

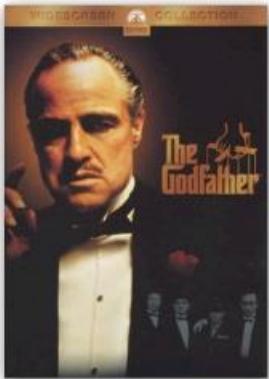
TÀI LIỆU LÍ THUYẾT CTDL & GT

Cấu Trúc Dữ Liệu Cơ Bản

Nội dung

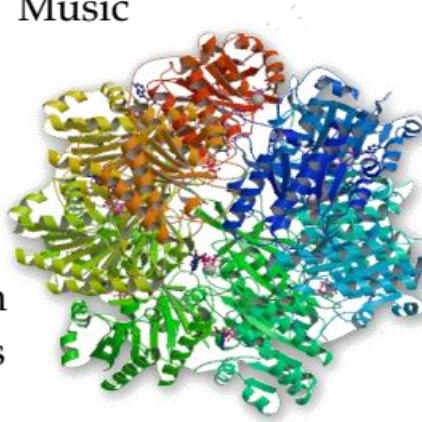
- Giới thiệu dữ liệu và cấu trúc dữ liệu
- Kiểu dữ liệu
 - Kiểu dữ liệu hệ thống
 - Kiểu dữ liệu người dùng
- Kiểu dữ liệu trừu tượng
- Tổng quan về cấu trúc dữ liệu
- Phân loại cấu trúc dữ liệu
- Một số cấu trúc dữ liệu cơ bản

Dữ liệu số



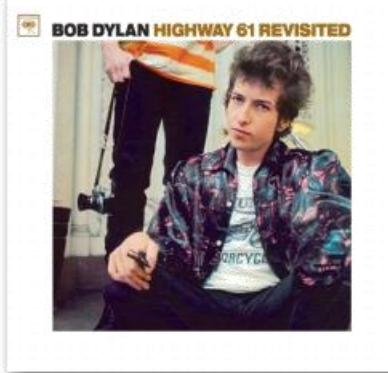
Movies

Protein
Shapes



DNA

```
gatctttta tttaaacat ctcttatta gatctcttat taggatcatg atcctctgtg  
gataagtat tattcacatg gcagatcata taattaagga ggatcgttt tggtgagtga  
ccggtgatcg tattgcgtat aagctggat ctaaatggca tgatatgcac agtcactcg  
cagaatcaag gttttatgt ggatatctac tggtttacc ctgcctttaa gcatagttat  
acacattcgt tcgcgcgatc tttgagctaa ttagagtaaa ttaatccat cttgaccca
```



Music



Photos



Maps

001010100101010100100100101010000010010010100....

RAM

- Dữ liệu chủ yếu chứa trong RAM để xử lý
- **RAM = Kí tự và Con trỏ**

Về mặt vật lý, RAM là một dãy các bit có thể truy xuất ngẫu nhiên

16-bit words	
0	0100101001111011
2	0110111010100000
4	0010000100100011
6	1000010001010001
8	0000000000000100
10	1001010110001010
12	1000000111000001
14	1111111111111111

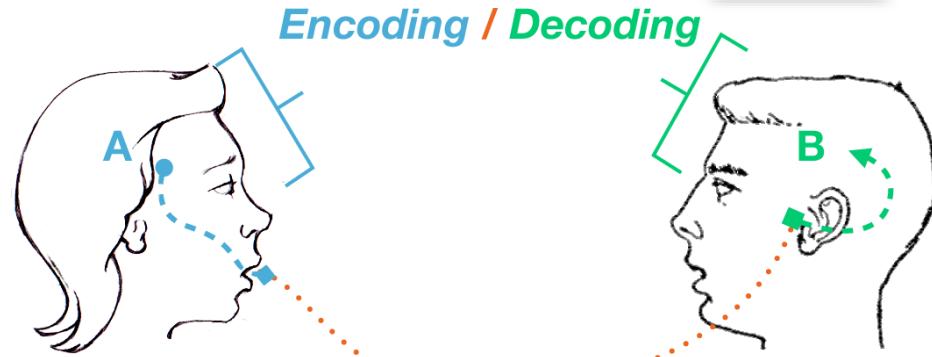
Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	SP
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	\$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29)

Bảng ASCII: sử dụng bit mang ý nghĩa nhất định

Sử dụng bit như
địa chỉ (con trỏ)

Dữ liệu cần phải ...

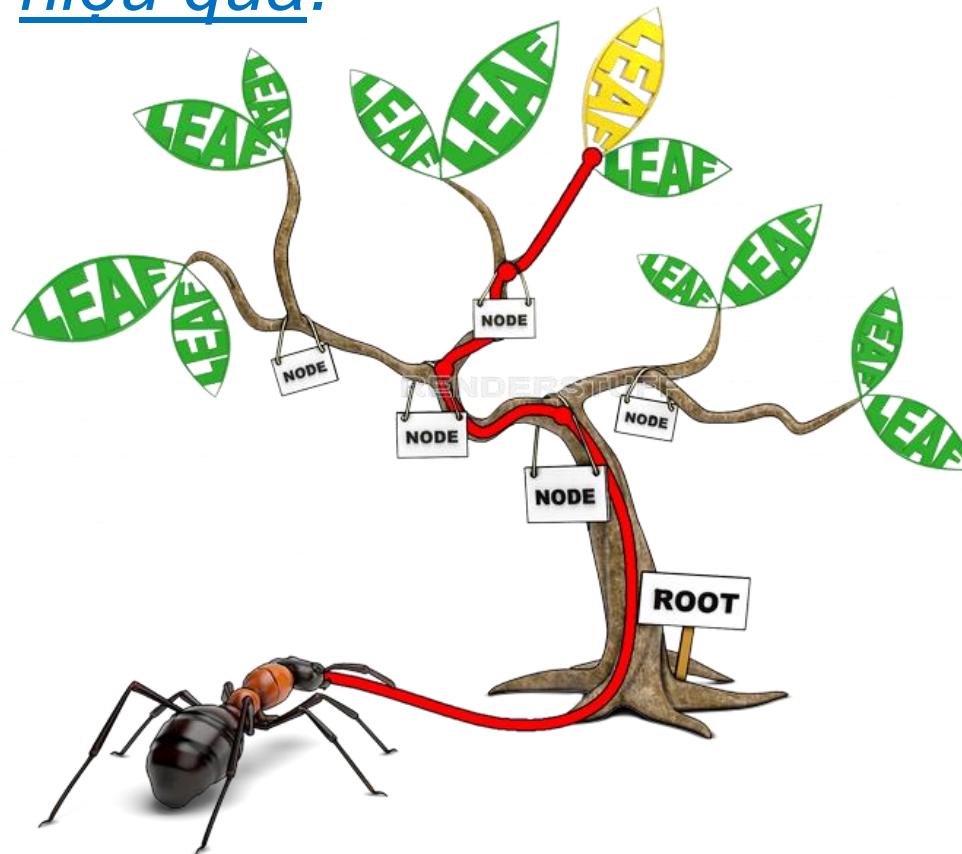
- Mã hóa
 - VD: 100 0001 \leftrightarrow A
- Sắp xếp
 - Chứa có thứ tự trong bộ nhớ/ đĩa
- Truy xuất
 - Thêm, xóa, sửa dữ liệu
 - Tìm kiếm dữ liệu
- Xử lý
 - Các thuật toán: tìm đường đi ngắn nhất, ...



Cấu trúc ...dữ liệu

- Cấu trúc ...dữ liệu?

- Là cách chúng ta tổ chức thông tin để có thể tìm kiếm, cập nhật, thêm và xóa các phần của nó một cách hiệu quả.



Cấu trúc ...dữ liệu (tt)

- Bao gồm các bước:
 - *Như thế nào để chứa đối tượng/tập đối tượng vào bộ nhớ*
 - *Những thực thi nào có thể áp dụng lên dữ liệu*
 - *Thuật toán thực thi*
 - *Tiêu tốn bao nhiêu tài nguyên như thời gian, không gian.*
- Ví dụ, kiểu vector trong C++:
 - Chứa các đối tượng một cách tuần tự
 - Có thể truy xuất, thay đổi, chèn hay xóa đối tượng
 - Thuật toán chèn hay xóa đối tượng sẽ dời các phần tử nếu cần
 - Độ phức tạp không gian: $O(n)$, thời gian truy xuất/cập nhật $O(1)$, Thêm/Xóa là $O(n)$

Ví dụ Cấu Trúc Dữ Liệu

- Google thực hiện lưu trữ dữ liệu và tìm kiếm như thế nào để có tốc độ nhanh?
- Làm sao để chuyển gói tin (file) trên mạng một cách nhanh chóng?
- Hệ điều hành điều phối bộ nhớ như thế nào?
- Trong game, bằng cách nào máy tính xác định phần cần hiển thị trên màn hình để giả lập không gian 3D.

Ví dụ của bạn?

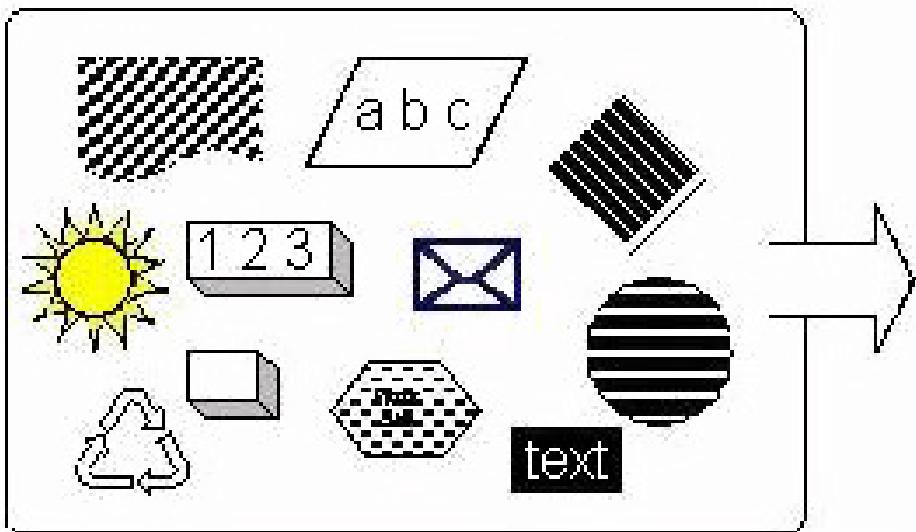
Nội dung

- Giới thiệu dữ liệu và cấu trúc dữ liệu
- **Kiểu dữ liệu**
 - Kiểu dữ liệu hệ thống
 - Kiểu dữ liệu người dùng
- Kiểu dữ liệu trừu tượng
- Tổng quan về cấu trúc dữ liệu
- Phân loại cấu trúc dữ liệu
- Một số cấu trúc dữ liệu cơ bản

Kiểu dữ liệu

Vậy thực sự, máy tính chứa gì?

Your Data

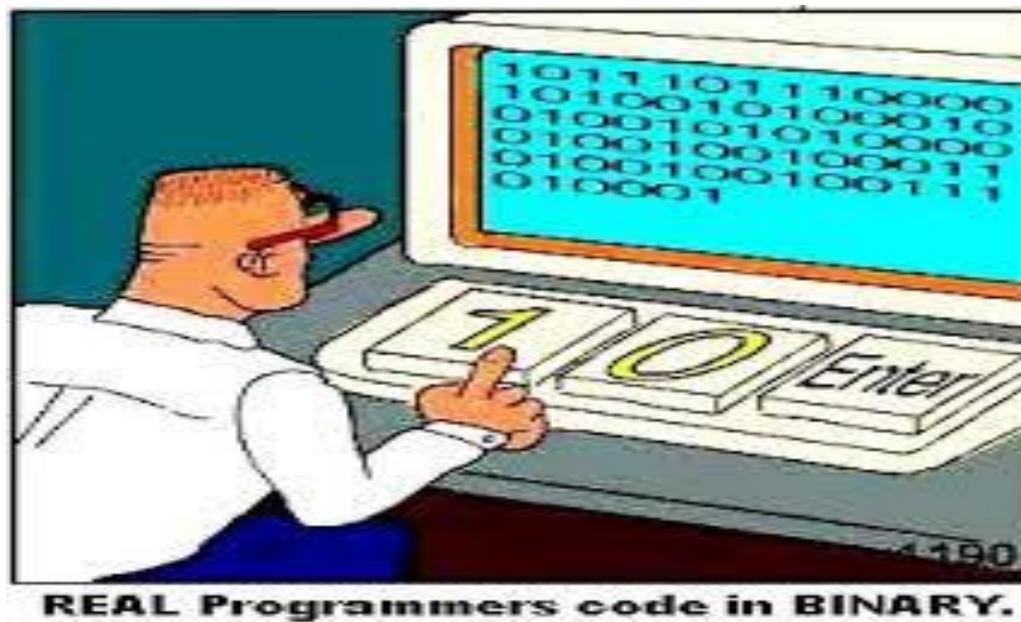


Computer Data

```
01110101011010101  
10100101011010101  
01010101011010101  
01000101011010101  
01101010101001100  
00101011101100111  
10101001010101010
```

Kiểu dữ liệu (tt)

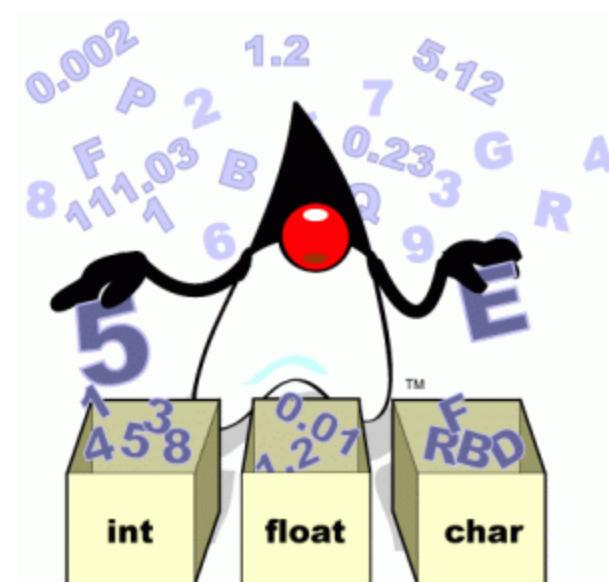
- Hãy tưởng tượng, bạn chỉ có kiểu dữ liệu với hai số [0,1], hãy viết chương trình để giải quyết một bài toán nào đó?



→ Để dễ dàng, ngôn ngữ lập trình và trình biên dịch cung cấp **kiểu dữ liệu**.

Kiểu dữ liệu (tt)

- *Kiểu dữ liệu* trong ngôn ngữ lập trình là tập dữ liệu với các giá trị có đặc trưng đã được định trước.
- Ví dụ:
 - Integer: ...0, 1, 2, ...
 - Floating Point: ...
 - Character: ...
 - String: ...



Kiểu dữ liệu (tt)

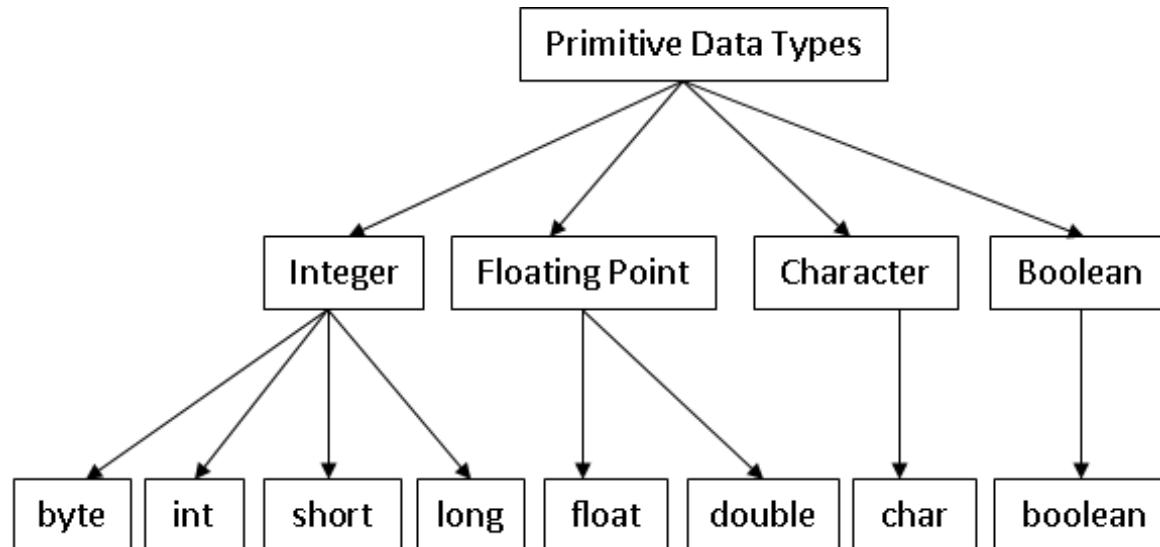
- Cách thức ngôn ngữ lập trình và trình biên dịch cung cấp *kiểu dữ liệu*:
 - **Integer** chiếm 2 byte (phụ thuộc vào trình biên dịch) trong bộ nhớ nghĩa là chúng liên kết 16 bit lại và gọi nó là *integer*.
 - **Float** chiếm 4 byte nghĩa là chúng liên kết 32 bit lại và gọi nó là *float*.

