Chương 5:
NGĂN XẾP – HÀNG ĐỢI
(Stack - Queue)

Nội dung

- Ngăn xếp (Stack)
  - Khái niệm Stack
  - Các thao tác trên Stack
  - Hiện thực Stack
  - Ứng dụng của Stack
- Hàng đợi (Queue)

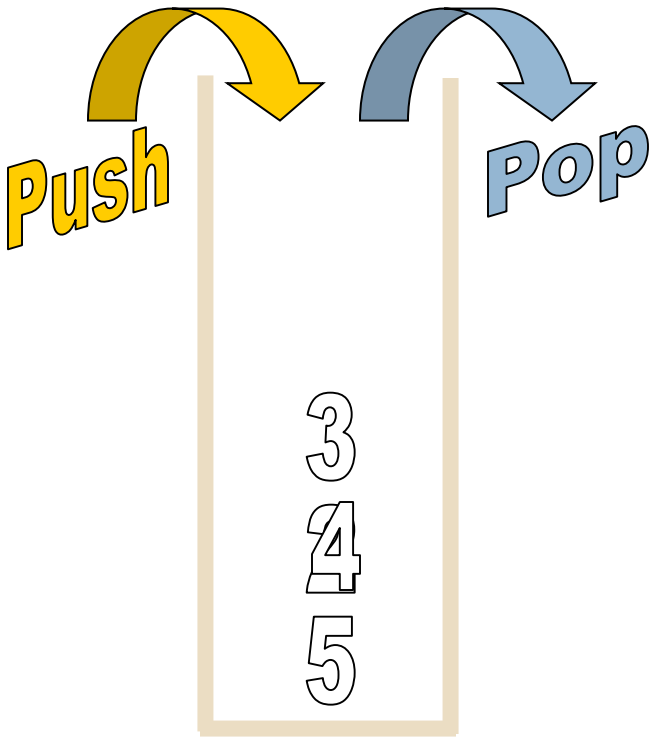
Stack - Khái niệm

- Stack là một danh sách mà các đối tượng được **thêm vào** và **lấy ra** chỉ ở một đầu của danh sách (A stack is simply a list of elements with insertions and deletions permitted at one end)
- Vì thế, việc thêm một đối tượng vào Stack hoặc lấy một đối tượng ra khỏi Stack được thực hiện theo cơ chế LIFO (Last In First Out - *Vào sau ra trước*)
- Các đối tượng có thể được thêm vào Stack bất kỳ lúc nào nhưng chỉ có đối tượng **thêm vào sau cùng** mới được phép lấy ra khỏi Stack

Stack – Các thao tác

- Stack hỗ trợ 2 thao tác chính:
  - "**Push**": Thao tác **thêm** 1 đối tượng vào Stack
  - "**Pop**": Thao tác **lấy** 1 đối tượng ra khỏi Stack
- Ví dụ:

  5 2 3 - - 4



Stack – Các thao tác

- Stack cũng hỗ trợ một số thao tác khác:
  - **isEmpty**(): Kiểm tra xem Stack có rỗng không
  - **Top**(): Trả về giá trị của phần tử nằm ở đầu Stack mà không hủy nó khỏi Stack. Nếu Stack rỗng thì lỗi sẽ xảy ra

Stack – Hiện thực Stack
(Implementation of a Stack)

Mảng 1 chiều

Kích thước stack khi quá thiếu, lúc quá thừa

Push / Pop hơi phức tạp

Danh sách LK

Cấp phát động!

Push/Pop khá dễ dàng

Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

- Có thể tạo một Stack bằng cách khai báo một mảng 1 chiều với kích thước tối đa là N (ví dụ: N =1000)
  - Stack có thể chứa tối đa N phần tử đánh số từ 0 đến N-1
- Phần tử nằm ở đỉnh Stack sẽ có chỉ số là top
- Như vậy, để khai báo một Stack, ta cần một mảng 1 chiều, và 1 biến số nguyên top cho biết chỉ số của đỉnh Stack:

```
struct Stack {
   DataType  list[N];
   int   top;
};
```

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

- Tạo một Stack S rỗng: top = 0
- Giá trị của top sẽ cho biết số phần tử hiện hành có trong Stack
- Khi cài đặt bằng mảng 1 chiều, Stack bị giới hạn kích thước nên cần xây dựng thêm một thao tác phụ cho Stack:
  - **isFull**(): Kiểm tra xem Stack có đầy chưa, vì khi Stack đầy, việc gọi đến hàm **Push()** sẽ phát sinh ra lỗi

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

- **Khởi tạo Stack:**

```
void Init (Stack &s)
{
    s.top = 0;
}
```

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

- **Kiểm tra Stack rỗng hay không:**
  - **Rỗng: hàm trả về 1**
  - **Ngược lại: hàm trả về 0**

```
int isEmpty(Stack s)
{
   if (s.top==0)
        return 1; // stack rỗng
   else
        return 0;
}
```

Hiện thực Stack dùng mảng

(Implementation of a Stack using Array)

- Kiểm tra Stack đầy hay không:
  - Đầy: hàm trả về 1
  - Ngược lại: hàm trả về 0

```
int isFull(Stack s)
{
   if (s.top>=N)
        return 1;
   else
        return 0;
}
```

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

- Thêm một phần tử x vào Stack

```
void Push (Stack &s, DataType x)
{
    if (!isFull(s)) // stack chưa đầy
    {
        s.list[s.top]=x;
        s.top++;
    }
}
```

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

- Trích thông tin và hủy phần tử ở đỉnh Stack

```
DataType Pop(Stack &s)
{
    DataType x;
    if (!isEmpty(s)) // stack khác rỗng
    {
        s.top--;
        x = s.list[s.top];
    }
    return x;
}
```

Hiện thực Stack dùng mảng
(Implementation of a Stack using Array)

Nhận xét:

- Các thao tác trên đều làm việc với chi phí O(1)
- Việc cài đặt Stack thông qua mảng một chiều đơn giản và khá hiệu quả
- Tuy nhiên, hạn chế lớn nhất của phương án cài đặt này là giới hạn về kích thước của Stack (N)
  - Giá trị của N có thể quá nhỏ so với nhu cầu thực tế hoặc quá lớn sẽ làm lãng phí bộ nhớ

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Có thể tạo một Stack bằng cách sử dụng một danh sách liên kết đơn (DSLK)
- Khai báo các cấu trúc:

```
struct Node
{
    DataType data;
    Node *pNext;
};
struct Stack
{
    Node *top;
};
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- **Khởi tạo Stack:**

```
void Init(Stack &t)
{
    t.top = NULL;
}
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- Kiểm tra xem Stack có rỗng không:

```
int  isEmpty (Stack t)
{
    return  t.top == NULL ? 1 : 0;
}
```

Hiện thực Stack dùng DSLK
(Implementation of a Stack using Linked List)

