

# Nhập Môn Lập Trình Cấu Trúc Mảng

TS. Lê Nguyên Khôi  
Trường Đại học Công nghệ, ĐHQGHN

# Nội Dung

---

- ▶ Khái niệm cơ bản
- ▶ Truyền mảng vào hàm
- ▶ Lập trình với mảng

# Bảng – Dữ Liệu

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18
0	3	6	9	12	15	18	21	24	27
0	4	8	12	16	20	24	28	32	36
0	5	10	15	20	25	30	35	40	45
0	6	12	18	24	30	36	42	48	54
0	7	14	21	28	35	42	49	56	63
0	8	16	24	32	40	48	56	64	72
0	9	18	27	36	45	54	63	72	81

# Bảng – Vị Trí

start →

00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

← end

# Bảng – Ví Dụ

---

	a	b	c	d	e	f	g	h	
8									8
7									7
6									6
5									5
4									4
3									3
2									2
1									1
	a	b	c	d	e	f	g	h	

# Bảng – Ưu Điểm

---

- ▶ Biến không phù hợp để biểu diễn và xử lý những dữ liệu phức tạp, ví dụ:
  - ▶ danh sách điểm thi của một sinh viên
  - ▶ danh sách sinh viên của lớp học
  - ▶ sắp xếp, tìm kiếm, tính toán trên chuỗi dữ liệu
- ▶ Ngôn ngữ lập trình (C++) cung cấp kiểu dữ liệu có cấu trúc để xử lý các vấn đề trên
  - ▶ mảng là cấu trúc dữ liệu thông dụng nhất
  - ▶ mảng dùng để lưu trữ dữ liệu **cùng kiểu**
  - ▶ mảng tránh phải sử dụng nhiều biến

# Khái Niệm Chung

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...

- ▶ Mảng là chuỗi dữ liệu có cùng kiểu
- ▶ Mỗi phần tử mảng lưu một dữ liệu
- ▶ Mảng được đặt tại vùng nhớ liên tiếp
- ▶ Phần tử mảng được đánh số liên tiếp từ 0
- ▶ Kích thước của mảng không thay đổi

# Truy Cập Dữ Liệu

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...

- ▶ Dùng chỉ số **index** để truy cập phần tử
  - ▶ **value[3]** (=2) , **value[6]** (=9) , **value[9]** (=5)
- ▶ Truy cập phần tử mảng, sử dụng:
  - ▶ tên mảng (**value**)
  - ▶ chỉ số phần tử **index** là một giá trị/biểu thức nguyên
  - ▶ **value [index=5]**: truy cập phần tử
    - chỉ số là **5**, dữ liệu là **3**



# Truy Cập Dữ Liệu

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...
iterator							i					

```
value [3] = 9; // 2 -> 9
```

```
// với iterator i=5 -> value [6] = 5
```

```
value [i+1] = 5; // 9 -> 5
```

```
// thực hiện tính toán
```

```
value [9] = value [9] - value [i]; // 5 -> 2
```

```
//in ra ???
```

```
cout << value [3] << value [6] << value [9];
```

# Truy Cập Dữ Liệu

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...
iterator		$i \longrightarrow i$										
iterator		$j \longleftarrow j$										

```
for (int i_iter = 0; i_iter < 10; i_iter++)
{
    value [i_iter] = ... ..;
}



for (int j_iter = 10-1; j_iter > -1; j_iter--)
{
    value [j_iter] = ... ..;
}
```

# Truy Cập Dữ Liệu

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...
iterator				i	→				i			
iterator				j	←				j			
				m					n			


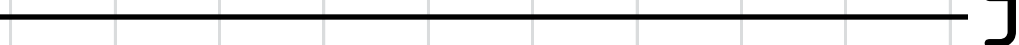
```
for (int i = m; i < n; i++) {  
    value [i] = ... ..;  
}  
  
for (int j = n; j > m; j--) {  
    value [j] = ... ..;  
}
```

