



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Cấu trúc dữ liệu và thuật toán

Nguyễn Khánh Phương

**Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn**

Nội dung khóa học

Chương 1. Các khái niệm cơ bản

Chương 2. Các sơ đồ thuật toán

Chương 3. Các cấu trúc dữ liệu cơ bản

Chương 4. Cây

Chương 5. Sắp xếp

Chương 6. Tìm kiếm

Chương 7. Đồ thị



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Chương 3. Các cấu trúc dữ liệu cơ bản

Nguyễn Khánh Phương

**Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn**

Kiểu dữ liệu (Data types)

- **Kiểu dữ liệu (data type)** được đặc trưng bởi:
 - Tập các giá trị (a set of *values*);
 - Cách biểu diễn dữ liệu (*data representation*) được sử dụng chung cho tất cả các giá trị này và
 - Tập các phép toán (set of *operations*) có thể thực hiện trên tất cả các giá trị.

Các kiểu dữ liệu dựng sẵn

(Built-in data types)

- Trong các ngôn ngữ lập trình thường có một số kiểu dữ liệu nguyên thủy đã được xây dựng sẵn. Ví dụ:
 - Kiểu số nguyên (Integer numeric types)
 - byte, char, short, int, long
 - Kiểu số thực dấu phẩy động (floating point numeric types)
 - float, double
 - Các kiểu nguyên thủy khác (Other primitive types)
 - bool
 - Kiểu mảng (Array type)
 - mảng các phần tử cùng kiểu

Các kiểu dữ liệu dựng sẵn trong C/C++

(Built-in data types)

- Các kiểu dữ liệu phải biết
 - `bool`: biến bun (true/false)
 - `char`: biến nguyên 8 bit (thường được sử dụng để biểu diễn các ký tự ASCII)
 - `string`: biến chuỗi ký tự
 - `int`: biến nguyên 32 bit
 - `long`: biến nguyên 32 bit
 - `float`: biến thực 32 bit
 - `double`: biến thực 64 bit
- Các modifier
 - `signed`
 - `unsigned char`, `signed char`, `unsigned int`, `signed int`, `unsigned long`, `signed long`
 - `unsigned`
 - `short int`, `unsigned short int`,
 - `short`
 - `long int`, `unsigned long int`, `long double`
 - `long`
 - ...

Các kiểu dữ liệu dựng sẵn trong C/C++

(Built-in data types)

DATA TYPE	SIZE (IN BYTES)	RANGE
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	$\approx -3.4 \times 10^{-38}$ to $\approx 3.4 \times 10^{-38}$ ≈ 7 chữ số
double	8	$\approx -1.7 \times 10^{-308}$ to $\approx 1.7 \times 10^{-308}$ ≈ 14 chữ số
long double	12	

Trên trình biên dịch GCC 64 bit

Phép toán đối với kiểu dữ liệu nguyên thủy

- **Đối với kiểu: `byte`, `char`, `short`, `int`, `long`**
 - ✓ `+`, `-`, `*`, `/`, `%`, đổi thành xâu, ...
- **Đối với kiểu: `float`, `double`**
 - ✓ `+`, `-`, `*`, `/`, `round`, `ceil`, `floor`, ...
- **Đối với kiểu: `bool`**
 - ✓ kiểm giá trị `true`, hay kiểm giá trị `false`
- Nhận thấy rằng: Các ngôn ngữ lập trình khác nhau có thể sử dụng mô tả kiểu dữ liệu khác nhau. Chẳng hạn, PASCAL và C có những mô tả các dữ liệu số khác nhau.

Nội dung

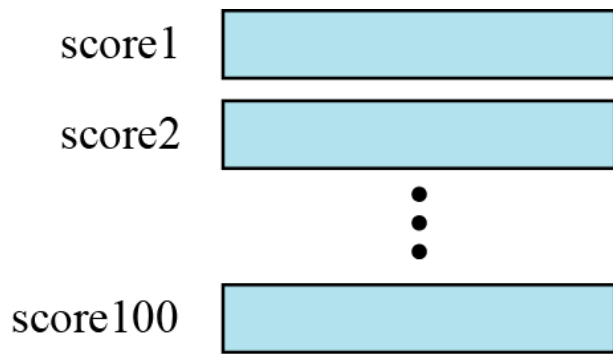
1. Mảng (Array)
2. Bản ghi (Record)
3. Danh sách liên kết (Linked List)
4. Ngăn xếp (Stack)
5. Hàng đợi (Queue)

Nội dung

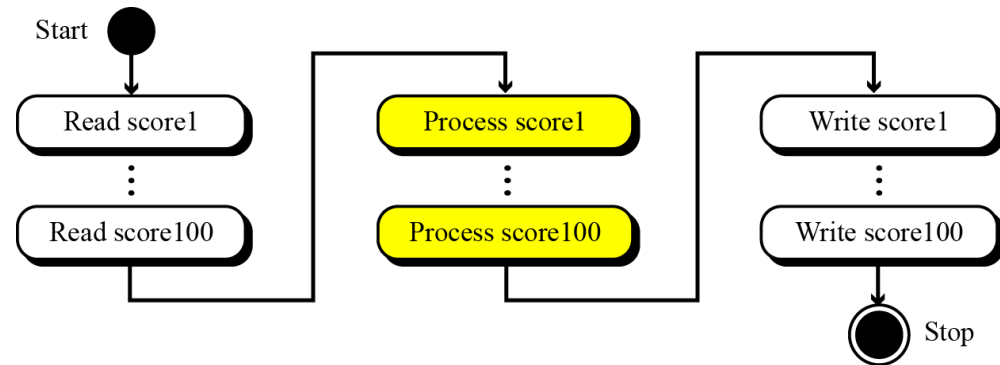
- 1. Mảng (Array)**
2. Bản ghi (Record)
3. Danh sách liên kết (Linked List)
4. Ngăn xếp (Stack)
5. Hàng đợi (Queue)

1. Mảng (Array)

- Giả sử có 100 tỉ số (scores) trong một giải đấu bóng chày. Chúng ta cần tiến hành các thao tác: lấy thông tin của 100 tỉ số đó (read them), rồi đem xử lý các tỉ số này (process them), và cuối cùng là in chúng (print/write them).
- Để làm được điều này, ta cần lưu trữ 100 tỉ số này vào bộ nhớ trong suốt quá trình thực hiện chương trình. Do đó, ta sẽ sử dụng 100 biến, mỗi biến một tên tương ứng với 1 tỉ số: (Xem hình 1)



Hình 1 Sử dụng 100 biến

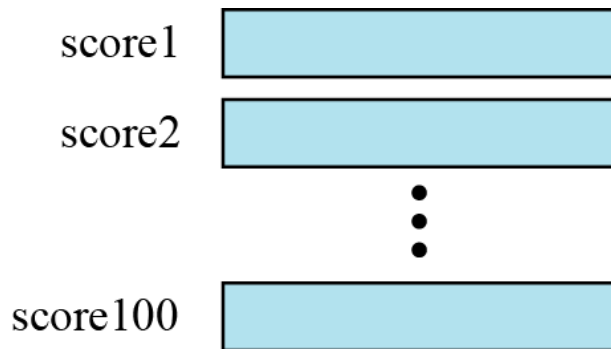


Hình 2 Xử lý dữ liệu tỉ số trên 100 biến

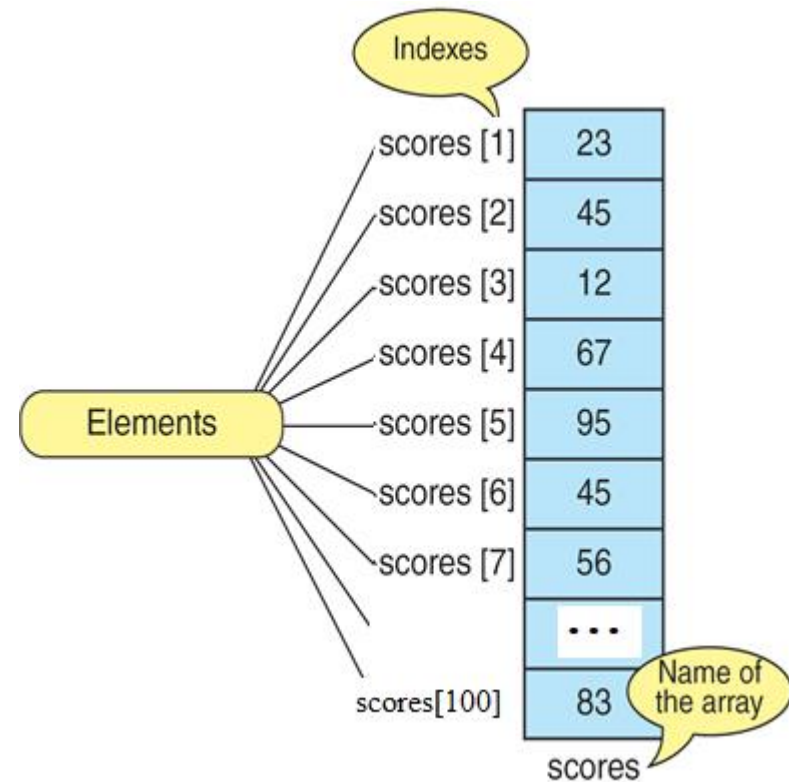
- Sử dụng 100 biến dẫn đến vấn đề: ta cần phải viết lệnh đọc (read) 100 lần, lệnh xử lý (process) 100 lần và lệnh in (write) 100 lần (xem Hình 2).

1. Mảng (Array)

- Thay vì khai báo biến một cách rời rạc 100 biến `score1`, `score2`, ..., `score100`, ta có thể khai báo một mảng các giá trị như `scores[1]`, `scores[2]` và ... `scores[100]` để biểu diễn các giá trị riêng biệt.



Hình 1 Sử dụng 100 biến



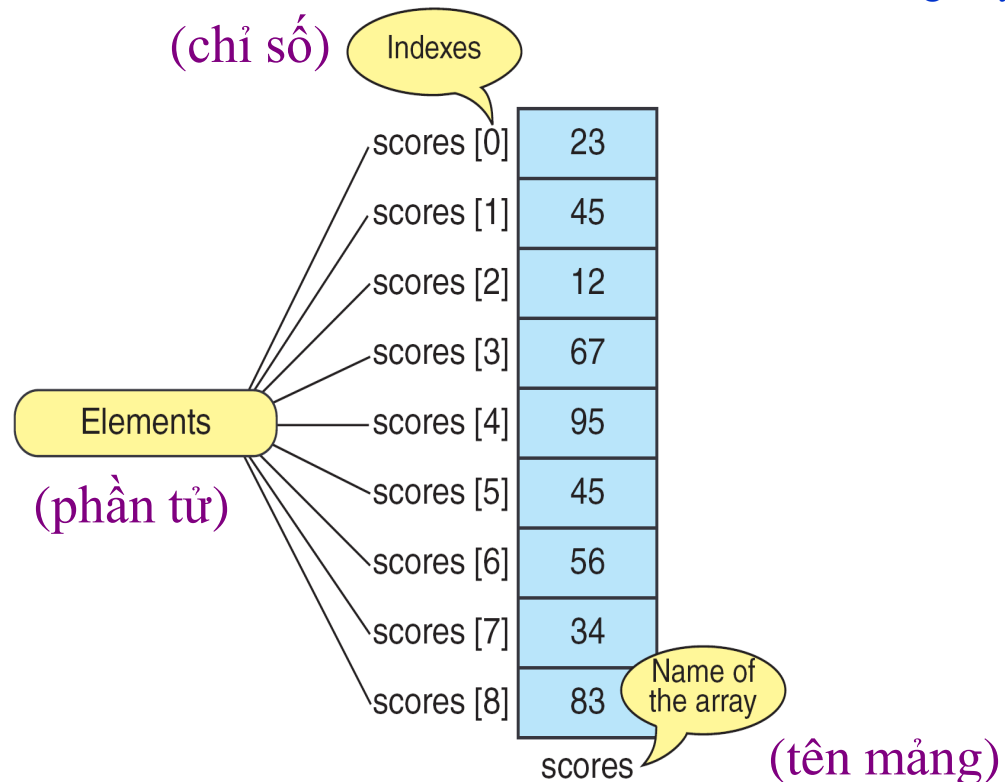
Hình 2 Sử dụng mảng `scores` gồm 100 phần tử

1. Mảng (Array)

Mảng là một kiểu dữ liệu chứa dãy các phần tử được đánh chỉ số, thông thường các phần tử này có cùng kiểu dữ liệu.