- **Thêm một phần tử vào Stack:**

```
void  Push (Stack &t, DataType x)
{
        Node *p = new Node;
        if (p==NULL)  { cout<<"Khong du bo nho"; return; }
        p->data = x;
        p->pNext = NULL;
        if (t.top==NULL)        // if (isEmpty(t))
                t.top = p;
        else{
                p->pNext = t.top;
                t.top = p;
            }
}
```

Thêm phần tử vào đầu danh sách

Hiện thực Stack dùng DSLK
(Implementation of a Stack Using Linked List)

- Trích thông tin và hủy phần tử ở đỉnh Stack:

```
DataType Pop (Stack &t)
{
      if (t.top==NULL){
            cout<<"Stack rỗng";
            return 0;
      }
      DataType x;
      Node *p = t.top;
      t.top = t.top->pNext;
      x = p->data;
      delete p;
      return x;
}
```

Lấy và xóa phần tử ở đầu danh sách

Stack - Ứng dụng

- Stack thích hợp lưu trữ các loại dữ liệu mà trình tự truy xuất ngược với trình tự lưu trữ
- Một số ứng dụng của Stack:
  - Trong trình biên dịch (thông dịch), khi thực hiện các thủ tục, Stack được sử dụng để lưu môi trường của các thủ tục
  - Lưu dữ liệu khi giải một số bài toán của lý thuyết đồ thị (như tìm đường đi)
  - Khử đệ qui
  - Ứng dụng trong các bài toán tính toán biểu thức
  - …

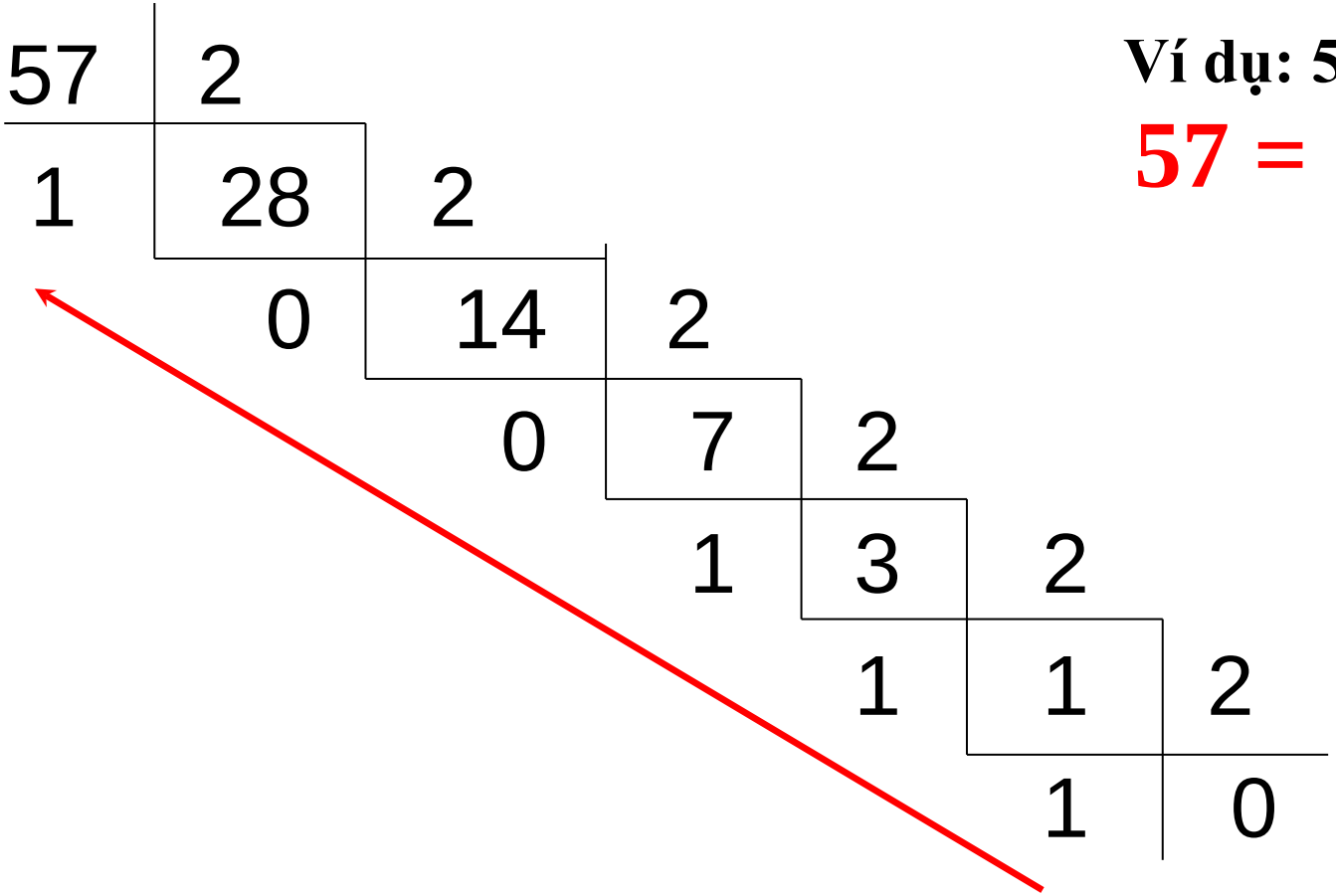
Stack - Ứng dụng

Ví dụ: thủ tục Quick_Sort dùng Stack để khử đệ qui:

- Bước 1. l=1; r=n;
- Bước 2. Chọn phần tử giữa x=a[(l+r) / 2]
- Bước 3. Phân hoạch (l, r) thành (l1, r1) và (l2, r2) bằng cách xét:
  - y thuộc (l1, r1) nếu y≤x
  - y thuộc (l2, r2) ngược lại
- Bước 4. Nếu phân hoạch (l2, r2) có nhiều hơn 1 phần tử thì thực hiện:
  - Cất (l2, r2) vào Stack
  - Nếu (l1, r1) có nhiều hơn 1 phần tử thì thực hiện:
    - l = l1
    - r = r1
    - Quay lên bước 2
  - Ngược lại
    - Lấy (l, r) ra khỏi Stack, nếu Stack khác rỗng thì quay lên bước 2, ngược lại thì dừng

Stack - Ứng dụng

Bài tập: đổi số từ cơ số 10 sang cơ số x



Ví dụ: 57 = $???_2$

$57 = 111001_2$

```
void main()
{
    Stack s;
    int coso, so, sodu;
    Init(s);
    // Nhập số cần chuyển vào biến so …
    // Nhập cơ số cần chuyển vào biến coso…
    while (so != 0)
    {
        sodu = so % coso;
        Push (s, sodu);  // push so du vao stack
        so = so/coso;
    }
    cout<<"Kết quả: ";
    while (!isEmpty(s))
        cout<<Pop(s);  // pop so du ra khoi stack
}
```

Stack - Ứng dụng

- Thuật toán Ba Lan ngược
  (Reverse Polish Notation – RPN)
  - Định nghĩa RPN:
    - Biểu thức toán học trong đó các toán tử được viết sau toán hạng và không dùng dấu ngoặc
  - Phát minh bởi Jan Lukasiewics một nhà khoa học Ba Lan vào những năm 1950

