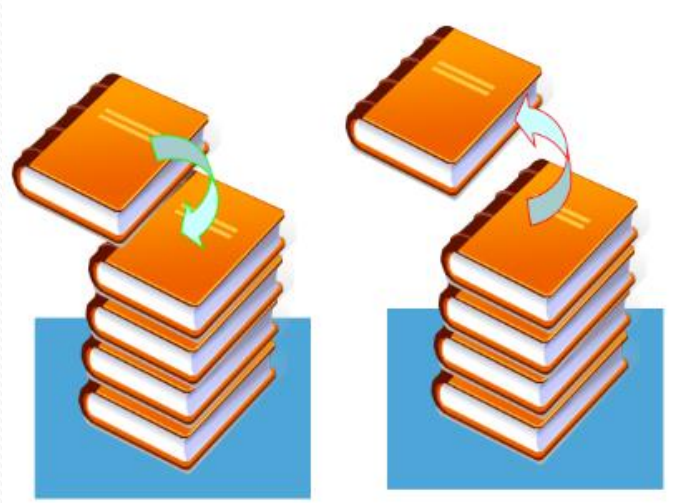
Bài 4. Ngăn xếp và Hàng đợi

Ngăn xếp (Stack)

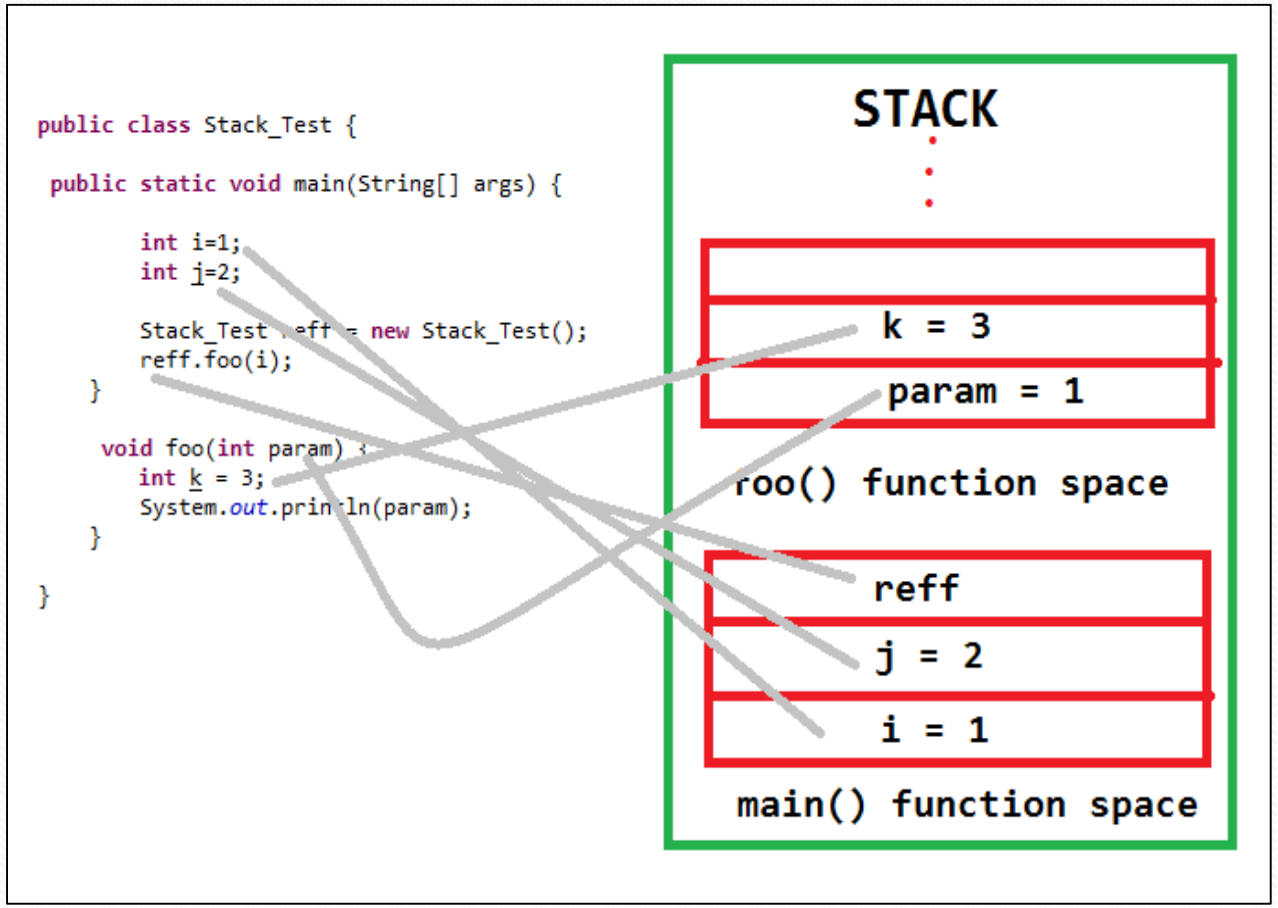
Giới thiệu

- Ngăn xếp là danh sách giới hạn trên 2 thao tác: thêm vào cuối và lấy ra phần tử cuối.
- Ngăn xếp còn gọi là danh sách LIFO (last in first out)