- Mỗi phần tử của mảng có một chỉ số cố định duy nhất
 - Chỉ số nhận giá trị trong khoảng từ một **cận dưới** đến một **cận trên** nào đó

Ví dụ: Trong ngôn ngữ C, mảng scores kích thước $N = 9$, mỗi phần tử được đánh một chỉ số duy nhất i với điều kiện $0 \leq i < N$, tức là chỉ số của mảng luôn luôn được bắt đầu từ 0 và phải nhỏ hơn kích thước của mảng. Để truy cập đến mỗi phần tử của mảng ta chỉ cần biết chỉ số của chúng, khi ta viết **scores[i]** có nghĩa là ta đang truy cập tới phần tử thứ i của mảng scores



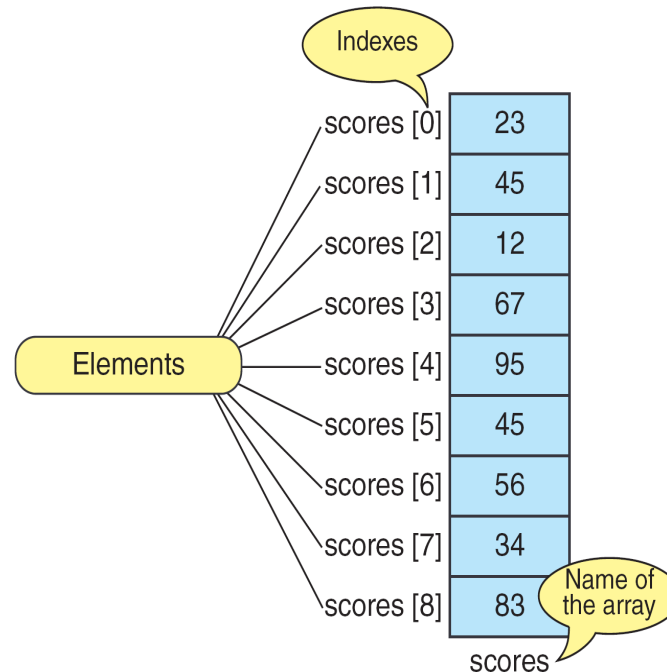
Tên mảng vs. Tên phần tử của mảng

Trong 1 mảng, ta có 2 kiểu định danh:

- Tên của mảng
- Tên của các phần tử riêng biệt thuộc mảng.

Tên của mảng là tên của toàn bộ cấu trúc, còn tên của một phần tử cho phép ta truy cập đến phần tử đó.

Ví dụ:

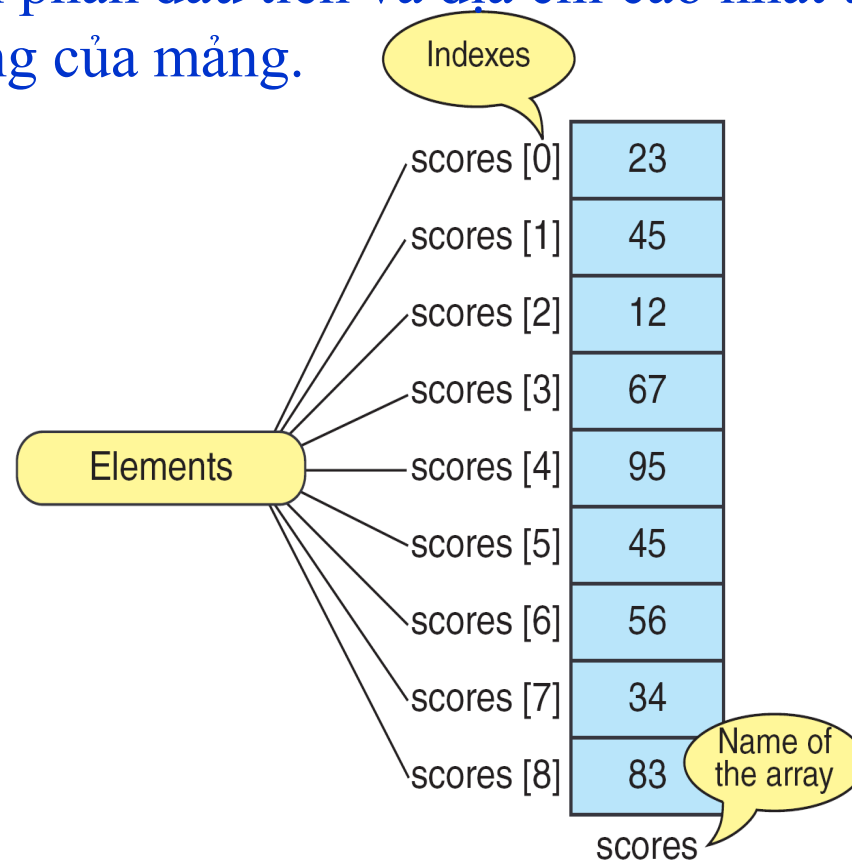


Tên của mảng: *scores*,

Tên mỗi phần tử của mảng: gồm tên của mảng theo sau là chỉ số của phần tử, ví dụ *scores[0]*, *scores[1]*, ...

1. Mảng (Array)

- Mảng (Array): tập các cặp (**chỉ số (index)** và **giá trị (value)**)
 - Cấu trúc dữ liệu: mỗi chỉ số, sẽ tương ứng với 1 giá trị.
 - Lưu trữ trong bộ nhớ: mảng được lưu trữ ở các ô nhớ liên kề nhau (cách lưu trữ này được gọi là *lưu trữ kế tiếp*). Địa chỉ thấp nhất tương ứng với thành phần đầu tiên và địa chỉ cao nhất tương ứng với thành phần cuối cùng của mảng.



Mảng trong các ngôn ngữ lập trình

- Các chỉ số có thể là số nguyên (C/C++, Java) hoặc là các giá trị kiểu rời rạc (Pascal, Ada)
- Cận dưới là 0 (C/C++, Java), 1 (Fortran), hoặc tùy chọn bởi người lập trình (Pascal, Ada)
- Trong hầu hết các ngôn ngữ, mảng là **thuần nhất** (nghĩa là tất cả các phần tử của mảng có cùng một kiểu); trong một số ngôn ngữ (như Lisp, Prolog) các thành phần có thể là không thuần nhất (có các kiểu khác nhau)

Khai báo mảng 1 chiều (one-dimensional array)

Khi khai báo mảng, ta cần kiểu mảng (type), tên mảng (arrayName) và số phần tử trong mảng (arraySize > 0) :

type arrayName [arraySize] ;



Ví dụ: khai báo **int A[5] ;**

tạo mảng A gồm 5 phần tử kiểu số nguyên (vì là kiểu nguyên, nên mỗi phần tử chiếm 4 bytes trong bộ nhớ)

- Nếu kích thước mảng **arraySize** là hằng số thì cho ta mảng có độ dài cố định (fixed length array), nếu là 1 biến (variable) thì cho ta mảng có độ dài thay đổi (variable-length arrays)

– Ví dụ: `double A[10] ;`

Mảng A cố định gồm 10 phần tử

`int n ;`

`double B[n] ;`


Mảng B có độ dài thay đổi qua giá trị của biến *n*

- Chú ý: trước khi sử dụng một mảng, ta luôn phải khai báo và khởi tạo nó

Con trỏ (pointers)

- Giá trị các biến được lưu trữ trong bộ nhớ máy tính, có thể truy cập tới các giá trị đó qua tên biến, đồng thời cũng có thể qua địa chỉ của chúng trong bộ nhớ.
- Con trỏ thực chất là 1 biến mà nội dung của nó là địa chỉ của 1 đối tượng khác (Biến, hàm, nhưng không phải 1 hằng số) ➔ Việc sử dụng con trỏ cho phép ta truy nhập tới 1 đối tượng gián tiếp qua địa chỉ của nó.
- Có nhiều kiểu biến với các kích thước khác nhau, nên có nhiều kiểu con trỏ. Con trỏ `int` để trỏ tới biến hay hàm kiểu `int`.
- Cách khai báo:

```
type *pointer_name;
```



Chỉ rằng đây là con trỏ

Sau khi khai báo, ta được con trỏ NULL, vì nó chưa trỏ tới 1 đối tượng nào.

- Để sử dụng con trỏ, ta dùng toán tử lấy địa chỉ &

```
pointer_name = &var_name;
```

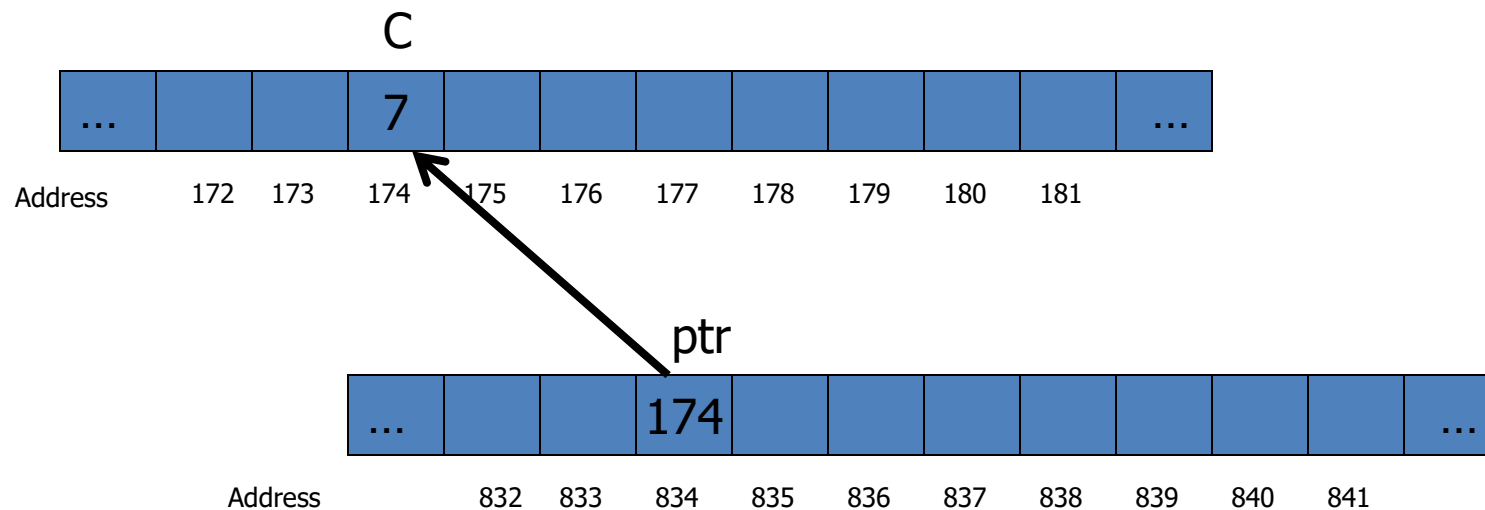
- Để lấy nội dung biến do con trỏ trỏ tới, ta dùng toán tử lấy nội dung *

```
*pointer_name
```