# Truy Cập Dữ Liệu

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...
iterator												
iterator												

```
print_all_pairs()
{
    for (int i = 0; i < _SIZE_-1; i++)
    {
        for (int j = i+1; j < _SIZE_; j++)
        {
            cout << value [i] << value [j];
        }
    }
}
```

# Truy Cập Dữ Liệu

data	...	1	0	6	2	8	3	9	4	7	5	...
iterator		<b>i</b>  <b>i</b>										
index		0	1	2	3	4	5	6	7	8	9	
iterator		<b>j</b>  <b>j</b>										
data	...	5	7	4	9	3	8	2	6	0	1	...

```
print_selected_pairs() {  
    for (int i = 0; i < _SIZE_; i++) {  
        for (int j = _SIZE_-1; j > -1; j--) {  
            if (old_data [i] == new_data [j]) {  
                cout << i << j << endl;  
                break;  
            }  
        }  
    }  
}
```

# Khai Báo Mảng

---

- ▶ Cú pháp:

**data\_type array\_name [\_SIZE\_] ;**

- ▶ Ví dụ:

**int value [10] ;**

- ▶ khai báo mảng tên **value**, kích thước **10** phần tử, lưu trữ dữ liệu kiểu **int**

- ▶ Lưu ý:

- ▶ mảng phải được khai báo trước khi sử dụng
- ▶ khi khai báo phải xác định số phần tử mảng
- ▶ số phần tử của mảng không thay đổi sau khai báo

# Khởi Tạo Mảng

---

- ▶ Mảng có thể được khởi tạo như sau:

```
int value [6] = {1, 0, 6, 2, 8, 3};
```

- ▶ lưu ý: số dữ liệu trong danh sách khởi tạo (trong **{ }**) không được vượt quá kích thước mảng (trong **[ ]**)

- ▶ Mảng cũng có thể được khởi tạo như sau:

```
int value [] = {1, 0, 6, 2, 8, 3};
```

- ▶ lưu ý: C++ sẽ tự xác định độ dài của mảng dựa trên độ dài của danh sách khởi tạo

- ▶ Mảng không được khởi tạo, phần tử mảng có dữ liệu không xác định (giống biến)

# Thao Tác Dữ Liệu Mảng

index		0	1	2	3	4	5	6	7	8	9	
data	...	1	0	6	2	8	3	9	4	7	5	...

```
const int _SIZE_ = 10;
int main() {
    int sum [_SIZE_];
    // int value [] = {1,0,6,2,8,3,9,4,7,5};
    int value [_SIZE_];
    for (int i = 0; i < _SIZE_; i++)
        value [i] = rand() % 10;
    sum [0] = value [0];
    for (int i = 1; i < _SIZE_; i++)
        sum [i] = sum [i-1] + value [i];
    return 0;
}
```



# Thao Tác Dữ Liệu Mảng

---

- ▶ Thống kê số lần xuất hiện các mặt từ 1 đến 6, sau 600 lần tung xúc xắc

```
const int _DICE_ = 6;
const int _ROLL_ = 600;
int main()
{
    int face_count [_DICE_] = {0};
    for (int i = 0; i < _ROLL_; i++)
    {
        int face = rand() % _DICE_;
        face_count [face] ++;
    }
    return 0;
}
```

# Quản Lý Chỉ Số Phần Tử

---

- ▶ Khai báo mảng có kích thước **`_SIZE_`**
  - ▶ chỉ số các phần tử từ **0** đến **`_SIZE_-1`**
- ▶ Tất cả các chỉ số khác không hợp lệ
- ▶ Truy cập vào các chỉ số không hợp lệ
  - ▶ ví dụ: **`value [-1]`** , **`value [_SIZE_]`**
  - ▶ truy cập ra vùng bộ nhớ ngoài mảng
  - ▶ không gây lỗi cú pháp (lỗi dịch)
  - ▶ có thể gây lỗi khi chạy chương trình
    - chương trình chạy sai
    - dừng đột ngột
  - ▶ lập trình viên có trách nhiệm kiểm soát giá trị chỉ số

