


# CÁC KIỂU DỮ LIỆU PHỨC TẠP

# NỘI DUNG CHÍNH

---

- Kiểu mảng **array**
  - Kiểu véc-tơ **vector**
  - Kiểu cấu trúc **struct**
  - Kiểu chuỗi kí tự **string**
- 

# Kiểu mảng

- Kiểu mảng và cách sử dụng mảng

# ĐẶT VẤN ĐỀ

---

- Chương trình cần lưu trữ **3** số nguyên?  
=> Khai báo **3** biến **int a1, a2, a3;**
- Chương trình cần lưu trữ **100** số nguyên?  
=> Khai báo **100** biến kiểu số nguyên!!!
- Người dùng muốn nhập **n** số nguyên?  
=> **n** không biết trước!

# GIẢI PHÁP

---

- Kiểu dữ liệu để **lưu trữ** và **xử lý** chuỗi **giá trị cùng kiểu** một cách **tuần tự** trong bộ nhớ như:
  - **Arrays (C)**
  - **Vectors (C++)**
- Một số kiểu dữ liệu khác:
  - Ngăn xếp - stacks
  - Hàng đợi – queues
  - Danh sách liên kết
  - Cây nhị phân

# GIẢI PHÁP

---

- Chương trình cần lưu trữ **3** số nguyên?  
=> Khai báo **mảng có 3 phần tử kiểu số nguyên;**
- Chương trình cần lưu trữ **100** số nguyên?  
=> Khai báo mảng **100** phần tử kiểu số nguyên!!!
- Người dùng muốn nhập **n** số nguyên?  
=> **Dùng mảng hoặc véc-tơ**

# MẢNG MỘT CHIỀU

- **Mảng**
  - Là một nhóm các vùng nhớ liên nhau
  - Cùng tên và cùng kiểu dữ liệu
  - Ví dụ: mảng c có n (n=10) phần tử
- Truy cập vào từng phần tử của mảng  
<ten\_mang>[<so\_thu\_tu>]
  - Tên mảng và thứ tự trong mảng.
  - Ví dụ: c[0], c[1], ...c[n-1]
  - Phần tử đầu tiên thường có thứ tự là 0

Tên mảng

c[0]

-56

c[1]

10

c[2]

2

c[3]

100

c[4]

2345

c[5]

54

c[6]

87

c[7]

-876

c[8]

1734

c[9]

10

Chỉ số của phần tử  
trong mảng c

# MẢNG MỘT CHIỀU

---

- Mỗi phần tử của mảng như một biến bình thường

```
c[0] = 3;
```

```
cout << c[0];
```

- Chỉ số của các phần tử có thể là **hằng số, biến, biểu thức có kiểu rời rạc**
- Ví dụ nếu  $i = 3$  thì  $c[5-2]$ ,  $c[3]$ ,  $c[i]$  là tương đương nhau
- Thường dùng một biến kiểu rời rạc làm chỉ số chạy qua các phần tử của mảng



# KHAI BÁO MẢNG

---

- Khi khai báo mảng cần chỉ ra `<kieu_mang> <ten_mang>[<so_phan_tu>];`
  - Kiểu của các phần tử
  - Tên mảng
  - Số lượng phần tử

- Ví dụ:

```
int c[10];  
float myArray[3284];
```

- Khai báo các mảng cùng kiểu
  - Giống như khai báo các biến cùng kiểu
  - Ví dụ:

```
int b[100], x[27];
```

# KHỞI TẠO MẢNG

---

- Khai báo mảng có giá trị khởi đầu

```
int n[5] = {1,2,3,4,5}
```

- Nếu không đủ thì các phần tử bên phải nhất nhận giá trị 0
  - Ví dụ: `int n[5] = {1}` thì phần tử đầu tiên có giá trị 1; các phần tử còn lại có giá trị 0
- Nếu số lượng giá trị nhiều hơn số phần tử => **xảy ra lỗi**
- Nếu không chỉ định số phần tử thì số lượng các giá trị khởi tạo sẽ là số phần tử của mảng
  - Với `int n[] = {1,2,3,4,5}` mảng n có 5 phần tử

# KHỞI TẠO MẢNG

[\*] vd3-1.cpp

```
1  // khai bao mang mot chieu
2  #include <iostream>
3  #include <conio.h>
4
5  using namespace std;
6
7  main() {
8      // khai bao mang co 5 phan tu, trong do 3 phan tu co gia tri ban dau
9      int A[5] = {2, 6, 14};
10     cout << "Cac phan tu cua mang la: ";
11     for (int i = 0; i < 5; i++) {
12         cout << A[i] << " ";
13     }
14     getch();
15     return 0;
16 }
```

# KHỞI TẠO MẢNG

vd3-2.cpp

```
1 // vi du 3-2
2 // nhap gia tri cua mang 1 chieu su dung vong for
3 #include <iostream>
4 #include <conio.h> // de lay ham getch()
5 #include <cstdlib> // de lay cac ham srand(), rand()
6 #include <ctime> // de lay ham time()
7 using namespace std;
8
9 main() {
10     int A[5];
11     // nhap gia tri ngau nhien cho cac phan tu cua mang
12     srand(time(NULL)); // ham khoi tao bo sinh so ngau nhien
13     // su dung vong lap for de gan cac gia tri cho cac phan tu cua mang
14     for (int i = 0; i < 5; i++) {
15         A[i] = rand();
16     }
17     // dua gia tri cua cac phan tu ra man hinh
18     cout << "Cac phan tu cua mang la: ";
19     for (int i = 0; i < 5; i++) {
20         cout << A[i] << " ";
21     }
22     getch();
23     return 0;
24 }
```