Phân loại kiểu dữ liệu

- Về cơ bản, kiểu dữ liệu được phân làm 2 loại:
 - Kiểu được định nghĩa bởi hệ thống (*primitive data type*)
 - Kiểu do người dùng định nghĩa (*user defined data type*)

Kiểu dữ liệu của hệ thống

- *Kiểu dữ liệu của hệ thống* như int, float, char, double, bool, ...
- Số lượng bit của mỗi kiểu phụ thuộc vào ngôn ngữ lập trình, trình biên dịch và hệ điều hành.
 - Ví dụ: int có thể 2 byte, 4 byte, 8 byte



Kiểu dữ liệu người dùng định nghĩa

- Khi kiểu dữ liệu hệ thống không đủ, hầu hết các ngôn ngữ lập trình đều cho phép người dùng định nghĩa *kiểu dữ liệu mới* dựa trên các kiểu dữ liệu sẵn có.
- Ví dụ:

```
struct myType {  
    int x;  
    float y;  
    char z;  
};
```

Cấu trúc dữ liệu và Kiểu dữ liệu

Cấu Trúc Dữ Liệu	Kiểu Dữ Liệu
<ul style="list-style-type: none">- Tập trung mô tả cách tổ chức dữ liệu.- Mô tả thực thi ở mức phức tạp hơn	<ul style="list-style-type: none">- Tập trung mô tả tập các giá trị có thể có.

Ví dụ:

int là *kiểu dữ liệu* nhưng không là *cấu trúc*
struct point { int x,y;} vừa là *kiểu dữ liệu* cũng vừa là *cấu trúc*.

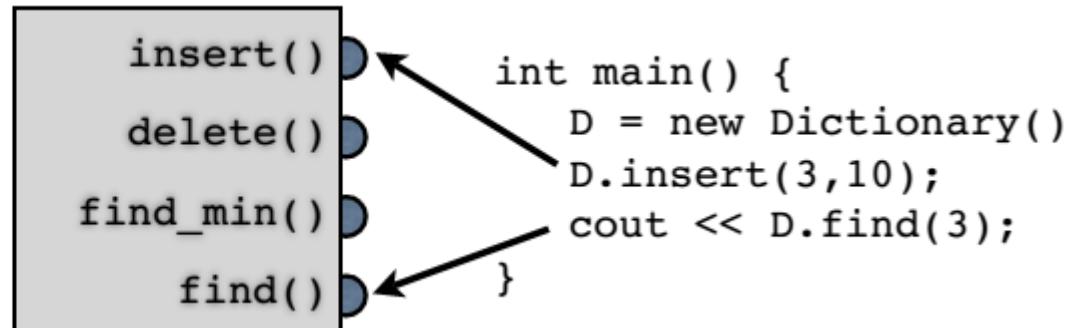
Nội dung

- Giới thiệu dữ liệu và cấu trúc dữ liệu
- Kiểu dữ liệu
 - Kiểu dữ liệu hệ thống
 - Kiểu dữ liệu người dùng
- **Kiểu dữ liệu trừu tượng**
- Tổng quan về cấu trúc dữ liệu
- Phân loại cấu trúc dữ liệu
- Một số cấu trúc dữ liệu cơ bản

Kiểu dữ liệu trừu tượng (ADT)

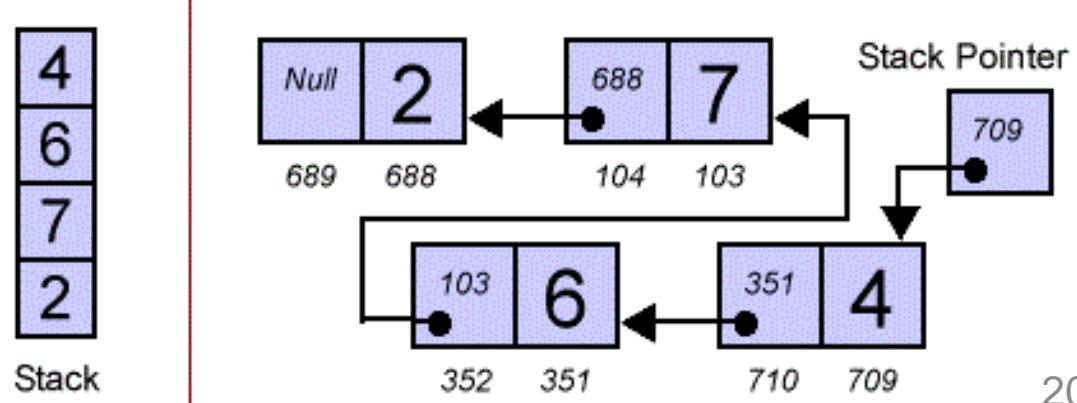
- Với kiểu dữ liệu hệ thống (int, float, ...) cũng hỗ trợ những thực thi cơ bản:
 - +, -, *, /, div, mod, >, <, !=, ...
- Với kiểu dữ liệu người dùng, ta cũng cần định nghĩa các thực thi của chúng.
→ Người ta gọi đó là *kiểu dữ liệu trừu tượng* (Abstract Data Types)

```
class Dictionary {  
    Dictionary();  
    void insert(int x, int y);  
    void delete(int x);  
    ...  
}
```

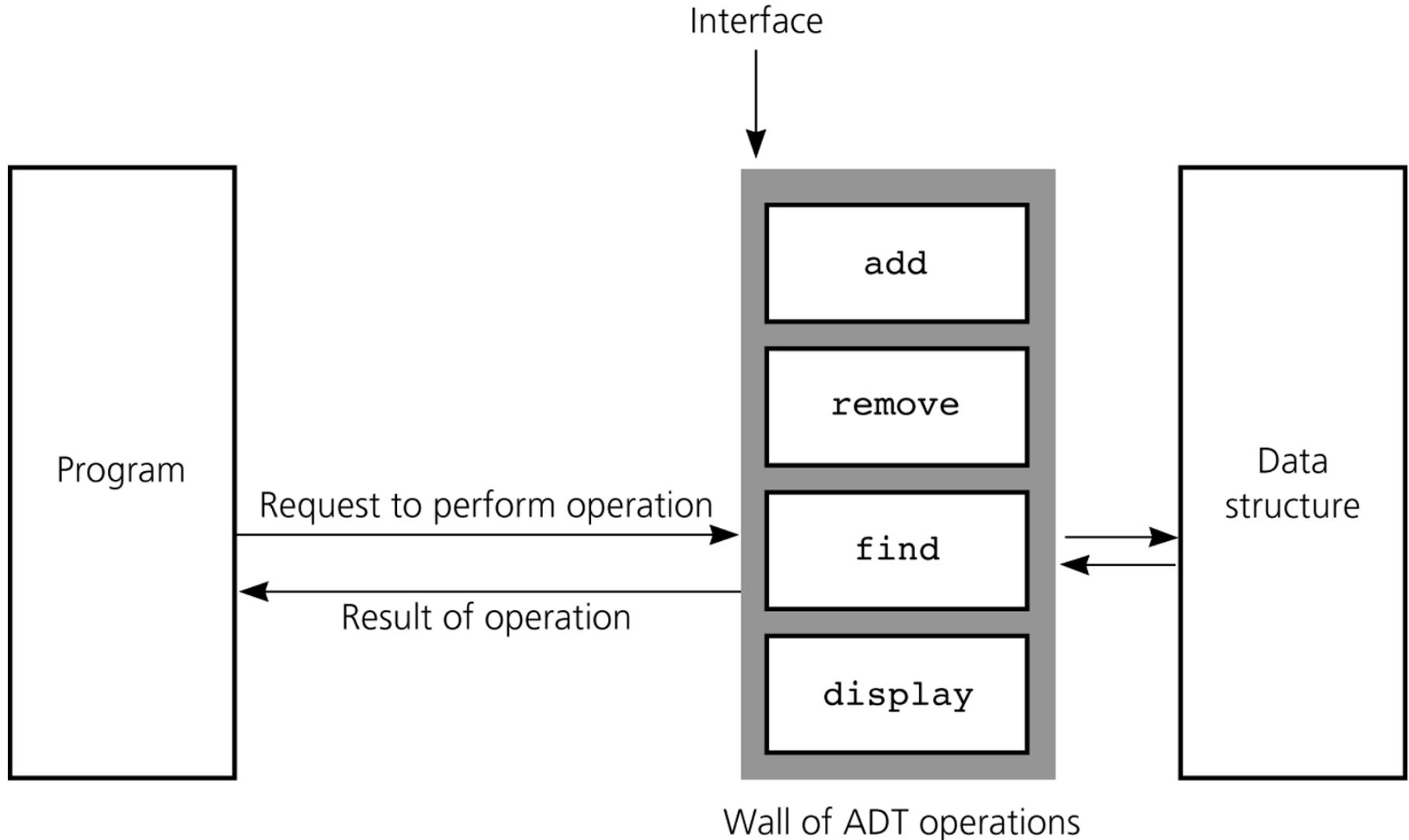


Kiểu dữ liệu trùu tượng (tt)

- *Kiểu dữ liệu trùu tượng* gồm:
 - Phần dữ liệu (Value - V)
 - Phần thực thi (Operator - O)
- Chỉ tập trung mô tả chứ không đi sâu vào cách thức thực thi chi tiết trong code.
 - Có nhiều cách cài đặt khác nhau. Ví dụ: ngăn xếp (stack) có thể sử dụng mảng hoặc danh sách liên kết.



Kiểu DL triều tượng

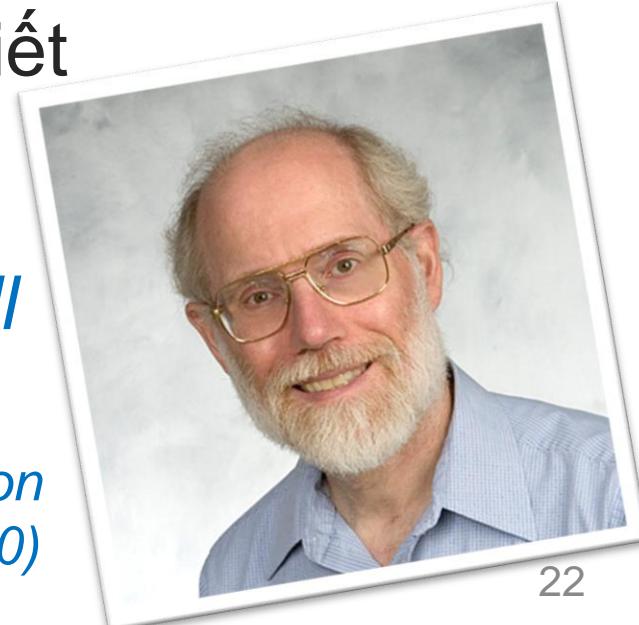


Kiểu dữ liệu trừu tượng (tt)

- Đặc điểm:
 - ADT chứa dữ liệu và cho phép rất nhiều thực thi trên dữ liệu để truy xuất và thay đổi nó.
 - ADT không phải hoàn toàn là lớp (class), nó giống như lớp trừu tượng (abstract class) hay giao diện (interface)
 - Không cần thực thi ở mức chi tiết

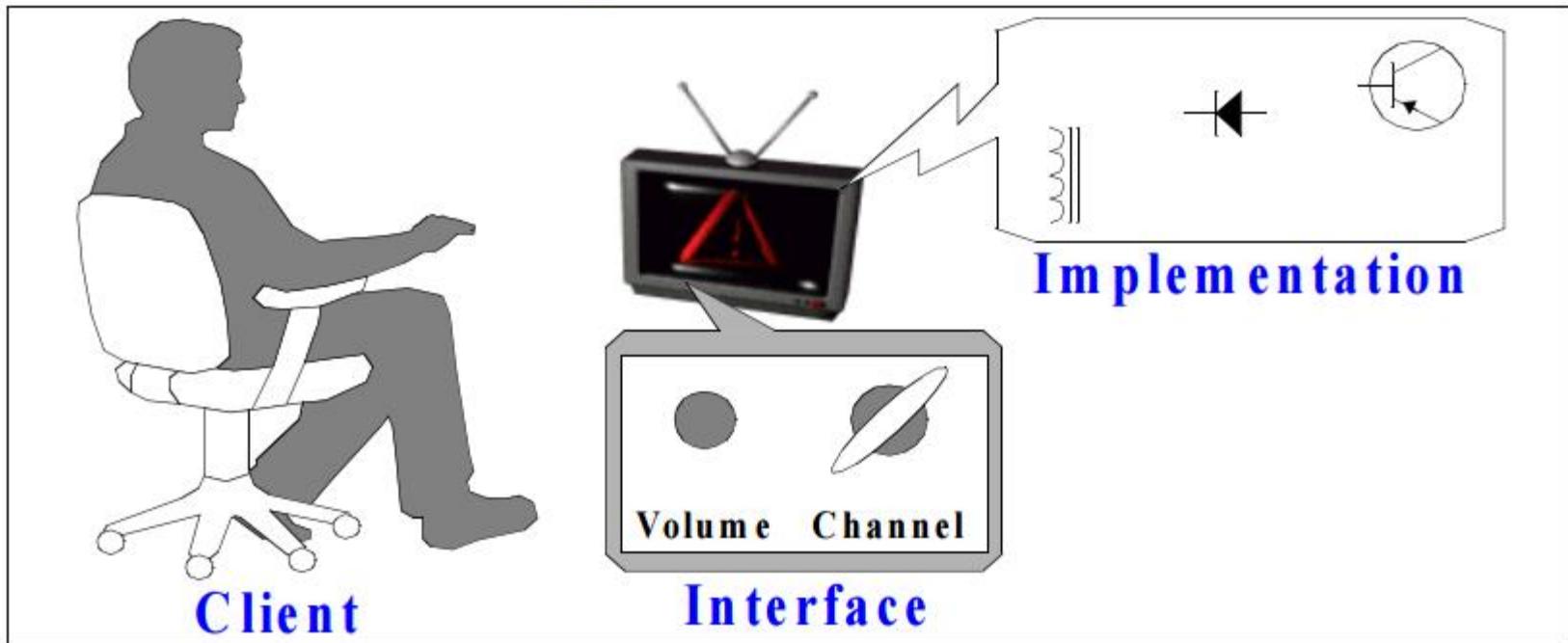
“Get your data structures correct first, and the rest of the program will write itself”

*David S. Johnson
(winner of Knuth's Prize in 2010)*



Kiểu DL trừu tượng

- Mỗi ADT nên như một bản hợp đồng (contract hay specification) mà:
 - Xác định tập dữ liệu hợp lệ
 - Xác định thực thi:
 - Tên của nó
 - Loại tham số
 - Loại kết quả trả về nếu có
 - Cách cung cấp có thể quan sát được của nó
 - Không xác định:
 - Thể hiện dữ liệu
 - Thuật toán để thực hiện các thực thi



- Separate implementation from specification
 - INTERFACE:** specify the allowed operations
 - IMPLEMENTATION:** provide code for ops
 - CLIENT:** code that uses them

Ví dụ

- Danh sách (List):

- Ý niệm:

- Danh sách là một tập hợp hữu hạn các phần tử có cùng một kiểu. Ta biểu diễn danh sách như là một chuỗi các phần tử của nó: a_1, a_2, \dots, a_n với $n \geq 0$. Nếu $n=0$ ta nói danh sách rỗng. Nếu $n > 0$ ta gọi a_1 là phần tử đầu tiên và a_n là phần tử cuối cùng của danh sách. Số phần tử của danh sách ta gọi là độ dài của danh sách.