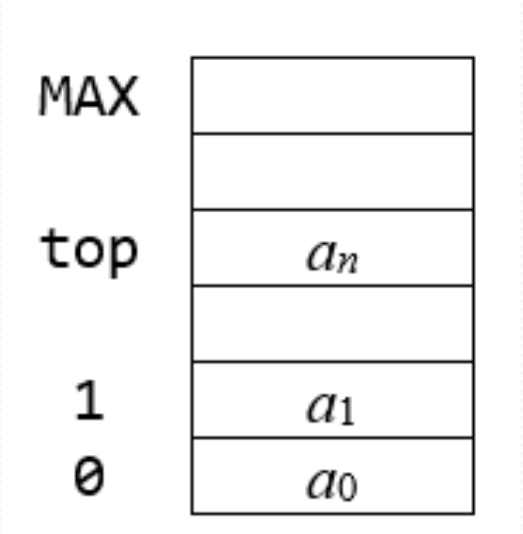


- Vùng nhớ cấp phát cho các biến của hàm thực hiện theo nguyên tắc của ngăn xếp



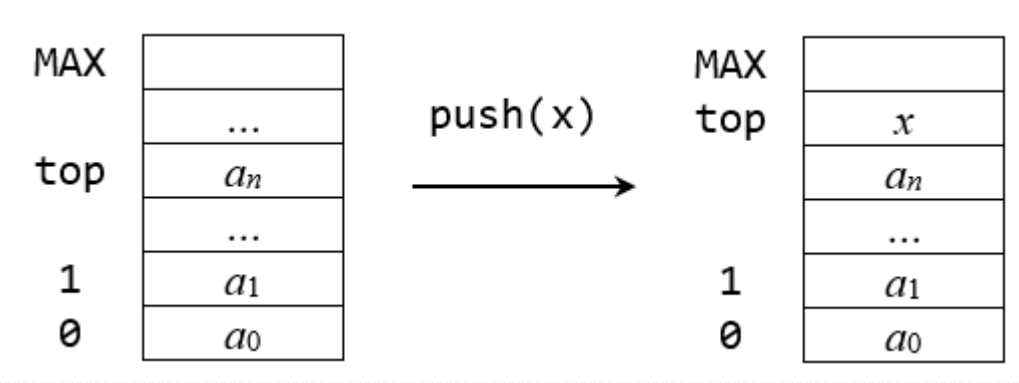
Tổ chức ngăn xếp bằng mảng

```
struct Stack{
   ElementType arr[MAX];
   int top;
};
```