# NHẬP GIÁ TRỊ MẢNG

[\*] vd3-3.cpp

```
1 // vi du 3-3
2 // nhap gia tri cua mang 1 chieu su dung vong for
3 #include <iostream>
4 #include <conio.h> // de lay ham getch()
5 using namespace std;
6
7 main() {
8     int A[5];
9     int n; // so luong phan tu
10    cout << "Cho biet n: "; cin >> n;
11    // su dung vong lap for de nhap cac phan tu cua mang
12    for (int i = 0; i < n; i++) {
13        cout << "A[" << i << "] = ";
14        cin >> A[i];
15    }
16    // dua gia tri cua cac phan tu ra man hinh
17    cout << "Cac phan tu cua mang da nhap la: ";
18    for (int i = 0; i < n; i++) {
19        cout << A[i] << " ";
20    }
21    getch();
22    return 0;
23 }
```

# XUẤT GIÁ TRỊ MẢNG

```
1 // vi du 3-4: dua ra tri cua mang ra man hinh
2 #include <iostream>
3 #include <conio.h> // de lay ham getch()
4 #define SIZE 100
5 using namespace std;
6
7 main() {
8     int A[SIZE];
9     int n; // so luong phan tu
10    cout << "Cho biet n: "; cin >> n;
11    for (int i = 0; i < n; i++) {
12        cout << "A[" << i << "]= ";
13        cin >> A[i];
14    }
15    // dua gia tri cua cac phan tu ra man hinh
16    cout << "Cac phan tu cua mang da nhap la: ";
17    for (int i = 0; i < n; i++) {
18        cout << A[i] << " ";
19    }
20    getch();
21    return 0;
22 }
```

Định nghĩa số phần tử tối đa của mảng bằng #define

Khai báo mảng với hằng số SIZE đã định nghĩa

Dùng vòng lặp để nhập giá trị cho các phần tử của mảng

Dùng vòng lặp để đưa các phần tử ra màn hình

# MẢNG HAI CHIỀU

---

- Mảng hai chiều = ma trận kích thước  $n \times m$  (n: số dòng, m: số cột)

- Cú pháp khai báo:

`<kieu_mang> <ten_mang>[<so_dong>][<so_cot>;`

- Ví dụ: *int b[2][2], a[3][5];*

- Truy cập đến một phần tử:

`<ten_mang>[<chi_so_dong>][<chi_so_cot>]`

- Ví dụ: *b[1][1], a[0][4]*

- Lưu ý: Không dùng *b[i,j]*, phải dùng *b[i][j]*

# MẢNG HAI CHIỀU

---

- Khai báo có giá trị khởi đầu
- Nếu các giá trị không đủ, các phần tử còn lại được gán bằng 0

```
int b[2][2] = { {1, 2}, {3, 4} };
```

1 2  
3 4

1 3  
2 4



# MẢNG NHIỀU CHIỀU

---

- Đối với mảng từ hai chiều trở lên, cũng được khai báo và sử dụng theo cách trên
- Chẳng hạn mảng ba chiều: `int A[10][10][10]`
- Truy cập các phần tử: `A[i][j][k]`
- Có thể quan niệm mảng nhiều chiều là mảng của mảng



# NHẬP/XUẤT MẢNG HAI CHIỀU

vd3-5.cpp

```
1 // vi du 3-4: dua ra tri cua mang ra man hinh
2 #include <iostream>
3 #include <conio.h> // de lay ham getch()
4 #define SIZE 100 // khai bao hang so SIZE
5 using namespace std;
6
7 main() {
8     int A[SIZE][SIZE];
9     int n, m; // kích thước của mảng 2 chiều
10
11     // nhập kích thước, số dòng n, số cột m
12     cout << "Cho biết n, m: ";
13     cin >> n >> m;
14     // nhập các giá trị của ma trận 2 chiều
15     for (int i = 0; i < n; i++)
16         for (int j = 0; j < m; j++) {
17             cout << "A[" << i << "][" << j << "]= ";
18             cin >> A[i][j];
19         }
20 }
```

Khai báo mảng hai chiều

Nhập kích thước mảng

Nhập các giá trị của mảng sử dụng 2 vòng for lồng nhau

Xuất mảng hai chiều dưới dạng ma trận, sử dụng 2 vòng for lồng nhau

```
21 // dua mang 2 chieu ra man hinh
22 // duoi dang mot ma tran n x m
23 cout << "Cac phan tu cua mang da nhap la: ";
24 cout << endl;
25 for (int i = 0; i < n; i++) {
26     for (int j = 0; j < m; j++) {
27         cout << A[i][j] << " ";
28     }
29     cout << endl;
30 }
31 getch();
32 return 0;
33 }
```

# LỖI THƯỜNG GẶP KHI SỬ DỤNG MẢNG

```
1 // Out-of-Range Errors.
2 /* Chương trình minh họa hành vi bất thường gây
3    ra bởi lỗi out-of-range -----
4    -----*/
5 #include <iostream>                // cout, <<
6 using namespace std;
7 #define SIZE 4
8
9 int main()
10 {
11     int a[] = {0, 1, 2, 3};
12     int below = -3, above = 6;
13     int i;
14
15     for (i=0; i<SIZE; i++)
16         cout << a[i] << " ";
17     cout << endl;
18
19     a[below] = -999;
20     a[above] = 999;
21
22     for (i=0; i<SIZE; i++)
23         cout << a[i] << " ";
24     cout << endl;
25 }
```

## Kết quả chạy thử:

a =           0           1           2           3

a =           0           1    999           3

hoặc báo lỗi

# BÀI TẬP THỰC HÀNH

---

## Bài 3-1.

Cho một dãy số thực gồm  $N$  ( $N < 1000$ ) phần tử, hãy lập trình thực hiện các việc sau:

- Nhập  $N$  và dãy số từ bàn phím
- Tìm số nhỏ thứ 2 và lớn thứ 2 trong dãy
- Nhân đôi giá trị của các phần tử ở vị trí chia hết cho 3 (giả sử dãy được đánh số từ 1)
- Hiển thị dãy trước và sau khi nhân lên màn hình

## Bài 3-2.

Cho một ma trận kích thước  $N \times M$  ( $N, M < 100$ ), mỗi phần tử là một số nguyên. Hãy lập trình thực hiện các việc sau:

- Nhập  $N$ ,  $M$  và các phần tử của ma trận từ bàn phím
- Hiển thị dữ liệu vừa nhập dưới dạng một ma trận
- Tìm số lớn nhất và các vị trí xuất hiện của nó trong ma trận
- Cho số nguyên  $k$ , sắp xếp lại dòng  $k$  theo chiều không giảm. Hiển thị ma trận sau khi sắp xếp dòng  $k$  lên màn hình
- Đảo dòng thành cột, cột thành dòng; hiển thị ma trận sau khi đảo lên màn hình.