# Truyền Mảng Cho Hàm

- ▶ Dùng tên mảng làm tham số truyền cho hàm
- ▶ Hàm thay đổi dữ liệu trong mảng
  - ▶ kết thúc hàm, dữ liệu của mảng thay đổi
- ▶ Mảng được truyền theo kiểu tham chiếu
  - ▶ nhưng không sử dụng toán tử **&**

```
void nhapMang(int value []) {  
    for (int i = 0; i < _SIZE_; i++)  
        cin >> value [i];  
}  
int main() {  
    int value [_SIZE_];  
    nhapMang(value);  
    return 0;  
}
```

# Truyền Mảng Cho Hàm

- ▶ Nếu không muốn hàm thay đổi dữ liệu của mảng
  - ▶ dùng khai báo hằng số trong chữ ký hàm
  - ▶ thay đổi dữ liệu mảng (gán, nhập) gây lỗi dịch

```
double dtb(const int value [], int _SIZE_) {  
    double sum = 0.0;  
    for (int i = 0; i < _SIZE_; i++)  
        sum = sum + value [i];  
    return (sum / _SIZE_);  
}  
  
int main() {  
    int value [_SIZE_];  
    nhapMang(value);  
    cout << "dtb: " << dtb(value, _SIZE_);  
    return 0;  
}
```

# Lập Trình Với Mảng

---

- ▶ Nhập dữ liệu
- ▶ In dữ liệu
- ▶ Tìm dữ liệu
- ▶ Thêm dữ liệu
- ▶ Xóa dữ liệu
- ▶ Tìm dữ liệu nhỏ nhất
- ▶ Sắp xếp dữ liệu

# Nhập & In Dữ Liệu

---

```
void nhapMang(int value [], int size) {
    for (int i = 0; i < size; i++)
        cin >> value [i];
}

void inMang(const int value [], int size) {
    for (int i = 0; i < size; i++)
        cout << value [i] << endl;
}

int main() {
    const int _SIZE_ = 10;
    int value [_SIZE_];
    nhapMang(value, _SIZE_);
    inMang(value, 5);
    return 0;
}
```

# Tìm Dữ Liệu

- ▶ Bắt đầu từ cuối mảng, kiểm tra từng phần tử
- ▶ Tìm thấy dữ liệu, trả về chỉ số phần tử
- ▶ Hết mảng, không tìm thấy, trả về **-1**

0	...	1	0	6	2	8	3	...	...	...
?	i=5									
0	...	1	0	6	2	8	3	...	...	...
?	i=4									
0	...	1	0	6	2	8	3	...	...	...
?	i=3									
0	...	1	0	6	2	8	3	...	...	...
?	i=2									
0	...	1	0	6	2	8	3	...	...	...
i=1	i=1									
0	...	1	0	6	2	8	3	...	...	...

# Tìm Dữ Liệu

9	...	1	0	6	2	8	3	...	...	...
?	i=5									
9	...	1	0	6	2	8	3	...	...	...
?	i=4									
9	...	1	0	6	2	8	3	...	...	...
?	i=3									
9	...	1	0	6	2	8	3	...	...	...
?	i=2									
9	...	1	0	6	2	8	3	...	...	...
?	i=1									
9	...	1	3	6	2	8	3	...	...	...
?	i=0									
9	...	1	0	6	2	8	3	...	...	...
i=-1	i=-1									
9	...	1	0	6	2	8	3	...	...	...