Thuật toán Ba Lan ngược - RPN

Infix : toán tử viết giữa toán hạng
Postfix (RPN): toán tử viết sau toán hạng
Prefix : toán tử viết trước toán hạng

Examples:

INFIX	RPN (POSTFIX)	PREFIX
A + B	A B +	+ A B
A * B + C	A B * C +	+ * A B C
A * (B + C)	A B C + *	* A + B C
A - (B - (C - D))	A B C D - - -	- A - B - C D
A - B - C - D	A B - C - D -	- - - A B C D

Lượng giá biểu thức RPN

***Kỹ thuật gạch dưới*:**

1. Duyệt từ trái sang phải của biểu thức cho đến khi gặp toán tử.

2. Gạch dưới 2 toán hạng ngay trước toán tử và kết hợp chúng bằng toán tử trên

3. Lặp đi lặp lại cho đến hết biểu thức.

Ví dụ 2*((3+4)-(5-6))

2 3 4 + 5 6 - - *

→ 2 3 4 + 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 -1 - *

→ 2 7 -1 - * → 2 8 * → 2 8 * → 16

Thuật toán tính giá trị

1. Khởi tạo Stack rỗng *(chứa hằng hoặc biến)*.
2. Lặp cho đến khi kết thúc biểu thức:

    Đọc 01 phần tử của biểu thức *(hằng, biến, phép toán)*.

    Nếu phần tử là *hằng* hay *biến*: đưa vào Stack.

    Ngược lại:

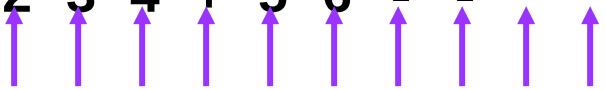
        Lấy ra 02 phần tử của Stack.

        Áp dụng phép toán cho 02 phần tử vừa lấy ra.

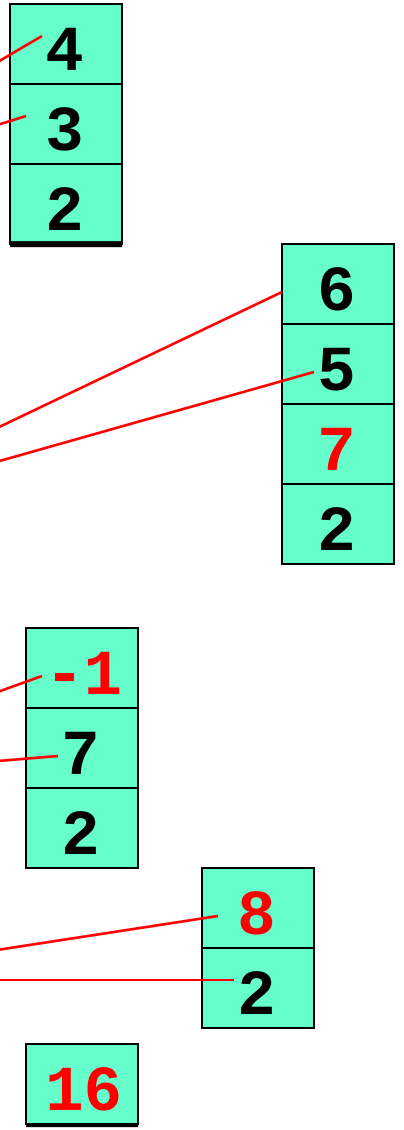
        Đưa kết quả vào Stack.
3. Giá trị của biểu thức chính là phần tử cuối cùng của Stack.

2*((3+4)-(5-6))

Example: `2 3 4 + 5 6 - - *`



- Push 2
- Push 3
- Push 4
- Read +
  - Pop 4, Pop 3, 3 + 4 = 7
- Push 7
- Push 5
- Push 6
- Read -
  - Pop 6, Pop 5, 5 - 6 = -1
- Push -1
- Read -
  - Pop -1, Pop 7, 7 - -1 = 8
- Push 8
- Read *
  - Pop 8, Pop 2, 2 * 8 = 16
- Push 16



Chuyển infix thành postfix

1. Khởi tạo Stack rỗng (chứa các phép toán)
2. Lặp cho đến khi kết thúc biểu thức:

   Đọc 01 phần tử của biểu thức *(01 phần tử có thể là hằng, biến, phép toán, ")" hay "(")*

   Nếu phần tử là:

   2.1 **"("**: đưa vào Stack.

   2.2 **")"**: lấy các phần tử của Stack ra cho đến khi gặp "(" trong Stack.

Chuyển infix thành postfix

2.3 Một phép toán: + - * /

Nếu Stack rỗng: đưa vào Stack.

Nếu Stack khác rỗng và **phép toán có độ ưu tiên cao hơn phần tử ở đầu Stack**: đưa vào Stack.

Nếu Stack khác rỗng và phép toán có độ ưu tiên thấp hơn hoặc bằng phần tử ở đầu Stack:

- lấy phần tử từ Stack ra;
- sau đó lặp lại việc so sánh với phần tử ở đầu Stack.

2.4 Hằng hoặc biến: đưa vào kết quả.

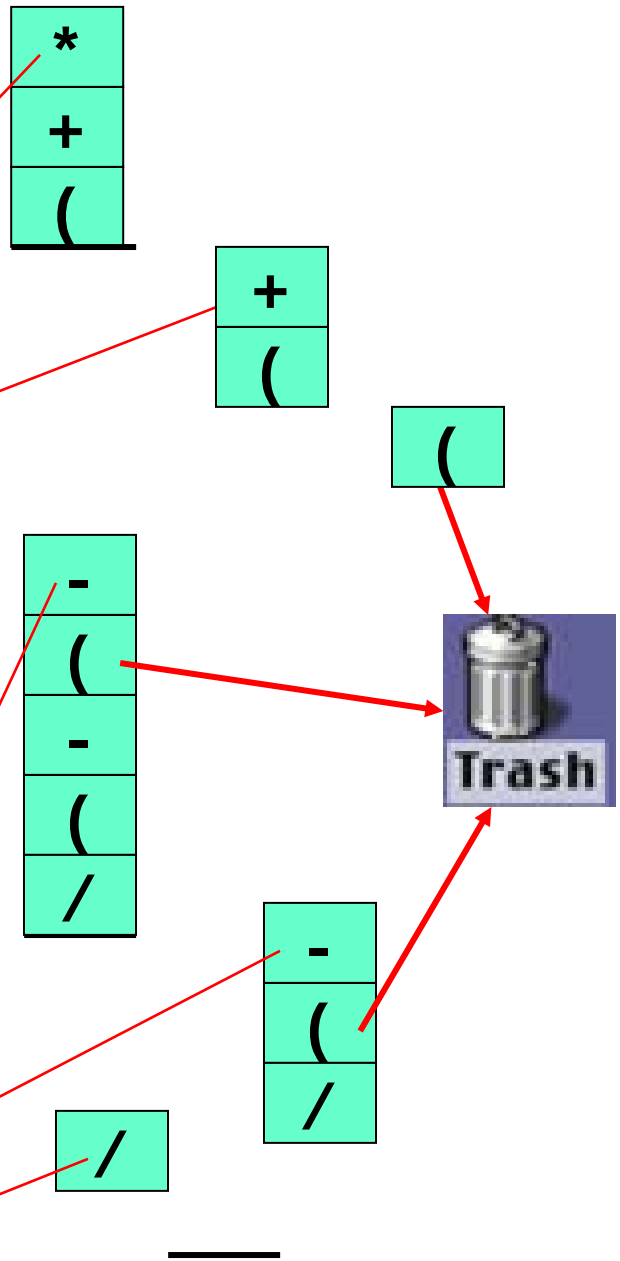
3. Lấy hết tất cả các phần tử của Stack ra.