# VÍ DỤ 1

---

- Cho một dãy số gồm  $n$  số nguyên ( $n < 100$ ), hãy viết chương trình thực hiện các việc sau đây (có sử dụng hàm)
  - Nhập  $n$  và dãy số nguyên từ bàn phím
  - Đưa ra dãy số đã nhập
  - Tính tổng các phần tử chẵn và chia hết cho 3
  - Tìm giá trị nhỏ thứ 2 trong dãy số đã nhập
  - Sắp xếp lại dãy số đã nhập theo chiều không giảm

## VÍ DỤ 2

---

- Cho một ma trận kích thước  $n \times m$  ( $n, m < 100$ ). Mỗi phần tử của ma trận là 1 số thực. Hãy viết chương trình thực hiện các việc sau đây:
  - Nhập,  $n$ ,  $m$  và các phần tử
  - Đưa ma trận ra màn hình
  - Tính tổng các phần tử trên cột thứ  $k$  của ma trận ( $k$  nhập từ bàn phím).
  - Đếm số lượng phần tử có giá trị dương trong ma trận

# Kiểu VEC-TƠ VÀ CÁCH SỬ DỤNG

- Khai báo véc-tơ
- Sử dụng các hàm có sẵn

# VÌ SAO DÙNG VÉC TƠ THAY MẢNG

---

- Véc tơ là cấu trúc lưu trữ dữ liệu giống như mảng
- Không cần biết trước kích thước (khác mảng)
- Kích thước véc tơ tự động tăng khi thêm phần tử mới
- Là một lớp có sẵn trong thư viện C++, không có trong C, cần khai báo `#include <vector>` ở đầu chương trình
- Là một mẫu (template), có thể cài đặt cho từng kiểu dữ liệu



# KHAI BÁO VÀ KHỞI TẠO VEC TƠ

---

- Khai báo:

```
vector<int> v1;
```

```
vector<float> v2;
```

- Có thể thiết lập kích thước khởi tạo cho véc tơ:

```
vector<int> v3(20);
```

```
vector<float> v4(10);
```

- Khởi tạo giá trị cho véc tơ:

```
vector<int> v5(100,1);
```

Khai báo véc tơ  
v3 có 20 phần  
tử

Khai báo véc tơ v5 có  
100 phần tử, tất cả  
đều có giá trị là 1

# MỘT SỐ THAO TÁC CƠ BẢN VỚI VÉC TƠ

---

- `empty()`
  - Kiểm tra véc tơ rỗng?
  - Trả về giá trị `true` hoặc `false`
- `size()`
  - Trả về số phần tử của véc tơ
- `at(i)`:
  - Trả về phần tử thứ  $i$  của véc tơ
- `push_back(<value>)`:
  - Thêm một phần tử có giá trị `<value>` vào cuối véc tơ
- `pop_back()`:
  - Loại bỏ phần tử ở cuối véc tơ
- `resize(ts1, ts2)`:
  - Thay đổi kích cỡ véc tơ

# CÁCH GỌI PHƯƠNG THỨC CỦA VÉC TƠ

---

- Véc tơ là một **lớp (class)** trong C++
- Các thao tác là các **phương thức (methods)** của lớp
- Sử dụng toán tử **"."** để truy cập đến các thành phần của lớp
- Cú pháp:  
`<ten_vector>.<ten_phuong_thuc>(<tham_so_neu_co>)`
- Ví dụ: `v.push_back(100)`, `v.pop_back()`

# THÊM PHẦN TỬ

```
1 // Them bot phan tu cua vec to
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5
6 int main() {
7     vector<int> v;
8     int i;
9     //take input
10    for (i = 0; i < 10; i++) v.push_back(i);
11    // Take out the last element
12    // If (!v.empty())
13    // v.pop_back();
14    cout<< "Current size of vector " << v.size() << endl;
15 }
16
```

Dùng vòng lặp để thêm phần tử vào cuối vector bằng hàm push\_back()

Xem kích thước véc tơ

# KIỂM TRA RỎNG VÀ BỚT PHẦN TỬ

```
1 // Them bot phan tu cua vec to
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5
6 int main() {
7     vector<int> v;
8     int i;
9     //take input
10    for (i = 0; i < 10; i++) v.push_back(i);
11    // Take out the last element
12    if (!v.empty())
13        v.pop_back();
14    cout<< "Current size of vector " << v.size() << endl;
15 }
```

Kiểm tra để chắc chắn véc  
tơ không rỗng

Dùng vòng lặp và  
pop\_back() để lấy bớt  
phần tử ở cuối của véc  
tơ

# THAY ĐỔI KÍCH THƯỚC

Câu lệnh cho kết quả gì?

```
1 // Thay doi kích thước và khởi tạo Vector
2 #include <iostream>
3 #include <vector>
4
5 using namespace std;
6 int main() {
7     vector<int> v(3);
8     v.resize(5, 1);
9     cout << "Resized vector : ";
10    for (int i = 0; i < v.size(); i++) {
11        cout << v[i] << " ";
12    }
13    cout << endl;
14 }
```