# Tìm Dữ Liệu

- ▶ Bắt đầu từ cuối mảng, kiểm tra từng phần tử
- ▶ Tìm thấy dữ liệu, trả về chỉ số phần tử
- ▶ Hết mảng, không tìm thấy, trả về **-1**

```
int search(const int value [], int size,  
           int data)  
{  
    int i = size-1;  
    while (i > -1)  
    {  
        if (value [i] == data) return i;  
        i = i - 1;  
    }  
    return -1;  
}
```

# Thêm & Xóa Dữ Liệu

---

- ▶ Đặt vấn đề:
  - ▶ sinh viên đăng ký lớp môn học
  - ▶ sinh viên lần lượt đăng ký (thêm vào danh sách)
  - ▶ sinh viên rút khỏi lớp môn học (xóa khỏi danh sách)
  - ▶ kích thước của danh sách lớp môn học ban đầu
  - ▶ số lượng sinh viên thực sự của lớp môn học

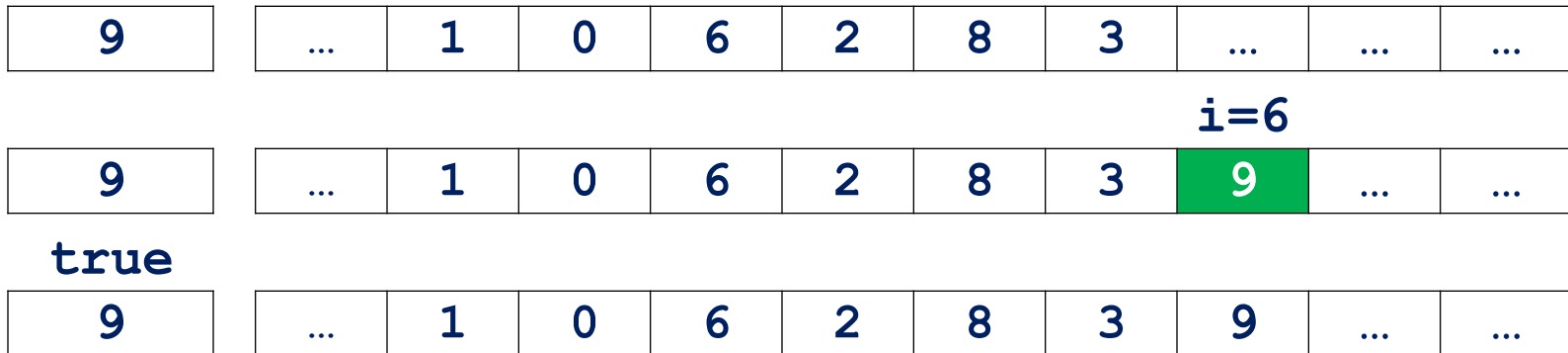
# Thêm Dữ Liệu

---

- ▶ Thêm dữ liệu vào mảng, làm thay đổi số lượng phần tử mảng, không thay đổi kích thước
  - ▶ biến số quản lý số phần tử của mảng
  - ▶ biến số này trong khoảng chỉ số hợp lệ
  - ▶ chương trình phải quản lý biến số này
- ▶ Thêm dữ liệu
  - ▶ tăng số phần tử mảng
  - ▶ phải kiểm tra mảng có đầy hay không

# Thêm Dữ Liệu – Mảng Chưa Sắp Xếp

- ▶ Thêm vào sau phần tử cuối cùng
- ▶ Tăng biến quản lý số phần tử
- ▶ Nếu mảng chưa đầy, thêm được, trả về **true**



# Thêm Dữ Liệu – Mảng Chưa Sắp Xếp

- ▶ Thêm vào sau phần tử cuối cùng
- ▶ Tăng biến quản lý số phần tử
- ▶ Nếu mảng chưa đầy, thêm được, trả về **true**

```
bool insert(int value [], int & no_ele,  
            int data)  
{  
    if (no_ele < _SIZE_)  
    {  
        value [no_ele] = data;  
        no_ele++;  
        return true;  
    }  
    return false;  
}
```