- Định nghĩa ADT:

Ví dụ

- Định nghĩa ADT cho danh sách:
 - Dữ liệu: tập các phần tử cùng kiểu
 - Thực thi:
 - **insertList(x, p, L)**: thêm phần tử x vào vị trí p trong danh sách L
 - **locate(x, L)**: xác định vị trí của phần tử x trong danh sách L
 - **retrieve(p, L)**: lấy phần tử ở vị trí p
 - **deleteList(p,L)**: xóa phần tử ở vị trí p
 - **next(p, L)**: lấy phần tử kế tiếp vị trí p
 - **previous(p,L)**: lấy phần tử trước vị trí p
 - **first(L)**: lấy phần tử đầu tiên

Ví dụ

- Hãy đặc tả ADT cho:

Danh sách sinh viên

Ví dụ

- ADT cho danh sách sinh viên:
 - Dữ liệu: name, age, sex, address
 - Thự thi:
 - createStudent (name, age, sex, address)
 - compare (student1, student2)
 - getName (student)
 - getAge (student)
 - getSex (student)
 - getAdd (student)

Ví dụ

- Hãy đặc tả ADT cho:

Danh sách lớp học

Ví dụ

- ADT cho danh sách lớp học:
 - Dữ liệu: className, numberStudent, studentArr, address
 - Thực thi:
 - addStudent (studentClass, student)
 - findStudent (studentClass, student)
 - deleteStudent (studentClass, student)
 - getClassName (studentClass)
 - getNumberStudent (studentClass)
 - getStudentArr (studentClass)
 - getClassAddress (studentClass)

Ví dụ

- Hãy đặc tả ADT cho:

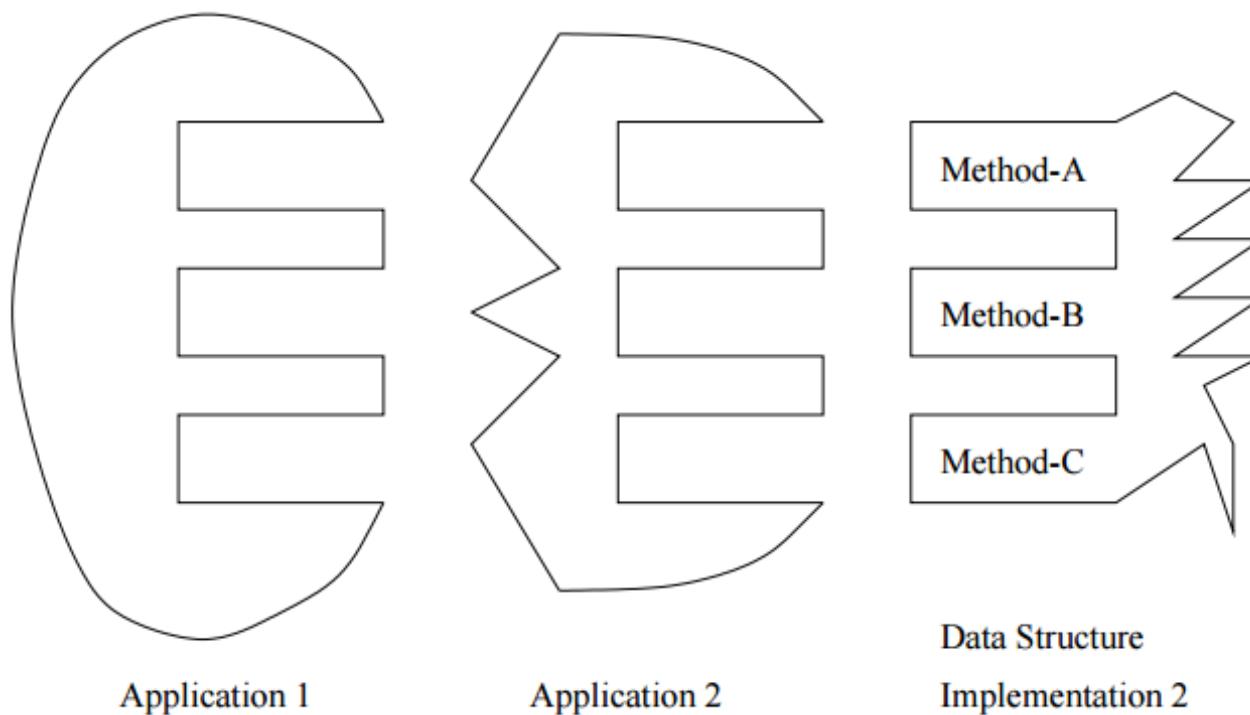
Tùy Diễn

Ví dụ

- ADT cho danh sách lớp học:
 - Dữ liệu: các từ (chuỗi kí tự)
 - Thực thi:
 - find (key)
 - insert(key,data)
 - remove(key)

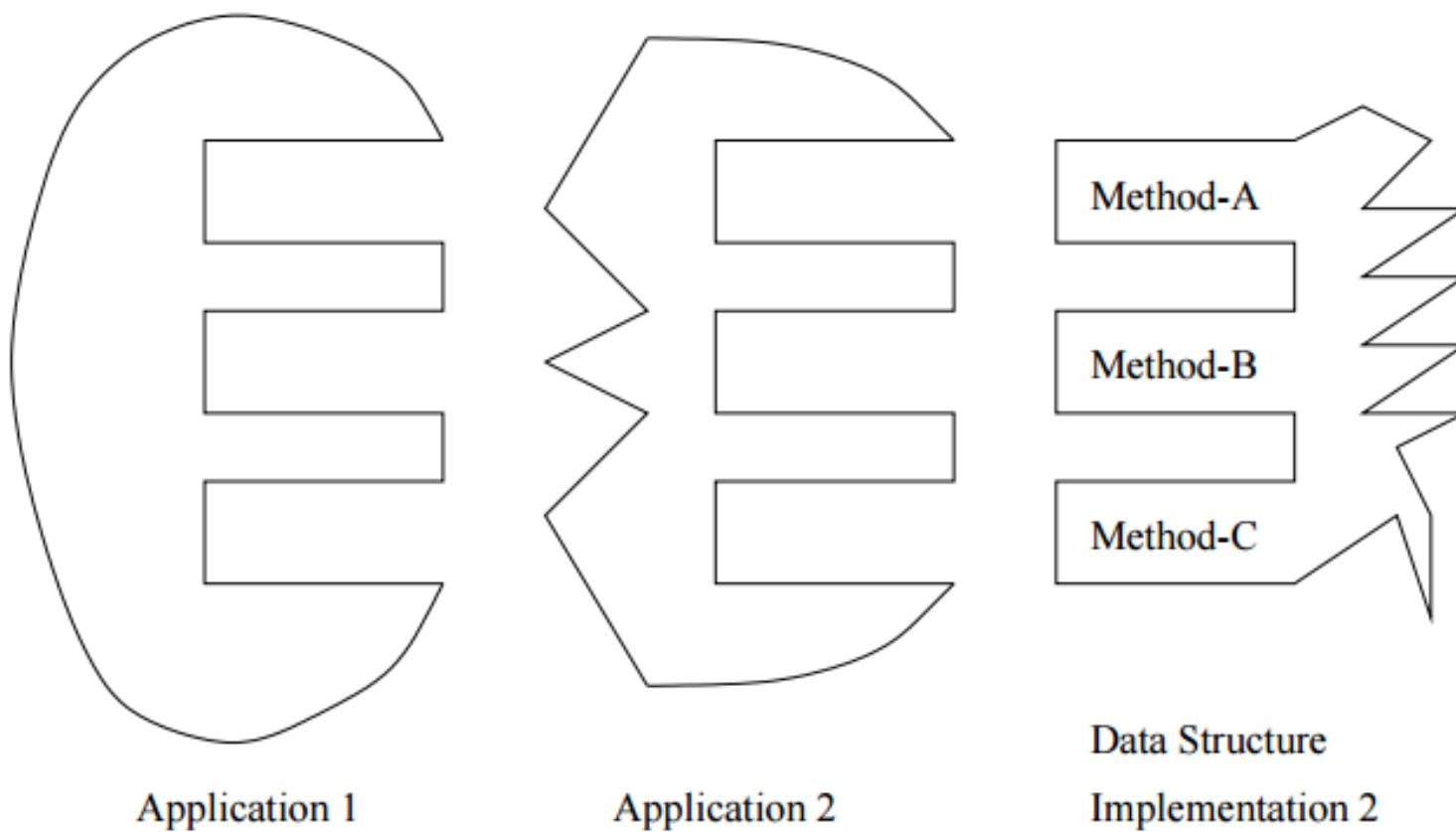
Ý nghĩa của ADT

- Giúp cho việc chia nhỏ bài toán
- Che chi tiết thực thi
- Tái sử dụng



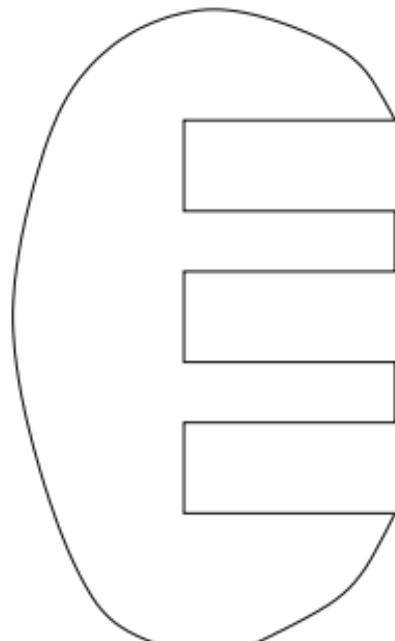
Ý nghĩa của ADT

- Che chi tiết thực thi
- Tái sử dụng

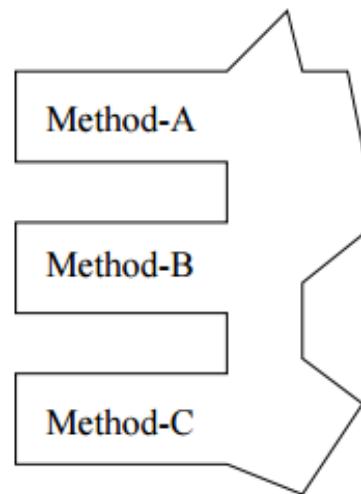


Ý nghĩa của ADT

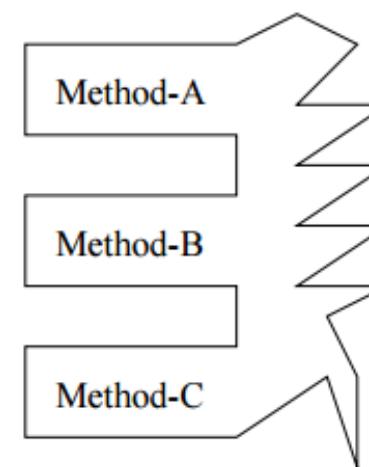
- Che chi tiết thực thi:
 - Người sử dụng không cần biết chi tiết, chỉ cần biết nó làm được gì
 - Người lập trình có thể thay thế thực thi mà không ảnh hưởng ứng dụng sử dụng nó



Application



Data Structure
Implementation 1



Data Structure
Implementation 2

Ý nghĩa của ADT – ví dụ 1

Java Interface for ADT Dictionary

```
public interface Dictionary {  
  
    public Object find(Object key);  
  
    public void insert(Object key, Object data)  
        throws DuplicatedKeyException;  
  
    public void remove(Object key)  
        throws NoKeyException;  
  
}
```

Ý nghĩa của ADT – ví dụ 1

A Java Implementation for ADT Dictionary

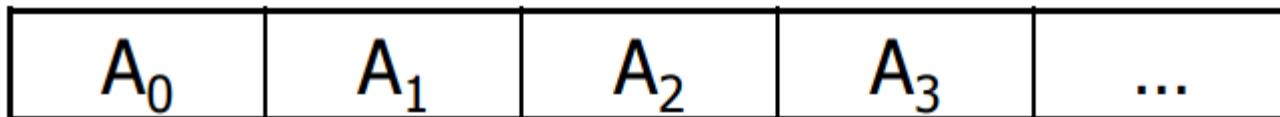
```
public class LinkedListDictionary implements Dictionary {  
    protected int size;  
    protected DNode head, tail;  
    public LinkedListDictionary() {  
        size = 0;  
        head = new DNode(null, null, null);  
        tail = new DNode(null, null, null);  
        head.setNext(tail);  
    }  
    public Object find(Object key) {  
        if (size == 0) return null;  
        else {  
            :  
        }  
    }  
    public void insert (Object key, Object data) throws  
        DuplicatedKeyException {  
        :  
    }
```

Ý nghĩa của ADT – ví dụ 2

- ADT của danh sách (list):
 - Dữ liệu: a₀, a₁, ..., a_n
 - Thực thi:
 - insert(X,k),
 - remove(k),
 - find(X),
 - findKth(k),
 - printList()

Ý nghĩa của ADT – ví dụ 2

- Cài đặt ADT danh sách sử dụng mảng:



Operations

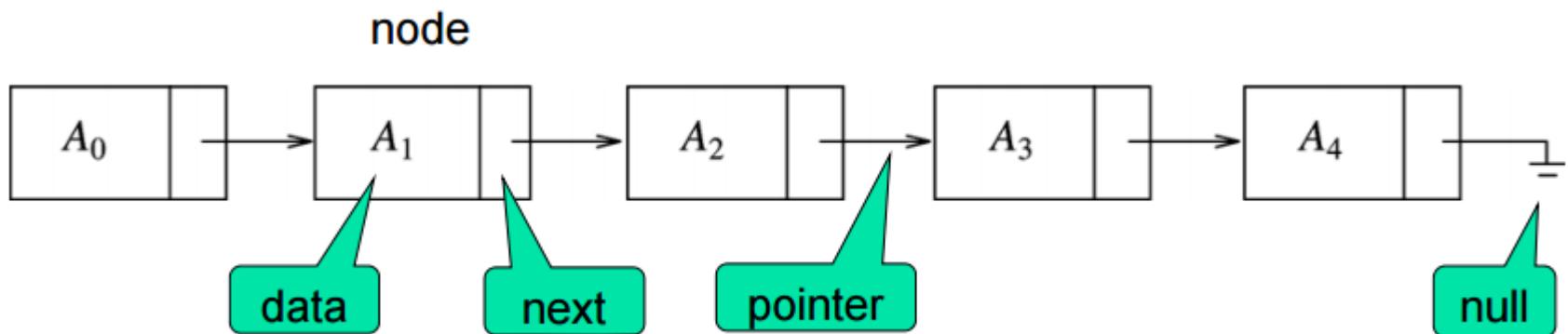
- $\text{insert}(X, k) : O(N)$
- $\text{remove}(k) : O(N)$
- $\text{find}(X) : O(N)$
- $\text{findKth}(k) : O(1)$
- $\text{printList()} : O(N)$

• Read as “order N”
(means that runtime is proportional to N)

• Read as “order 1”
(means that runtime is a constant – i.e., not dependent on N)

Ý nghĩa của ADT – ví dụ 2

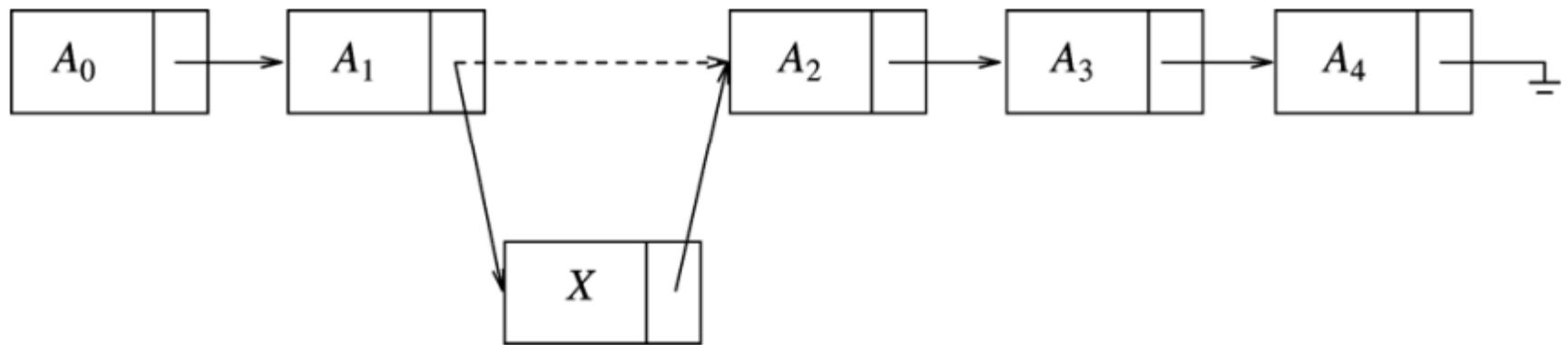
- Cài đặt ADT danh sách sử dụng danh sách liên kết đơn:
 - Các phần tử không chứa liên tục
 - Mỗi phần tử có thêm con trỏ đến phần tử tiếp theo.