Thuật toán Ba Lan ngược - Độ ưu tiên

- + , - 1
- *, / 2
- ^ 3

Example: `(A+B*C)/(D-(E-F))`

	Output
Push (	
Display A	A
Push +	
Display B	AB
Push *	
Display C	ABC
Read)	
Pop *, Display *,	ABC*
Pop +, Display +, Pop (	ABC*+
Push /	
Push (	
Display D	ABC*+D
Push -	
Push (	
Display E	ABC*+DE
Push -	
Display F	ABC*+DEF
Read)	
Pop -, Display -, Pop (	ABC*+DEF-
Read)	
Pop -, Display -, Pop (	ABC*+DEF--
Pop /, Display /	ABC*+DEF--/



Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[];

KQ=“”

Ví dụ

A + (B*C - (D/E^F) * G) * H

A + (B*C - (D/E^F) * G) * H

S=[];

KQ="A"

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(*;];

KQ=ABC

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(*-/^FEDG*CBH;

KQ=ABC*-DEF^/G*+H*

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-[;

KQ=ABC*

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(];

KQ=ABC*D

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/;

KQ=ABC*D

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/];

KQ=ABC*DE

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+((-((/^]:;

KQ=ABC*DE

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/^];

KQ=ABC*DEF

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-(/^;];

KQ=ABC*DEF^/

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-*;];

KQ=ABC*DEF^/

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+(-*];

KQ=ABC*DEF^/G

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+[[;]*;];

KQ=ABC*DEF^/G*-

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+]*;

KQ=ABC*DEF^/G*-

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[+*];

KQ=ABC*DEF^/G*-H

Ví dụ

A + (B*C - (D/E^F) * G) * H

S=[];

KQ=ABC*DEF^/G*-H*+

Nội dung

- Ngăn xếp (Stack)
- Hàng đợi (Queue)
  - Khái niệm Queue
  - Các thao tác trên Queue
  - Hiện thực Queue
  - Ứng dụng Queue

Queue - Khái niệm

- Queue là một danh sách mà các đối tượng được **thêm vào** ở một đầu của danh sách và **lấy ra** ở một đầu kia của danh sách (*A queue is also a list of elements with insertions permitted at one end and deletions permitted from the other end)*
- Việc **thêm** một đối tượng vào Queue luôn diễn ra ở **cuối** Queue và việc **lấy** một đối tượng ra khỏi Queue luôn diễn ra ở **đầu** Queue
- Vì thế, việc thêm một đối tượng vào Queue hoặc lấy một đối tượng ra khỏi Queue được thực hiện theo cơ chế FIFO (First In First Out - *Vào trước ra trước*)

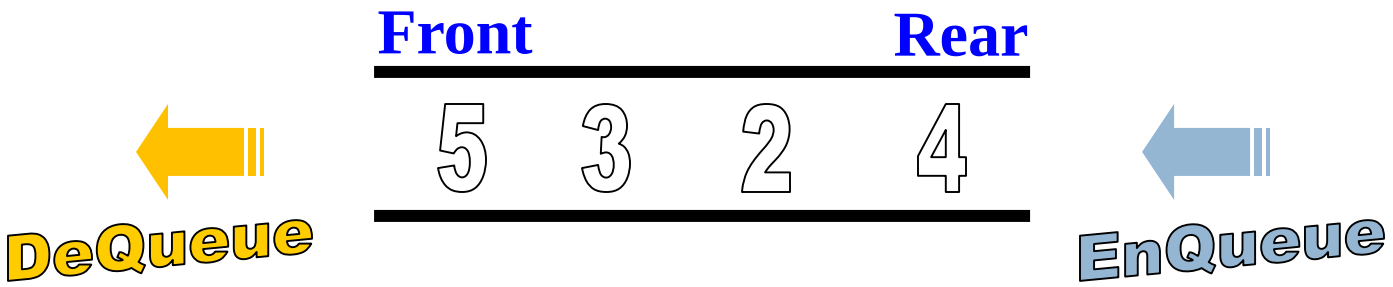
Queue - Khái niệm

Imaging

Queue – Các thao tác

- Hàng đợi hỗ trợ các thao tác:
  - **EnQueue**(): **Thêm** đối tượng vào cuối (rear) Queue
  - **DeQueue**(): **Lấy** đối tượng ở đầu (front) Queue ra khỏi Queue
- Ví dụ:

  5 3 2 - - 4



Queue – Các thao tác

- Queue còn hỗ trợ các thao tác:
  - **isEmpty**(): Kiểm tra xem hàng đợi có rỗng không
  - **Front**(): Trả về giá trị của phần tử nằm ở đầu hàng đợi mà không hủy nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra

Queue – Hiện thực Queue
(Implementation of a Queue)

Mảng 1 chiều

Kích thước queue khi quá thiếu, lúc quá thừa

EnQueue/DeQueue hơi phức tạp

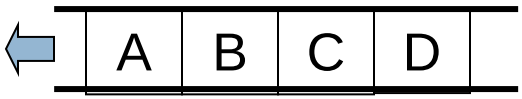
Danh sách LK

Cấp phát động!

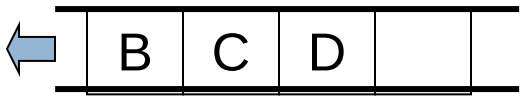
EnQueue/DeQueue khá dễ dàng

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Hai cách hiện thực:
  - **Khi lấy một phần tử ra thì đồng thời dời các ô phía sau nó lên một vị trí:**



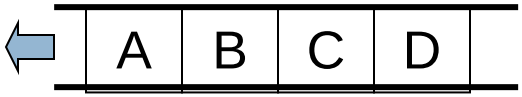
Ban đầu



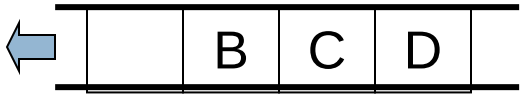
Lấy ra 1 phần tử:
dời tất cả về trước để
trống chỗ thêm vào

Thêm vào 1 phần tử

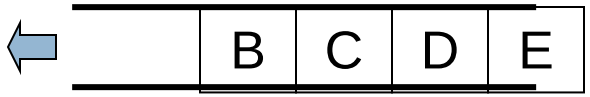
  - **Khi lấy một phần tử ra thì không dời ô lên (xoay vòng):**