# SỬ DỤNG HÀM AT()

- Thay toán tử [] bằng hàm at()
- Kết quả khác nhau???
- Lưu ý: lỗi out-of-range!!

```
1 // Thay đổi kích thước và khởi tạo Vector
2 #include <iostream>
3 #include <vector>
4
5 using namespace std;
6 int main() {
7     vector<int> v(3);
8     v.resize(5, 1);
9     cout << "Resized vector : ";
10    for (int i = 0; i < v.size(); i++) {
11        // cout << v[i] << " ";
12        cout << v.at(i) << " ";
13    }
14    cout << endl;
15 }
```

# PHÉP GÁN VÉC TƠ

- Sử dụng toán tử "=" như phép gán đối với các kiểu dữ liệu chuẩn
- Sao chép toàn bộ dữ liệu từ 1 véc tơ sang 1 véc tơ khác
- Không thể thực hiện với mảng

```
1 // Gán vec to
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5
6 int main() {
7     vector<int> a = { 1, 2, 3 };
8     vector<int> b;
9     b = a;
10    for (int i = 0; i < b.size(); i++)
11    {
12        cout << b[i] << " ";
13    }
14    cout << endl;
15 }
```



# ƯU ĐIỂM CỦA VÉC TƠ

- Véc tơ: dễ dùng hơn mảng
  - Cần ghi nhớ và quản lý ít hơn
  - Khó làm sai
- Ví dụ: So sánh kết quả khi dùng toán tử [] và dùng `push_back()`

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main (int, char *[]) {
6      vector<int> v;
7      // Cac cau lenh gan co the se gap loi....
8      // v[0] = 1; v[1] = 2; v[2] = 4;
9      // ... nhưng dùng push_back thì ok
10     v.push_back (1);
11     v.push_back (2);
12     v.push_back (4);
13     // In gia tri phan tu vecto
14     for (size_t s = 0; s < v.size(); ++s) {
15         cout << "v[" << s << "] is "
16             << v[s] << endl;
17     }
18     return 0;
19 }
```

# ƯU ĐIỂM CỦA VÉC TƠ

---

- Véc-tơ làm hầu hết các công việc quản lý bộ nhớ (khó khăn) thay cho lập trình viên
  - Sử dụng `new[]` và `delete[]` ngẫm
  - `resize` tránh rò rỉ bộ nhớ
- Chú ý một số điểm:
  - Hàm `push_back` cấp phát thêm bộ nhớ nhưng toán tử `[]` thì không
  - Véc tơ sao chép và sở hữu các đối tượng chứa trong nó

# BÀI TẬP THỰC HÀNH

---

## Bài 3-3.

Cho hai số nguyên A và B, mỗi số nguyên có thể lên đến 100 chữ số. Bạn hãy viết chương trình thực hiện phép toán cộng hai số nguyên nói trên. Hiển thị kết quả lên màn hình.

Chú ý: trong chương trình cần sử dụng dữ liệu kiểu vector.

## Bài 3-4.

Cho hai dãy số thực X và Y; dãy X gồm có n phần tử, dãy Y có m phần tử. Viết chương trình thực hiện các việc sau đây:

- Nhập vào hai dãy X, Y
- Sắp xếp hai dãy X và Y theo chiều không giảm.
- **Ghép nối** các phần tử của dãy X và dãy Y để được dãy mới Z có  $n+m$  phần tử, sao cho dãy Z cũng là một dãy được sắp xếp theo chiều không giảm.

Chú ý: Không được nối hai dãy X và Y lại rồi sắp xếp để được dãy Z.

# Kiểu Xâu Ký Tự

- Khai báo biến xâu
- Các phép toán xử lý xâu
- Một số bài toán ứng dụng

# XÂU KÝ TỰ

---

- Một dãy các ký tự liên tiếp
- Xâu có thể rỗng (gồm 0 ký tự)
- Mỗi ký tự cũng được coi là một xâu gồm đúng 1 ký tự đó



# KHAI BÁO XÂU KÝ TỰ

---

- Khai báo sử dụng thư viện string (chú ý **không phải cstring**)
- Kiểu dữ liệu cho chuỗi ký tự là `std::string`
- Nếu trong chương trình đã khai báo **`using namespace std;`** thì có thể sử dụng **`string`** thay cho **`std::string`**

```
#include <string>  
using namespace std;
```

```
string s = "Hello";
```

# CÁCH VIẾT HẰNG XÂU KÝ TỰ

---

- Viết các ký tự nằm trong dấu nháy kép
- Các ký tự đặc biệt được chấp nhận như bảng ký tự đặc biệt

```
Kết quả in ra:  
Me di lam  
Tu sang som  
Day thoi com  
Kho thit ca
```

```
#include <iostream>  
#include <string>  
using namespace std;  
int main()  
{  
    string s = "Me di lam  
\\n Tu sang som \\n Day thoi  
com \\n Kho thit ca";  
    cout << s;  
}
```

# BẢNG MÃ KÝ TỰ ĐẶC BIỆT

Mã C++	Tên ký tự	Mã ASCII
\n	Xuống dòng	10 hoặc 13+10
\r	Về đầu dòng (Enter)	13
\t	Tab	9
\v	Tab dọc	
\b	Lùi trái (Backspace)	8
\f	Hết form	
\a	alert (beep)	7
\'	Nháy đơn '	39
\"	Nháy kép "	34
\?	Dấu hỏi ?	63
\\	Dấu sổ ngược \	92



# BẢN CHẤT CỦA KIỂU STD::STRING

---

- `string` là một lớp, mỗi biến kiểu `string` là một đối tượng, bao gồm cả dữ liệu và các phương thức xử lý dữ liệu
- Có thể coi `s` (khai báo như hình bên) là một mảng các ký tự (`char`)
- Tuy nhiên khác với mảng:
  - `s` không phải địa chỉ của dãy ký tự được lưu
  - `s + i` cũng không phải địa chỉ của ký tự thứ `i`

Chú ý: Không được dùng `memset`, `fill_n`, `fill` trực tiếp trên biến `s` mà phải dùng các phương thức đặc thù của `string`