# Xóa Dữ Liệu

---

- ▶ Xóa dữ liệu khỏi mảng, làm thay đổi số lượng phần tử mảng, không thay đổi kích thước
  - ▶ biến số quản lý số phần tử của mảng
  - ▶ biến số này trong khoảng chỉ số hợp lệ
  - ▶ chương trình phải quản lý biến số này
- ▶ Xóa dữ liệu
  - ▶ giảm số phần tử mảng
  - ▶ phải kiểm tra phần tử có trong mảng hay không

# Xóa Dữ Liệu – Mảng Chưa Sắp Xếp

- ▶ Phần tử bị xóa có chỉ số là **index** (cần phải tìm trước)
- ▶ Đổi chỗ phần tử này với phần tử cuối cùng
- ▶ Giảm biến quản lý số phần tử thực sự

2	...	1	0	6	2	8	3	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

**i=5**

2	...	1	0	6	2	8	3	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

**i=4**

2	...	1	0	6	2	8	3	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

**i=3**

2	...	1	0	6	2	8	3	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

2	...	1	0	6	3	8	2	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

2	...	1	0	6	3	8	...	...	...	...
---	-----	---	---	---	---	---	-----	-----	-----	-----

# Xóa Dữ Liệu – Mảng Chưa Sắp Xếp

- ▶ Phần tử bị xóa có chỉ số là **index** (cần phải tìm trước)
- ▶ Đổi chỗ phần tử này với phần tử cuối cùng
- ▶ Giảm biến quản lý số phần tử thực sự

```
bool delete(int value [], int & no_ele,
            int data) {
    int index = search(value, no_ele, data);
    if (index > -1)
    {
        value [index] = value [no_ele-1];
        no_ele--;
        return true;
    }
    return false;
}
```



# Sắp Xếp Dữ Liệu

1	0	6	2	8	3	9	4	7	5
1	0	6	2	8	3	9	4	7	5
1	0	6	2	8	3	5	4	7	9
1	0	6	2	7	3	5	4	8	9
1	0	6	2	4	3	5	7	8	9
1	0	5	2	4	3	6	7	8	9
1	0	3	2	4	5	6	7	8	9
1	0	3	2	4	5	6	7	8	9
1	0	3	2	4	5	6	7	8	9
1	0	2	3	4	5	6	7	8	9
1	0	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

# Sắp Xếp Lựa Chọn

---

- ▶ Coi mảng là 2 phần:
  - ▶ phần sau: các phần tử đã được sắp xếp đúng chỗ
  - ▶ phần đầu: các phần tử còn lại cần được sắp xếp
  - ▶ một chỉ số để xác định điểm phân cách 2 phần
- ▶ Vòng lặp
  - ▶ tìm phần tử lớn nhất trong mảng chưa sắp xếp
  - ▶ đổi chỗ xuống cuối mảng chưa sắp xếp
- ▶ Dừng vòng lặp
  - ▶ khi mảng chưa sắp xếp còn 1 phần tử

# Tìm Dữ Liệu Lớn Nhất

- ▶ Sử dụng **iMax**, đánh dấu chỉ số phần tử lớn nhất
- ▶ Trả về **iMax**, chỉ số của phần tử lớn nhất