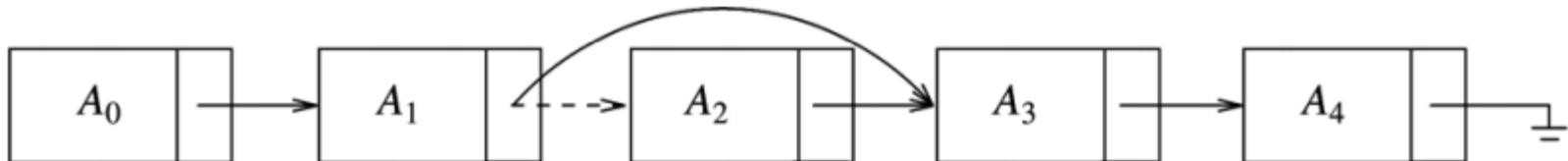
Ý nghĩa của ADT – ví dụ 2

- Thực thi ADT danh sách sử dụng danh sách liên kết đơn:

- **Insert(X,A) – O(1)**



- **Remove(A) – O(1)**

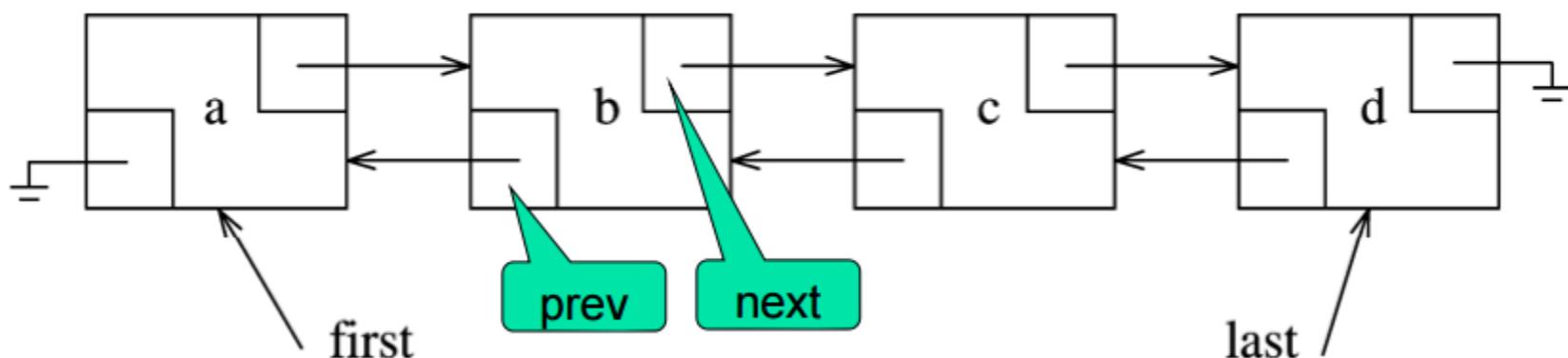


Ý nghĩa của ADT – ví dụ 2

- Thực thi ADT danh sách sử dụng danh sách liên kết đơn:
 - $\text{find}(X)$ – $O(N)$
 - $\text{findKth}(k)$ – $O(N)$
 - $\text{printList}()$ – $O(N)$

Ý nghĩa của ADT – ví dụ 2

- Cài đặt ADT danh sách sử dụng danh sách liên kết đôi:
 - Tương tự danh sách liên kết đơn nhưng bổ sung thêm con trỏ ngược lại phần tử trước



Ý nghĩa của ADT – ví dụ 2

- Thực thi ADT danh sách sử dụng danh sách liên kết:
 - $\text{find}(X)$ – $O(N)$
 - $\text{findKth}(k)$ – $O(N)$
 - $\text{printList}()$ – $O(N)$

Kiểu dữ liệu trừu tượng (tt)

- Các kiểu dữ liệu trừu tượng phổ biến:
 - Danh sách liên kết (linked list)
 - Ngăn xếp (Stacked)
 - Hàng đợi (Queues)
 - Hàng đợi ưu tiên (Priority Queues)
 - Cây (Tree)
 - Từ điển (Dictionary)
 - Bảng băm (Hash table)
 - Đồ thị (Graph)...

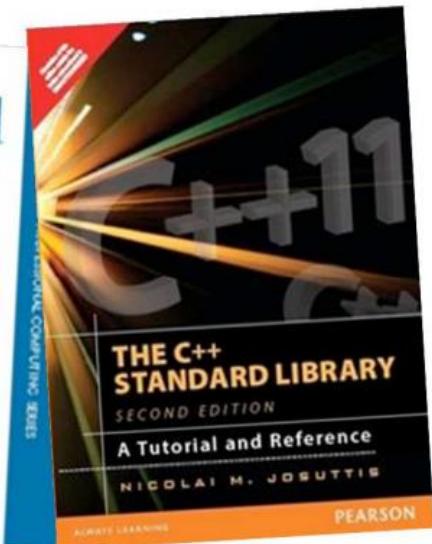
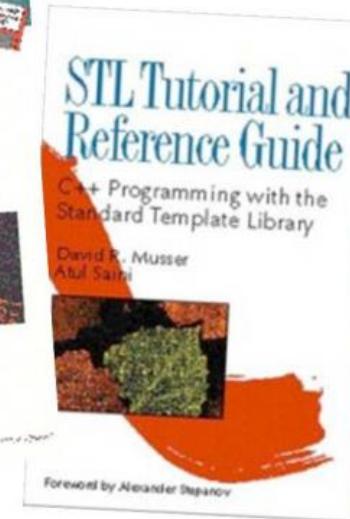
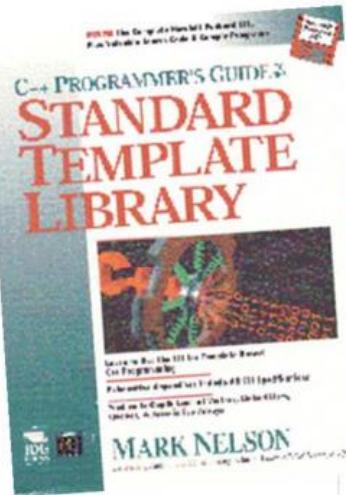
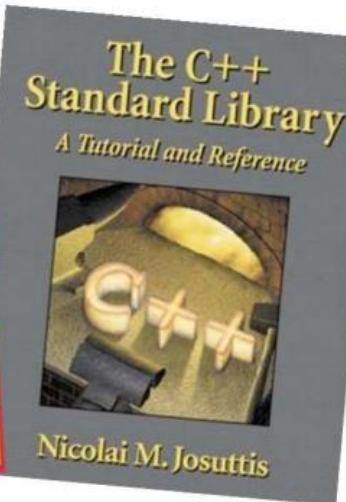
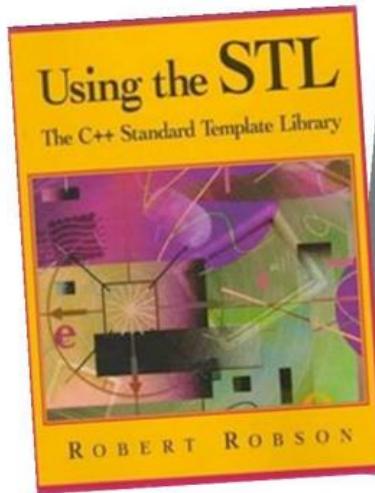
Kiểu dữ liệu trùu tượng (tt)

- Một số thao tác phổ biến:

- Trả về phần tử xác định
- Thêm phần tử vào
- Xóa phần tử
- Số phần tử tồn tại
- Sao chép
- Tìm kiếm
- Sắp xếp
- Phân chia
- Cung cấp nhóm phần tử con
-

Built-in ADT

- Các ngôn ngữ lập trình cấp cao thường cài đặt sẵn các dữ liệu trừu tượng trên.
 - Standard Template Library (C++ STL)
 - Java Collections Framework (java.util.Stack, ...)

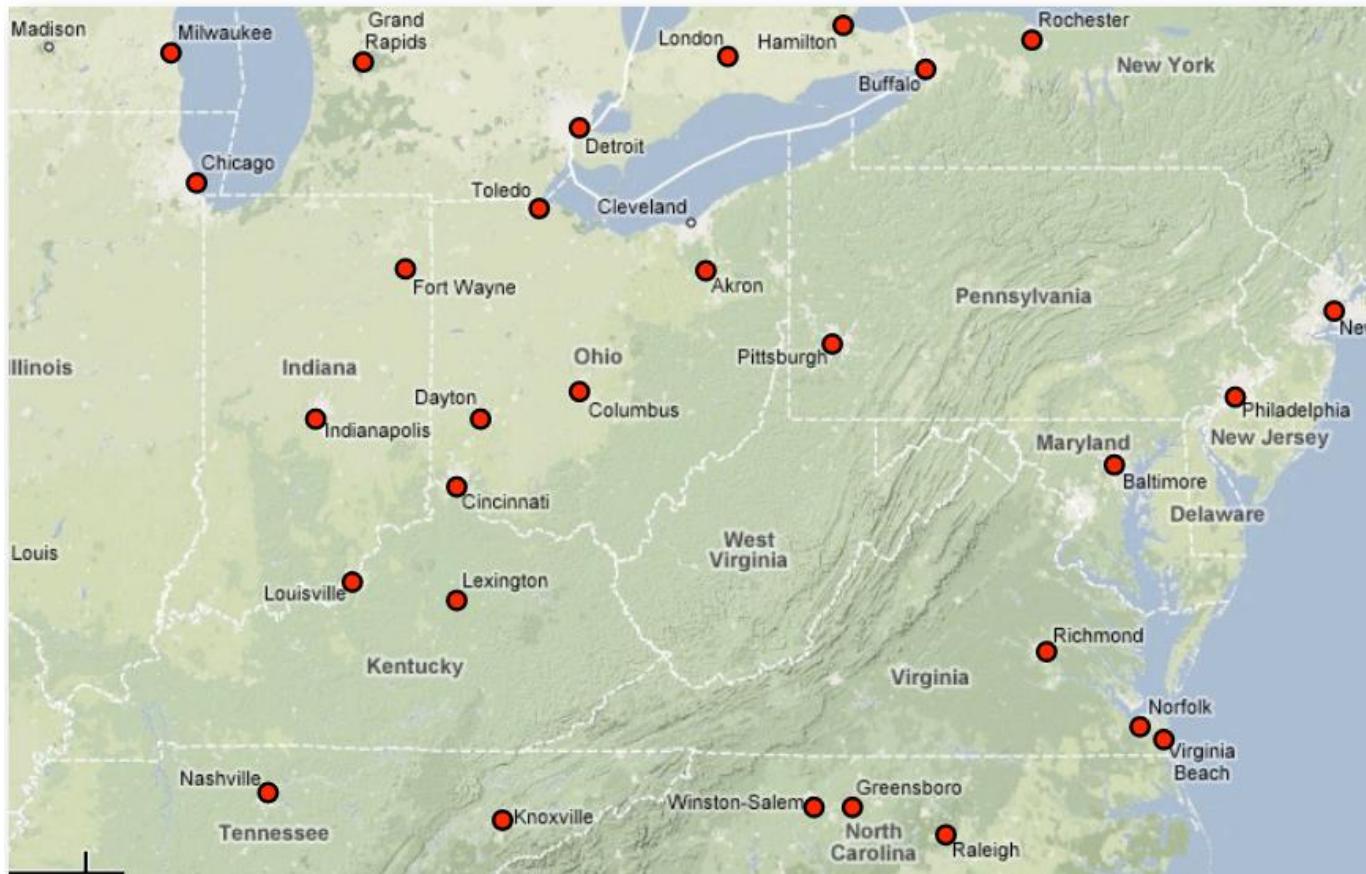


Thảo luận

- Khi đã có các thư viện cài đặt sẵn, vậy:
 - Tại sao cần học kiểu dữ liệu trừu tượng (ADT)?
 - Có nên cài đặt lại ADT?

Bài tập 1

- Giả sử bạn tạo ra Google Map và bạn đang chứa dữ liệu về các thành phố (vị trí, tọa độ, dân số,...)



Những thực thi nào mà cấu trúc dữ liệu của bạn có thể hỗ trợ?

Bài tập 2

Hãy mô tả và liệt kê các thao tác của các kiểu dữ liệu đã được học.

Nội dung

- Giới thiệu dữ liệu và cấu trúc dữ liệu
- Kiểu dữ liệu
 - Kiểu dữ liệu hệ thống
 - Kiểu dữ liệu người dùng
- Kiểu dữ liệu trừu tượng
- **Tổng quan về cấu trúc dữ liệu**
- Phân loại cấu trúc dữ liệu
- Một số cấu trúc dữ liệu cơ bản

Tổng quan về một số cấu trúc dữ liệu

Phân tích điểm mạnh và yếu của một số cấu trúc dữ liệu đã được học. Ví dụ:

- Mảng
- Mảng đã có thứ tự
- Stack, Queue
- Danh sách liên kết
- Cây nhị phân
- Cây đỏ đen
- Bảng băm
- Heap
- Đồ thị

....

Tổng quan về một số cấu trúc dữ liệu

CTDL	Điểm mạnh	Điểm Yếu
Mảng	<ul style="list-style-type: none">- Thêm nhanh- Truy xuất đến phần tử nhanh nếu có chỉ mục	<ul style="list-style-type: none">- Tìm kiếm chậm- Xóa chậm- Kích thước cố định
Mảng có thứ tự	<ul style="list-style-type: none">- Tìm kiếm nhanh hơn mảng không thứ tự	<ul style="list-style-type: none">- Thêm, xóa chậm- Kích thước cố định
Stack	<ul style="list-style-type: none">- Cung cấp cách thức truy xuất FILO	<ul style="list-style-type: none">- Truy xuất chậm đến các phần tử khác
Queue	<ul style="list-style-type: none">- Cung cấp cách thức truy xuất FIFO	<ul style="list-style-type: none">- Truy xuất chậm đến các phần tử khác

Tổng quan về một số cấu trúc dữ liệu

CTDL	Điểm mạnh	Điểm Yếu
Danh sách liên kết	- Thêm, xóa nhanh	- Tìm kiếm chậm
Cây nhị phân	- Tìm kiếm nhanh. - Thêm, xóa nhanh nếu cây còn cân bằng	- Xóa phức tạp
Cây đỏ đen	- Tìm, thêm, xóa nhanh do cây luôn cân bằng	- Phức tạp
Bảng băm	- Truy xuất nhanh nếu biết khóa. - Chèn nhanh	- Truy xuất chậm nếu không biết khóa. - Sử dụng bộ nhớ không hiệu quả

Nội dung

- Giới thiệu dữ liệu và cấu trúc dữ liệu
- Kiểu dữ liệu
 - Kiểu dữ liệu hệ thống
 - Kiểu dữ liệu người dùng
- Kiểu dữ liệu trừu tượng
- Tổng quan về cấu trúc dữ liệu
- **Phân loại cấu trúc dữ liệu**
- Một số cấu trúc dữ liệu cơ bản

Phân loại cấu trúc dữ liệu

- Nếu dựa trên *cách tổ chức* các phần tử, cấu trúc dữ liệu có thể phân làm 2 loại:
 - **Cấu trúc dữ liệu tuyến tính** (linear data structure): các phần tử được *truy xuất* một cách *tuần tự* như *không bắt buộc phải chứa liên tục*.
 - VD: danh sách liên kết, ngăn xếp, hàng đợi, ...
 - **Cấu trúc dữ liệu phi tuyến** (non-linear data structure): các phần tử được *chứa/truy xuất theo thứ tự phi tuyến*.
 - VD: Cây, đồ thị, ...

Phân loại cấu trúc dữ liệu

- Nếu dựa trên *cách phân bổ* các phần tử, cấu trúc dữ liệu có thể phân làm 2 loại:
 - **Cấu trúc cấp phát liên tục** (contiguously-allocated structure): các phần tử được *phân bổ trên một vùng nhớ đơn*.
 - VD: mảng, ma trận, heap, bảng băm, ...
 - **Cấu trúc dữ liệu liên kết** (linked data structure): các phần tử được *chứa trong các vùng nhớ phân biệt* được *liên kết với nhau bằng con trỏ*.
 - VD: danh sách liên kết, cây, đồ thị, ...