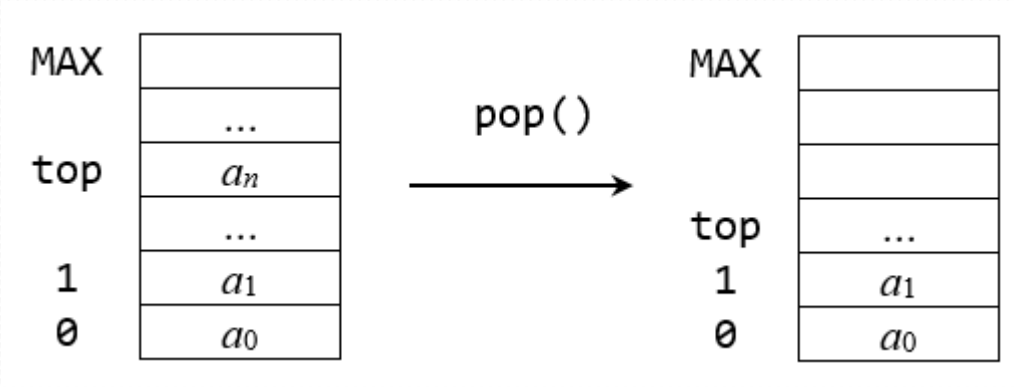



```
boolean empty(Stack s)
{
   return s.top == -1;
}
```

```
void push(Stack &s, ElementType x)
{
  if(s.top < MAX)
  {
      s.arr[++s.top] = x;
  }
}
```

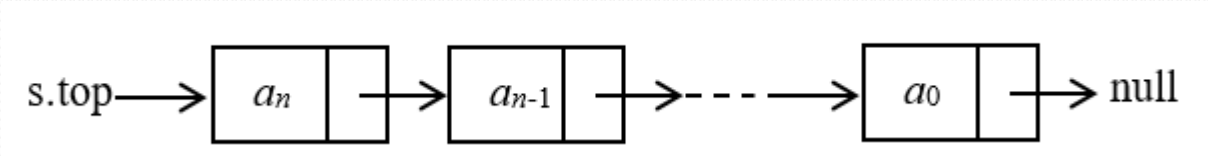



```
ElementType pop(Stack &s)
{
 if(! empty(s))
 {
      ElementType x = s.arr[s.top];
      s.top--;
      return x;
 }
}
```



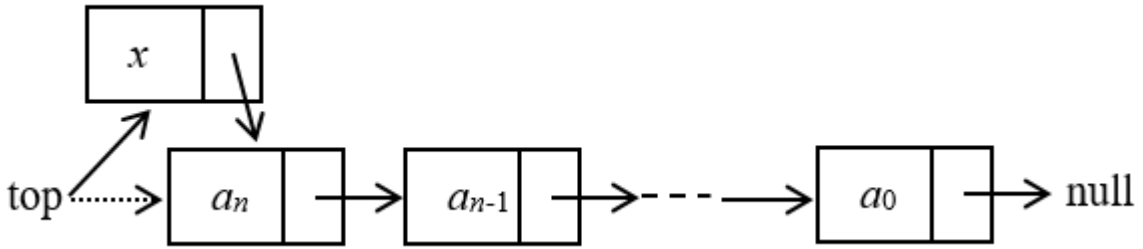
Tổ chức ngắn xếp bằng DSLK đơn

- Dùng DSLK đơn lưu các phần tử của ngăn xếp
- Phần tử cuối ngăn xếp lưu ở đầu DSLK đơn