Con trỏ: Ví dụ

```
int c;  
int *ptr; /* Khai báo biến ptr là con trỏ int */  
c = 7;  
ptr = &c;
```

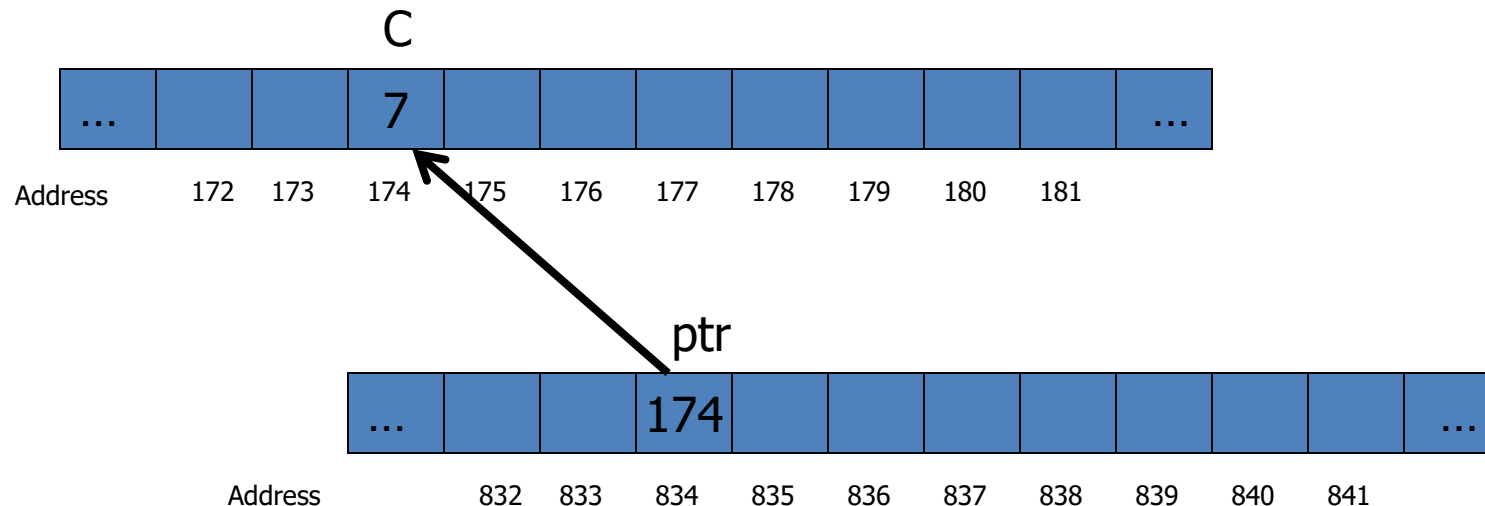
```
cout<<*ptr;  
*ptr = 80;  
cout<<c;
```



Con trỏ: Ví dụ

```
int c;  
int *ptr; /* Khai báo biến ptr là con trỏ int */  
c = 7;  
ptr = &c;
```

```
cout<<*ptr; /* in ra số 7 */  
*ptr = 80;  
cout<<c; /* in ra số 80 */
```



Con trỏ void*

- Là con trỏ không định kiểu.
- Nó có thể trỏ tới bất kì một loại biến nào.
- Thực chất một con trỏ void chỉ chứa một địa chỉ bộ nhớ mà không biết rằng tại địa chỉ đó có đối tượng kiểu dữ liệu gì. ➔ không thể truy cập nội dung của một đối tượng thông qua con trỏ void.
- Để truy cập được đối tượng thì trước hết phải ép kiểu biến trỏ void thành biến trỏ có định kiểu của kiểu đối tượng

Ví dụ:

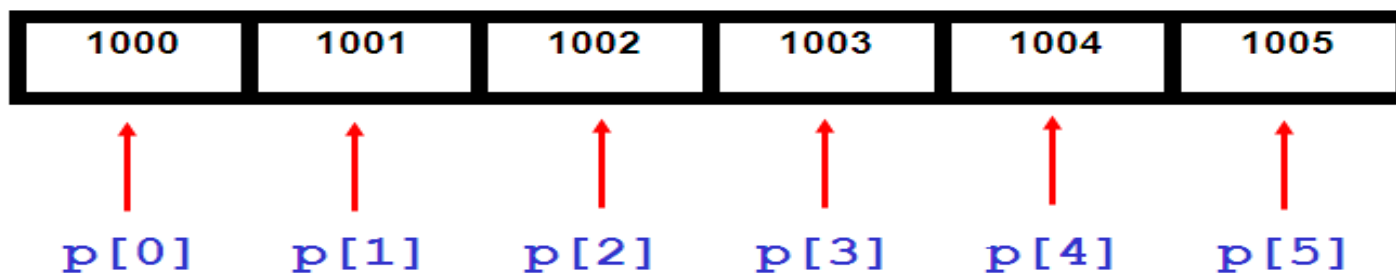
```
float x; int y;
void *p; // khai báo con trỏ void
p = &x; // p chứa địa chỉ số thực x
*p = 2.5; // báo lỗi vì p là con trỏ void
/* cần phải ép kiểu con trỏ void trước khi truy cập đối tượng
qua con trỏ */
*((float*)p) = 2.5; // x = 2.5
p = &y; // p chứa địa chỉ số nguyên y
*((int*)p) = 2; // y = 2
```

Khai báo mảng

- Điều gì xảy ra khi ta khai báo 1 mảng **p**?

Bộ nhớ (Memory):

- Các phần tử của mảng được lưu trữ liên kế tiếp nhau trong bộ nhớ.
- Về cơ bản, máy tính sẽ dựa trên địa chỉ **GỐC** (trỏ bởi biến **p** – cũng là địa chỉ của phần tử đầu tiên của mảng **p**) của mảng, rồi đếm tuần tự lần lượt.



Biểu diễn mảng 1 chiều trong ngôn ngữ C

Bộ nhớ



start_address (địa chỉ gốc)

Ví dụ: Mảng 1 chiều $x = [a, b, c, d]$

- Lưu trữ vào một khối bộ nhớ liên tiếp (contiguous memory locations)
- Địa chỉ($x[i]$) = $\text{start_address} + W * i$

biết rằng:

- start_address: địa chỉ phần tử đầu tiên trong mảng
- W: kích thước của mỗi phần tử trong mảng

Ví dụ: Xét khai báo **float list[5];**

- Khai báo trên sẽ khai báo biến mảng tên list với 5 phần tử có kiểu là số thực (4 byte).
- **Địa chỉ của các phần tử trong mảng một chiều**

<code>list[0]</code>	địa chỉ gốc = α
<code>list[1]</code>	$\alpha + \text{sizeof}(\text{float})$
<code>list[2]</code>	$\alpha + 2 * \text{sizeof}(\text{float})$
<code>list[3]</code>	$\alpha + 3 * \text{sizeof}(\text{float})$
<code>list[4]</code>	$\alpha + 4 * \text{size}(\text{float})$

Ví dụ

Chương trình trên ngôn ngữ C đưa ra địa chỉ các phần tử trong mảng 1 chiều:

```
#include <stdio.h>
int main()
{   int A[ ] = {5, 10, 12, 15, 4};
    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", &A[i], A[i]);
}
```

$\&A[i]$: địa chỉ của phần tử $A[i]$
 $A[i]$: nội dung của phần tử $A[i]$

Result in DevC
(sizeof(int)=4)

Address	Contents
6487536	5
6487540	10
6487544	12
6487548	15
6487552	4

Memory $\text{Location}(A[i]) = \text{start_address} + W * i$



$\text{start_address} = 6487536$

Result in turboC
(sizeof(int)=2)

Address	Contents
65516	5
65518	10
65520	12
65522	15
65524	4

Ví dụ

Chương trình trên ngôn ngữ C đưa ra địa chỉ các phần tử trong mảng 1 chiều:

```
#include <stdio.h>
int main()
{   int A[ ] = {5, 10, 12, 15, 4};
    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", &A[i], A[i]);
```

$\&A[i]$: địa chỉ của phần tử $A[i]$
 $A[i]$: nội dung của phần tử $A[i]$

```
printf("Address    Contents\n");
for (int i=0; i < 5; i++)
    printf("%8d %5d\n", A+i, *(A+i));
```

$A+i$: địa chỉ của phần tử $A[i]$
 $*(A+i)$: nội dung của phần tử $A[i]$

```
/* print the address of 1D array by using pointer */
int *ptr = A;
printf("Address    Contents\n");
for (int i=0; i < 5; i++)
    printf("%8d %5d\n", ptr+i, *(ptr+i));
```

$ptr+i$: địa chỉ của phần tử $A[i]$
 $*(ptr+i)$: nội dung của phần tử $A[i]$

```
}
```

Ví dụ

Chương trình trên ngôn ngữ C đưa ra địa chỉ các phần tử trong mảng 1 chiều:

```
#include <stdio.h>
int main()
{   int A[ ] = {5, 10, 12, 15, 4};
    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", &A[i], A[i]);

    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", A+i, *(A+i));

    /* print the address of 1D array by using pointer */
    int *ptr = A;
    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", ptr+i, *(ptr+i));
}
```

&A[i]

A[i]

A+i

*(A+i)

ptr+i

*(ptr+i)

Address	Contents
6487536	5
6487540	10
6487544	12
6487548	15
6487552	4
Address	Contents
6487536	5
6487540	10
6487544	12
6487548	15
6487552	4
Address	Contents
6487536	5
6487540	10
6487544	12
6487548	15
6487552	4

```

#include <stdio.h>

int main()
{
    int A[ ] = {5, 10, 12, 15, 4};
    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", &A[i], A[i]);

    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", A+i, *(A+i));

    /* print the address of 1D array by using pointer */
    int *ptr = A;
    printf("Address    Contents\n");
    for (int i=0; i < 5; i++)
        printf("%8d %5d\n", ptr+i, *(ptr+i));
}

```

So sánh `int *ptr` và `int A[5]`:

Giống nhau: `ptr` và `A` đều là **con trỏ**.

Khác nhau: `A` chiếm giữ **5 vị trí trong bộ nhớ (5 locations)**.

Kí hiệu:

`A` : 1 con trỏ trỏ tới phần tử `A[0]` (a pointer to `A[0]`)

`(A + i)` : 1 con trỏ trỏ tới phần tử `A[i]` (cũng có thể viết là **`&A[i]`**)

`*(A + i)` : nội dung chứa trong phần tử **`A[i]`**

Mảng trong C

```
int list[5], *plist[5];
```

list[5]: mảng gồm 5 số nguyên

list[0], list[1], list[2], list[3], list[4]

*plist[5]: một mảng gồm 5 con trỏ, mỗi con trỏ trỏ tới 1 số nguyên

plist[0], plist[1], plist[2], plist[3], plist[4]

Các thành phần của mảng list được lưu trữ trong bộ nhớ tại các địa chỉ:

list[0] địa chỉ gốc = α

list[1] $\alpha + \text{sizeof}(\text{int})$

list[2] $\alpha + 2 * \text{sizeof}(\text{int})$

list[3] $\alpha + 3 * \text{sizeof}(\text{int})$

list[4] $\alpha + 4 * \text{size}(\text{int})$

- So sánh `int *list1` và `int list2[5]`:

Giống nhau: list1 và list2 đều là **con trỏ**.

Khác nhau: list2 chiếm giữ **5 vị trí trong bộ nhớ (5 locations)**.