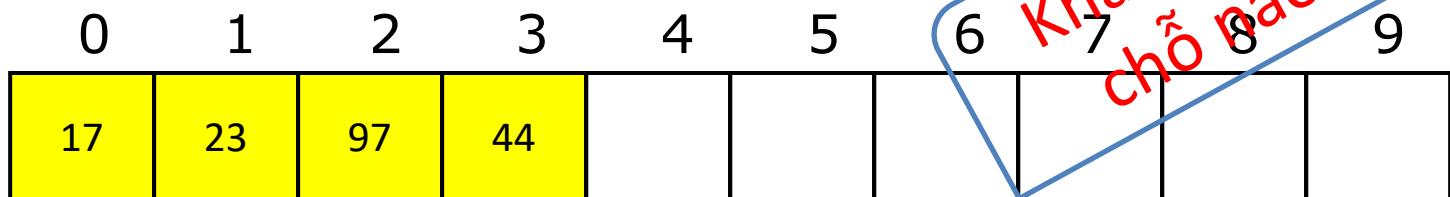
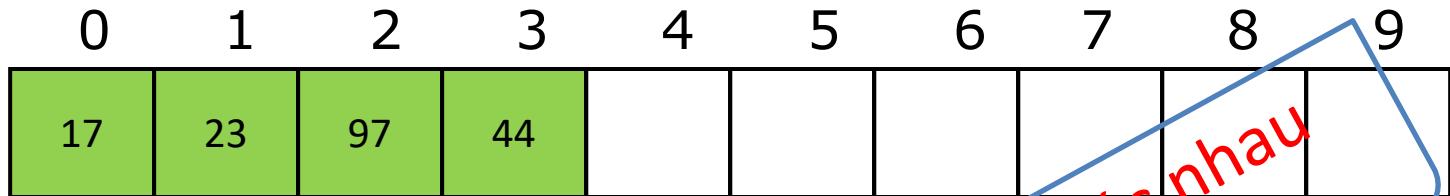
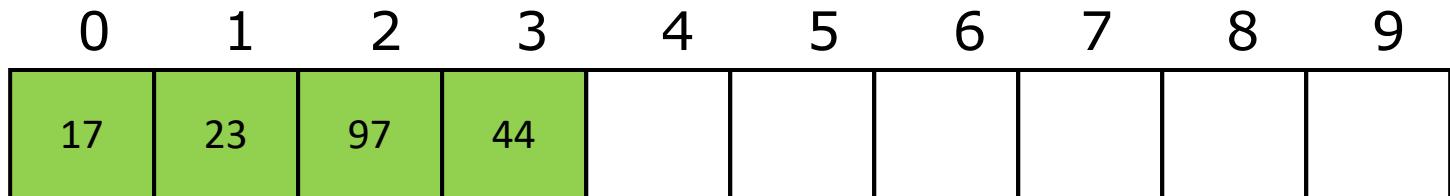
Nội dung

- Giới thiệu dữ liệu và cấu trúc dữ liệu
- Kiểu dữ liệu
 - Kiểu dữ liệu hệ thống
 - Kiểu dữ liệu người dùng
- Kiểu dữ liệu trừu tượng
- Tổng quan về cấu trúc dữ liệu
- Phân loại cấu trúc dữ liệu
- **Một số cấu trúc dữ liệu cơ bản**

ADT: Bag, Set và List

- CSDL đơn giản nhất là **Bag** (túi)
 - Các phần tử có thể được thêm, xóa và truy cập
 - Không quan tâm đến thứ tự các phần tử
 - Cho phép trùng
- **Set** (tập hợp)
 - Tương tự bag, nhưng không cho phép trùng
 - Thực hiện các phép hội, giao, hiệu và tập con
- **List** (danh sách)
 - Các phần tử có thứ tự và cho phép trùng nhau

ADT: Mảng, Stack, Queue



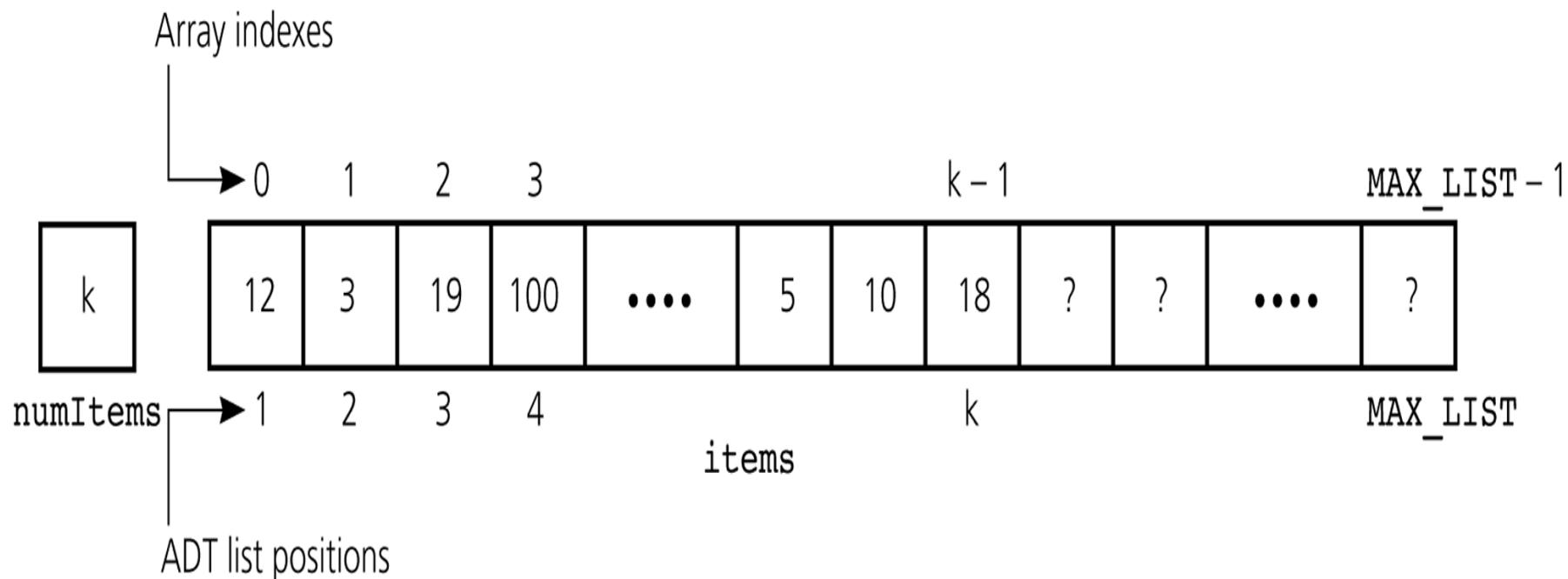
public static int
public static int

M[10];
nM = 10;

public static int
public static int
public static int
public static int

S[10];
nS = 4;
Q[10];
nQ = 4;

Mảng



- **Mảng:**
 - Thuộc loại cấu trúc dữ liệu cấp phát liên tục
 - Kích thước cố định
 - Mỗi phần tử được đặt ở chỉ mục/địa chỉ xác định
- Ví dụ: nhà đường phố với địa chỉ là chỉ mục

Mảng (tt)

- *Thuận lợi của mảng:*

- Truy xuất tới phần tử với thời gian hằng: do chỉ cần dựa trên chỉ mục/địa chỉ là có thể nhảy đến.
- Hiệu quả không gian: do kích thước cố định
- Xác định bộ nhớ dễ dàng: do phần tử được đặt liên tục.

- *Bất lợi của mảng:*

- Không thể thay đổi kích thước → giải quyết bằng mảng động → bất lợi khi phải tiêu tốn thời gian sao chép khi cấp phát thêm phần tử.

- *Độ phức tạp khi tìm kiếm:* $O(n)$

Các thao tác của mảng

- Các thao tác cơ bản của mảng:
 - Khởi tạo
 - Thêm
 - Xóa
 - Sửa
 - Tìm kiếm
- Hãy dùng mã giả để mô tả các bước thực hiện thao tác trên

Mảng 2 chiều

- Mảng 2 chiều được ánh xạ vào bộ nhớ liên tục theo 2 cách:

	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20

Row-major order

	1	2	3	4	5
1	1	5	9	13	17
2	2	6	10	14	18
3	3	7	11	15	19
4	4	8	12	16	20

Column-major order

$$\text{Addr}(i,j) = \text{Base} + 5(i-1) + (j-1)$$

$$\text{Addr}(i,j) = \text{Base} + (i-1) + 4(j-1)$$

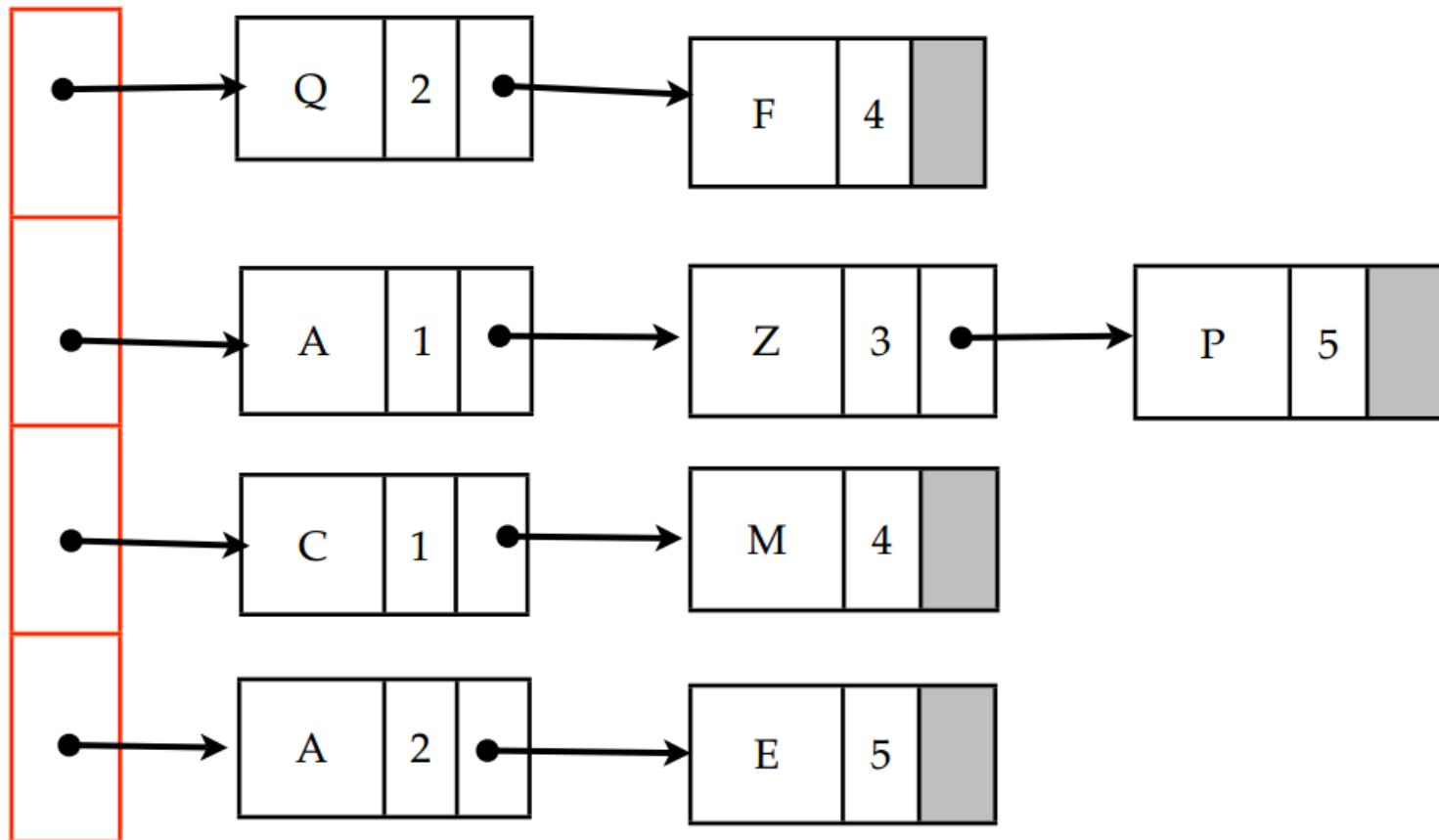
Ma trận thưa

- Rất nhiều vị trí không có giá trị hay vô nghĩa

$$\begin{pmatrix} 1.0 & 0 & 5.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 & 0 & 11.0 & 0 \\ 0 & 0 & 0 & 0 & 9.0 & 0 & 0 & 0 \\ 0 & 0 & 6.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.0 & 0 & 0 & 0 & 0 \\ 2.0 & 0 & 0 & 0 & 0 & 10.0 & 0 & 0 \\ 0 & 0 & 0 & 8.0 & 0 & 0 & 0 & 0 \\ 0 & 4.0 & 0 & 0 & 0 & 0 & 0 & 12.0 \end{pmatrix}$$