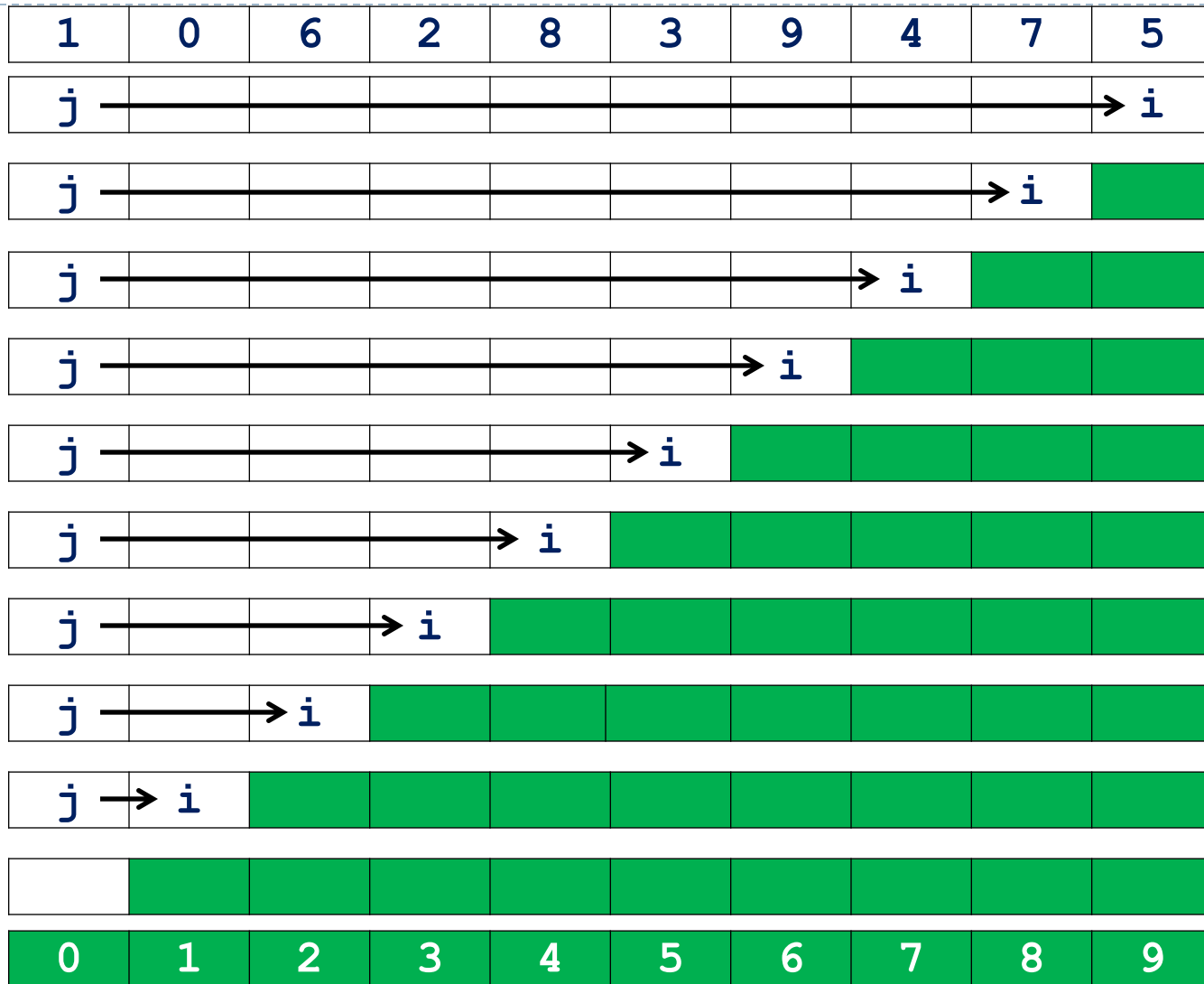
iMax=5	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...
iMax=4	...	1	0	6	2	8	3	...	...	...

# Tìm Dữ Liệu Lớn Nhất

- ▶ Sử dụng **iMax**, đánh dấu chỉ số phần tử lớn nhất
- ▶ Trả về **iMax**, chỉ số của phần tử lớn nhất
- ▶ Bắt đầu từ cuối mảng, kiểm tra từng phần tử

```
int search_max(const int value [], int size) {  
    int iMax = size-1;  
    int dataMax = value [iMax];  
    for (int i = size-1; i > -1; i--) {  
        if (value [i] > dataMax) {  
            iMax = i;  
            dataMax = value [iMax];  
        }  
    }  
    return iMax;  
}
```

