NGÔN NGỮ LẬP TRÌNH C++

Slide 2.2: Array and vector

Giảng viên: Th.S Bùi Văn Kiên

Nội dung

- Mảng
  - Khai báo mảng
  - Số phần tử của mảng
  - Truy cập phần tử của mảng
  - Truyền tham số mảng cho hàm
  - Mảng 2 chiều
  - Mảng và con trỏ
  - Cấp phát tĩnh và cấp phát động
- Vector

3. Mảng

Khái niệm

- Mảng là một loại cấu trúc dữ liệu trong ngôn ngữ lập trình C/C++, nó lưu trữ một tập hợp tuần tự các phần tử cùng kiểu với độ dài cố định.
- Mảng thường được sử dụng để lưu trữ tập hợp dữ liệu, nhưng nó cũng hữu dụng khi dùng để lưu trữ một tập hợp biến có cùng kiểu.

3.1 Khai báo mảng

- Để khai báo một mảng trong C/C++, lập trình viên chỉ định loại phần tử và số lượng phần tử mà một mảng yêu cầu như sau
- <kiểu dữ liệu> arrayName [arraySize];
- Đây được gọi là mảng một chiều. arraySize phải là một hằng số nguyên lớn hơn 0 và kiểu có thể là bất kỳ kiểu dữ liệu nào.
- Ví dụ:

```
double balance[10];
```

3.1 Khai báo mảng

- Mảng nhiều chiều là một mảng của các mảng.
- Mảng có thể có số lượng kích thước bất kỳ.
- Mảng hai chiều: Mảng 2D còn được gọi là ma trận (bảng gồm các hàng và cột).
- Ví dụ mảng 1 chiều:

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

- Để tạo một mảng 2D các số nguyên, hãy xem ví dụ sau:

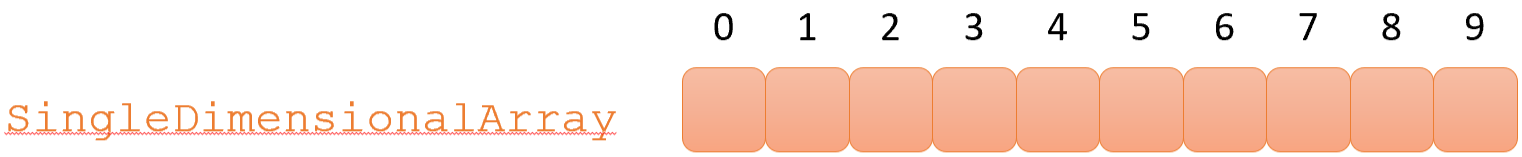
```
int arr[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

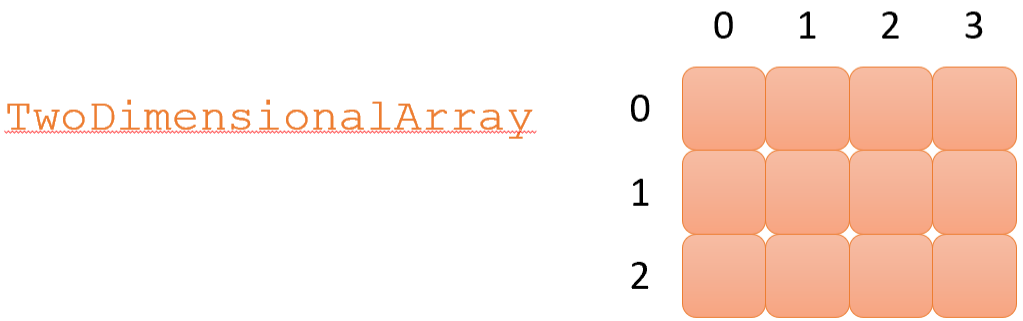
3.1 Khai báo mảng

Khai báo mảng:

```
int SingleDimensionalArray[10];
```




```
int TwoDimensionalArray[3][4];
```



3.2 Số phần tử của mảng

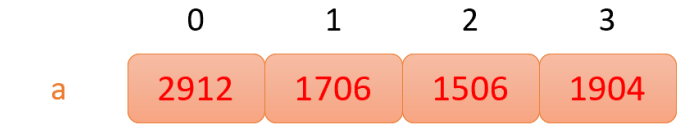
- Phải chỉ định số lượng phần tử tại thời điểm khai báo

int n1 = 10; int a[n1];

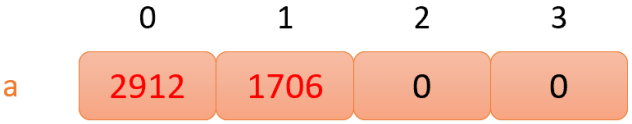
Int b[20];

- Khởi tạo toàn bộ phần tử của mảng
- Khởi tạo một số phần tử của mảng