Kí hiệu:

list2 : 1 con trỏ trỏ tới list2[0] (a pointer to list2[0])

(list2 + i) : 1 con trỏ trỏ tới phần tử list2[i] (cũng có thể viết là **&list2[i]**)

*(list2 + i) : nội dung chứa trong phần tử **list2[i]**

Khai báo mảng 2 chiều (two-dimensional array)

- Cách khai báo:

`<element-type> <arrayName> [size1][size2];`

Ví dụ: `double a[4][3];`
giống như bảng 4 dòng 3 cột

dòng 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>
dòng 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>
dòng 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>
dòng 3	<code>a[3][0]</code>	<code>a[3][1]</code>	<code>a[3][2]</code>
	cột 0	cột 1	cột 2

- Cách khởi tạo:

Ví dụ: `int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};`

```
int a[3][4] = {  
    {1, 2, 3, 4}, /* khởi tạo giá trị cho hàng có chỉ mục là 0 */  
    {5, 6, 7, 8}, /* khởi tạo giá trị cho hàng có chỉ mục là 1 */  
    {9, 10, 11, 12} /* khởi tạo giá trị cho hàng có chỉ mục là 2 */  
};
```

<code>a[0][0] = 1</code>	<code>a[0][1]=2</code>	<code>a[0][2]=3</code>	<code>a[0][3]=4</code>
<code>a[1][0] = 5</code>	<code>a[1][1]=6</code>	<code>a[1][2]=7</code>	<code>a[1][3]=8</code>
<code>a[2][0] = 9</code>	<code>a[2][1]=10</code>	<code>a[2][2]=11</code>	<code>a[2][3]=12</code>

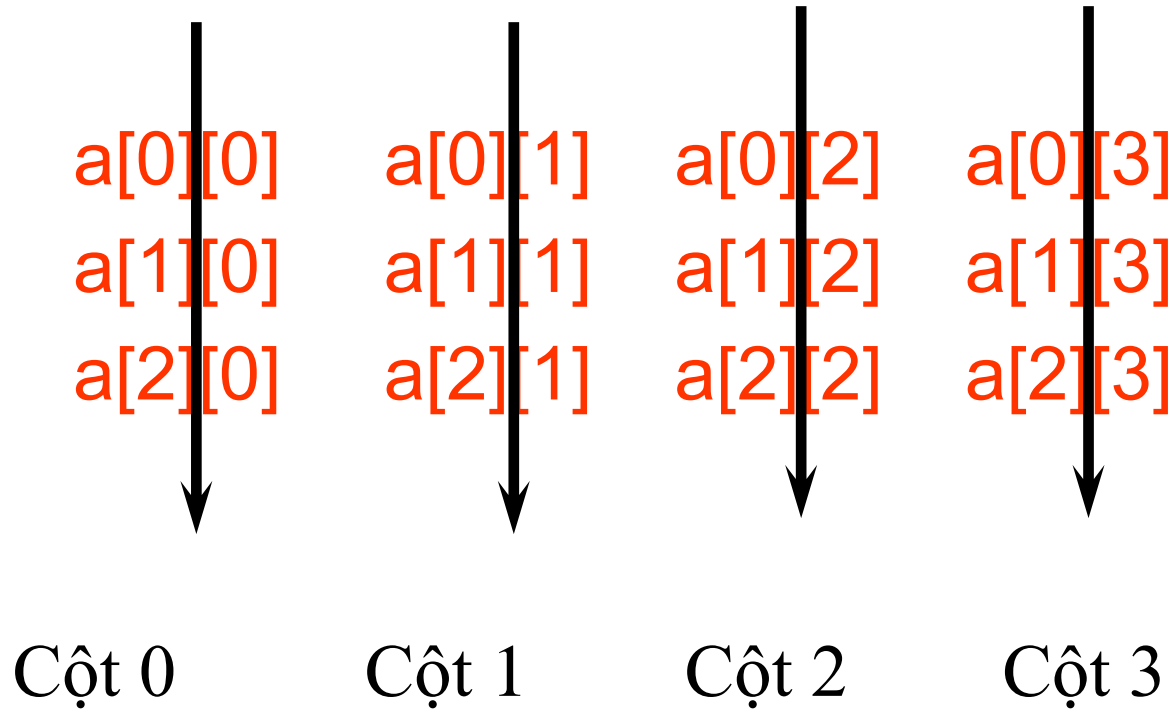
- Truy cập vào 1 phần tử của mảng:

- `a[2][1];`
- `a[i][j];`

Các dòng của mảng 2 chiều

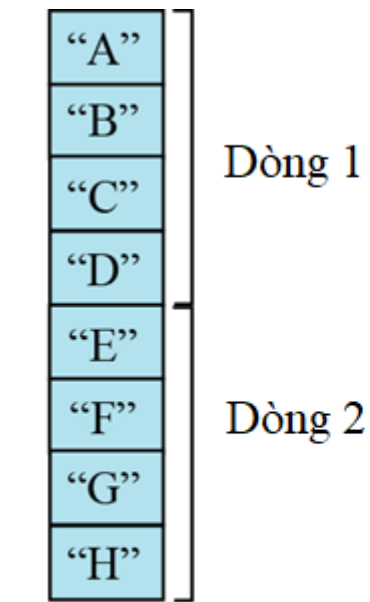
a[0][0]	a[0][1]	a[0][2]	a[0][3]	→ dòng 0
a[1][0]	a[1][1]	a[1][2]	a[1][3]	→ dòng 1
a[2][0]	a[2][1]	a[2][2]	a[2][3]	→ dòng 2

Các cột của mảng 2 chiều



Phân bổ bộ nhớ cho mảng

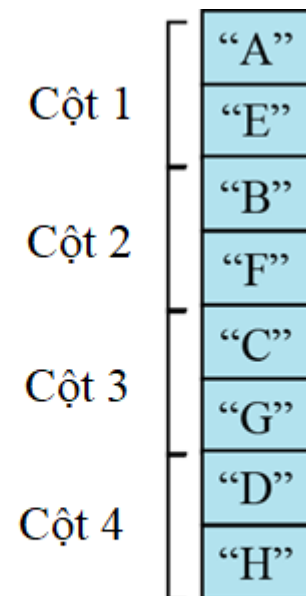
- Trong bộ nhớ, các phần tử của mảng đa chiều thường được lưu trữ kế tiếp nhau theo một trong 2 cách sau:
 - Hết dòng này đến dòng khác: thứ tự này được gọi là thứ tự ưu tiên dòng (**row major order**)
 - Hết cột này đến cột khác: thứ tự này được gọi là thứ tự ưu tiên cột (**column major order**)



Lưu trữ theo
thứ tự ưu tiên dòng

	[1]	[2]	[3]	[4]
[1]	"A"	"B"	"C"	"D"
[2]	"E"	"F"	"G"	"H"

User's View



Lưu trữ theo
thứ tự ưu tiên cột

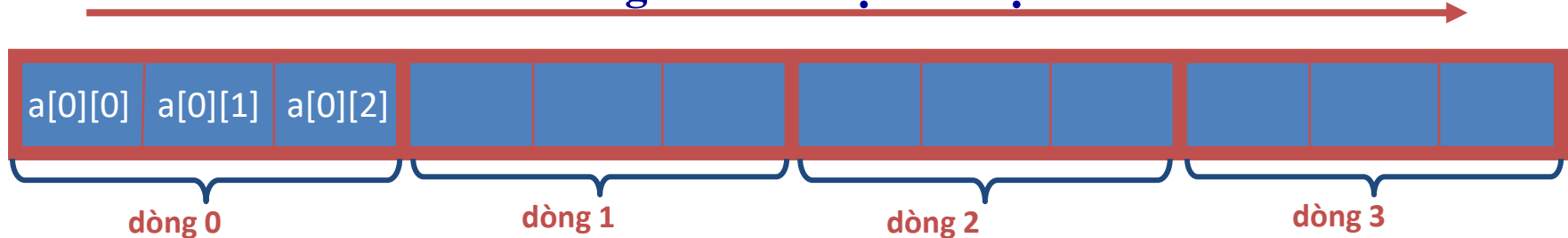
Thứ tự ưu tiên dòng (Ví dụ: Pascal, C/C++)

- Các phần tử của mảng nhiều chiều được lưu trữ kế tiếp nhau theo bộ nhớ, hết dòng này đến dòng khác. Vì vậy, phần tử ở dòng đầu tiên sẽ chiếm tập các vị trí ô nhớ đầu tiên của mảng, các phần tử ở dòng thứ 2 sẽ chiếm tập các ô nhớ tiếp theo,... cho đến dòng cuối cùng

Các phần tử của dòng 0	Các phần tử của dòng 1	Các phần tử của dòng 2	Các phần tử của dòng i
------------------------	------------------------	------------------------	------	------------------------	-------

- Ví dụ: `int a[4][3]`

Theo chiều tăng dần của địa chỉ bộ nhớ



- Ví dụ mảng 3 x 4:
- | | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

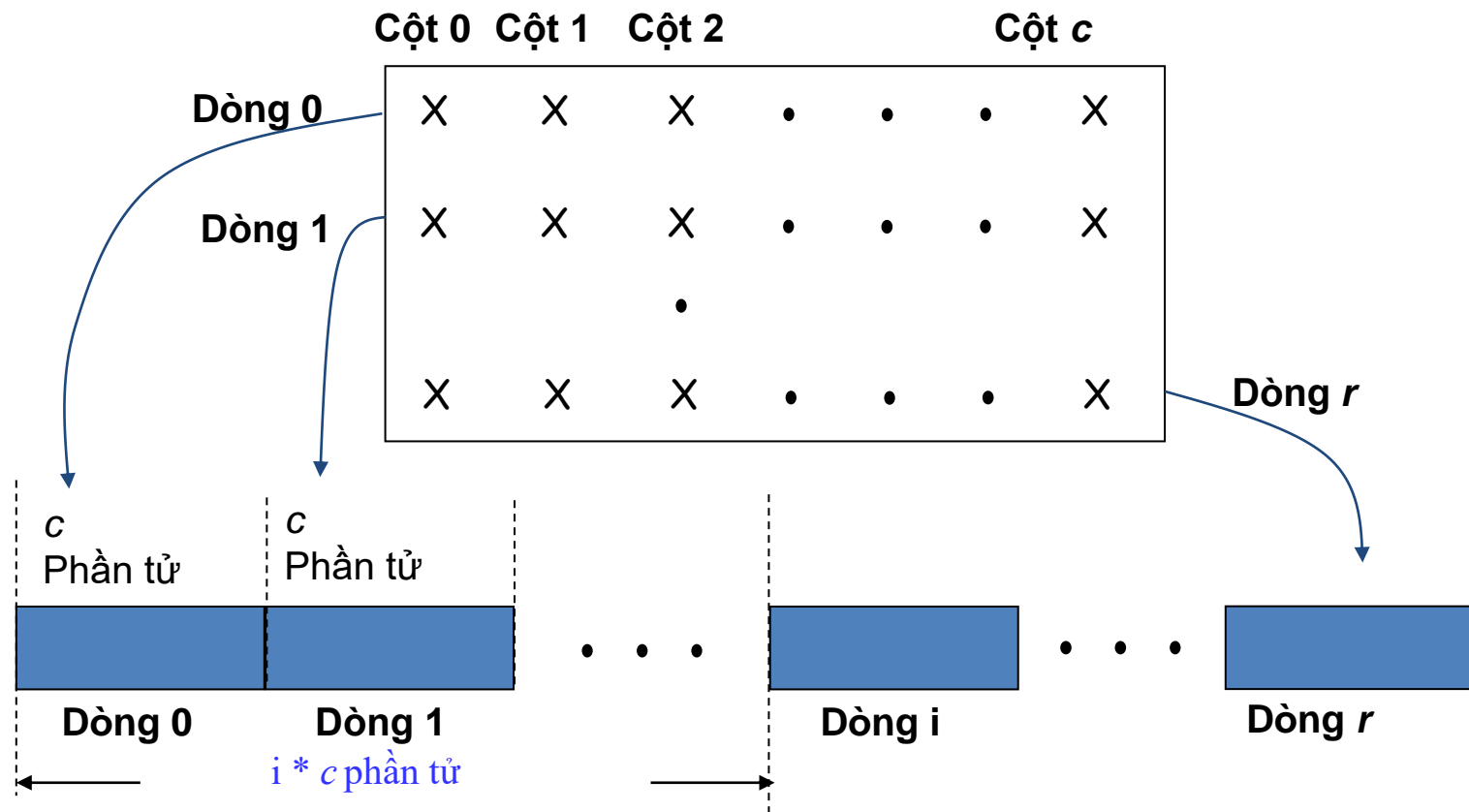
được đổi về mảng 1 chiều **Y** bằng cách gom các phần tử theo dòng:

- Trong mỗi dòng: các phần tử được gom từ trái sang phải.
- Các dòng được gom từ trên xuống dưới.