# Sắp Xếp Lựa Chọn



# Sắp Xếp Lựa Chọn

- ▶ Sắp xếp dữ liệu tăng dần
- ▶ Tìm phần tử lớn nhất, đổi xuống cuối

```
void selection_sort(int value [], int size) {  
    for (int i = size-1; i > 0; i--) {  
        // int iMax = search_max(value, i+1);  
        int iMax = 0;  
        for (int j = 0; j <= i; j++) {  
            if (value [j] > value [iMax])  
                iMax = j;  
        }  
        int temp_value = value [i];  
        value [i] = value [iMax];  
        value [iMax] = temp_value;  
    }  
}
```

# Thêm Dữ Liệu – Mảng Đã Sắp Xếp

- ▶ Tìm vị trí thêm, từ đó lùi các phần tử ra sau một chỉ số
- ▶ Tăng biến quản lý số phần tử thực sự

2	...	0	1	4	5	6	8	...	...	...
							i=5			
2	...	0	1	4	5	6	8	8	...	...
							i=4			
2	...	0	1	4	5	6	6	8	...	...
							i=3			
2	...	0	1	4	5	5	6	8	...	...
							i=2			
2	...	0	1	4	4	5	6	8	...	...
							i=1			
2	...	0	1	2	4	5	6	8	...	...

# Thêm Dữ Liệu – Mảng Đã Sắp Xếp

- ▶ Tìm vị trí thêm, từ đó lùi các phần tử ra sau một chỉ số
- ▶ Tăng biến quản lý số phần tử thực sự

```
bool insert(int value [], int & no_ele,
            int data) {
    if (no_ele < _SIZE_) {
        int i = no_ele-1;
        while (i > -1 && data < value [i]) {
            value [i+1] = value [i]; i--;
        }
        value [i+1] = data;
        no_ele++;
        return true;
    }
    return false;
}
```



# Xóa Dữ Liệu – Mảng Đã Sắp Xếp

- ▶ Tìm vị trí phần tử để xóa
- ▶ Từ vị trí đó dồn các phần tử lên trước một chỉ số
- ▶ Giảm biến quản lý số phần tử thực sự

5	...	0	1	4	5	6	8	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

$i=5$

5	...	0	1	4	5	6	8	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

$i=4$

5	...	0	1	4	5	6	8	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

$i=3$

5	...	0	1	4	5	6	8	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

$i=3$

5	...	0	1	4	6	6	8	...	...	...
---	-----	---	---	---	---	---	---	-----	-----	-----

$i=4$

5	...	0	1	4	6	8	...	...	...	...
---	-----	---	---	---	---	---	-----	-----	-----	-----

# Xóa Dữ Liệu – Mảng Đã Sắp Xếp

- ▶ Tìm vị trí phần tử để xóa
- ▶ Từ vị trí đó dồn các phần tử lên trước một chỉ số
- ▶ Giảm biến quản lý số phần tử thực sự

```
bool delete(int value [], int & no_ele,
            int data) {
    int index = search(value, no_ele, data);
    if (index > -1) {
        for (int i = index; i < no_ele-1; i++)
            value [i] = value [i + 1];
        no_ele--;
        return true;
    }
    return false;
}
```

# Tìm Dữ Liệu – Mảng Đã Sắp Xếp

---

- ▶ Binary search

# Mảng Nhiều Chiều

<code>mang[0]</code>		O							
<code>mang[1]</code>			X						
<code>mang[2]</code>				X	O				
<code>mang[3]</code>			O	X	X	X	X	O	
<code>mang[4]</code>				O	O	X			
<code>mang[5]</code>				X	O	O	O	O	X
<code>mang[6]</code>									
<code>mang[7]</code>									
<code>mang[8]</code>									
<code>mang[9]</code>									