Ban đầu



Lấy ra 1 phần tử

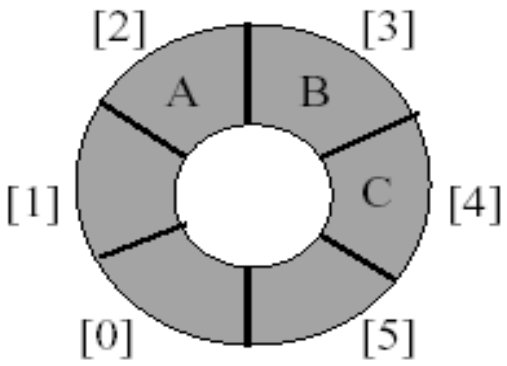


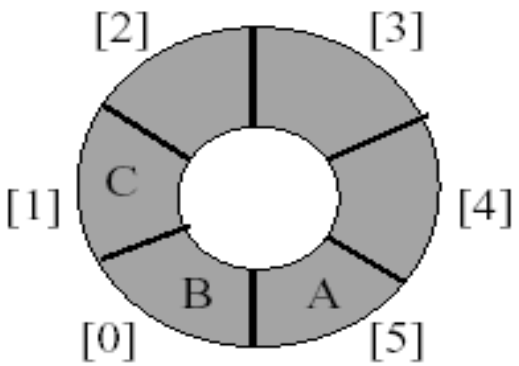
Thêm vào 1 phần tử

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

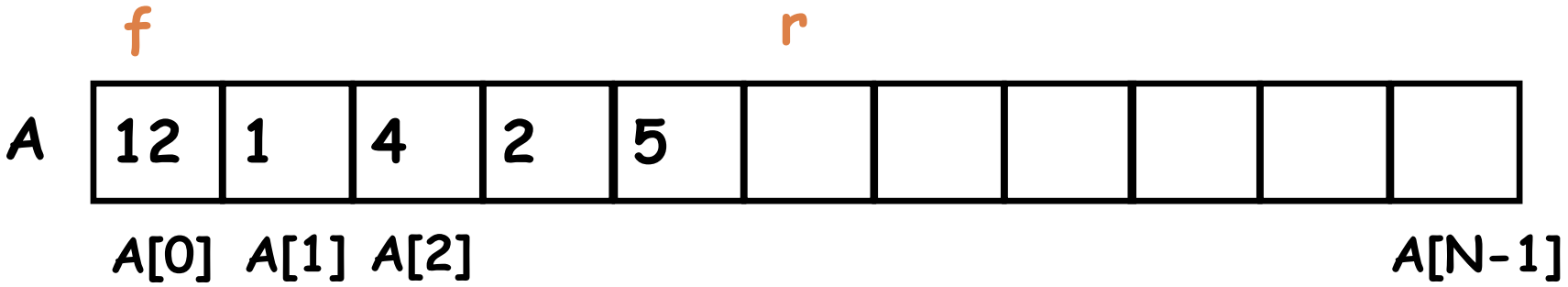
- Trạng thái Queue lúc bình thường:

- Trạng thái Queue lúc xoay vòng:





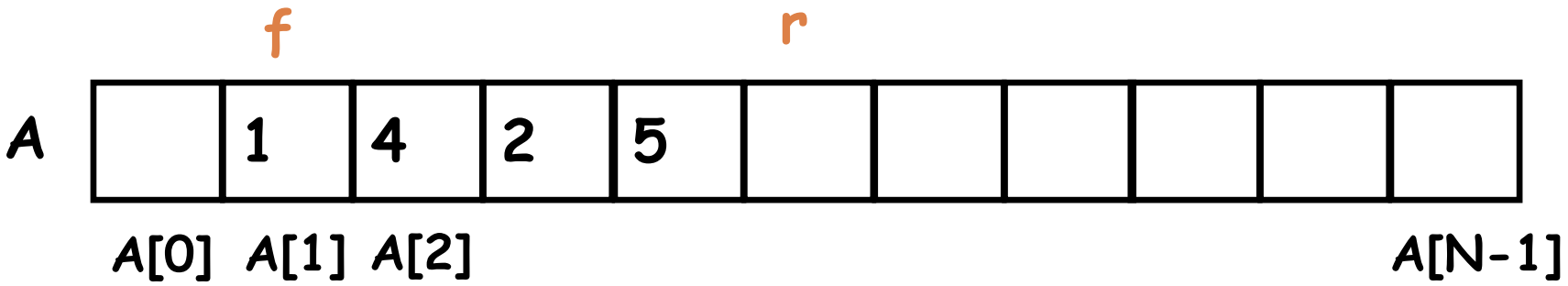
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)

Cách dùng mảng 2

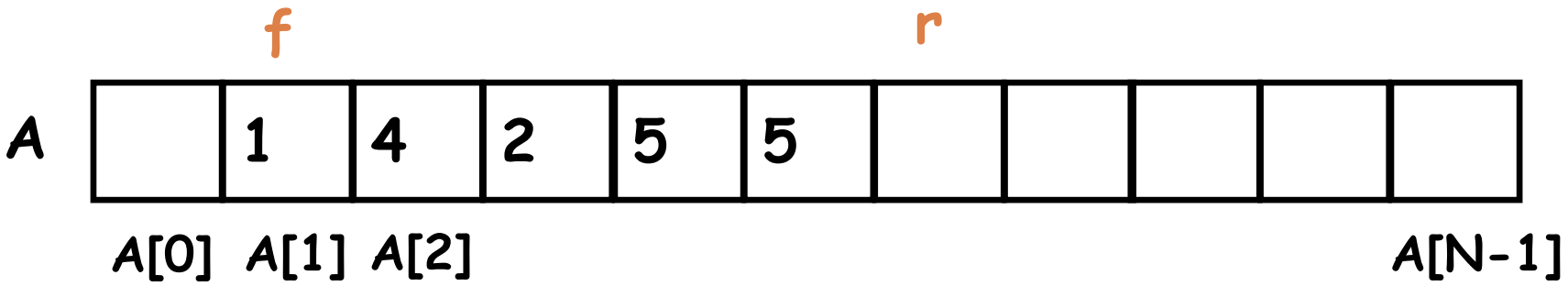
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)