Khi đó, ta có mảng **Y**[] =

{a, b, c, d, e, f, g, h, i, j, k, l}

Mảng 2 chiều theo thứ tự ưu tiên dòng



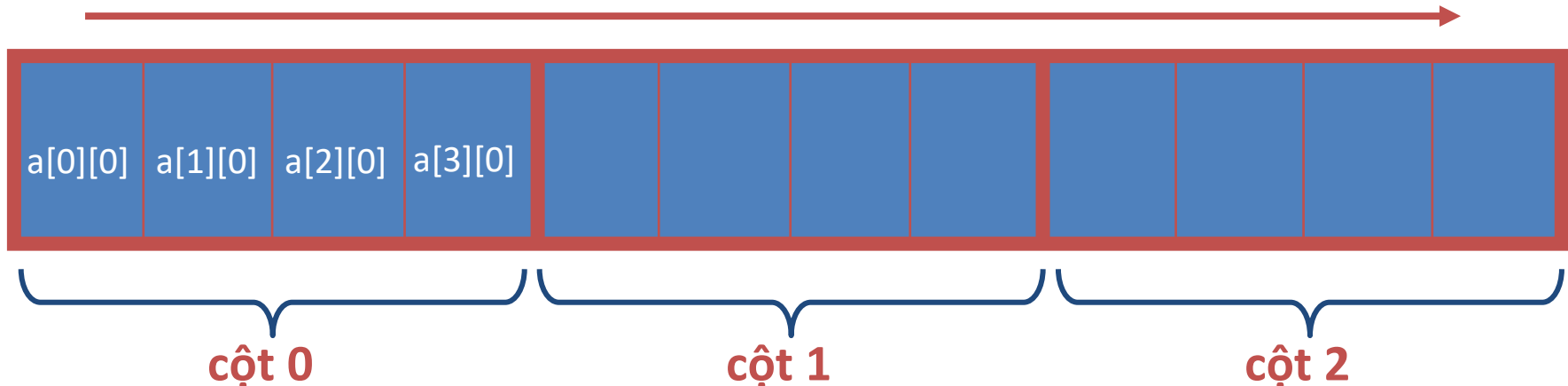
Thứ tự ưu tiên cột (ví dụ: Matlab, Fortran)

- Các phần tử của mảng nhiều chiều được lưu trữ kế tiếp nhau theo bộ nhớ, hết cột này đến cột khác. Vì vậy, các phần tử ở cột đầu tiên sẽ chiếm tập các vị trí ô nhớ đầu tiên của mảng, các phần tử ở cột thứ 2 sẽ chiếm tập các ô nhớ tiếp theo,... cho đến cột cuối cùng

Các phần tử của cột 0	Các phần tử của cột 1	Các phần tử của cột 2	Các phần tử của cột i
-----------------------	-----------------------	-----------------------	------	-----------------------	-------

- Ví dụ: `int a[4][3];`

Theo chiều tăng dần của địa chỉ bộ nhớ



Thứ tự ưu tiên cột (ví dụ: Matlab, Fortran)

- Các phần tử của mảng nhiều chiều được lưu trữ kế tiếp nhau trong bộ nhớ, hết cột này đến cột khác. Vì vậy, các phần tử ở cột đầu tiên sẽ chiếm tập các vị trí ô nhớ đầu tiên của mảng, các phần tử ở cột thứ 2 sẽ chiếm tập các ô nhớ tiếp theo,... cho đến cột cuối cùng

Các phần tử của cột 0	Các phần tử của cột 1	Các phần tử của cột 2	Các phần tử của cột i
-----------------------	-----------------------	-----------------------	------	-----------------------	-------

- Ví dụ mảng 3 x 4 :

a b c d

e f g h

i j k l

Được chuyển về mảng 1 chiều **Y** bằng cách gom các phần tử theo cột.

- Trong cùng 1 cột: các phần tử được gom từ trên xuống dưới.
- Các cột được gom từ trái sang phải.

Vì vậy, ta thu được mảng **Y**[] =

{a, e, i, b, f, j, c, g, k, d, h, l}

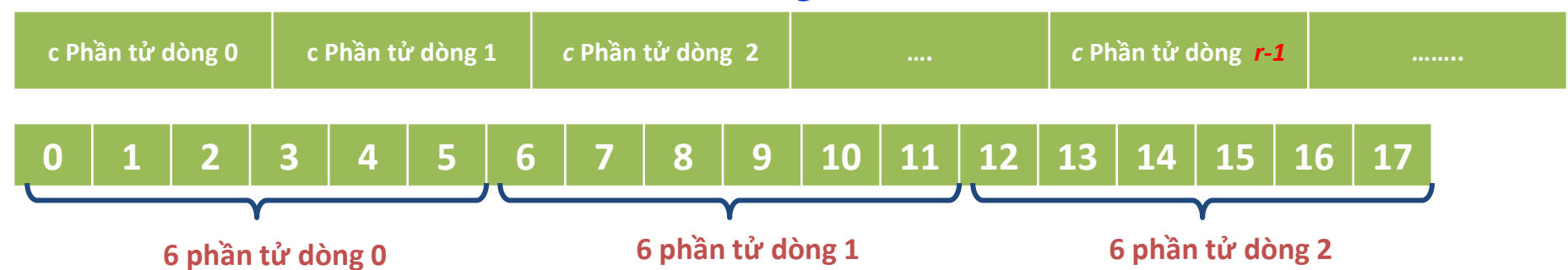
Ví dụ: Phân bổ bộ nhớ mảng 2 chiều: theo thứ tự ưu tiên dòng và thứ tự ưu tiên cột

Mảng 2 chiều: r dòng, c cột

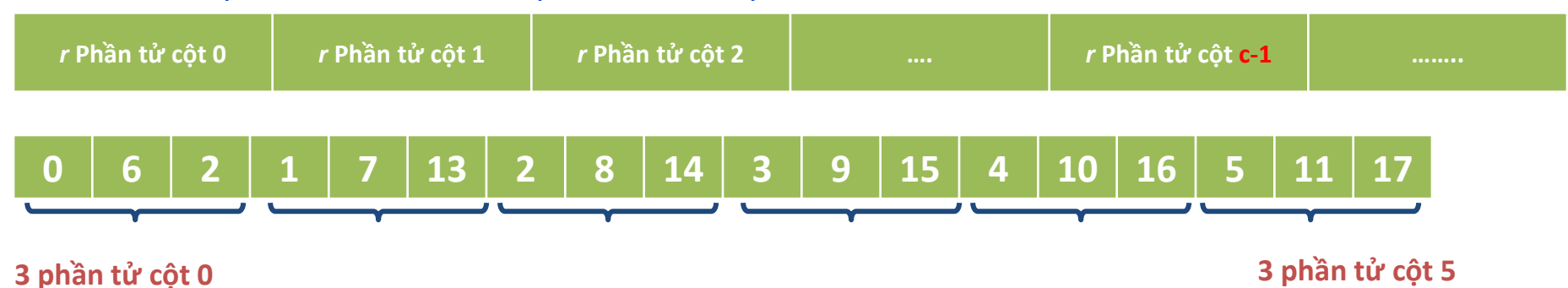
Ví dụ: `int a[3][6]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17};`

`a[0][0]=0` `a[0][1]=1` `a[0][2]=2` `a[0][3]=3` `a[0][4]=4` `a[0][5]=5`
`a[1][0]=6` `a[1][1]=7` `a[1][2]=8` `a[1][3]=9` `a[1][4]=10` `a[1][5]=11`
`a[2][0]=12` `a[2][1]=13` `a[2][2]=14` `a[2][3]=15` `a[2][4]=16` `a[2][5]=17`

Phân bổ bộ nhớ theo thứ tự ưu tiên dòng



Phân bổ bộ nhớ theo thứ tự ưu tiên cột



Xác định địa chỉ phần tử $x[i][j]$: thứ tự ưu tiên dòng

- Giả sử mảng x :
 - gồm r dòng và c cột (tức là, mỗi dòng gồm c phần tử)

c Phần tử dòng 0	c Phần tử dòng 1	c Phần tử dòng 2	c Phần tử dòng $r-1$
------------------	------------------	------------------	------	----------------------	-------

- Xác định địa chỉ phần tử $x[i][j]$:
 - i dòng nằm trước dòng $j \rightarrow$ có $i*c$ phần tử nằm trước phần tử $x[i][0]$
 - $x[i][j]$ đặt ở vị trí: $i*c + j$ của mảng 1 chiều
 - Địa chỉ của phần tử $x[i][j]$:

$$\text{Location}(x[i][j]) = \text{start_address} + W (i*c + j)$$

Với

- start_address: địa chỉ của phần tử đầu tiên ($x[0][0]$) trong mảng
- W : kích thước 1 phần tử
- c : số cột của mảng
- r : số dòng của mảng

Ví dụ

Chương trình trên ngôn ngữ C in địa chỉ của các phần tử thuộc mảng 2 chiều:

```
#include <stdio.h>
int main()
{
    int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
    printf("Address    Contents\n");
    for (int i=0; i < 3; i++)
        for (int j=0; j < 4; j++)
            printf("%8d %5d\n", &a[i][j], a[i][j]);
}
```

Result in DevC
(sizeof(int)=4)

Address	Contents
6487488	1
6487492	2
6487496	3
6487500	4
6487504	5
6487508	6
6487512	7
6487516	8
6487520	9
6487524	10
6487528	11
6487532	12

Bộ nhớ

$$\text{Location}(a[i][j]) = \text{start_address} + W * [(i * \text{cols}) + j]$$



↑
start_address=6487488

Location(a[1][2]) = ?