```
#include <string>
using namespace std;

string s = "Hello";
```

## ĐỌC/IN XÂU KÝ TỰ

---

- Toán tử `<<` có thể dùng để đẩy một xâu ký tự ra luồng xuất (`stdout`)
- Toán tử `>>` có thể dùng để đọc một xâu khác rỗng và không chứa dấu cách
- Ví dụ

```
cin >> s;
```

```
cout << s;
```

A decorative pattern of light gray diamond shapes arranged in a grid, located at the bottom of the slide.

## ĐỌC/IN XÂU KÝ TỰ

---

- Nếu muốn đọc một dòng từ luồng nhập vào một xâu (có thể dòng chứa dấu cách)
- Dùng `getline(stream, s, delim)`
  - `stream`: Luồng nhập (cin, ifstream, ...)
  - `s`: Xâu ký tự
  - `delim`: Ký tự kết thúc dòng, nếu không chỉ ra `delim`, coi như `delim` là ký tự xuống dòng (`\n`)

# ĐỌC/IN XÂU KÝ TỰ

---

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     string name;
7     cout << "Full Name: ";
8     getline(cin, name);
9     cout << name;
10 }
```

Full Name: James Bond  
James Bond

## ĐỌC/IN XÂU KÝ TỰ

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     string fname, lname;
7     cout << "Full name: ";
8     getline(cin, fname, ' ');
9     getline(cin, lname);
10    cout << fname << '\n' << lname;
11 }
```

Full Name: James Bond

James

Bond

# ĐỘ DÀI XÂU

---

- Hàm *s.length()*
  - Trả về số ký tự trong xâu *s* (độ dài xâu *s*)
- Hàm *s.resize(n)*
  - Đặt chiều dài xâu *s* là *n*
  - Nếu *s.length() < n*, máy sẽ thêm các ký tự vào cuối xâu cho đủ độ dài *n*, các ký tự thêm vào không xác định (rác)
  - Nếu *s.length() > n*, các ký tự cuối xâu sẽ bị cắt bỏ cho vừa độ dài *n*
- Hàm *s.resize(n, c)*
  - Giống như hàm *s.resize(n)*, có điều các ký tự thêm vào cuối xâu (nếu có) được đặt bằng ký tự *c*

# ĐỘ DÀI XÂU

---

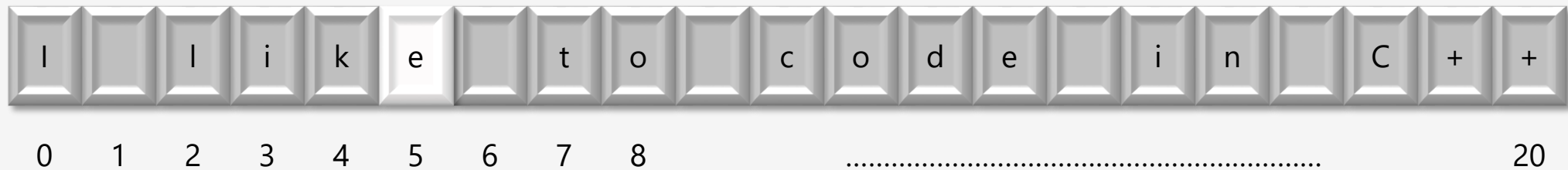
- Tạo một chuỗi `s` gồm 1000 dấu `*`

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     string s = "";
8     s.resize(1000, '*');
9     cout << s << '\n';
10    cout << "Number of \"*\": " << s.length();
11 }
```

# TRUY CẬP KÝ TỰ

- Các ký tự được đánh số từ 0 tới  $s.length() - 1$
- $s[i]$ : Ký tự thứ  $i$  trong chuỗi  $s$

```
#include <iostream>
#include <string>
using namespace std;
string s = "I like to code in C++";
int main()
{
    cout << s[5];
}
```





# PHÉP GÁN XÂU

---

- $s = t$ : Gán xâu  $s$  bằng xâu  $t$
- Chú ý:
  - Máy phải cấp phát bộ nhớ và copy các ký tự từ xâu  $t$  sang xâu  $s$  (Độ phức tạp tính toán bằng độ dài xâu  $t$ )

# PHÉP CỘNG XÂU

---

- $s + t$ : Với  $s, t$  là hai xâu, phép  $s + t$  trả về xâu tạo thành bằng cách ghép xâu  $t$  vào cuối xâu  $s$
- Chú ý
  - Phép cộng xâu không có tính giao hoán:  $s + t \neq t + s$
  - Có thể thực hiện toán tử +=: Viết  $s += t$  thay cho  $s = s + t$
  - Máy phải cấp phát bộ nhớ và copy các ký tự từ xâu  $s$  và  $t$  sang xâu kết quả (Độ phức tạp tính toán bằng độ dài xâu kết quả)

