VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Kỹ Thuật Lập Trình (Cơ bản và nâng cao C++)

KTLT1 - HK242

TASK 3 ARRAY, STRING, STRUCT, FILE

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 2/2025

Mục lục

1 Kiểu dữ liệu Array

Mảng (*array*) là một cấu trúc dữ liệu bao gồm một tập hợp các phần tử có cùng kiểu dữ liệu, được lưu trữ liên tiếp trong bộ nhớ và có thể truy cập thông qua chỉ số (*index*).

Quản lý nhiều phần tử một cách dễ dàng

Nếu không có mảng, muốn lưu 100 số nguyên, anh phải khai báo 100 biến riêng lẻ:

```
int num1, num2, num3, ..., num100;
```

Việc này không thực tế và rất khó quản lý. Với mảng, chỉ cần khai báo một lần:

```
int num[100]; // Mảng chứa 100 số nguyên
```

Đặc điểm của mảng:

1. **Kích thước cố định:** Khi khai báo, mảng có kích thước cố định và không thể thay đổi trong quá trình chạy chương trình (đối với mảng tĩnh).
2. **Các phần tử có cùng kiểu dữ liệu:** Tất cả phần tử trong mảng đều thuộc cùng một kiểu dữ liệu (ví dụ: số nguyên, số thực, ký tự, v.v.).
3. **Đánh số chỉ mục từ 0:** Trong hầu hết các ngôn ngữ lập trình (C, C++, Java, Python...), chỉ mục của mảng bắt đầu từ 0.
4. **Truy cập nhanh:** Có thể truy cập phần tử bất kỳ trong mảng bằng chỉ số trong thời gian **O(1)**.

Cách lưu trữ mảng một chiều

Mỗi phần tử trong mảng đều có một địa chỉ bộ nhớ riêng, được tính dựa trên địa chỉ của phần tử đầu tiên.

Giả sử **array** được cấp phát bắt đầu từ địa chỉ **0x1000**, thì cách lưu trữ trong bộ nhớ có thể như sau:

```
int arr[5] = {10, 20, 30, 40, 50};
```

Chỉ mục (i)	Giá trị ($arr[i]$)	Địa chỉ bộ nhớ
0	10	0x1000
1	20	0x1004
2	30	0x1008
3	40	0x100C
4	50	0x1010

Với giả định rằng mỗi phần tử kiểu `int` chiếm 4 byte trong bộ nhớ, địa chỉ của phần tử thứ i được tính theo công thức:

$$\text{address}(arr[i]) = \text{address}(arr[0]) + i \times \text{sizeof(int)}$$

Khai báo, truy xuất và thao tác với mảng một chiều

Khai báo mảng một chiều

Mảng một chiều trong C++ có thể được khai báo bằng cú pháp:

```
<kiểu dữ liệu> <tên mảng>[<số phần tử>];
```

Ví dụ khai báo mảng số nguyên có 5 phần tử:

```
int arr[5];  // Khai báo mảng 5 phần tử kiểu int
```

Ngoài ra, ta có thể khai báo và khởi tạo giá trị ban đầu:

```
int arr[5] = {10, 20, 30, 40, 50};
```

Nếu số phần tử không được chỉ định, trình biên dịch sẽ tự động xác định:

```
int arr[] = {10, 20, 30, 40, 50};  // Tự động nhận kích thước là 5
```

Truy xuất phần tử trong mảng

Các phần tử trong mảng có thể được truy xuất bằng chỉ mục, với chỉ mục bắt đầu từ 0. Cú pháp:

```
<tên mảng>[<chỉ mục>];
```

Ví dụ, để truy xuất và in ra giá trị của phần tử thứ hai (chỉ mục 1):

```
cout << arr[1];  // Xuất ra 20
```

Cập nhật giá trị phần tử trong mảng

Ta có thể thay đổi giá trị của một phần tử bằng cách gán giá trị mới cho chỉ mục tương ứng:

```
arr[2] = 100;  // Thay đổi giá trị phần tử thứ ba thành 100
```

Sau khi cập nhật, mảng sẽ trở thành:

```
{10, 20, 100, 40, 50}
```

Nhập và xuất mảng từ bàn phím

Ta có thể nhập giá trị cho mảng từ bàn phím bằng vòng lặp:

```
int arr[5];
for (int i = 0; i < 5; i++) {
    cin >> arr[i];  // Nhập giá trị cho từng phần tử
}
```