Chương trình trên ngôn ngữ C in địa chỉ của các phần tử thuộc mảng 2 chiều:

```
#include <stdio.h>

int main()
{
    int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

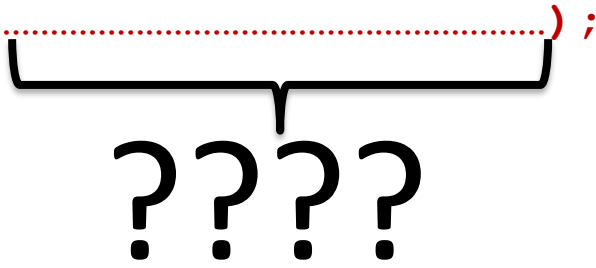
```
    printf("Address      Contents\n");
    for (int i=0; i < 3; i++)
        for (int j=0; j < 4; j++)
            printf("%8d %5d\n", &a[i][j], a[i][j]);
```

Bộ nhớ $\text{Location}(a[i][j]) = \text{start_address} + W * [(i * \text{cols}) + j]$



↑
start_address = địa chỉ phần tử a[0][0]

```
int *ptr = &a[0][0];
printf("Address      Contents\n");
for (int i=0; i < rows; i++)
    for (int j=0; j < cols; j++)
        printf("%8d %5d\n",
```



Ví dụ thứ tự ưu tiên dòng: phân bổ bộ nhớ cho mảng 2 chiều (kiểu số nguyên int)

- Địa chỉ của các phần tử trong mảng 2 chiều:

int a[4][3]

a[0][0]	địa chỉ = α
a[0][1]	$\alpha + 1 * \text{sizeof}(\text{int})$
a[0][2]	$\alpha + 2 * \text{sizeof}(\text{int})$
a[1][0]	$\alpha + 3 * \text{sizeof}(\text{int})$
a[1][1]	$\alpha + 4 * \text{sizeof}(\text{int})$
a[1][2]	$\alpha + 5 * \text{sizeof}(\text{int})$
a[2][0]	$\alpha + 6 * \text{sizeof}(\text{int})$
...	

Tổng quát: Khai báo

int a[m][n];

- Giả sử: địa chỉ của phần tử đầu tiên (a[0][0]) là α .
- Khi đó, địa chỉ của phần tử a[i][j] là:

$\alpha + (i * n + j) * \text{sizeof}(\text{int})$

Xác định địa chỉ phần tử $x[i][j]$: thứ tự ưu tiên cột

- Giả sử mảng x :

....	r Phần tử của cột 0	r Phần tử của cột 1	r Phần tử của cột 2	r Phần tử của cột j
------	-----------------------	-----------------------	-----------------------	------	-------------------------	-------

 - có r dòng và c cột (mỗi cột có r phần tử)

- Địa chỉ của phần tử $x[i][j]$:
 - j cột nằm trước cột $j \rightarrow$ do đó có $j*r$ phần tử nằm trước phần tử $x[0][j]$
 - $x[i][j]$ nằm ở vị trí: $j*r + i$ tính từ phần tử đầu tiên của mảng
 - $\text{Location}(x[i][j]) = \text{start_address} + W (j*r + i)$

Với

- start_address : địa chỉ của phần tử đầu tiên $x[0][0]$ của mảng
- W : kích thước mỗi phần tử
- c : số lượng cột của mảng
- r : số lượng dòng của mảng

Ví dụ: mảng : `int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};`

Tìm địa chỉ phần tử $a[1][2]$ trong bộ nhớ nếu $\text{start_address} = 6487488$

Ví dụ 1: Thứ tự ưu tiên dòng và thứ tự ưu tiên cột

Mảng 2 chiều:

```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

$a[0][0] = 1$	$a[0][1] = 2$	$a[0][2] = 3$	$a[0][3] = 4$
$a[1][0] = 5$	$a[1][1] = 6$	$a[1][2] = 7$	$a[1][3] = 8$
$a[2][0] = 9$	$a[2][1] = 10$	$a[2][2] = 11$	$a[2][3] = 12$

Phân bố bộ nhớ theo thứ tự ưu tiên dòng

$$\text{Location}(a[i][j]) = \text{start_address} + W * [(i * \text{cols}) + j]$$

		1	2	3	4	5	6	7	8	9	10	11	12			
--	--	---	---	---	---	---	---	---	---	---	----	----	----	--	--	--



start_address=6487488

Location($a[1][2]$) = ?

Phân bố bộ nhớ theo thứ tự ưu tiên cột

$$\text{Location}(a[i][j]) = \text{start_address} + W * [(j * \text{rows}) + i]$$

		1	5	9	2	6	10	3	7	11	4	8	12			
--	--	---	---	---	---	---	----	---	---	----	---	---	----	--	--	--



start_address=6487488

Location($a[1][2]$) = ?

Ví dụ 2: Thứ tự ưu tiên dòng và thứ tự ưu tiên cột

Mảng 2 chiều: r dòng, c cột

Ví dụ: `int a[3][6]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17};`

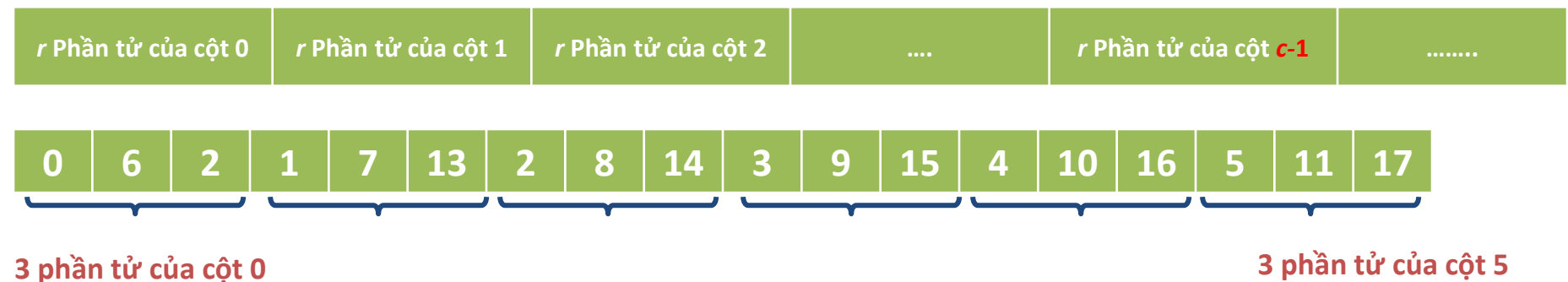
`a[0][0]=0` `a[0][1]=1` `a[0][2]=2` `a[0][3]=3` `a[0][4]=4` `a[0][5]=5`
`a[1][0]=6` `a[1][1]=7` `a[1][2]=8` `a[1][3]=9` `a[1][4]=10` `a[1][5]=11`
`a[2][0]=12` `a[2][1]=13` `a[2][2]=14` `a[2][3]=15` `a[2][4]=16` `a[2][5]=17`

Thứ tự ưu tiên dòng

`start_address = 1000` → `Location(a[1][4]) = ?`

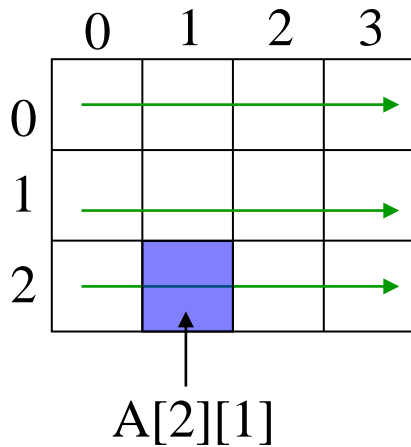


Thứ tự ưu tiên cột



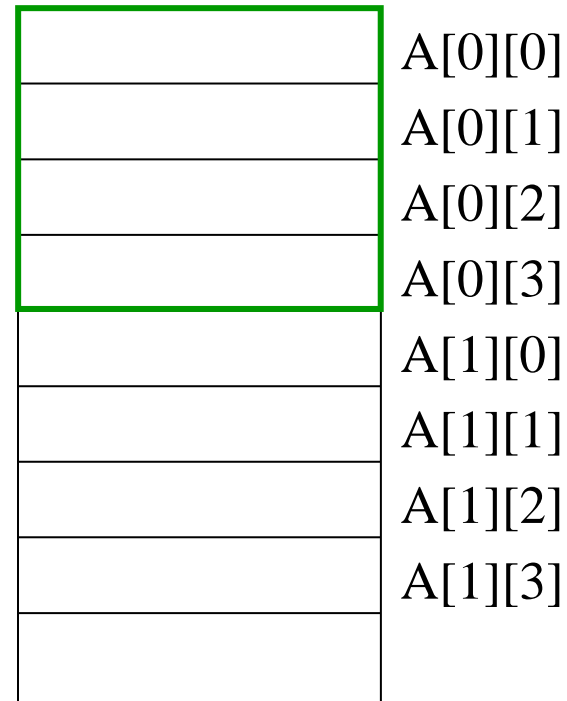
Phân bổ bộ nhớ

- `char A[3][4];`
- Cấu trúc logic



// Thứ tự ưu tiên dòng

Cấu trúc vật lý



Phân bổ bộ nhớ:

$$\text{Location}(A[i][j]) = \text{Location}(A[0][0]) + i*4 + j$$

Các thao tác trên mảng

- Các thao tác cơ bản trên mảng bao gồm: tìm kiếm (search), thêm/chèn (insert), xóa (delete), truy xuất thông tin (Retrieval), duyệt (traversal).

Ví dụ: Mảng S gồm n số nguyên: $S[0], S[1], \dots, S[n-1]$

- **Tìm kiếm** : tìm xem giá trị **key** có xuất hiện trong mảng S hay không
function `Search(S, key)` trả về giá trị `true` nếu **key** có trong S ; `false` nếu ngược lại
- **Truy xuất thông tin**: xác định giá trị của phần tử tại chỉ số **i** trong mảng S
function `Retrieve(S, i)` : trả về giá trị của phần tử $S[i]$ nếu $0 \leq i \leq n-1$
- **Duyệt**: in ra giá trị của tất cả các phần tử thuộc mảng S
function `PrintArray(S, n)`
- **Thêm**: thêm giá trị **key** vào mảng S
- **Xóa**: xóa phần tử tại chỉ số **i** của mảng S

Các thao tác trên mảng

- Các thao tác cơ bản trên mảng bao gồm: tìm kiếm (search), thêm/chèn (insert), xóa (delete), truy xuất thông tin (Retrieval), duyệt (traversal).
- Các thao tác:
 - tìm kiếm, truy xuất thông tin và duyệt mảng là những thao tác đơn giản;
 - thêm và xóa phần tử của mảng tốn thời gian hơn, vì:
 - Trước khi thêm 1 phần tử vào mảng, ta cần dịch chuyển các phần tử sau vị trí chèn về đằng sau (bên phải) 1 vị trí;
 - Sau khi xóa 1 phần tử thuộc mảng, ta cần đẩy các phần tử sau phần tử xóa về phía trước (bên trái) 1 vị trí

Các thao tác trên mảng

Các thao tác trên mảng đã thảo luận trong phần trước cho ta gợi ý khi nào thì nên sử dụng mảng ?:

- Nếu chúng ta có một danh sách và cần phải thực hiện rất nhiều thao tác thêm/xóa trên danh sách này thì không nên sử dụng mảng
- Nên dùng mảng khi chỉ cần thực hiện ít các thao tác thêm/xóa, nhưng nhiều thao tác tìm kiếm/truy xuất thông tin



Mảng là cấu trúc phù hợp cho trường hợp chỉ thực hiện một số lượng nhỏ các thao tác thêm/xóa, nhưng cần thực hiện nhiều thao tác tìm kiếm/truy xuất thông tin.

Thư viện STL trong C++: Vector

`#include <vector>` Vector có thể xem như là một mảng động (dynamic array): một mảng có thể thay đổi kích thước.