Cách dùng mảng 2

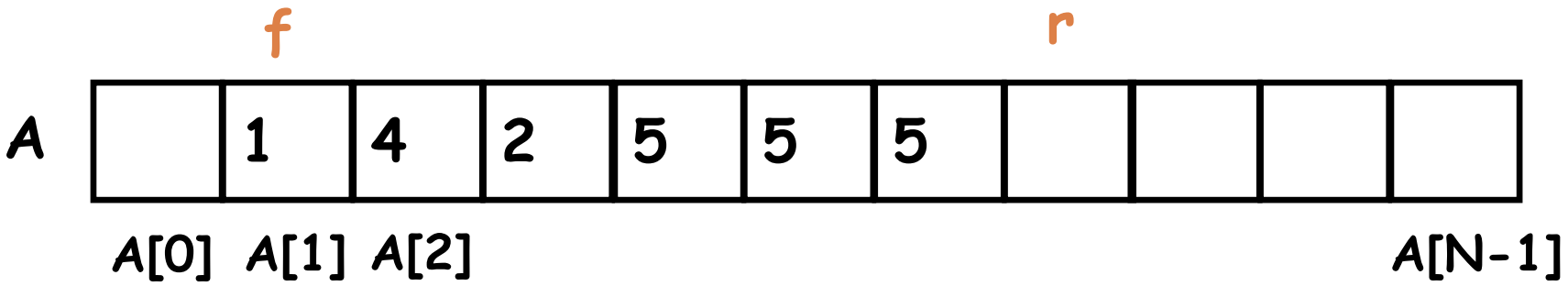
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)

Cách dùng mảng 2

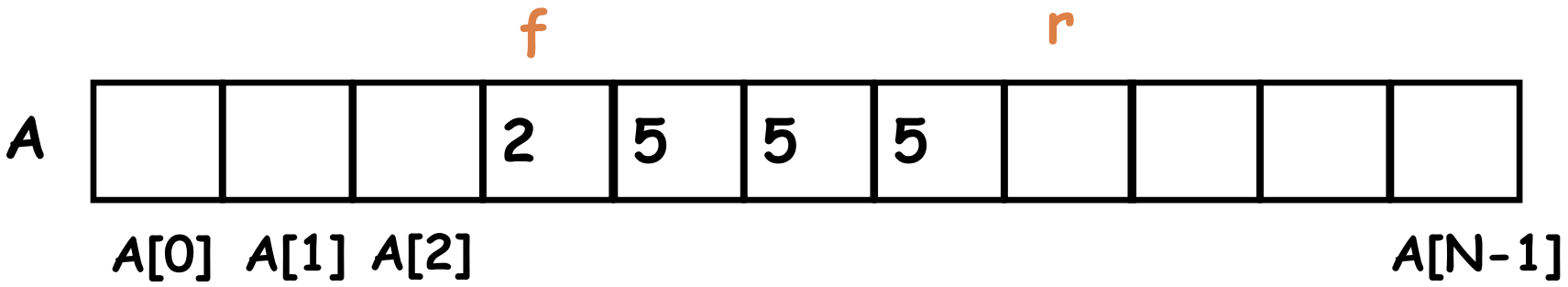
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)
DeQueue(Q)
DeQueue(Q)

Cách dùng mảng 2

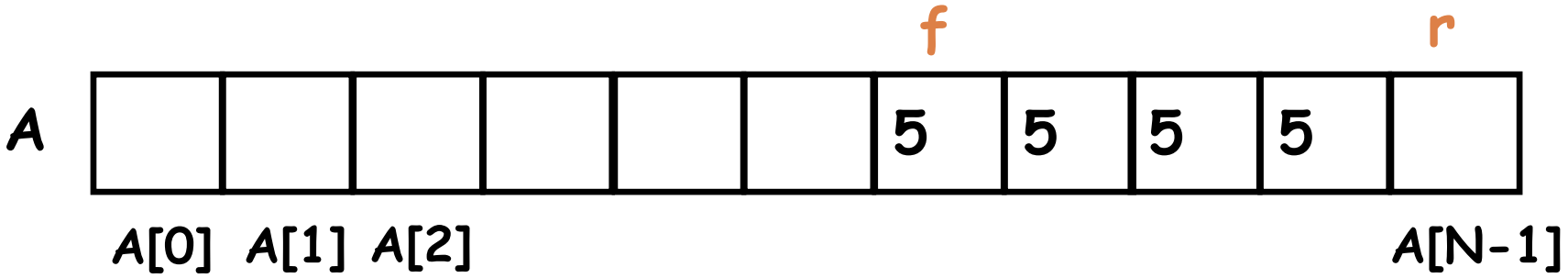
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)
DeQueue(Q)
DeQueue(Q)
DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),…….

Cách dùng mảng 2

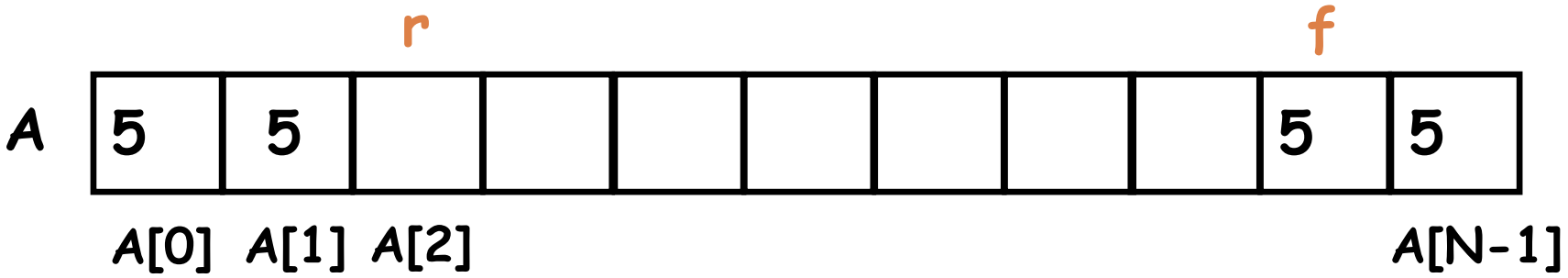
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q)
EnQueue(5,Q)
EnQueue(5,Q)
DeQueue(Q)
DeQueue(Q)
DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),…….

Cách dùng mảng 2

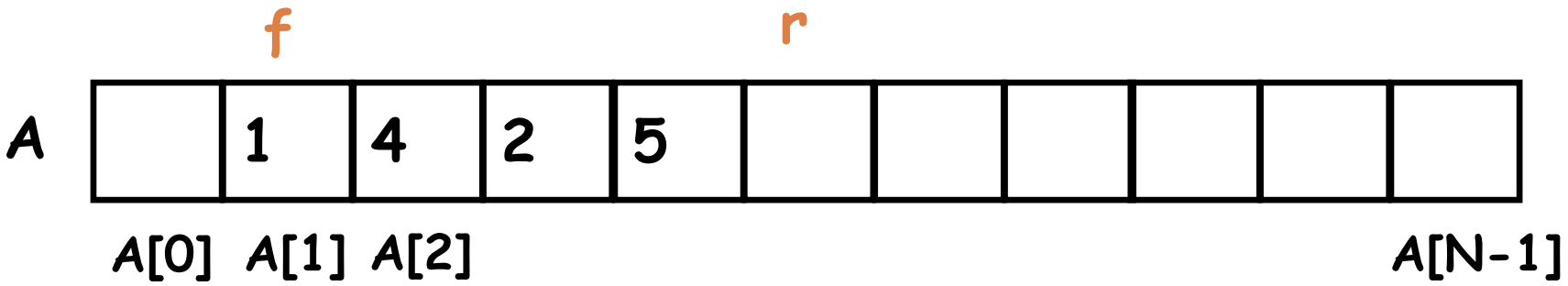
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



DeQueue(Q), EnQueue(5,Q), DeQueue(Q), EnQueue(5,Q),…….