Duyệt mảng bằng vòng lặp

Dưới đây là cách duyệt qua tất cả các phần tử trong mảng bằng vòng lặp:

```
for (int i = 0; i < 5; i++) {
    cout << arr[i] << " ";
}
```

Cách duyệt bằng vòng lặp `range-based for` trong C++11 trở lên:

```
for (int x : arr) {
    cout << x << " ";
}
```

Mảng 2 chiều

Mảng hai chiều là một mảng có các phần tử được tổ chức thành hàng và cột, giống như một ma trận.

1. Khai báo mảng 2D

   ```
   <kiểu dữ liệu> <tên mảng>[số hàng][số cột];
   ```

2. Khởi tạo mảng 2D

   ```
   int matrix[3][3] = {
       {1, 2, 3},
       {4, 5, 6},
       {7, 8, 9}
   };

   int matrix[][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
   ```

3. Truy cập phần tử trong mảng 2D

   ```
   cout << matrix[1][2];  // Lấy phần tử hàng 1, cột 2 (giá trị là 6)
   ```

4. Mảng ba chiều (3D array)

```
int array3D[2][3][4] = {
    {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    },
    {
        {13, 14, 15, 16},
        {17, 18, 19, 20},
        {21, 22, 23, 24}
    }
};
```

2 Kiểu dữ liệu xâu kí tự

2.1 Mảng kí tự char

Trong C++, chuỗi ký tự có thể được lưu trữ bằng mảng kiểu `char`. Một chuỗi ký tự kết thúc bằng ký tự `'\0'` (null-terminator).

Khai báo và khởi tạo chuỗi:

```
char str1[] = "Hello";  // Có '\0' tự động thêm vào cuối
char str2[6] = "Hello"; // Phải đủ kích thước để chứa '\0'
char str3[] = {'H', 'e', 'l', 'l', 'o', '\0'}; // Cách khai báo thủ công
char str4[10];  // Mảng ký tự chưa khởi tạo
```

2.2 Thư viện String

Trong C++, string là một kiểu dữ liệu dùng để lưu trữ và thao tác với chuỗi ký tự. Kiểu này thuộc thư viện <string> và cung cấp nhiều phương thức hỗ trợ xử lý chuỗi một cách dễ dàng, thay thế cho mảng ký tự (char[]).

Cách khai báo và sử dụng string

Để sử dụng kiểu dữ liệu `string`, cần khai báo thư viện `<string>`:

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

Có thể khai báo và khởi tạo chuỗi theo nhiều cách khác nhau:

- Gán trực tiếp chuỗi:

  ```
  string s1 = "Hello";
  ```

- Dùng constructor:

  ```
  string s2("World");
  ```

- Khai báo chuỗi rỗng:

  ```
  string s3;
  ```

Nhập và xuất chuỗi

Việc nhập chuỗi có thể thực hiện bằng `cin` hoặc `getline()`.

Nhập một từ:

```
string name;
cout << "Enter your name: ";
cin >> name; // Chỉ nhận một từ, không bao gồm khoảng trắng
cout << "Hello, " << name << "!" << endl;
```

Nhập cả dòng (bao gồm khoảng trắng):

```
string fullName;
cout << "Enter full name: ";
getline(cin, fullName);
cout << "Welcome, " << fullName << "!" << endl;
```

Các thao tác trên chuỗi

1. **Nối chuỗi**:

   ```
   string first = "Hello";
   string second = "World";
   string combined = first + " " + second; // "Hello World"
   ```

2. **Tính độ dài chuỗi**:

```
string text = "Programming";
cout << "minted: " << text.length() << endl;
```

3. **Truy cập ký tự trong chuỗi**:

```
string word = "Hello";
cout << word[1];       // 'e'
cout << word.at(2);    // 'l'
```

4. **Tìm kiếm chuỗi con**:

```
string str = "Hello World";
size_t pos = str.find("World");
if (pos != string::npos)
    cout << "Found at index: " << pos << endl;
```

5. **Trích xuất chuỗi con**:

```
string sentence = "Hello World";
string sub = sentence.substr(6, 5); // "World"
```