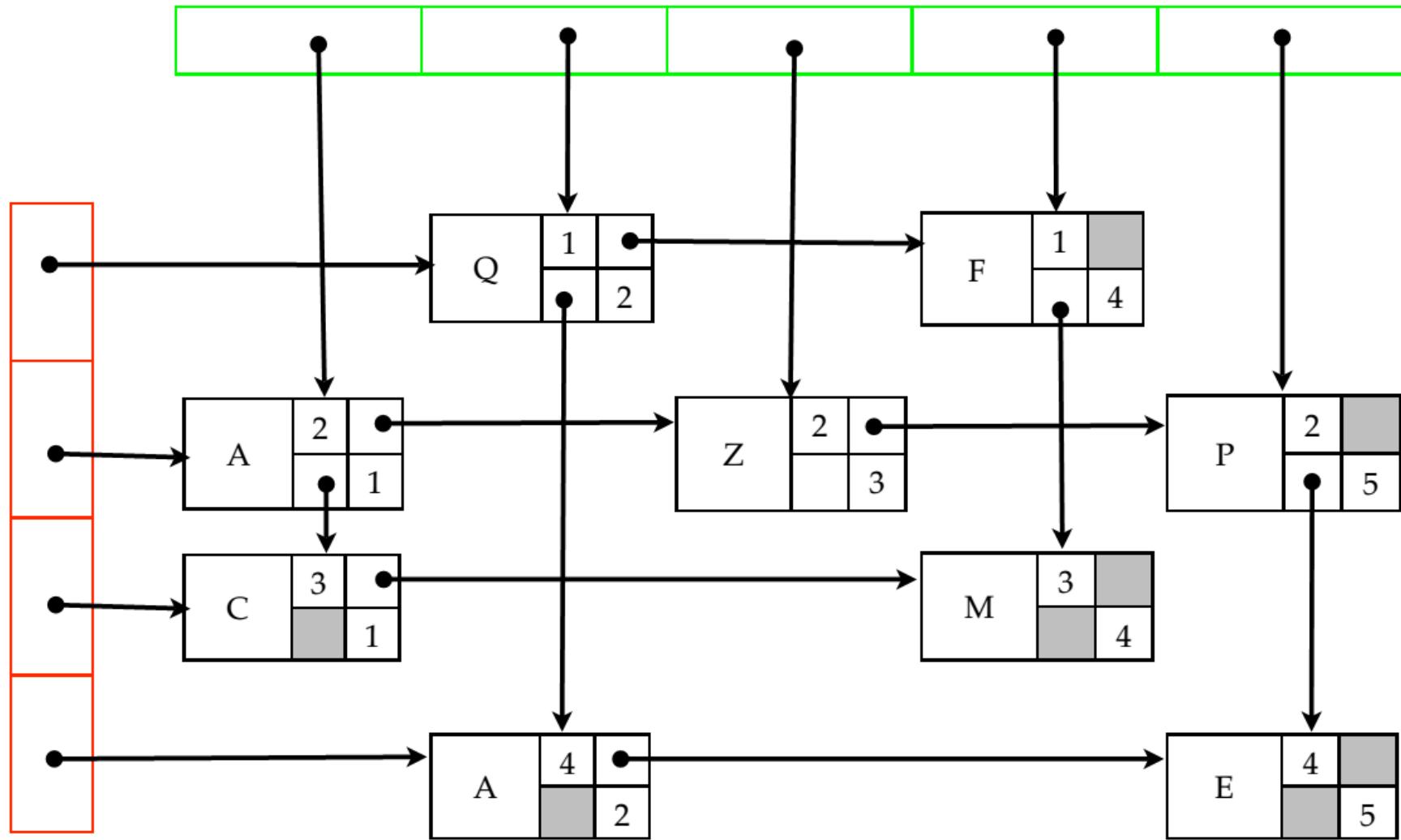
Giải pháp tối ưu ma trận thưa

- Sử dụng danh sách liên kết



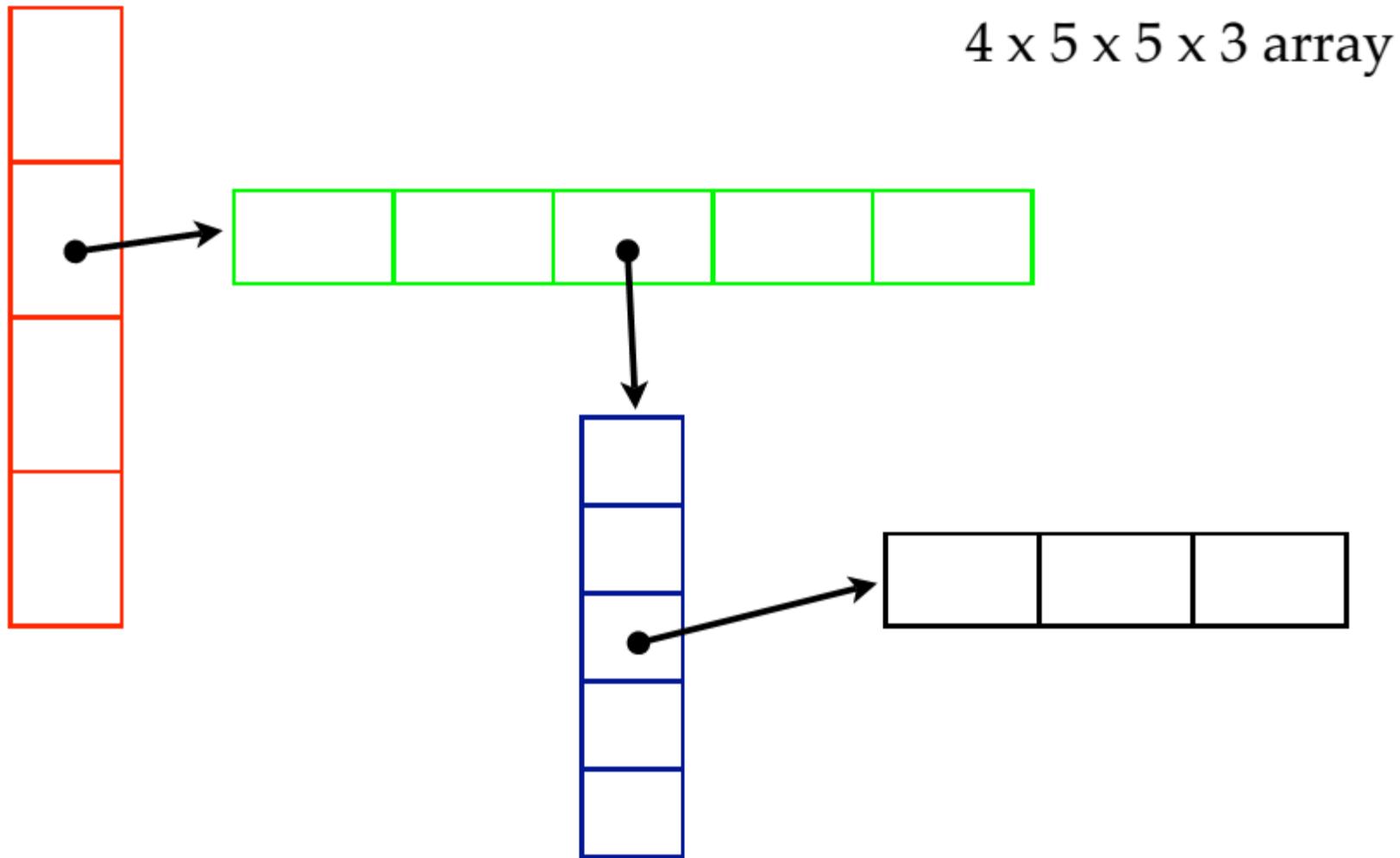
Giải pháp tối ưu ma trận thưa

- Sử dụng danh sách liên kết



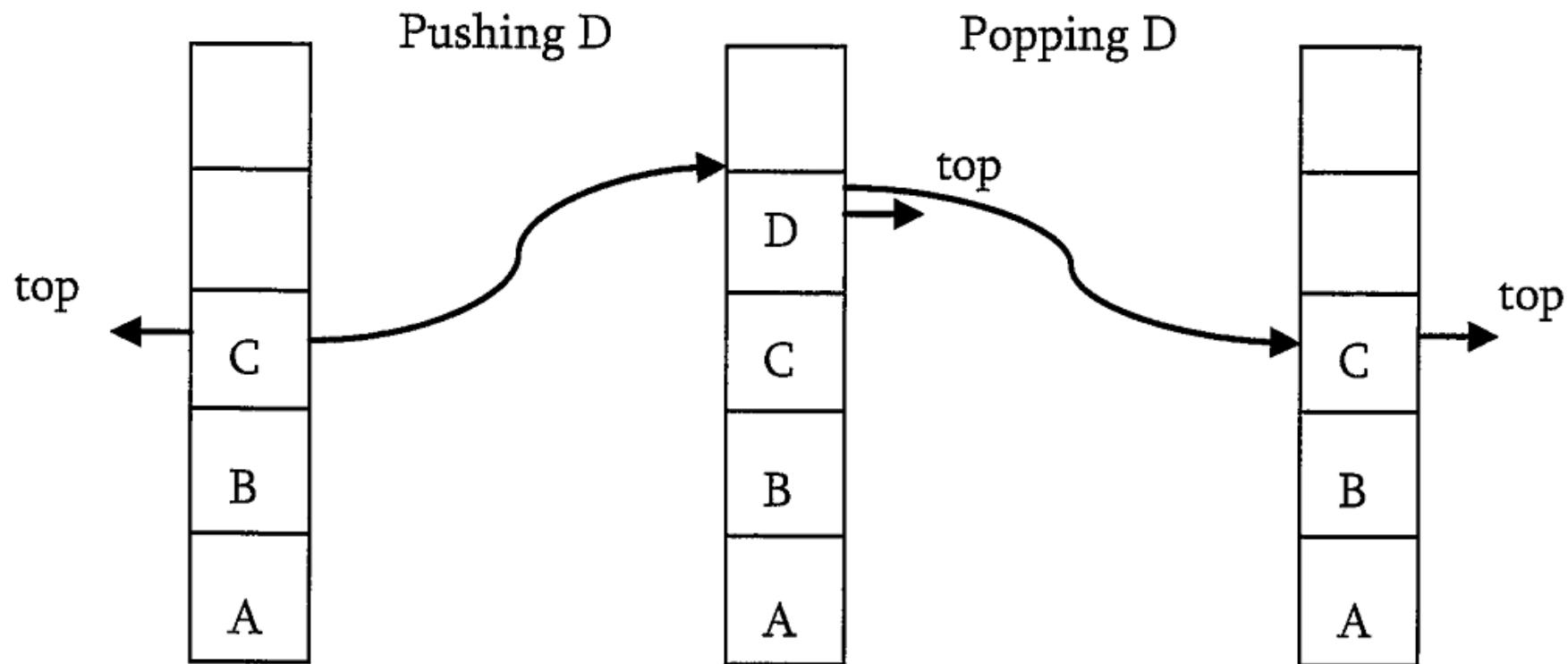
Giải pháp tối ưu ma trận thưa

- Sử dụng danh sách liên kết

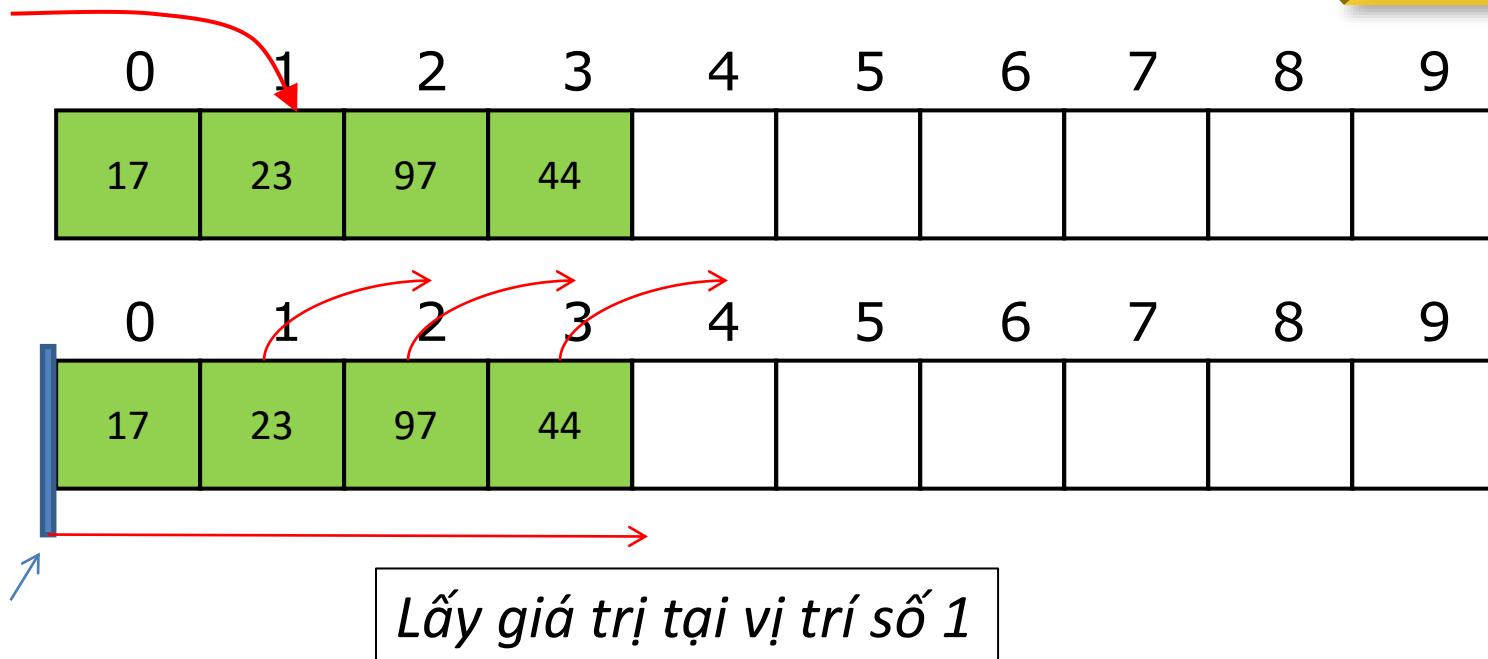


Stack (Ngăn xếp)

- **Stack**: là một danh sách có thứ tự và các phần tử được đưa vào và lấy ra theo cơ chế FILO hay LIFO



Stack vs. Mảng



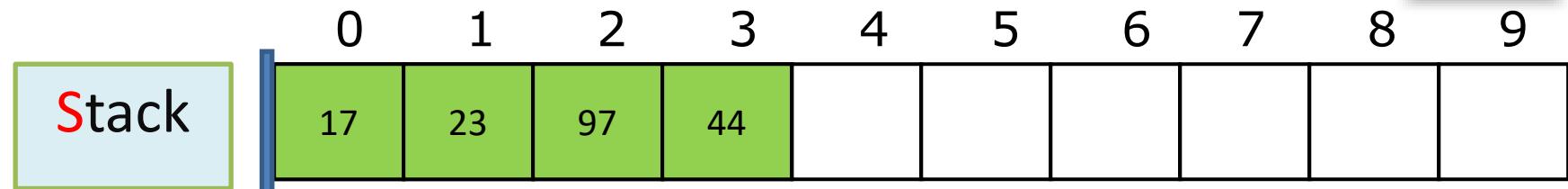
- Nhảy đến vị trí cần lấy và thực hiện thao tác trực tiếp.
M[1]

- Cơ chế FILO (First In Last Out – Vào Trước Ra Sau).
- Để truy cập đến phần tử 1, thì các phần tử sau phải được lấy ra khỏi stack trước.

Các thao tác trên Stack

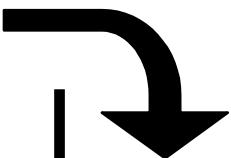
- Các thao tác cơ bản trên Stack:
 - Khởi tạo
 - Đưa phần tử vào (Push)
 - Lấy phần tử ra (Pop)
 - Xem phần tử trên cùng (Top)
 - Kích thước của Stack (Size)
 - Kiểm tra Stack rỗng (IsEmptyStack)
 - Kiểm tra Stack đầy (IsFullStack)
- Hãy sử dụng mã giả để thực thi các thao tác trên.

Các thao tác trên Stack - Push



Push

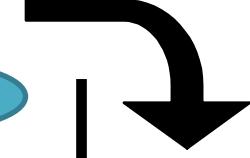
17



$nS=0$

Push

23



$nS=1$

Đáy Stack

17

$nS=4$

44
97
23
17

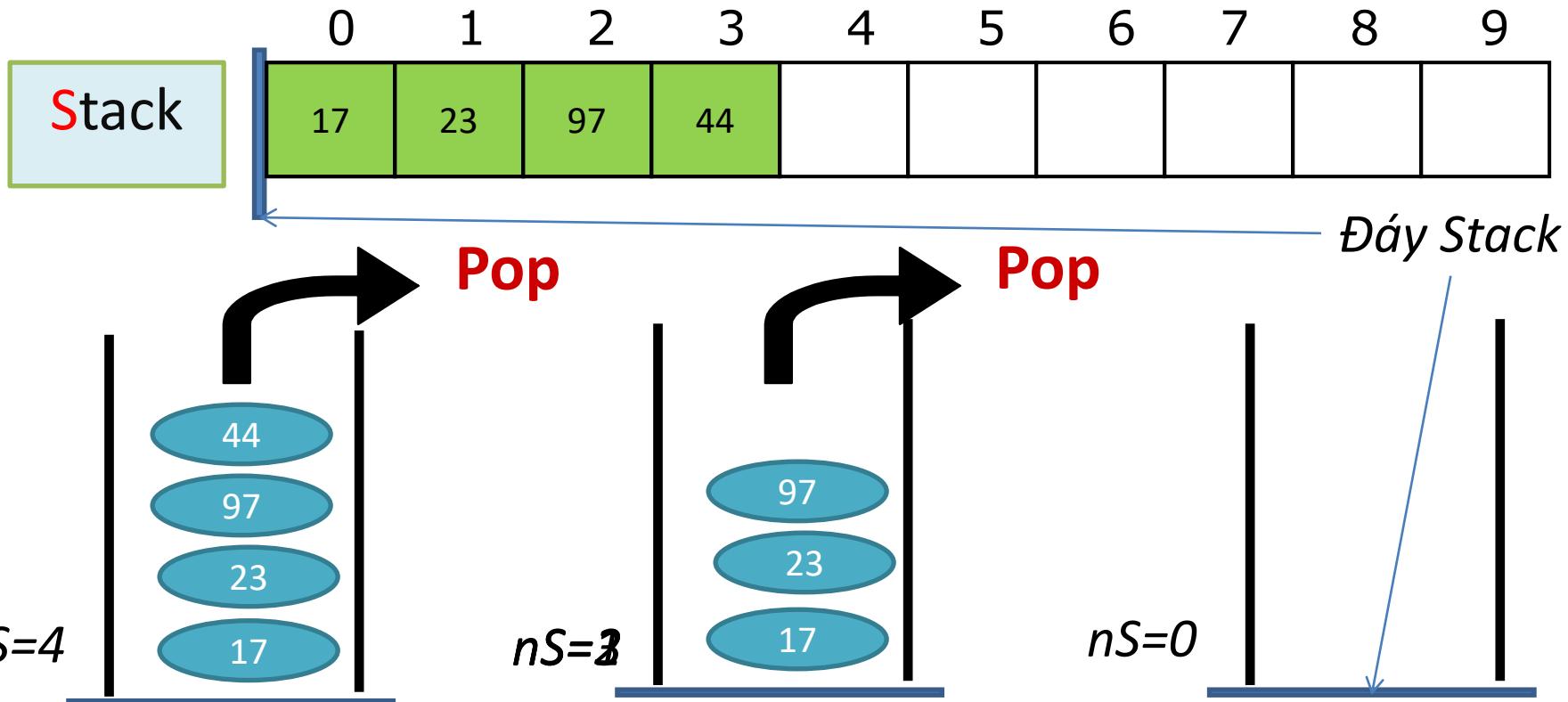
C++

C#

```
void Push(int x)
{
    S[nS] = x;
    nS++;
}
```

```
public static void Push(int x)
{
    S[nS] = x;
    nS++;
}
```

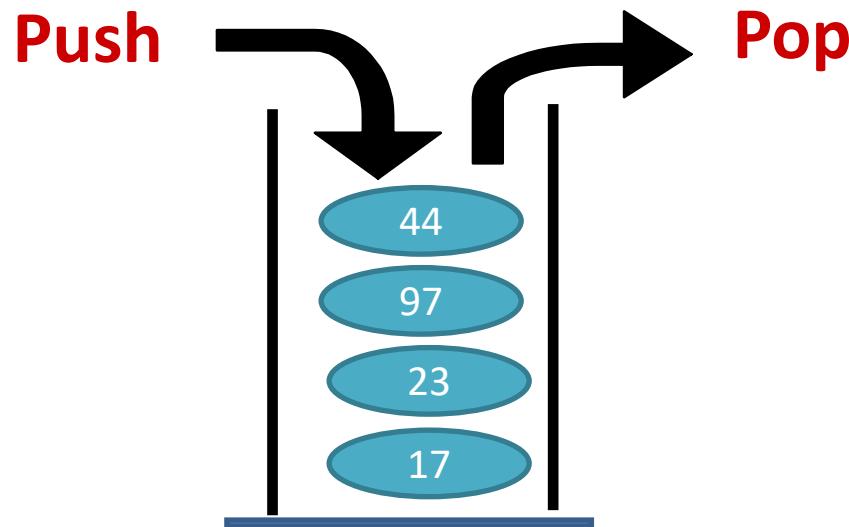
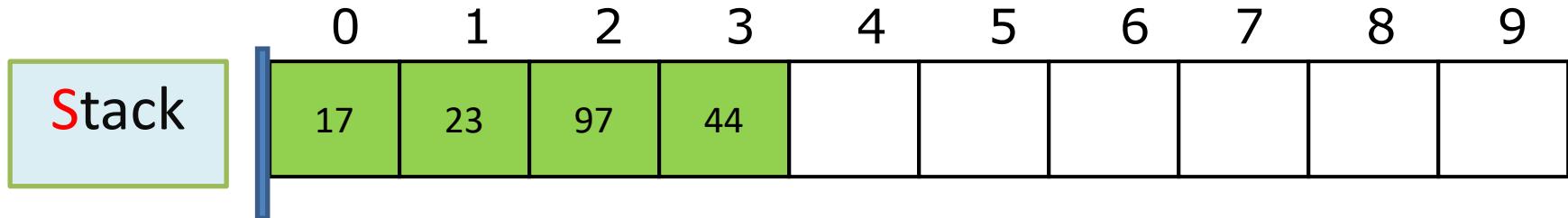
Các thao tác trên Stack - Pop



```
int Pop() {  
    if(nS == 0)      return -1; //rỗng  
    int x = S[nS-1];  
    nS--;  
    return x;        }  
C++
```

```
public static int Pop(){  
    if(nS == 0)      return -1; //rỗng  
    int x = S[nS-1];  
    nS--;  
    return x;        }  
C#
```