Cách dùng mảng 2

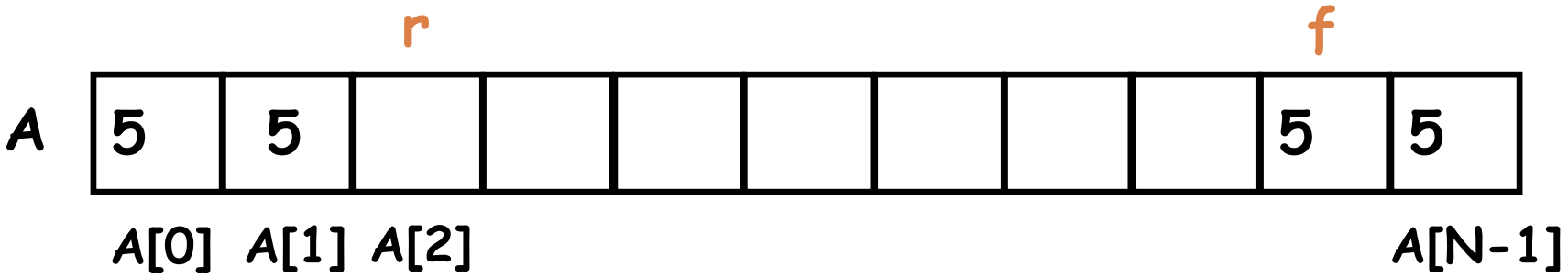
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)




```
size(Q):  if (r >= f) then return (r-f)
          else return N-(f-r)
```

Cách dùng mảng 2

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)



size(Q): if (r >= f) then return (r-f)
else return N-(f-r)

Cách dùng mảng 2

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Để khai báo một Queue, ta cần khai báo:
  - 1 mảng một chiều list,
  - 2 số nguyên front, rear cho biết chỉ số của đầu và cuối của hàng đợi,
  - hằng số N cho biết kích thước tối đa của Queue
- Hàng đợi có thể được khai báo cụ thể như sau:

```
struct Queue
{
    DataType  list[N];
    int   front, rear;
};
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Do khi cài đặt bằng mảng một chiều, hàng đợi bị giới hạn kích thước nên cần xây dựng thêm một thao tác phụ cho hàng đợi:
  - **isFull**(): Kiểm tra xem hàng đợi có đầy chưa

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Khởi tạo Queue:

```
void   Init(Queue &q)
{
     q.front = q.rear = 0;
}
```

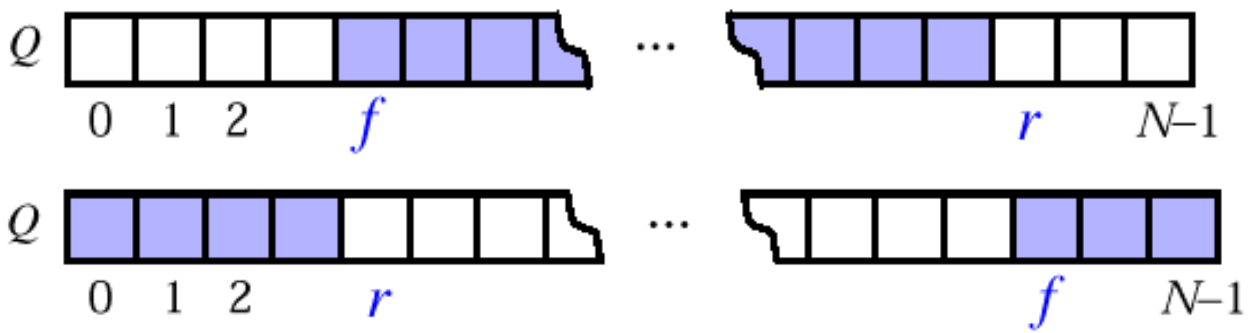
- Kiểm tra xem Queue có rỗng không:

```
int isEmpty(Queue q)
{
      if (q.front==q.rear && q.rear==0)
            return 1;
      if (q.front == q.rear) return 1;
      return 0;
}
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Kiểm tra xem Queue có đầy hay không:

```
int isFull(Queue q)
{
  if (q.front == q.rear) return 1;
  return 0;
}
```



Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Nhận xét:
  - Khi `front = rear` thì queue có thể đầy hoặc rỗng
  - Không thể phân biệt được queue đầy hoặc rỗng trong trường hợp này

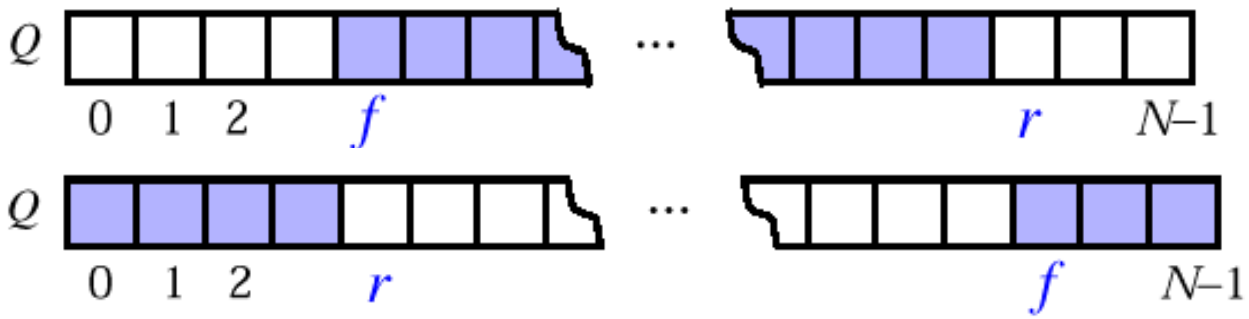
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Cách giải quyết:
  1. Không để queue đầy
     - Tăng kích thước mảng khi thêm mà không còn chỗ
  2. Định nghĩa thêm 1 biến để tính số pt hiện hành trong queue (NumElements)
     - Mỗi khi thêm 1 pt vào queue thì NumElements++
     - Mỗi khi lấy 1 pt khỏi queue thì NumElements—
     - Queue rỗng khi (`front = rear` và `NumElements!=N`)
     - Queue đầy khi (`front = rear` và `NumElements==N`)

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Thêm một phần tử x vào cuối Queue:

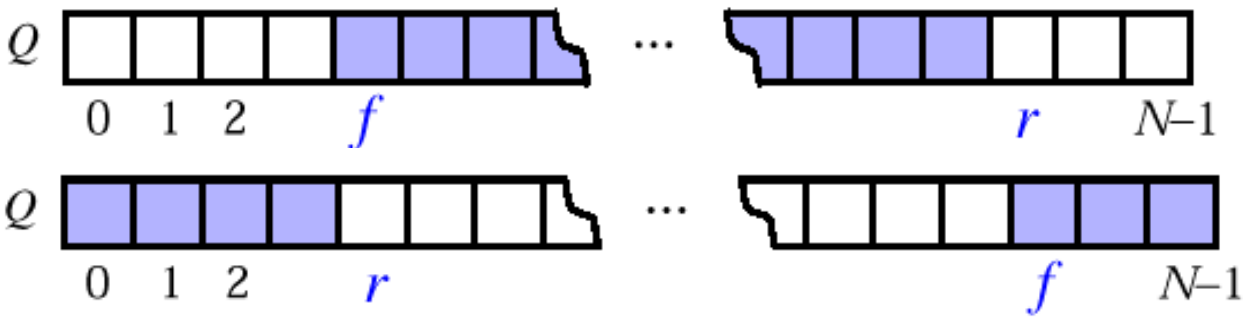
```
int  EnQueue(Queue &q, DataType x)
{
    if (isFull(q))
        return 0; // không thêm được vì Queue đầy
    q.list[q.rear] = x;
    q.rear++;
    if (q.rear==N)      q.rear=0;
    return 1;
}
```



Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Trích, huỷ phần tử ở đầu Queue:

```
DataType  DeQueue(Queue &q)
{
      if (isEmpty(q)){
            cout<<"Queue rong";
            return 0;}
      DataType t = q.list[q.front];
      q.front++;
      if (q.front==N) q.front = 0;
      return t;
}
```



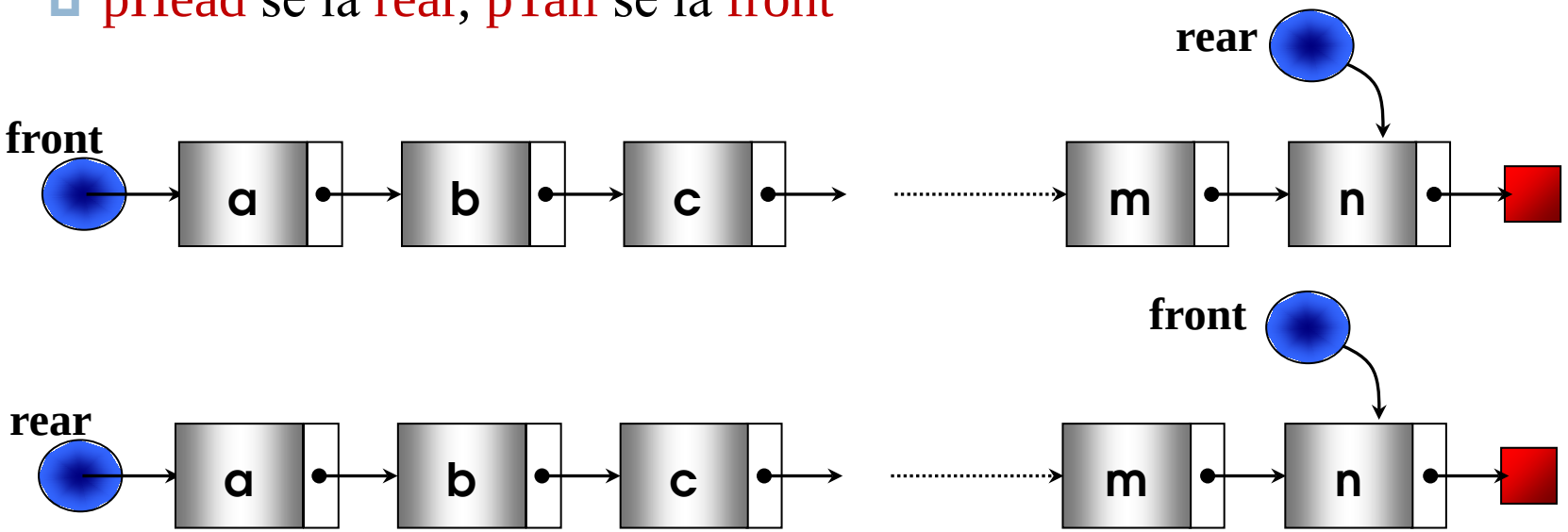
Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Xem thông tin của phần tử ở đầu Queue:

```
DataType Front(Queue q)
{
   if (isEmpty(q))
   {
        cout<<"Queue rong";
        return 0;
   }
   return q.list[q.front];
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Có thể biểu diễn Queue bằng cách sử dụng DSLK đơn
- Có 2 lựa chọn (cách nào tốt nhất?):
  - pHead sẽ là front, pTail sẽ là rear
  - pHead sẽ là rear, pTail sẽ là front



Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Khai báo các cấu trúc:

```
struct Node
{
    DataType data;
    Node *pNext;
};
struct Queue
{
    Node *front, *rear;
};
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Khởi tạo Queue rỗng:

```
void Init(Queue &q)
{
    q.front = q.rear = NULL;
}
```

- Kiểm tra hàng đợi rỗng :

```
int  isEmpty(Queue &q)
{
    if (q.front==NULL)
        return 1;
    else
        return 0;
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Thêm một phần tử p vào cuối Queue:

```
int EnQueue(Queue &q, DataType x)
{
   Node *p = new Node;
   if (p==NULL) return 0; //Khong du bo nho
   p->pNext = NULL;
   p->data = x;
   if (q.front==NULL) // TH Queue rỗng
      q.front = q.rear = p;
   else
   {
      q.rear->pNext = p;
      q.rear = p;
   }
   return 1;
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

- Trích và huỷ phần tử ở đầu Queue:

```
DataType DeQueue(Queue &q)
{
    if (isEmpty(q)) {
        cout<<"Queue rong";return 0;
    }
    Node *p = q.front;
    DataType x = p->data;
    q.front = q.front->pNext;
    if (q.front==NULL) q.rear = NULL;
    delete p;
    return x;
}
```

Hiện thực Queue dùng mảng
(Implementation of a Queue using Array)

- Xem thông tin của phần tử ở đầu Queue:

```
DataType Front(Queue q)
{
   if (isEmpty(q))
   {
        cout<<"Queue rong";
        return 0;
   }
   return q.front->data;
}
```

Hiện thực Queue dùng DSLK
(Implementation of a Queue using Linked List)

Nhận xét:

- Các thao tác trên Queue biểu diễn bằng danh sách liên kết làm việc với chi phí O(1)
- Nếu không quản lý phần tử cuối xâu, thao tác **Dequeue** sẽ có độ phức tạp O(n)

Queue - Ứng dụng

- Queue có thể được sử dụng trong một số bài toán:
  - Bài toán "sản xuất và tiêu thụ" (ứng dụng trong các hệ điều hành song song)
  - Bộ đệm (ví dụ: Nhấn phím ⇒ Bộ đệm ⇒ CPU xử lý)
  - Xử lý các lệnh trong máy tính (ứng dụng trong HĐH, trình biên dịch), hàng đợi các tiến trình chờ được xử lý, ….