# Noughts & Crosses – Tic Tac Toe

---

- ▶ Khởi tạo bàn chơi
- ▶ In bàn chơi
- ▶ Kiểm tra ô có được phép không
- ▶ Chọn ô chơi ngẫu nhiên
- ▶ Kiểm tra có ai thắng chưa
- ▶ Chọn ô có thể được chơi

# Noughts & Crosses – Tic Tac Toe

```
void initBoard(char board[][COL], int ROW) {
    for (int _row = 0; _row < ROW; _row++)
        for (int _col = 0; _col < COL; _col++)
            board[_row][_col] = ' ';
}

void printBoard(char board[][COL], int ROW) {
    for (int _row = 0; _row < ROW; _row++)
        for (int _col = 0; _col < COL; _col++)
            cout << board[_row][_col];
}

bool play(char board[][COL], char player,
          int _row, int _col) {
    board[_row][_col] = player;
}
```

# Noughts & Crosses – Tic Tac Toe

---

```
int main()
{
    char board[ROW][COL];
    initBoard(board);
    printBoard(board);
    int _row, _col;
    char player = O_PLAYER;
    do {
        _row = rand() % ROW; // <cstdlib>
        _col = rand() % COL; // <cstdlib>
        play(board, _row, _col, player);
        system("CLS"); // <stdlib.h>
        printBoard(board);
    } while (true);
    return 0;
}
```

# Sắp Xếp Lựa Chọn – Giảm Dần

1	0	6	2	8	3	9	4	7	5
1	0	6	2	8	3	9	4	7	5
1	5	6	2	8	3	9	4	7	0
7	5	6	2	8	3	9	4	1	0
7	5	6	4	8	3	9	2	1	0
7	5	6	4	8	9	3	2	1	0
7	5	6	9	8	4	3	2	1	0
7	8	6	9	5	4	3	2	1	0
7	8	9	6	5	4	3	2	1	0
9	8	7	6	5	4	3	2	1	0
9	8	7	6	5	4	3	2	1	0
9	8	7	6	5	4	3	2	1	0



# Tìm Dữ Liệu Nhỏ Nhất

- ▶ Sử dụng **iMin**, đánh dấu chỉ số phần tử nhỏ nhất
- ▶ Trả về **iMin**, chỉ số của phần tử nhỏ nhất

iMin=5						i=5			
3	...	1	0	6	2	8	3	...	...

iMin=5					i=4				
3	...	1	0	6	2	8	3	...	...

iMin=3				i=3					
2	...	1	0	6	2	8	3	...	...

iMin=3			i=2						
2	...	1	0	6	2	8	3	...	...

iMin=1		i=1							
0	...	1	0	6	2	8	3	...	...

iMin=1	i=0								
0	...	1	0	6	2	8	3	...	...

# Tìm Dữ Liệu Nhỏ Nhất

- ▶ Sử dụng **iMin**, đánh dấu chỉ số phần tử nhỏ nhất
- ▶ Trả về **iMin**, chỉ số của phần tử nhỏ nhất
- ▶ Bắt đầu từ cuối mảng, kiểm tra từng phần tử

```
int search_min(const int value [], int size) {  
    int iMin = size-1;  
    int dataMin = value [iMin];  
    for (int i = size-1; i > -1; i--) {  
        if (value [i] < dataMin) {  
            iMin = i;  
            dataMin = value [iMin];  
        }  
    }  
    return iMin;  
}
```

# Sắp Xếp Lựa Chọn

- ▶ Sắp xếp dữ liệu giảm dần
- ▶ Tìm phần tử nhỏ nhất, đổi xuống cuối

```
void selection_sort(int value [], int size) {  
    for (int i = size-1; i > 0; i--) {  
        // int iMin = search_min(value, i+1);  
        int iMin = 0;  
        for (int j = 0; j <= i; j++) {  
            if (value [j] < value [iMin])  
                iMin = j;  
        }  
        int temp_value = value [i];  
        value [i] = value [iMin];  
        value [iMin] = temp_value;  
    }  
}
```