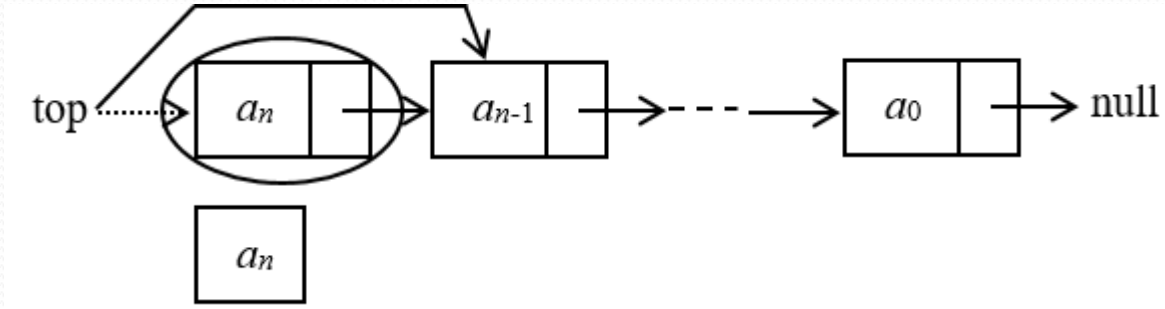

```
struct Node {
   ElementType data;
   Node       *next;
};
struct Stack {
  Node *top;
};
```

```
bool empty(Stack s)
{
   return (s.top == nullptr);
}
```




```
void push(Stack &s, ElementType x)
{
   Node *newNode;
   newNode = new Node;
   newNode->data = x;
   newNode->next = s.top;
   s.top = newNode;
}
```

```
ElementType pop(Stack &s)
{
   if (!empty(s))
   {
      Node *q;
      ElementType x;
      q = s.top;
      x= q->data;
      s.top = q->next;
      delete q;
      return x;
   }
}
```



Ứng dụng của ngăn xếp

- Cách tính biểu thức:
  - Chuyển biểu thức trung tố thành biểu thức hậu tố tương đương.
  - Tính giá trị của biểu thức hậu tố.
- Ví dụ biểu thức: 3 * (5+2) – 7 + 1 tương đương với 3 5 2 + * 7 – 1 +.

Input: Biểu thức trung tố cần chuyển

Output: Biểu thức hậu tố tương đương biểu thức trung tố đầu vào

Sử dụng một ngăn xếp để lưu các thành phần của biểu thức khi chuyển đổi.

Action:

Khởi tạo ngăn xếp rỗng, biểu thức hậu tố rỗng.

Duyệt lần lượt từng thành phần của biểu thức trung tố:

Nếu là toán hạng thì đưa vào sau biểu thức hậu tố.

Nếu là dấu ngoặc mở thì đưa vào ngăn xếp.

Nếu là dấu ngoặc đóng thì lấy lần lượt các phần tử ở đỉnh ngăn xếp đưa vào sau biểu thức hậu tố cho đến khi gặp dấu ngoặc mở.

Nếu là phép toán:

Lấy lần lượt các phép toán ở đỉnh ngăn xếp có độ ưu tiên cao hơn hoặc bằng phép toán đang xét (nếu có) đưa vào sau biểu thức hậu tố.

Đưa phép toán đang xét vào ngăn xếp.

Lấy lần lượt từng phần tử trong ngăn xếp đưa vào sau biểu thức hậu tố.

5 + (7 – 2)*3 – 8/2 + 1

Thành phần	Ngăn xếp	Biểu thức hậu tố
5		5
+	+	5
(	+ (	5
7	+ (	5 7
-	+ (-	5 7
2	+ (-	5 7 2
)	+	5 7 2 -
*	+ *	5 7 2 -
3	+ *	5 7 2 - 3
-	-	5 7 2 - 3 * +
8	-	5 7 2 – 3 * + 8
/	- /	5 7 2 – 3 * + 8
2	- /	5 7 2 – 3 * + 8 2
+	+	5 7 2 – 3 * + 8 2 / -
1	+	5 7 2 – 3 * + 8 2 / - 1
		5 7 2 – 3 * + 8 2 / - 1 +

Input: Biểu thức trung tố cần tính

Output: Giá trị của biểu thức

Sử dụng một ngăn xếp để lưu các giá trị trung gian trong khi tính toán.

Action:

Khởi tạo ngăn xếp rỗng.

Duyệt lần lượt từng thành phần của biểu thức hậu tố.

Nếu là toán hạng thì đưa vào ngăn xếp.

Nếu là phép toán thì:

Lấy lần lượt 2 phần tử ở đỉnh ngăn xếp thực hiện phép toán (phần tử lấy ra trước đặt ở bên phải phép toán, phần tử lấy ra sau đặt ở bên trái phép toán).