Tóm tắt các thao tác trên Stack



```
public static void Push(int x)
{
    S[nS] = x;
    nS++;
}
```

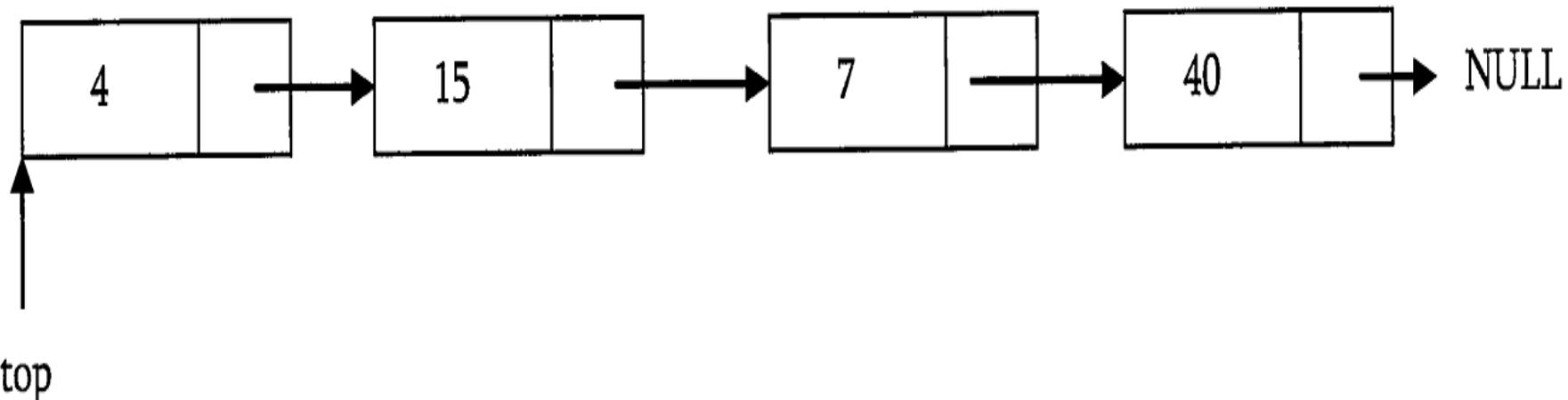
C#

```
public static int Pop(){
    if(nS == 0)    return -1; //rỗng
    int x = S[nS-1];
    nS--;
    return x;
}
```

C#

Thực thi Stack

- Có nhiều cách để thực thi stack:
 - Sử dụng mảng
 - Sử dụng mảng động (tăng kích thước khi stack đầy)
 - Sử dụng danh sách liên kết



Ứng dụng Stack

Một số ứng dụng quan trọng:

- Ứng dụng trực tiếp:
 - Cân bằng kí tự
 - Chuyển đổi trung tố thành hậu tố
 - Tính toán biểu thức hậu tố
 - Thực thi lời gọi hàm (bao gồm đệ quy)
 - Lịch sử viếng thăm web (back button)
 - Phục hồi kí tự trong trình soạn thảo văn bản.
 - So khớp các tag trong HTML và XML
- Ứng dụng gián tiếp:
 - Thuật toán duyệt cây

Bài tập

Hãy thực hiện/cài đặt các ứng dụng trong slide trước:

1. Kiểm tra cân bằng kí tự.

– Ví dụ:

Example	Valid?	Description
(A+B)+(C-D)	Yes	The expression is having balanced symbol
((A+B)+(C-D)	No	One closing brace is missing
((A+B)+[C-D])	Yes	Opening and immediate closing braces correspond
((A+B)+[C-D])	No	The last closing brace does not correspond with the first opening parenthesis

Bài tập

2. Chuyển trung tố thành hậu tố

– Ví dụ:

Infix	Prefix	Postfix
$A+B$	$+AB$	$AB+$
$A+B-C$	$-+ABC$	$AB+C-$
$(A+B)^*C-D$	$-^*+ABCD$	$AB+C^*D-$

Bài tập

3. Tính toán biểu thức hậu tố (nghịch đảo Balan)

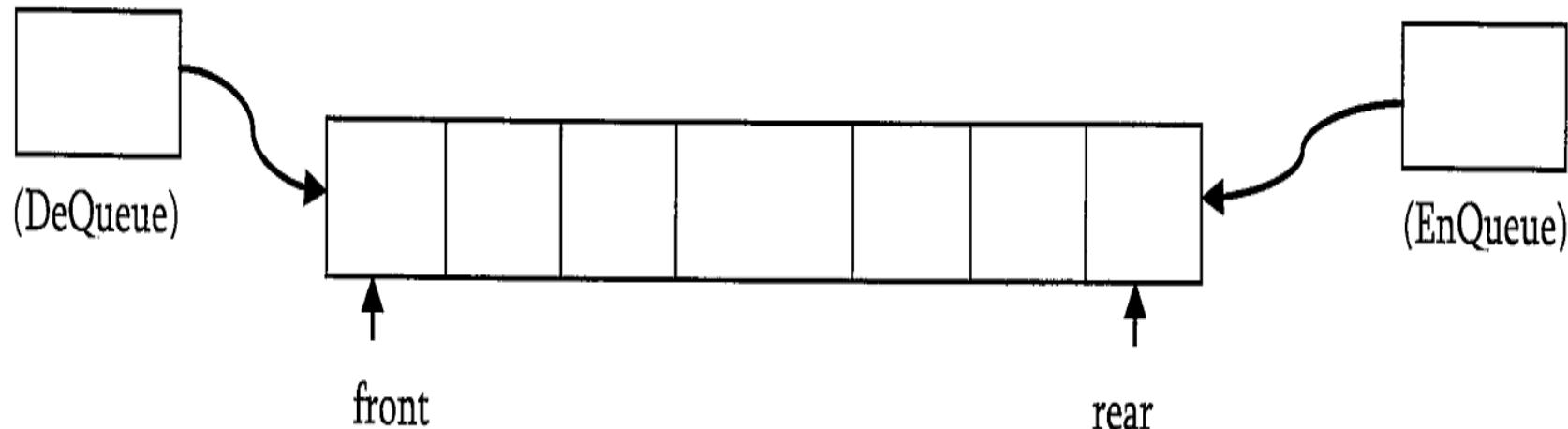
Ví dụ:

Postfix String : 123*+5-

Result : 2

Queue (Hàng đợi)

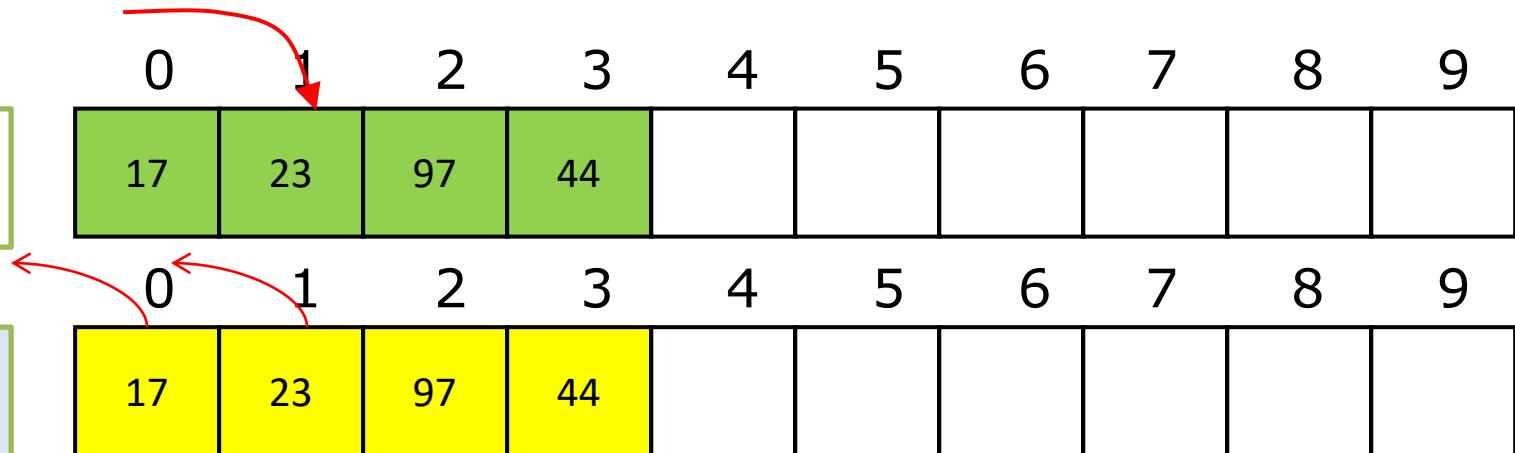
- **Queue** là một danh sách có thứ tự và các phần tử được đưa vào và lấy ra theo cơ chế FIFO hay LILO



Các thao tác trên Queue

- Các thao tác cơ bản trên Queue:
 - Khởi tạo
 - Đưa phần tử vào (EnQueue)
 - Lấy phần tử ra (DeQueue)
 - Xem phần tử đứng trước (Front)
 - Kích thước của Queue (QueueSize)
 - Kiểm tra Queue rỗng (IsEmptyQueue)
 - Kiểm tra Queue đầy (IsFullQueue)
- Hãy sử dụng mã giả để thực thi các thao tác trên.

Queue vs Mảng

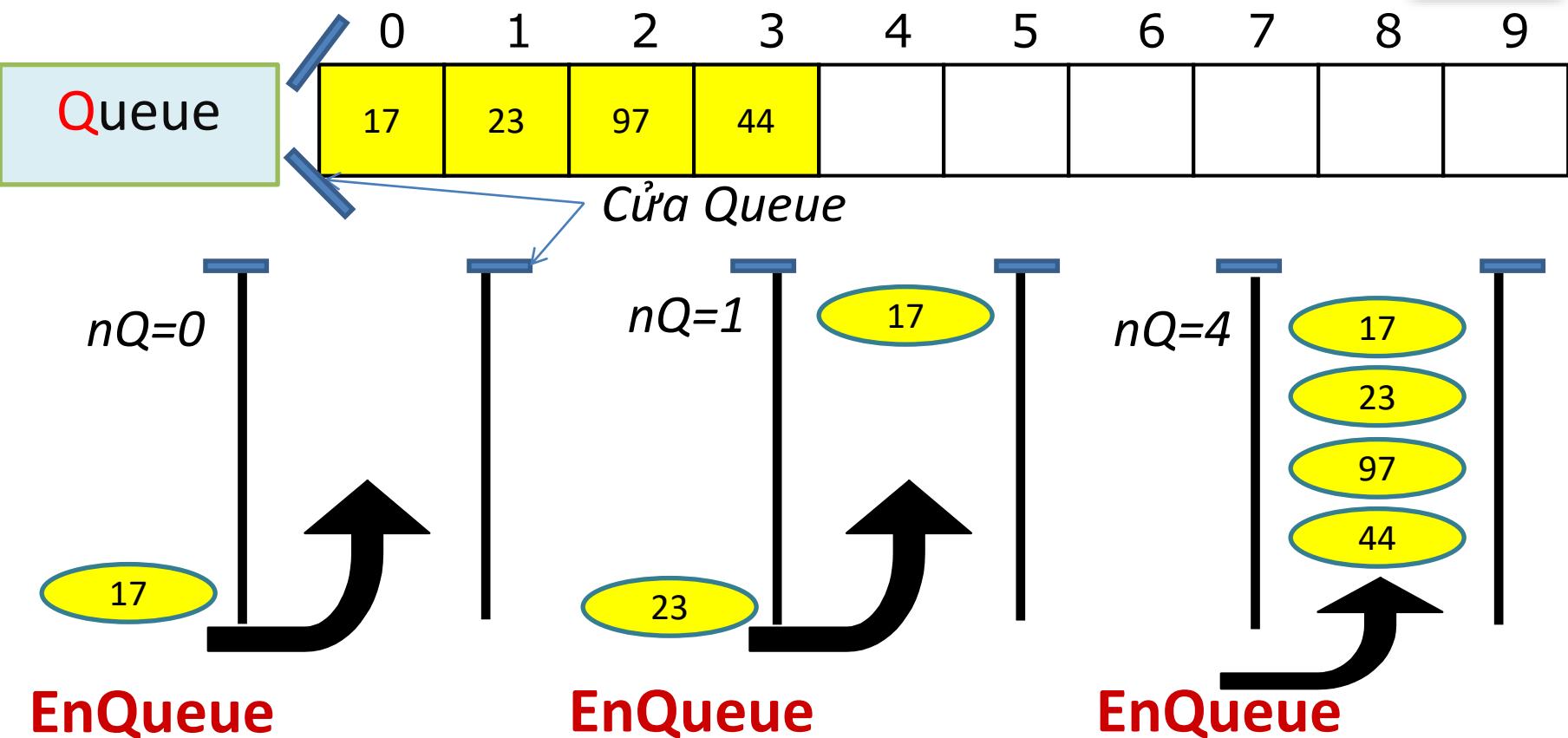


Lấy giá trị tại vị trí số 1

- Nhảy đến vị trí cần lấy và thực hiện thao tác trực tiếp.
M[1]

- Cơ chế FIFO (First In First Out – Vào Trước Ra Trước).
- Để truy cập đến phần tử 1, thì các phần tử **trước** phải được lấy ra khỏi queue trước.

Các thao tác trên Queue - EnQueue



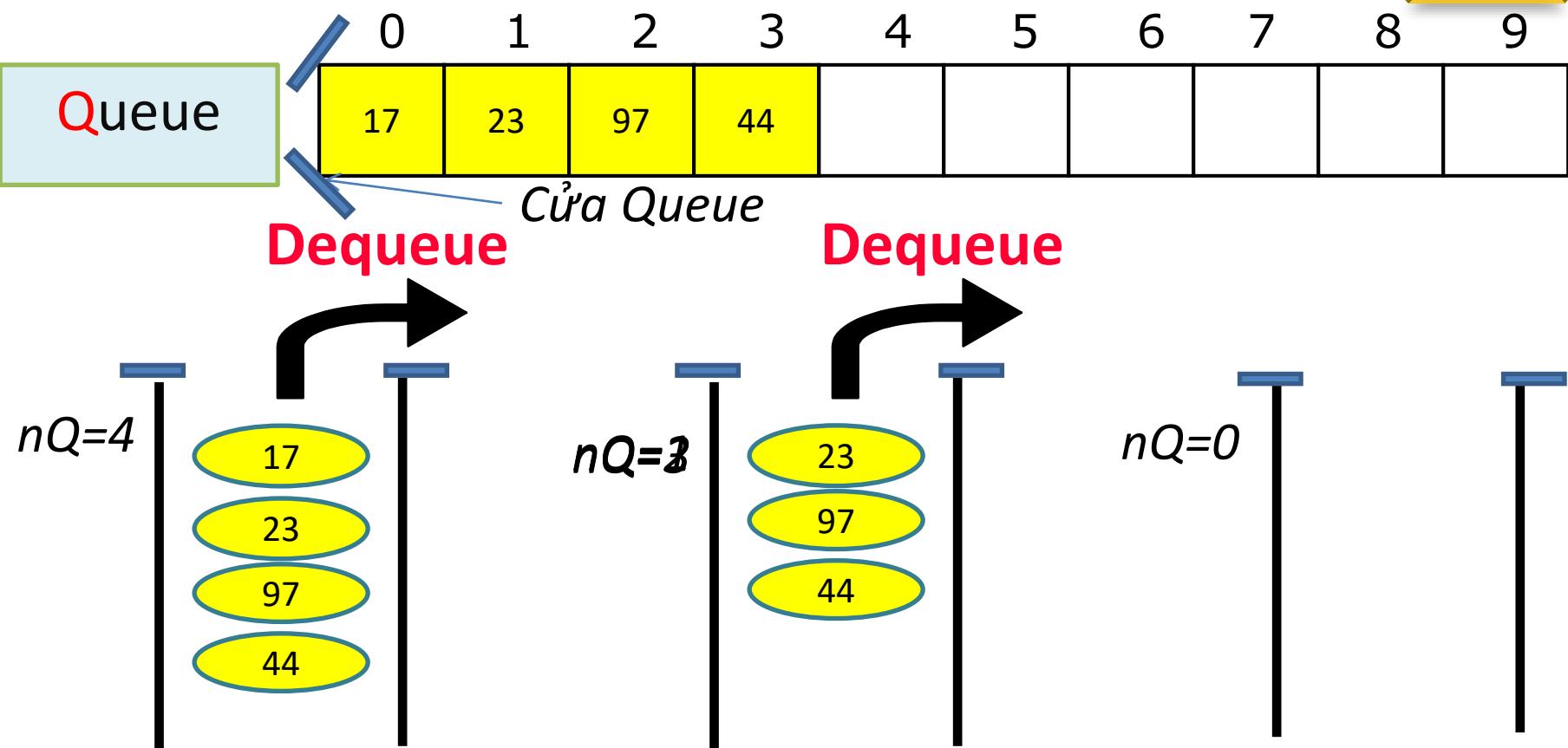
```
void DeQueue(int x)
{
    Q[nQ] = x;
    nQ++;
}
```