# PHÉP CỘNG XÂU

---

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     string a = "It's now";
7     string b = "or never";
8     string c = a + ' ' + b;
9     cout << c; //It's now or never
10 }
```

# THỨ TỰ TỪ ĐIỂN (LEXICOGRAPHICAL ORDER)

- Nguyên lý so sánh hai chuỗi  $A = a_0a_1 \dots a_m$  và  $B = b_0b_1 \dots b_n$
- So sánh từ đầu theo từng cặp ký tự  $(a_i, b_i)$ ,  $i = 1, 2, 3, \dots$ 
  - Nếu gặp  $a_i < b_i$ , kết luận ngay  $a < b$ , stop
  - Nếu gặp  $a_i > b_i$ , kết luận ngay  $a > b$ , stop
- Nếu không thấy cặp ký tự tương ứng khác nhau, chuỗi ngắn hơn được coi là nhỏ hơn

Chú ý: Tốc độ phép so sánh phụ thuộc vào vị trí  $i$  đầu tiên mà  $a_i \neq b_i$

"computer" > "calculator"  
"COMPUTER" < "calculator"  
"professional" < "professor"  
"10" < "9"  
"with" < "without"

# PHÉP SO SÁNH

---

- Trên xâu có đầy đủ các phép so sánh:
  - `==`; `!=`; `<`; `>`; `<=`; `>=`
- Các toán tử so sánh căn cứ theo thứ tự từ điển

# HÀM SUBSTR

---

- $s.substr(i, k)$
- Trả về một xâu con trong xâu  $s$ , bắt đầu từ ký tự thứ  $i$ , lấy  $k$  ký tự.
- Nếu không đủ  $k$  ký tự tính từ vị trí  $i$ , hàm chỉ lấy tới ký tự cuối cùng trong xâu  $s$
- Độ phức tạp tính toán bằng số ký tự trong xâu kết quả

---

a =

0	1	2	3	4	5	6	7	8	9	10
O	U	T	S	T	A	N	D	I	N	G

b = a.substr(3, 5)

b =

---

a =

0	1	2	3	4	5	6	7	8	9	10
O	U	T	S	T	A	N	D	I	N	G

b = a.substr(3, 100)

b =



# HÀM INSERT

---

- *s.insert(i, a)*
- Chèn chuỗi *a* vào chuỗi *s* tại vị trí *i* ( $0 \leq i \leq s.length()$ )
- Nếu  $i = s.length()$ , chuỗi *a* được thêm vào cuối chuỗi *s*

---

S =

0	1	2	3	4	5	6
C	O	S	T			

N	T	E
---	---	---

s.insert(2, "NTE");

---

0 1 2 3 4 5 6 7 8 9

S =

C	O	N	T	E	S	T			
---	---	---	---	---	---	---	--	--	--

A	N	T
---	---	---

```
s.insert(7, "ANT");
```

# HÀM ERASE

---

- *s.erase(i, k)*
- Xóa trong chuỗi *s* từ vị trí *i* đi *k* ký tự
- Chỉ xóa đến hết chuỗi dù từ vị trí *i* không còn đủ *k* ký tự.
- Chú ý: Máy phải cấp phát lại bộ nhớ và sao chép các ký tự không bị xóa sang vùng nhớ mới (độ phức tạp tính toán bằng độ dài chuỗi)

---

S =

0	1	2	3	4	5	6	7	8	9
C	O	N	T	E	S	T	A	N	T

```
s.erase(7, 100);
```

---

S =

0	1	2	3	4	5	6
C	O	N	T	E	S	T

`s.erase(2, 3);`

# HÀM REPLACE

---

- *s.replace(i, k, t)*
- Thay loạt ký tự trong xâu *s* tính từ vị trí *i* đi *k* ký tự bởi xâu *t*
- Nếu không đủ *k* ký tự tính từ vị trí *i*, hàm chỉ thay các ký tự cho đến hết xâu

---

S =

0	1	2	3	4	5	6	7
P	R	E	C	I	S	E	

A	C	T	I	C
---	---	---	---	---

```
s.replace(2, 4, "ACTIC");
```



---

0 1 2 3 4 5 6 7 8 9 10

S =

P	R	A	C	T	I	C	E			
---	---	---	---	---	---	---	---	--	--	--

O	G	R	A	M	M	I	N	G
---	---	---	---	---	---	---	---	---

```
s.replace(2, 100, "OGRAMMING");
```

# HÀM FIND

---

- $s.find(t, i)$
- Tìm vị trí xuất hiện đầu tiên của xâu  $t$  trong xâu  $s$  tính từ vị trí  $i$ , nếu không thấy trả về hằng `string::npos` (-1)
- Nếu không chỉ ra tham số  $i$ , việc tìm kiếm tiến hành từ đầu xâu  $s$  ( $i = 0$ )

```
s = "This is the first disk";
```

```
k = s.find("is"); //k = 2: Thisis is the first disk
```

```
k = s.find("is", 3); //k = 5: This is the first disk
```

```
k = s.find("is", 8); //k = 19: This is the first disk
```

```
k = s.find("is", 20); //k = -1: This is the first disk
```

# HÀM RFind

---

- *s.rfind(t, i)*
- Tìm vị trí xuất hiện cuối cùng của xâu *t* trong xâu *s* tính từ vị trí *i* trở về trước, nếu không thấy trả về hằng `string::npos` (-1)
- Nếu không chỉ ra tham số *i*, việc tìm kiếm tiến hành từ cuối xâu *s* (*i* = *s.length()*)