,

So sánh chuỗi

Chuỗi trong C++ có thể so sánh bằng các toán tử `==`, `!=`, `<`, `>`, `<=`, `>=`:

```
string a = "Apple", b = "Banana";
if (a < b) cout << "Apple comes before Banana";
```

Mảng `string` trong C++ Có thể khai báo mảng chứa nhiều chuỗi:

```
string names[3] = {"Alice", "Bob", "Charlie"};
cout << names[1];  // "Bob"
```

Duyệt mảng bằng vòng lặp:

```
for (int i = 0; i < 3; i++) {
    cout << names[i] << endl;
}
```

Hoặc dùng vòng lặp `range-based for`:

```
for (string name : names) {
    cout << name << endl;
}
```

Lưu ý về `string` trong C++

- `string` hỗ trợ quản lý bộ nhớ động, không cần cấp phát thủ công như mảng ký tự.
- Khi truyền chuỗi vào hàm, nếu không muốn copy dữ liệu, nên dùng tham chiếu:

```
void printString(const string &s) {
    cout << s << endl;
}
```

2.3 So sánh mảng ký tự `char` và kiểu dữ liệu `std::string`

Tiêu chí	**Mảng ký tự (`char` array)**	`std::string`
Khai báo	`char str1[] = "Hello";` `char str2[10] = "World";`	`std::string str = "Hello";`
Lưu trữ	Mảng ký tự, kết thúc bằng ký tự '\0'	Lớp 'std::string' động, không cần '\0'
Chiều dài	Dùng `strlen()`	Dùng `size()` hoặc `length()`
Gán giá trị	Không thể gán trực tiếp sau khi khai báo	Có thể gán trực tiếp
Nối chuỗi	Dùng `strcat()`	Dùng toán tử '+' hoặc `append()`
So sánh chuỗi	Dùng `strcmp()`	Dùng toán tử '==', '<', '>' trực tiếp
Duyệt từng ký tự	Dùng vòng lặp 'for' thông thường	Có thể dùng vòng lặp 'for' hoặc 'range-based for'
Dễ sử dụng	Cần thao tác với mảng	Dễ dàng sử dụng hơn nhờ các phương thức có sẵn
Quản lý bộ nhớ	Phải tự quản lý kích thước, tránh tràn bộ nhớ	Quản lý động, tự động mở rộng

Bảng 1: *So sánh `char` array và `std::string` trong C++*

3 Kiểu dữ liệu Struct

Struct (cấu trúc) là một kiểu dữ liệu do người dùng định nghĩa trong C++, cho phép nhóm nhiều biến có kiểu dữ liệu khác nhau vào cùng một đơn vị logic.

Lưu trữ thông tin có cấu trúc

Giả sử anh muốn lưu thông tin về một sinh viên, nếu không dùng `struct`, anh phải khai báo từng biến riêng lẻ:

```
string name;
int age;
float gpa;
```

Điều này gây khó khăn khi quản lý nhiều sinh viên. Với `struct`, ta có thể nhóm các thuộc tính lại với nhau:

```
struct Student {
    string name;
    int age;
    float gpa;
};
```

Đặc điểm của struct:

1. **Gom nhóm dữ liệu có liên quan:** Cho phép nhóm các biến khác kiểu vào cùng một đối tượng.
2. **Dễ quản lý và mở rộng:** Có thể tạo nhiều biến kiểu struct mà không cần khai báo lại từng thuộc tính.
3. **Truy cập bằng dấu chấm (`.`):** Các thành phần của struct được truy cập qua dấu chấm.
4. **Không hỗ trợ trực tiếp các phép toán:** Không thể dùng các toán tử như `+`, `-` trực tiếp trên struct.