Đưa kết quả vào ngăn xếp

Giá trị duy nhất còn lại trong ngăn xếp chính là giá trị của biểu thức hậu tố.

5 7 2 – 3 * + 8 2 / - 1 +

Thành phần	Ngăn xếp
5	5
7	5 7
2	5 7 2
-	5 5
3	5 5 3
*	5 15
+	20
8	20 8
2	20 8 2
/	20 4
-	16
1	16 1
+	17

Lớp stack trong C++

- #include <stack>
- Khai báo biến: std::stack<DataType> variable;
- Ví dụ:
  - std::stack<int> stackOfInt;
  - std::stack<Node*> stackOfNodePtr;

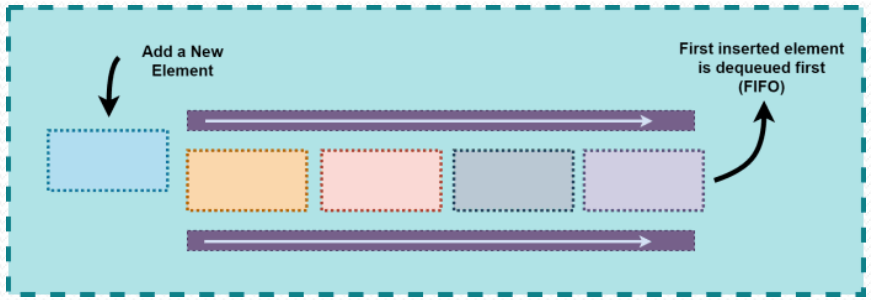
Phương thức	Chức năng	Độ phức tạp
size()	Số phần tử trong stack	O(1)
empty()	Cho biết stack có rỗng không	O(1)
swap(s)	Đổi các phần tử của stack với stack s	O(n)
push(e)	Thêm phần tử e vào đỉnh stack	O(1)
pop()	Xóa một phần tử ở đỉnh stack	O(1)
top()	Trả về phần tử ở đỉnh stack	O(1)

```
#include <iostream>
#include <stack>

int main() {
    std::stack<int> myStack;

    myStack.push(5);
    myStack.push(1);
    myStack.push(4);
    myStack.push(2);
    myStack.push(3);

    std::cout << "Elements of the stack: ";
    while (!myStack.empty()) {
        std::cout << myStack.top() << " ";
        myStack.pop();
    }
    std::cout << std::endl;

    return 0;
}
```

Hàng đợi (Queue)

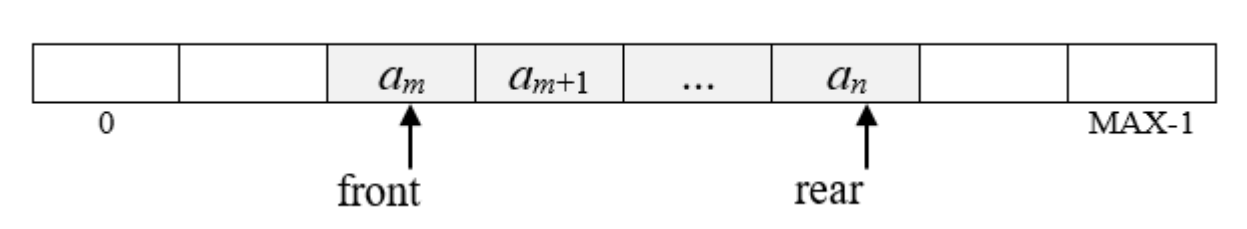
Giới thiệu

- Hàng đợi là danh sách giới hạn 2 thao tác: thêm vào cuối và lấy ra phần tử đầu
- Hàng đợi còn gọi là danh sách FIFO (first in first out)