```
s = "This is the first disk";
```

```
k = s.rfind("is"); //k = 19: This is the first disk
```

```
k = s.rfind("is", 10); //k = 5: This is the first disk
```

```
k = s.rfind("is", 2); //k = 2: This is the first disk
```

```
k = s.rfind("is", 1); //k = -1: This is the first disk
```

# BÀI TẬP THỰC HÀNH

---

## **Bài 3-5.**

- Viết một hàm lowercase(s) nhận vào một xâu s và trả về xâu tạo thành bằng cách chuyển tất cả các chữ cái hoa sang chữ thường
- Viết một hàm uppercase(s) nhận vào một xâu s và trả về xâu tạo thành bằng cách chuyển tất cả các chữ cái thường sang chữ hoa

## **Bài 3-6.**

Hãy viết chương trình nhập vào một chuỗi bất kỳ từ bàn phím và mã hóa chuỗi đó thành các số tương ứng biết quy tắc mã hóa dựa trên bố trí các phím trên điện thoại thông thường: Mỗi phím số tương ứng với các nhóm ký tự như sau: 2-ABC, 3-DEF, 4-GHI, 5-JKL, 6-MNO, 7-PQRS, 8-TUV, 9-WXYZ.

Ví dụ: chuỗi "CNTT – DHSPHN" được mã hóa thành "2688-347746".

# Kiểu bản ghi

- Khai báo kiểu bản ghi
- Sử dụng kiểu bản ghi

# BẢN GHI

---

- Kiểu dữ liệu chứa các trường (fields)
- Mỗi trường là thuộc một kiểu dữ liệu nhất định
- Hai trường khác nhau có thể thuộc hai kiểu dữ liệu khác nhau



# ĐỊNH NGHĨA VÀ KHAI BÁO BẢN GHI

- T: Kiểu bản ghi
- fi: Danh sách các tên trường (fields) thuộc kiểu Ti. Có thể coi mỗi trường là một biến trong cấu trúc bản ghi
- Các tên trường trong mỗi danh sách fi phải cách nhau bởi dấu phẩy ",".
- Chú ý: Cần ";" sau khai báo struct
- v: Biến kiểu bản ghi
- Truy cập trường: Viết tên biến kiểu bản ghi, dấu chấm, tiếp theo là tên trường...

```
//Định nghĩa  
kiểu
```

```
struct T  
{  
    T1 f1;  
    T2 f2;  
    ...  
    Tn fn;  
};
```

```
//Khai báo biến  
T v;
```

# VÍ DỤ KIỂU BẢN GHI

---

```
struct TMovie
{
    string title, genre;
    int year;
    double rate;
};
```

```
TMovie m;
```

```
m.title =
    "GONE WITH THE WIND";
m.genre =
    "drama, romance";
m.year = 1939;
m.rate = 8.2;
```



# NHẬP XUẤT KIỂU BẢN GHI

---

- Bản ghi không thể nhập xuất trực tiếp qua các luồng nhập xuất
- Việc nhập dữ liệu, in kết quả phải được thực hiện trên từng trường đối với kiểu bản ghi



# NHẬP XUẤT KIỂU BẢN GHI

---

```
struct TMovie
{
    string title, genre;
    int year;
    double rate;
};
TMovie a[100];
int n;
cout << "n = ";
cin >> n;
string dummy;
```

```
for (int i = 0; i < n; i++)
{
    getline(cin, dummy); //xuống dòng
    cout << "\nTitle: ";
    getline(cin, a[i].title);
    cout << "Genre: ";
    getline(cin, a[i].genre);
    cout << "Year: ";
    cin >> a[i].year;
    cout << "IMDB rate: ";
    cin >> a[i].rate;
```