Cách khai báo và sử dụng struct

Khai báo struct

Cú pháp khai báo một struct:

```
struct <tên struct> {
    <kiểu dữ liệu> <tên thuộc tính>;
    ...
};
```

Ví dụ khai báo struct `Student`:

```
struct Student {
    string name;
    int age;
    float gpa;
};
```

Khai báo biến kiểu struct

Sau khi định nghĩa, có thể khai báo biến kiểu struct như sau:

```
Student s1;  // Khai báo một biến kiểu Student
```

Truy xuất và cập nhật thuộc tính

Các thuộc tính trong struct được truy xuất bằng dấu chấm:

```
s1.name = "John Doe";
s1.age = 20;
s1.gpa = 3.5;
```

```
cout << "Name: " << s1.name << endl;
cout << "Age: " << s1.age << endl;
cout << "GPA: " << s1.gpa << endl;
```

Khởi tạo struct khi khai báo

Có thể gán giá trị ngay khi khai báo:

```
Student s2 = {"Alice", 19, 3.8};
```

Cách lưu trữ struct trong bộ nhớ

Mỗi struct được lưu trong bộ nhớ với các thuộc tính liên tiếp nhau. Giả sử struct sau được cấp phát bộ nhớ:

```
struct Student {
    char name[20];  // 20 bytes
    int age;        // 4 bytes
    float gpa;      // 4 bytes
};
```

Với **Student s1**, bộ nhớ có thể được lưu như sau:

Thuộc tính	Giá trị	Địa chỉ bộ nhớ
name	"John Doe"	0x1000 - 0x1013
age	20	0x1014 - 0x1017
gpa	3.5	0x1018 - 0x101B

Truy xuất địa chỉ của struct và các thành phần

Ta có thể dùng toán tử `&` để lấy địa chỉ:

```
cout << &s1;      // Địa chỉ struct
cout << &s1.name; // Địa chỉ của name
cout << &s1.age;  // Địa chỉ của age
```

Mảng struct

Có thể khai báo mảng struct để lưu danh sách sinh viên:

```
Student students[3] = {
    {"Alice", 19, 3.8},
    {"Bob", 20, 3.6},
    {"Charlie", 21, 3.9}
};

// Duyệt mảng struct
for (int i = 0; i < 3; i++) {
    cout << students[i].name << " - "
         << students[i].age << " - "
         << students[i].gpa << endl;
}
```

4 Kiểu dữ liệu Enum

Enum (liệt kê) là một kiểu dữ liệu trong C++ cho phép định nghĩa một tập hợp các giá trị nguyên có tên gọi, giúp mã nguồn dễ đọc và bảo trì hơn.

Lưu trữ thông tin có tính liệt kê

Giả sử anh muốn đại diện các ngày trong tuần bằng số, nếu không dùng `enum`, anh phải khai báo như sau:

```
const int Monday = 0;
const int Tuesday = 1;
const int Wednesday = 2;
```

Điều này dễ gây nhầm lẫn khi sử dụng. Với `enum`, ta có thể nhóm các giá trị này vào một tập hợp có tên:

```
enum Day {
    Monday,
    Tuesday,
    Wednesday
};
```

Đặc điểm của enum:

1. **Gom nhóm các giá trị có liên quan:** Enum giúp nhóm các hằng số có ý nghĩa thành một kiểu dữ liệu.
2. **Dễ đọc và bảo trì:** Thay vì dùng số, ta có thể dùng tên có ý nghĩa hơn.
3. **Mặc định bắt đầu từ 0:** Giá trị mặc định của phần tử đầu tiên trong enum là 0, các phần tử tiếp theo tăng dần.
4. **Có thể gán giá trị tùy chỉnh:** Ta có thể đặt giá trị cụ thể cho từng phần tử trong enum.

Cách khai báo và sử dụng enum

Khai báo enum

Cú pháp khai báo một enum:

```
enum <tên enum> {
    <tên giá trị> = <số nguyên>,
    ...
};
```

Ví dụ khai báo enum `Day`:

```
enum Day {
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3
};
```

Khai báo biến kiểu enum

Sau khi định nghĩa, có thể khai báo biến kiểu enum như sau:

```
Day today = Monday;  // Gán giá trị cho biến enum
```

Sử dụng enum trong chương trình

```
Day today = Tuesday;

if (today == Tuesday) {
    cout << "It's Tuesday!" << endl;
}
```

Gán giá trị tùy chỉnh cho enum

Có thể đặt giá trị bất kỳ cho các phần tử:

```
enum Status {
    SUCCESS = 200,
    ERROR = 500,
    NOT_FOUND = 404
};
```

Mảng enum

Có thể dùng enum kết hợp với mảng để ánh xạ tên giá trị:

```
const char* dayNames[] = { "Monday", "Tuesday", "Wednesday" };

Day d = Monday;
cout << "Today is " << dayNames[d] << endl;
```

5 Xử lý File trong C++

Trong C++, thư viện `fstream` cung cấp các lớp để làm việc với file, giúp đọc và ghi dữ liệu từ/tới file trên hệ thống.