Hàng đợi tổ chức bằng mảng

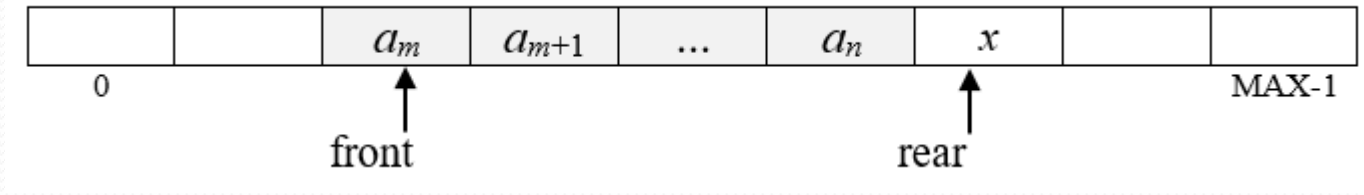
- Dùng mảng lưu các phần tử
- Dùng hai ô nhớ lưu vị trí đầu và cuối hàng đợi

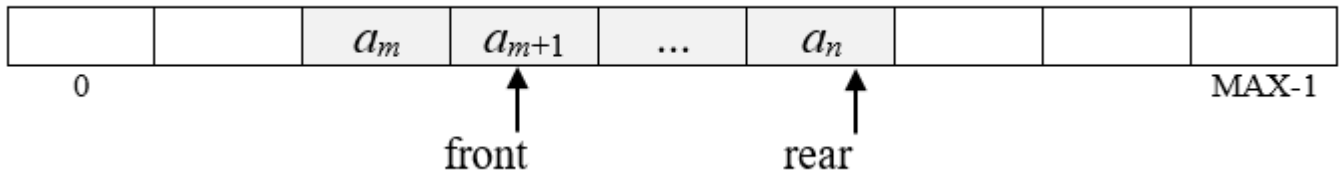



```
struct Queue{
   ElementType arr[MAX];
   int front, rear;
};
```

```
void initialize(Queue &q)
{
   q.front = 0;
   q.rear = -1;
}
```

```
boolean empty(Queue q)
{
   return q.rear == -1;
}
```

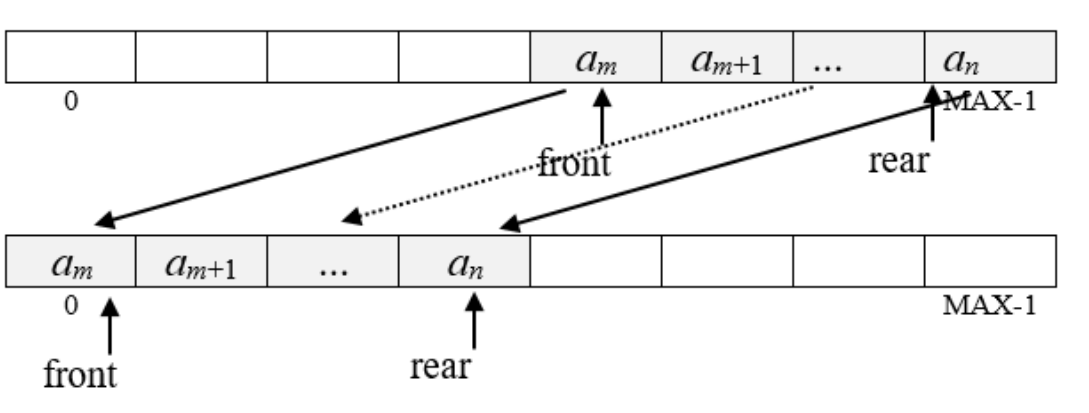





```
void add(Queue &q, ElementType x)
{
   if (q.rear < MAX)
   {
      q.rear++;
      q.arr[q.rear] = x;
   }
}
```

```
ElementType remove(Queue &q)
{
   if(!empty(q))
   {
      ElementType x = q.arr[q.front];
      q.front++;
      return x;
   }
}
```