```
}
```

n = 3

Title: GONE WITH THE WIND

Genre: drama, romance

Year: 1939

IMDB rate: 8.2

Title: FROM RUSSIA WITH LOVE

Genre: action, thriller

Year: 1963

IMDB rate: 7.5

Title: YELLOW FLOWERS ON THE GREEN GRASS

Genre: drama

Year: 2015

IMDB rate: 8.2

# BÀI TẬP THỰC HÀNH

---

## Bài 3-7.

Một cửa hàng có  $n$  sản phẩm, mỗi sản phẩm có các thông tin: *mã, tên, năm sản xuất, số lượng, giá bán*. Hãy viết chương trình thực hiện:

- Nhập thông tin về  $n$  sản phẩm
- Đưa ra tên của các sản phẩm sản xuất năm 2020
- Cho tên một sản phẩm, hãy cho biết giá bán của sản phẩm đó.
- Tính và đưa ra **tổng giá trị** của sản phẩm đã nhập

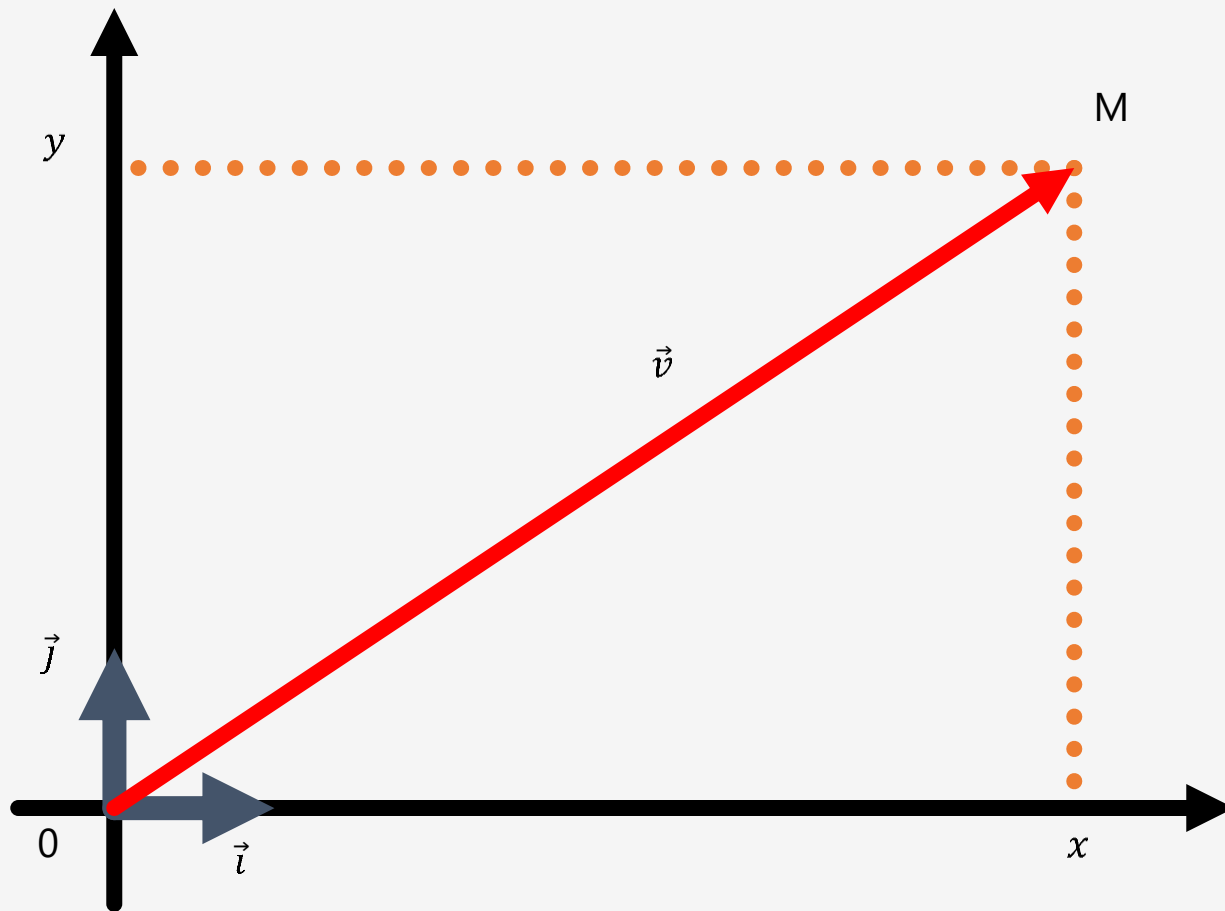
(giá trị của 1 sản phẩm = số lượng \* giá bán; tổng giá trị = tổng giá trị của tất cả các sản phẩm)

## Bài 3-8.

Viết chương trình thực hiện các yêu cầu sau:

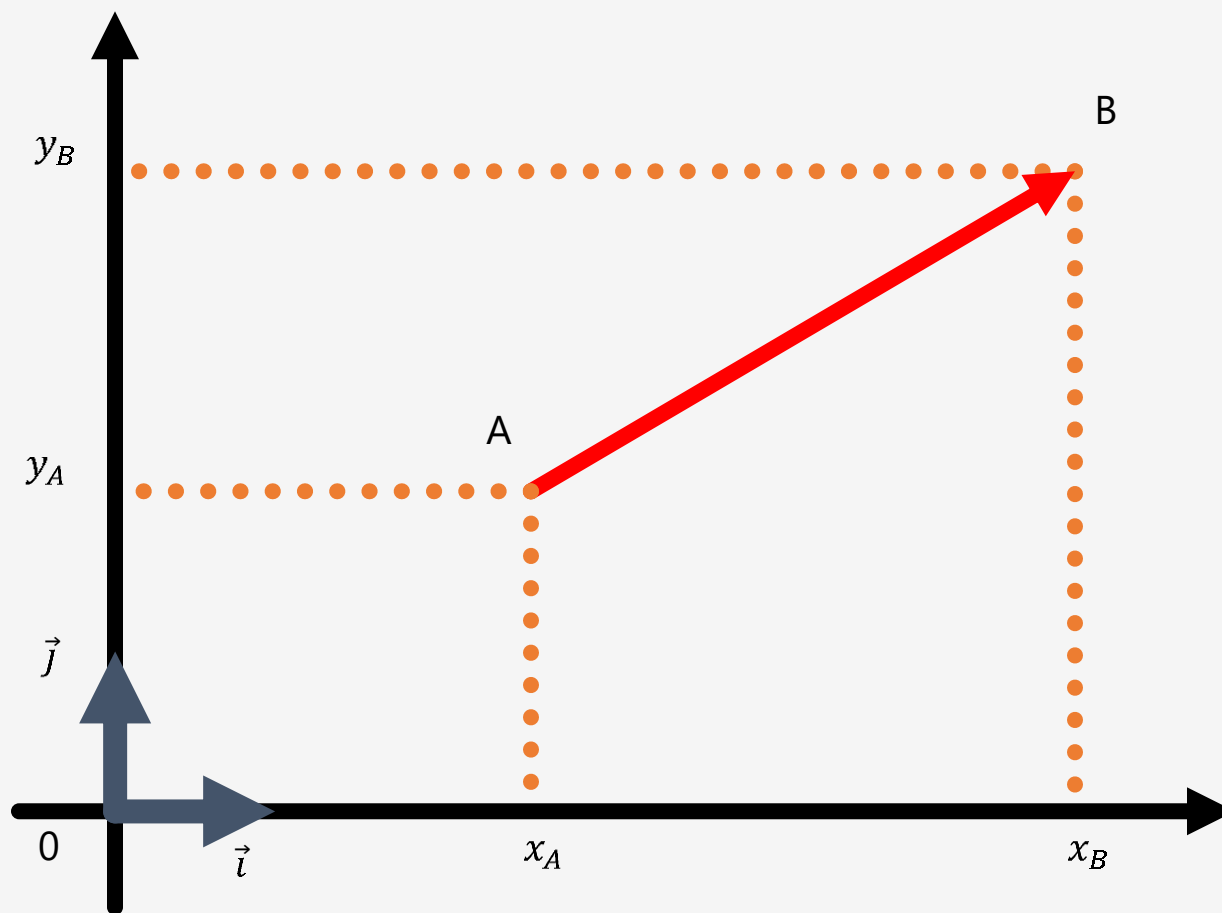
- Tạo lập một kiểu bản ghi biểu diễn các vector trong hệ tọa độ trực chuẩn 2 chiều
- Viết hàm tính độ dài của véc tơ
- Viết hàm tính tổng, hiệu của hai véc tơ

## GỢI Ý BÀI 3-8



- $\vec{v} = \overrightarrow{OM} = x.\vec{i} + y.\vec{j}$
- $\vec{v} = (x, y)$

## GỢI Ý BÀI 3-8



$$\begin{aligned}\overrightarrow{AB} &= (x_B - x_A, y_B - y_A) \\ &= \overrightarrow{OB} - \overrightarrow{OA}\end{aligned}$$

---

# QUESTION AND ANSWER