Các loại file trong C++

Có hai loại file chính trong C++:

- **File văn bản:** Chứa dữ liệu dạng ký tự (text), có thể mở bằng trình soạn thảo thông thường.
- **File nhị phân:** Lưu dữ liệu dưới dạng nhị phân, thường dùng để lưu trữ dữ liệu phức tạp như struct hoặc đối tượng.

Thư viện xử lý file

C++ sử dụng thư viện `fstream`, bao gồm ba lớp chính:

- `ifstream`: Đọc dữ liệu từ file.
- `ofstream`: Ghi dữ liệu vào file.
- `fstream`: Có thể vừa đọc vừa ghi file.

Mở và đóng file

File cần được mở trước khi thao tác và đóng sau khi hoàn thành:

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

ofstream outFile("example.txt"); // Mở file để ghi
if (outFile.is_open()) {
    outFile << "Hello, file!" << endl; // Ghi dữ liệu
    outFile.close(); // Đóng file
}
```

Đọc file văn bản

Sử dụng `ifstream` để đọc file:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ifstream inFile("example.txt"); // Mở file để đọc
    string line;

    if (inFile.is_open()) {
        while (getline(inFile, line)) { // Đọc từng dòng
            cout << line << endl;
        }
        inFile.close(); // Đóng file
    } else {
        cout << "Cannot open file!" << endl;
    }

    return 0;
}
```

Ghi file văn bản

Sử dụng `ofstream` để ghi file:

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream outFile("output.txt");

    if (outFile.is_open()) {
        outFile << "This is a test file." << endl;
        outFile.close();
    } else {
        cout << "Cannot open file!" << endl;
    }

    return 0;
}
```

Các chế độ mở file

Khi mở file, có thể sử dụng nhiều chế độ khác nhau:

- `ios::in` – Mở file để đọc.
- `ios::out` – Mở file để ghi, nếu file tồn tại thì nội dung cũ sẽ bị xóa.
- `ios::app` – Mở file để ghi, nhưng không xóa nội dung cũ (ghi tiếp vào cuối file).
- `ios::binary` – Mở file ở chế độ nhị phân.
- `ios::ate` – Mở file và di chuyển con trỏ đến cuối file.
- `ios::trunc` – Xóa nội dung file nếu file đã tồn tại.

Ví dụ mở file với nhiều chế độ:

```cpp
ofstream outFile("example.txt", ios::app);
```

Kiểm tra trạng thái file

Có thể kiểm tra trạng thái file bằng các hàm:

- `is_open()` – Kiểm tra xem file có mở thành công không.
- `eof()` – Kiểm tra xem đã đến cuối file chưa.
- `fail()` – Kiểm tra lỗi khi thao tác file.

Ví dụ kiểm tra lỗi:

```cpp
ifstream inFile("nonexistent.txt");

if (!inFile) {
    cout << "File not found!" << endl;
}
```

Các toán tử đọc và ghi file

- Toán tử « được sử dụng để ghi dữ liệu vào file với đối tượng `ofstream` hoặc `fstream`:

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream outFile("output.txt");
```

```
    if (outFile.is_open()) {
        outFile << "Hello, world!" << endl;
        outFile << 123 << endl;
        outFile << 3.14 << endl;
        outFile.close();
    } else {
        cout << "Cannot open file!" << endl;
    }

    return 0;
}
```

- Toán tử » được sử dụng để đọc dữ liệu từ file với đối tượng `ifstream` hoặc `fstream`:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ifstream inFile("output.txt");
    string text;
    int number;
    double pi;

    if (inFile.is_open()) {
        inFile >> text >> number >> pi;
        inFile.close();

        cout << "Text: " << text << endl;
        cout << "Number: " << number << endl;
        cout << "Pi: " << pi << endl;
    } else {
        cout << "Cannot open file!" << endl;
    }

    return 0;
}
```