Vector 1 chiều:

```
vector<int> vec1; //khai báo vector 1 chiều, rỗng, có kiểu dữ liệu là int
vector<int> vec2 (4, 100); //tạo vector có 4 phần tử int, khởi tạo giá trị cho cả 4
phần tử này là 100
vector<int> vec3 (vec2.begin(), vec2.end()); //khai báo vector kiểu int có
tên là vec3 sao chép từ đầu đến cuối vector vec2
vector<int> vec4(vec2); //tạo vector vec4 kiểu int và sao chép tất cả phần tử của
vec2; cách khai báo này cho kết quả giống như khai báo vec3
```

Vector 2 chiều:

```
vector<vector<int>> v; //khai báo vector 2 chiều, rỗng, có kiểu dữ liệu là int
vector<vector<int>> v(5,10); //khai báo vector 2 chiều kích thước 5*10
vector<vector<int>> v(5); khai báo vector 2 chiều, có 5 vector 1 chiều đều rỗng
vector<vector<int>> v(5, vector<int>(10,1)); //khai báo vector 5x10
với các phần tử khởi tạo giá trị đều là 1
```

Thư viện STL trong C++: Vector

`vector <int> v;`

`v.size()` : trả về số lượng phần tử

`v.empty()` : true/false

`v[i]` : phần tử thứ *i* trong vector *v*

`v.at(i)` : phần tử thứ *i* trong vector *v*

`v.front()` : phần tử đầu tiên trong vector *v*

`v.back()` : phần tử cuối cùng trong vector *v*

`v.push_back(x)` : thêm phần tử có giá trị *x* vào cuối vector *v*

`v.pop_back()` : loại bỏ phần tử cuối cùng ra khỏi vector *v*

Nội dung

1. Mảng (Array)
- 2. Bản ghi (Record)**
3. Danh sách liên kết (Linked List)
4. Ngăn xếp (Stack)
5. Hàng đợi (Queue)

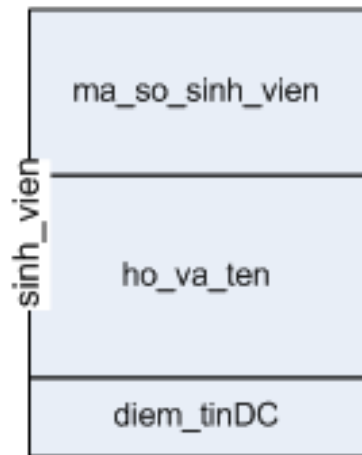
2. Bản ghi (record)

- Bản ghi (record) được tạo bởi một họ các trường (field) có thể có kiểu rất khác nhau

Khai báo kiểu dữ liệu bản ghi

- Khai báo kiểu bản ghi

```
struct  tên_bản_ghi{  
    <khai báo các trường >  
}
```



- Ví dụ

```
struct sinh_vien{  
    char ma_so_sinh_vien[10];  
    char ho_va_ten[30];  
    float diem_tinDC;  
}
```

```
struct point_3D{  
    float x;  
    float y;  
    float z;  
}
```

Khai báo biến bản ghi

- Cú pháp:

<tên_cấu_trúc> <tên_biến_cấu_trúc>;

- Ví dụ:

sinh_vien a, b, c;

- Kết hợp khai báo

```
struct [tên_cấu_trúc] {  
    <khai báo các trường dữ liệu>;  
} tên_biến_cấu_trúc;
```

```
struct sinh_vien{  
    char ma_so_sinh_vien[10];  
    char ho_va_ten[30];  
    float diem_tinDC;  
}sv1,sv2;
```

```
struct {  
    char ma_so_sinh_vien[10];  
    char ho_va_ten[30];  
    float diem_tinDC;  
}sv1,sv2;
```

Khai báo biến cấu trúc

- Các cấu trúc có thể được khai báo lồng nhau:

```
struct diem_thi {  
    float dToan, dLy, dHoa;  
};  
struct thi_sinh{  
    char SBD[10];  
    char ho_va_ten[30];  
    diem_thi ket_qua;  
} thi_sinh_1, thi_sinh_2;
```

- Có thể khai báo trực tiếp các trường dữ liệu của một cấu trúc bên trong cấu trúc khác:

```
struct thi_sinh{  
    char SBD[10];  
    char ho_va_ten[30];  
    struct diem_thi{  
        float dToan, dLy, dHoa;  
    } ket_qua;  
} thi_sinh_1, thi_sinh_2;
```

Khai báo biến cấu trúc

- Có thể gán giá trị khởi đầu cho một biến cấu trúc, theo nguyên tắc như kiểu mảng:

Ví dụ:

```
struct Date{
    int day;
    int month;
    int year;
};
struct{
    char Ten[20];
    struct Date NS;
} SV = {"Tran Anh", 20, 12, 1990 };
```

```
struct{
    char Ten[20];
    struct Date{
        int day;
        int month;
        int year;
    } NS;
} SV = {"Tran Anh", 20, 12, 1990 };
```


Xử lý dữ liệu bản ghi

- Truy cập các trường dữ liệu
- Phép gán giữa các biến bản ghi

Truy cập các trường dữ liệu

- Cú pháp

<tên_biến_bản_ghi>.<tên_trường>

- Lưu ý
 - Dấu “.” là toán tử truy cập vào trường dữ liệu trong bản ghi
 - Nếu trường dữ liệu là một bản ghi => sử dụng tiếp dấu “.” để truy cập vào thành phần mức sâu hơn

Ví dụ

```
#include <stdio.h>
void main() {
    struct {
        char Ten[20];
        struct Date {
            int day;
            int month;
            int year;
        } NS;
    } SV = {"Tran Anh", 20, 12, 1990 };

    printf(" Sinh vien %s (%d/%d/%d)",
SV.Ten, SV.NS.day, SV.NS.month, SV.NS.year) ;
}
```

Sinh vien Tran Anh (20/12/1990)

Phép gán giữa các biến bản ghi

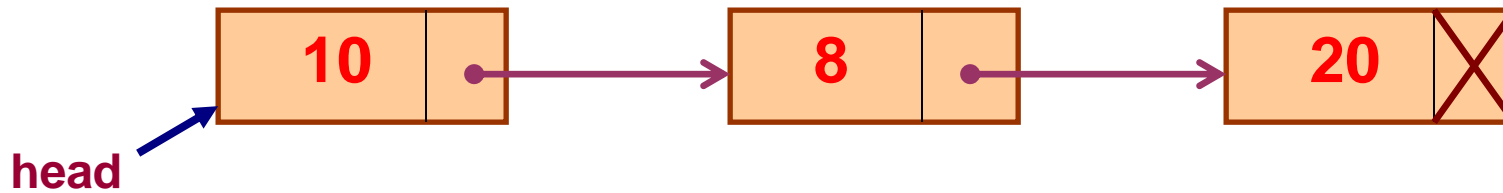
- Muốn sao chép dữ liệu từ biến bản ghi này sang biến bản ghi khác cùng kiểu
 - gán lần lượt từng trường trong hai biến bản ghi → “thủ công”
 - C cung cấp phép gán hai biến bản ghi cùng kiểu:
 $\text{biến_cấu_trúc_1} = \text{biến_cấu_trúc_2};$

Nội dung

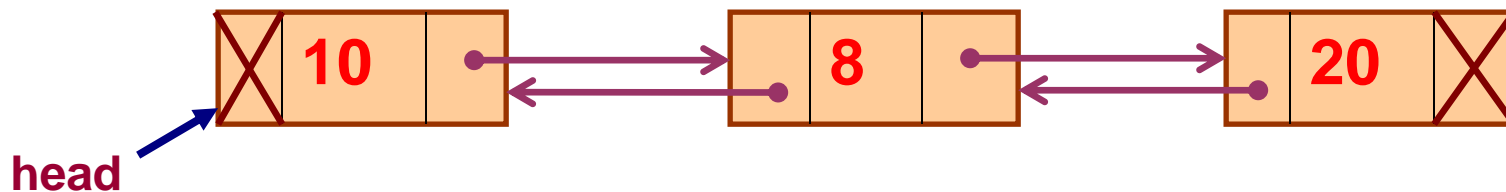
1. Mảng (Array)
2. Bản ghi (Record)
- 3. Danh sách liên kết (Linked List)**
4. Ngăn xếp (Stack)
5. Hàng đợi (Queue)

3. Danh sách liên kết/móc nối (Linked list)

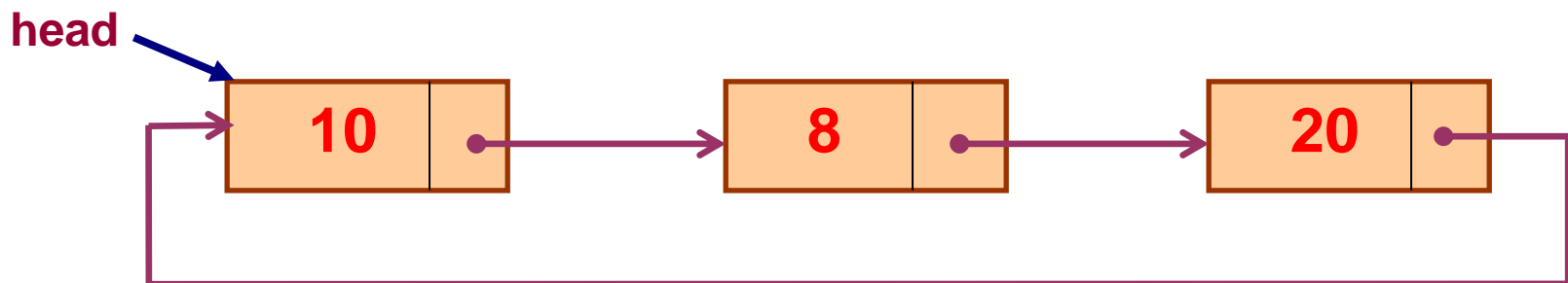
- Danh sách liên kết đơn (Singly linked list)



- Danh sách liên kết đôi (Doubly linked list)

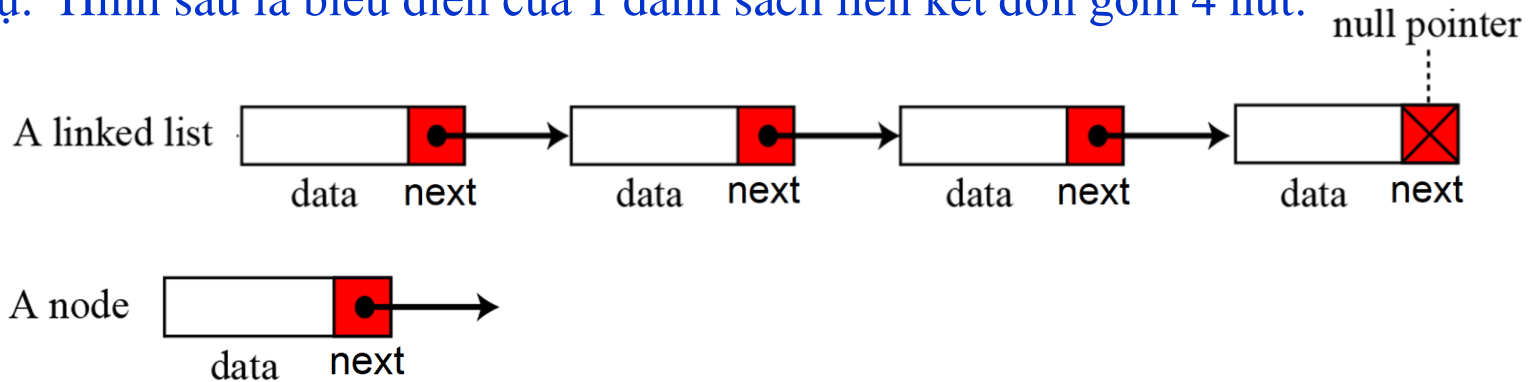


- Danh sách liên kết vòng (Circular linked list)