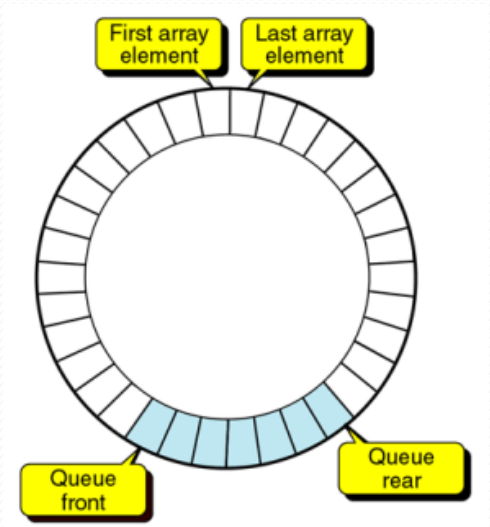
Xử lý tràn hàng đợi



Độ phức tạp O(n)

```
void add(Queue &q, ElementType x)
{
   int i;
   if (q.rear – q.front + 1 < MAX)
   {
      if (q.rear == MAX-1) //dời tịnh tiến
      {
       for (i = q.front; i<= q.rear; i++)
        q.arr[i-q.front] = q.arr[i];
       q.rear = q.rear – q.front;
       q.front = 0;
      }
      else
      {
       q.rear++;
       q.arr[q.rear] = x;
      }
   }
}
```

Di chuyển vòng

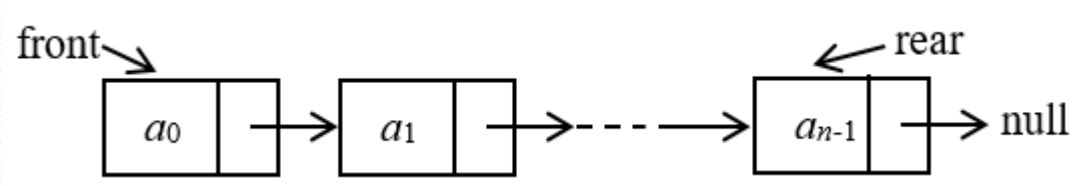



```
void add (Queue &q, ElementType x)
{
   if (q.front!=q.rear+1 && q.rear - q.front < MAX)
   {
      if (q.rear == MAX-1)
      q.rear = 0;
      else
      q.rear++;
      q.arr[q.rear] = x;
   }
}
```

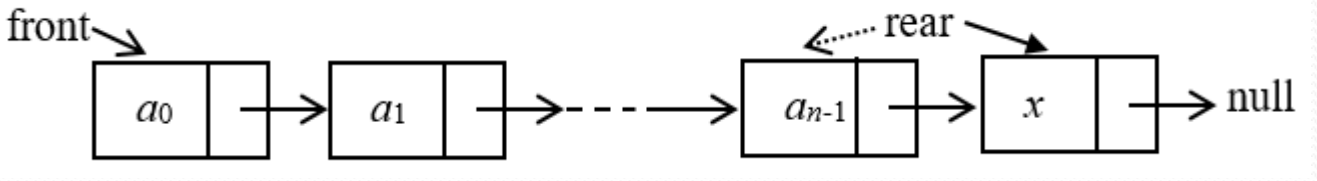
```
ElementType remove(Queue &q)
{
   ElementType x;
   if (!empty(q))
   {
      x = q.arr[q.front];
      if (q.front == q.rear) //hàng đợi còn 1 phần tử
      {
      q.front = 0;
      q.rear = -1;
      }
      else
      if (q.front == MAX-1)
              q.front = 0;
      else
              q.front++;
      return x;
   }
}
```