C++

```
public static void EnQueue(int x)
{
    Q[nQ] = x;
    nQ++;
}
```

C#

Các thao tác trên Queue - DeQueue



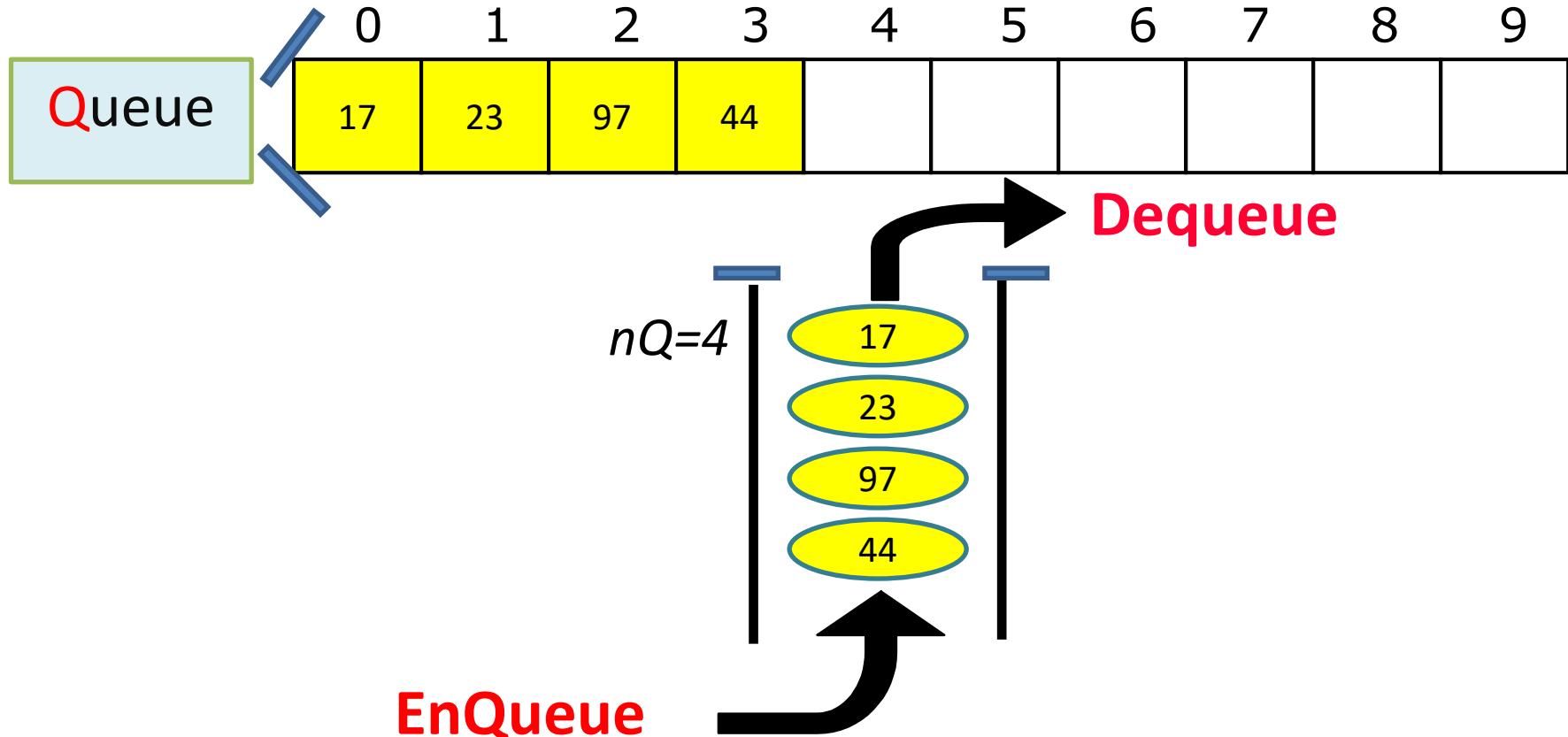
```
int DeQueue() {  
    if(nQ == 0)      return -1; //rỗng  
    int x = Q[0];  
    for (int i = 0; i <n; i++) Q[i] = Q[i+1];  
    nQ--;  
    return x;  
}
```

C++

```
public static void DeQueue(int x) {  
    if(nQ == 0)      return -1; //rỗng  
    int x = Q[0];  
    for (int i = 0; i <n; i++) Q[i] = Q[i+1];  
    nQ--;  
    return x;  
}
```

C#

Tóm tắt các thao tác trên Queue



```
public static void EnQueue(int x)
{
    Q[nQ] = x;
    nQ++;
}
```

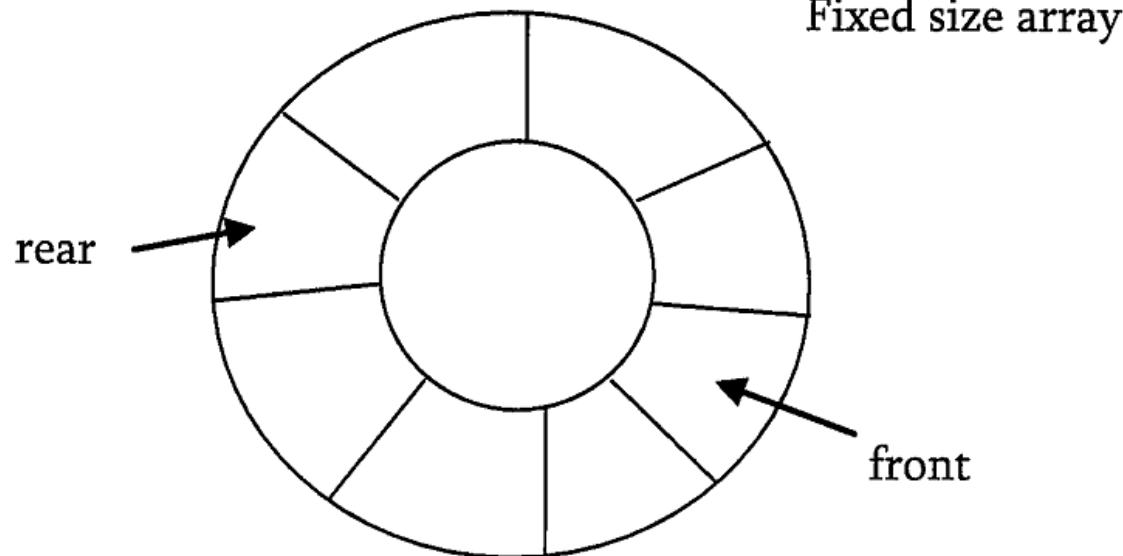
C#

```
public static void DeQueue(int x) {
    if(nQ == 0)      return -1; //rỗng
    int x = Q[0];
    for (int i = 0; i <n; i++) Q[i] = Q[i+1];
    nQ--;
    return x;
}
```

C#

Thực thi Queue

- Tương tự như Stack, Queue cũng có nhiều cách để thực thi như:
 - Mảng
 - Mảng động
 - Danh sách liên kết
 - Mảng vòng



Ứng dụng Queue

Một số ứng dụng quan trọng:

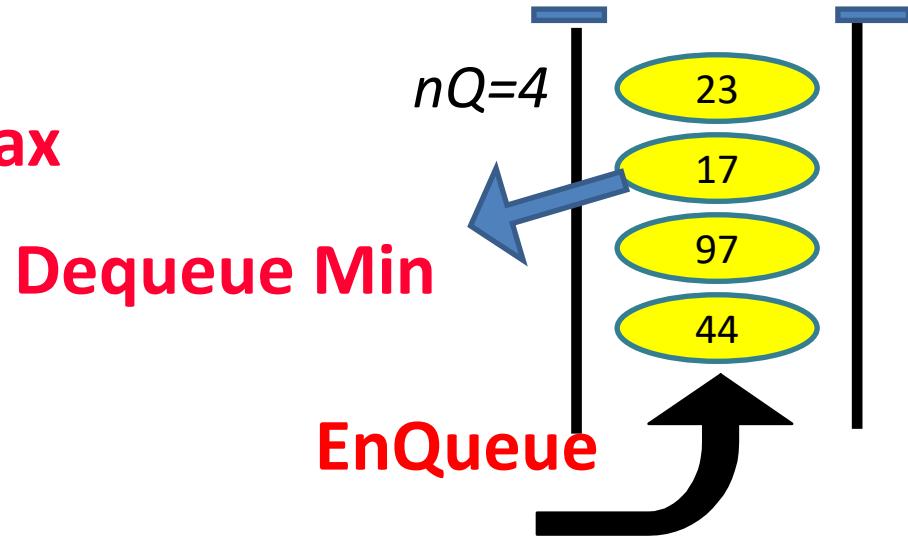
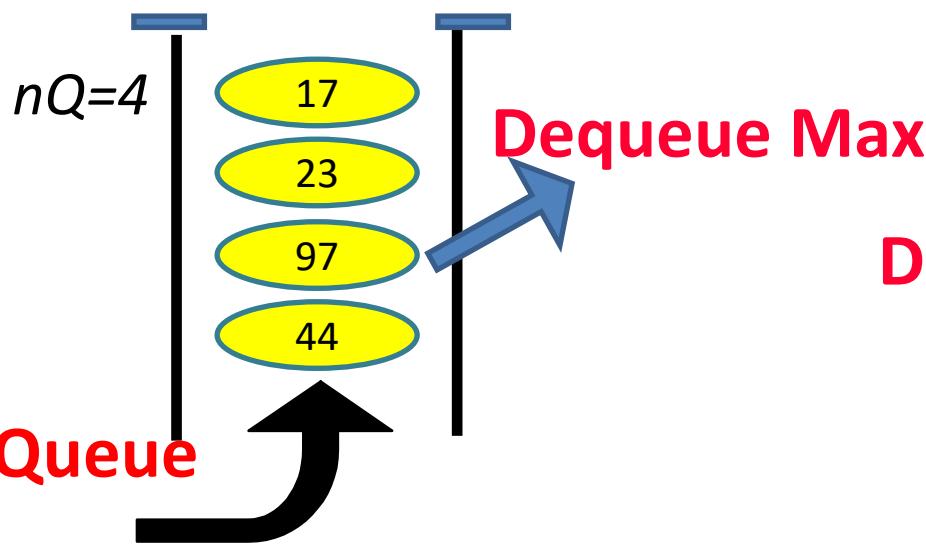
- Ứng dụng trực tiếp:
 - Lập lịch các chương trình hệ thống như in ấn
 - Chuyển gói tin bất đồng bộ
 - Đa tiêu trình
 - ...
- Ứng dụng gián tiếp:
 - Các thuật toán tìm kiếm
 - ...

Priority Queue – Hàng Đợi Có Độ Ưu Tiên



- Hàng đợi có độ ưu tiên (Priority Queue): phần tử vào giống như Queue, nhưng lúc ra, phần tử nào có độ ưu tiên lớn hơn sẽ ra trước mà không xét thứ tự trước sau nữa.
- VD 1: Độ ưu tiên là “phần tử nào lớn hơn có độ ưu tiên lớn hơn”.
→ Phần tử lớn nhất sẽ ra khỏi hàng đợi trước.
- VD 2: Độ ưu tiên là “phần tử nào nhỏ hơn có độ ưu tiên lớn hơn”.
→ Phần tử nhỏ nhất sẽ ra khỏi hàng đợi trước.

Priority Queue – Hàng Đợi Có Độ Ưu Tiên



```
public static void DeQueue(int x) {  
    if(nQ == 0)    return -1; //rỗng  
    int x = Q[0];  
    for (int i = 0; i <n; i++) Q[i] = Q[i+1];  
    nQ--;  
    return x; }
```

Queue

```
public static void DePriQueue(int x) {  
    if(nPriQ == 0)        return -1;  
    int x = TimMax(PriQ); //TimMin...  
    for (int i = 0; i <n; i++)  
        PriQ[i] = PriQ[i+1];  
    nPriQ--;  
    return x; }
```

Priority Queue

Bài tập

1. Hãy sử dụng 2 Stack để cài đặt cơ chế Queue

```
struct Queue {  
    struct Stack *S1;  
    struct Stack *S2;  
}
```

Bài tập

2. Tìm mảng con có tổng cực đại trong cửa sổ trượt

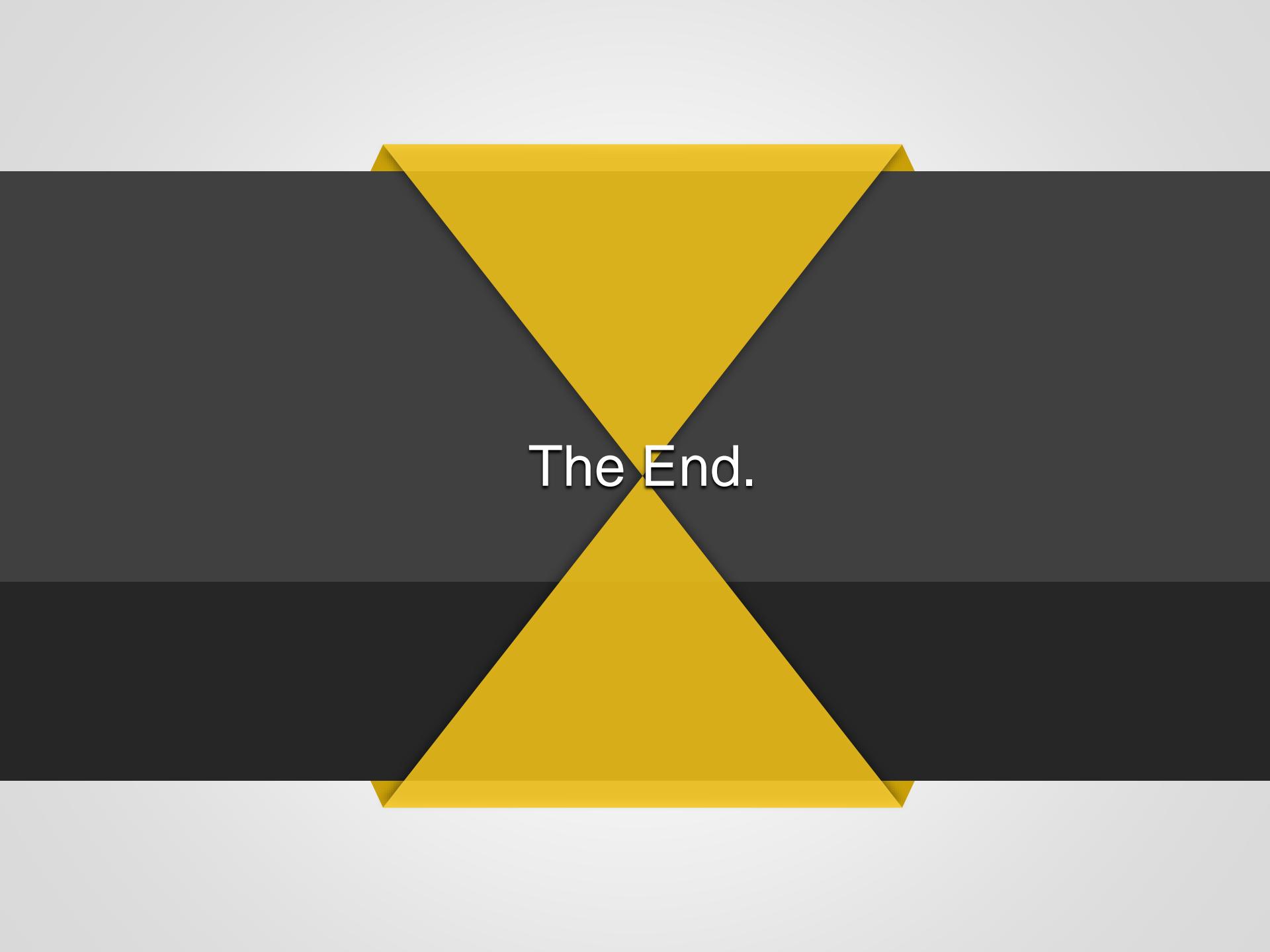
Cho một mảng a với cửa sổ trượt kích thước w. Cửa sổ di chuyển từ trái sang phải. Giả sử chúng ta chỉ nhìn thấy các số trong cửa sổ w. Mỗi lần cửa sổ di chuyển sang bên một vị trí.

Ví dụ: a [1 3 -1 -3 5 3 6 7] và w=3

Window position	Max
[1 3 -1] -3 5 3 6 7	3
1 [3 -1 -3] 5 3 6 7	3
1 3 [-1 -3 5] 3 6 7	5
1 3 -1 [-3 5 3] 6 7	5
1 3 -1 -3 [5 3 6] 7	6
1 3 -1 -3 5 [3 6 7]	7

Tìm hiểu thêm

- Bài tập về nhà: tìm hiểu thêm 6 cấu trúc được cài đặt sẵn trong C++:
 - Vector
 - Grid
 - Map
 - Lexicon
 - Scanner
 - Interator



The End.

DSA War – Gợi ý

- Từ khóa gợi ý:
 - Data
 - Data Type: int, float, ...
 - ADT
 - Pointer
 - Array
 - Vector