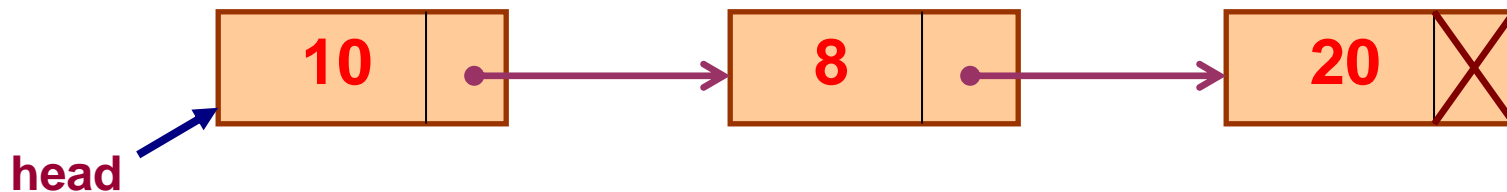


Danh sách liên kết đơn (Singly Linked list)

- Danh sách liên kết đơn gồm một dãy các nút (node), mỗi nút gồm 2 phần: **dữ liệu (data)** và **con trỏ** trỏ tới nút liền kề tiếp trong danh sách (con trỏ này chứa địa chỉ của nút tiếp theo).
- Ví dụ: Hình sau là biểu diễn của 1 danh sách liên kết đơn gồm 4 nút:

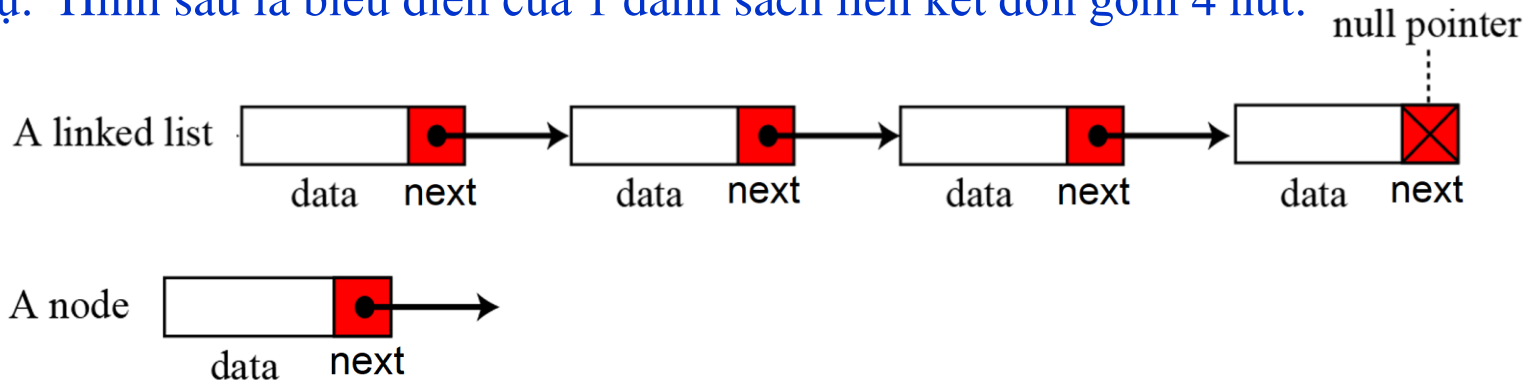


- Để định danh được danh sách liên kết đơn:
 - Ta cần biết con trỏ trỏ đến phần tử đầu tiên của danh sách (cần biết địa chỉ của nút đầu tiên) (thường con trỏ này được kí hiệu là *start* hoặc *head*)
 - Nếu con trỏ head = NULL, danh sách liên kết đơn rỗng (không có nút nào)



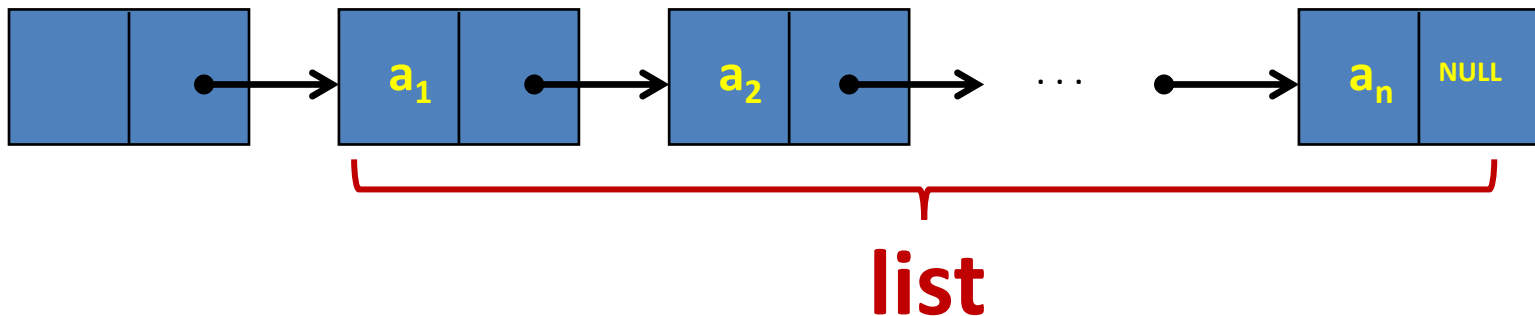
Danh sách liên kết đơn (Singly Linked list)

- Danh sách liên kết đơn gồm một dãy các nút (node), mỗi nút gồm 2 phần: dữ liệu (data) và con trỏ trỏ tới nút liền kề tiếp trong danh sách (con trỏ này chứa địa chỉ của nút tiếp theo).
- Ví dụ: Hình sau là biểu diễn của 1 danh sách liên kết đơn gồm 4 nút:



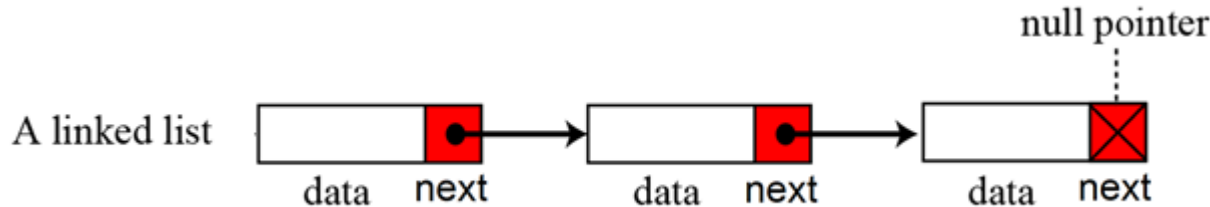
- Để định danh được danh sách liên kết đơn:
 - Ta cần biết con trỏ trỏ đến phần tử đầu tiên của danh sách (cần biết địa chỉ của nút đầu tiên) (thường con trỏ này được kí hiệu là *start* hoặc *head*)
 - Nếu con trỏ head = NULL, danh sách liên kết đơn rỗng (không có nút nào)

head

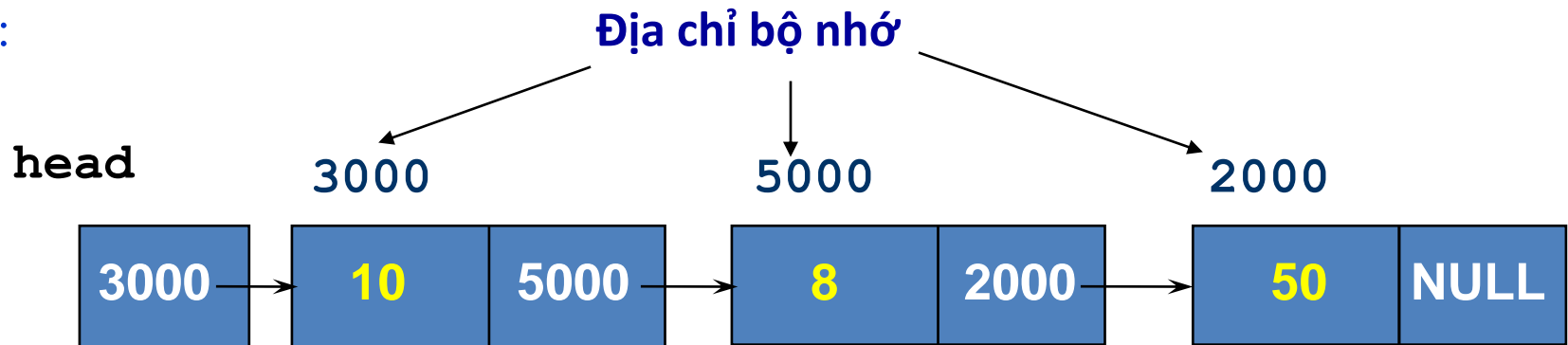


Danh sách liên kết đơn (Singly Linked list)

- Danh sách liên kết đơn gồm một dãy các nút (node), mỗi nút gồm 2 phần: **dữ liệu (data)** và **con trỏ** trỏ tới nút liền kề tiếp trong danh sách (con trỏ này chứa địa chỉ của nút tiếp theo).



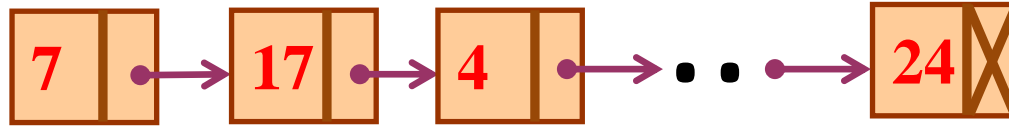
- Hình vẽ trên biểu diễn 1 danh sách gồm 3 nút. Mỗi nối giữa 2 nút liên tiếp được biểu diễn bởi 1 đường thẳng: một đầu có mũi tên, đầu còn lại là một chấm tròn.
- Ví dụ: 3 số nguyên 10, 8, 50 được lưu trữ bởi danh sách liên kết đơn gồm 3 nút như sau:



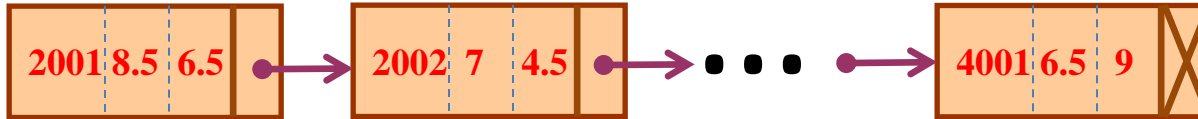
- Các phần tử thuộc danh sách liên kết có thể nằm ở vị trí bất kỳ trong bộ nhớ
(? So sánh với việc lưu trữ trong bộ nhớ của các phần tử thuộc cùng 1 mảng đã học ở phần trước)

Cách khai báo danh sách liên kết đơn trong ngôn ngữ C

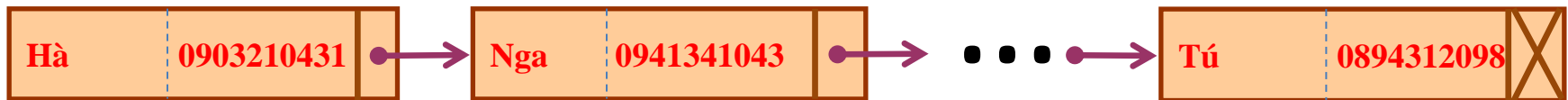
- Danh sách các số nguyên :



- Danh sách các sinh viên gồm các dữ liệu: mã sinh viên, điểm 2 môn toán và vật lý



- Danh sách các contact trong danh bạ điện thoại gồm các dữ liệu: tên, số điện thoại



➔ Cách khai báo 1 danh sách liên kết đơn:

- Đầu tiên cần khai báo kiểu của các dữ liệu lưu trữ trong 1 nút,
- Sau đó, khai báo danh sách liên kết đơn gồm (1) dữ liệu của nút, và (2) con trỏ lưu trữ địa chỉ của nút tiếp theo