Tổ chức hàng đợi bằng DSLK đơn

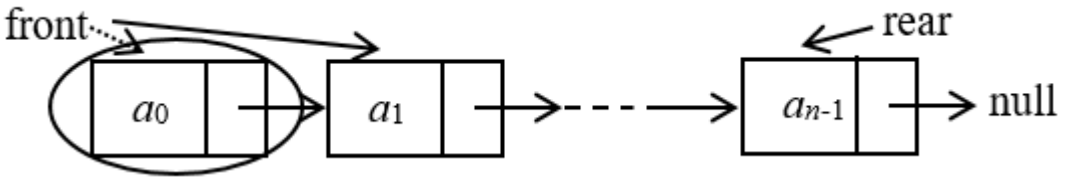
- Dùng DSLK đơn lưu các phần tử hàng đợi
- Lưu hai vị trí đầu và cuối hàng đợi




```
struct Node{
   ElementType data;
   Node* next;
};
struct Queue{
   Node *front, *rear;
};
```




```
void add(Queue &q, ElementType x)
{
      Node *newNode;
      newNode = new Node;
      newNode->data = x;
      newNode->next = nullptr;
      if (q.front == nullptr) //hàng đợi rỗng
      { q.front = q.rear = newNode; }
      else
      {
      q.rear->next = newNode;
      q.rear = newNode;
      }
}
```




```
ElementType remove(Queue &q)
{
      Node *p;
      ElementType x
      if (q.front != nullptr)
      {
      p = q.front;
      x = p->data;
      if (q.front == q.rear) //hàng đợi còn 1 phần tử
      { q.front = q.rear = nullptr;}
      else
            q.front = p->next;
      delete p;
      return x;
      }
}
```

Lớp queue trong C++

- #include <queue>
- Khai báo biến: std::queue<DataType> variable;
- Ví dụ:
  - std::queue<int> queueOfInt;
  - std::queue<Node*> queueOfNodePtr;

Phương thức	Chức năng	Độ phức tạp
size()	Số phần tử trong queue	O(1)
empty()	Cho biết queue có rỗng không	O(1)
push(e)	Thêm phần tử e vào cuối queue	O(1)
pop()	Xóa một phần tử ở đầu queue	O(1)
front()	Trả về phần tử đầu queue	O(1)
back()	Trả về phần tử cuối queue	O(1)

```cpp
#include <iostream>
#include <queue>

int main() {
    std::queue<int> myQueue;

    myQueue.push(5);
    myQueue.push(1);
    myQueue.push(4);

    while (!myQueue.empty()) {
        std::cout << myQueue.front() << " ";
        myQueue.pop();
    }

    return 0;
}
```

Bài tập

Bài 3.6 Sử dụng các kiểu stack và queue đã có, trình bày thuật toán và cài đặt các thao tác sau:

a) Lấy ra phần tử thứ k của hàng đợi

b) Thêm một phần tử vào vị trí k của ngăn xếp

c) Đảo ngược thứ tự của hàng đợi

d) Tìm khóa nhỏ nhất của ngăn xếp

Bài tập

Bài 3.7 Trình bày cách tổ chức dữ liệu cho hàng đợi hai đầu (deque), tức là thao tác thêm vào và lấy ra có thể thực hiện trên cả hai đầu của hàng đợi. Cài đặt các thao tác sau với độ phức tạp thời gian là O(1):

- push(x): thêm phần tử x vào đầu deque.
- pop(): xóa phần tử đầu khỏi deque và trả về phần tử này.
- inject(x): thêm phần tử x vào cuối deque.
- eject(): xóa phần tử cuối deque và trả về phần tử này.

Bài tập

Bài 3.8 Xây dựng cấu trúc dữ liệu hỗ trợ các thao tác push, pop của ngăn xếp và bổ sung thêm thao tác findMin, trả về kết quả là phần tử nhỏ nhất của cấu trúc dữ liệu với độ phức tạp trong trường hợp xấu nhất là O(1).

Bài 3.9 Trình bày cách cài đặt cấu trúc dữ liệu hai ngăn xếp bằng một mảng sao cho các ngăn xếp chỉ đầy khi mảng đã đầy.

Bài 3.10 Một cách khác để cài đặt hàng đợi là sử dụng danh sách liên kết vòng không chứa header. Cho phép sử dụng một con trỏ trỏ đến một nút trong danh sách liên kết vòng. Cho biết độ phức tạp của các thao tác trên hàng đợi trong trường hợp xấu nhất nếu:

a) Dùng con trỏ đến phần tử đầu của danh sách.

b) Dùng con trỏ đến phần tử cuối cùng của danh sách.