```
int a[4] = {2912, 1706, 1506, 1904};
```



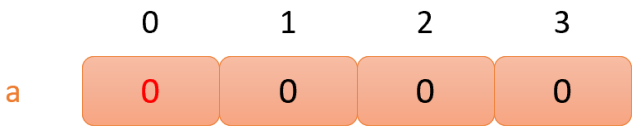

```
int a[4] = {2912, 1706};
```



3.2 Số phần tử của mảng

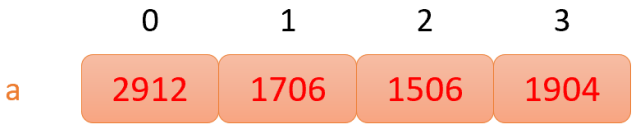
- Khởi tạo tất cả các phần tử của mảng bằng 0

```
int a[4] = {0};
```



- Tự động xác định số phần tử

```
int a[] = {2912, 1706, 1506, 1904};
```



3.3 Truy cập phần tử của mảng

- Sử dụng chỉ số

```
<array variable>[<index1>][<index2>]…[<indexn>]
```

- Ví dụ: cho mảng

  int a[4];

- Truy xuất
  - Hợp lệ: a[0], a[1], a[2], a[3]
  - Không hợp lệ: a[-1], a[4], a[5], …

→ Cho kết quả thường không như mong muốn!

3.3 Truy cập phần tử của mảng

Gán dữ liệu mảng

- Không thể gán thông thường mà phải gán trực tiếp giữa các phần tử tương ứng.

- Ví dụ:

```
#define MAX 3
int a[MAX] = {1, 2, 3}, b[MAX];
b = a;   // Sai
for (int i = 0; i < 3; i++) b[i] = a[i];
```

3.3 Truy cập phần tử của mảng

Một số lỗi thường gặp khi làm việc với mảng

- Khai báo không chỉ định số phần tử

int a[]; → int a[100];

- Khởi tạo tách biệt với khai báo

int a[4]; a = {2912, 1706, 1506, 1904};

→ int a[4] = {2912, 1706, 1506, 1904};

- Truy cập chỉ số mảng không hợp lệ → gây ra lỗi CE trên hệ thống code.ptit

int a[4];

a[-1] = 1; a[10] = 0;

3.4 Truyền tham số mảng cho hàm

- Tham số kiểu mảng trong khai báo hàm giống như khai báo biến mảng

void Sort(int a[100]);

- Tham số mảng truyền vào hàm là địa chỉ của phần tử đầu tiên của mảng
- Có thể bỏ số phần tử hoặc sử dụng con trỏ.

void Sort(int a[]);

void Sort(int *a);

- Khi truyền các mảng làm đối số cho hàm, tương đương với truyền địa chỉ bắt đầu của vùng nhớ được chuyển làm đối số)

→ mảng bị thay đổi giá trị nếu như các câu lệnh trong hàm có tác động lên mảng.

3.4 Truyền tham số mảng cho hàm

- mảng bị thay đổi giá trị nếu như các câu lệnh trong hàm có tác động lên mảng.

```
void Process(int a[], int n) {
        swap(a[1], a[2]);
}
int main(){
        int a[5] = {1, 2, 3, 4, 5};
        Process(a, 5);
        for (int i = 0; i <= 4; i++)
                cout << a[i] << " ";
}
```

3.4 Truyền tham số mảng cho hàm

- Số lượng phần tử sẽ được truyền qua biến khác:

```
void Sort(int a[100], int n);
void Sort(int a[], int n);
void Sort(int *a, int n);

void ReadArray(int a[], int &n);
void PrintArray(int a[], int n);
void main(){
        int a[100], n;
        ReadArray(a, n);
        PrintArray(a, n);
}
```

3.5 Mảng hai chiều

- Khai báo

<Kiểu dữ liệu> <Tên mảng> [<Số dòng tối đa>][< Số cột tối đa>];

- Ví dụ:

```
int A[10][10];      // Khai báo mảng 2 chiều kiểu int
                    //gồm 10 dòng, 10 cột
float b[10][10];    // Khai báo mảng 2 chiều kiểu
                    //float gồm 10 dòng, 10 cột
```

3.5 Mảng hai chiều

- Để truy xuất các thành phần của mảng hai chiều ta phải dựa vào chỉ số dòng và chỉ số cột.
- Ví dụ:

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

Với các khai báo như trên ta có:

```
A[0][0] = 2; A[0][1] = 3;
A[1][1] = 6; A[1][3] = 6;
```

3.6 Con trỏ và mảng

Mảng và con trỏ có quan hệ mật thiết

- Tên mảng coi như con trỏ hằng
- Con trỏ có thể thao tác trên chỉ số của mảng

- Truy cập các phần tử mảng bằng con trỏ

Phần tử b[n] có thể được truy cập bằng *(bPtr+n)

- Địa chỉ, ví dụ: &b[3]tương đương bPtr + 3
- Tên mảng có thể coi như con trỏ, ví dụ:

b[3] tươngđương *(b+3)

- Có thể đánh chỉ số cho con trỏ, ví dụ:

bPtr[3] tương đương b[3]

3.6 Con trỏ và mảng

- Khai báo mảng bằng con trỏ

```
int *a;
a = new int[SIZE];
int **a;
a = new int*[SIZE];
```

- Thu hồi bộ nhớ cấp phát

```
delete[] a;
```

3.7 Cấp phát tĩnh & động

Các vùng nhớ trong 1 chương trình C++

- Code segment: chứa mã máy của chương trình
- Data segment: chứa các biến global hoặc static
- Stack segment: Cơ chế First In – Last Out (FILO)

Stack là một phân khúc lưu trữ những biến tạm thời được CPU quản lý và tối ưu khá chặt chẽ. Khi một biến được khai báo trong hàm, nó sẽ được push vào Stack, khi hàm đó kết thúc, toàn bộ những biến đã được đẩy vào trong stack sẽ được pop và giải phóng. Lưu trên stack→ tính chất local.

Khi sử dụng, không cần quan tâm đến việc cấp phát và thu hồi biến đó.

3.7 Cấp phát tĩnh & động

Các vùng nhớ trong 1 chương trình C++

- Heap Segment:

Heap là phân khúc bộ nhớ cấp phát tự do. Tuy nhiên nó lại không được quản lý tự động, phải tự quản lý toàn bộ những vùng nhớ đã cấp phát trên Heap, nếu không còn sử dụng nữa mà không tự thu hồi sẽ gây ra hiện tượng rò rỉ bộ nhớ – memory leak.

Để cấp phát vùng nhớ trên Heap, có thể dùng malloc() – tức là memory allocation hoặc new. Để thu hồi vùng nhớ bạn có thể dùng free() – tức là memory deallocation hoặc delete.

Truy cập ở phân khúc Stack sẽ nhanh hơn một chút so với Heap nhưng bù lại phân khúc Stack này lại có kích thước có giới hạn tùy vào cấu hình với trình biên dịch.

3.7 Cấp phát tĩnh & động

- Cấp phát tĩnh: kiểu cấp phát đã xác định được kích thước cần sử dụng trước khi biên dịch.

int a = 6;

int arr[10];

Lưu ý: phía trong dấu [] phải là một hằng số.

- arr là:
  - Mảng tĩnh có 10 phần tử (hay 40 bytes) đã được xác định trước thời điểm biên dịch.
  - Các giá trị, biến này sẽ sẽ được lưu trữ trên Stack.
  - Các giá trị sẽ được tự động thu hồi khi ra khỏi tầm vực khai báo.

3.7 Cấp phát tĩnh & động

- Cấp phát động:
  - Tạo ra mảng hay 1 đối tượng với số lượng phần tử được xác định khi chương trình thực thi.
  - Các giá trị sẽ được lưu trữ trên Heap.
  - Khi không cần sử dụng phải chủ động thu hồi.
  - Để cấp phát động sử dụng hàm malloc/free (stdlib.h) trong C hoặc từ khóa new/delete trong C++.
  - Cấp phát động là thao tác chính để tạo nên các cấu trúc dữ liệu như cây nhị phân hay danh sách liên kết

3.7 Cấp phát tĩnh & động

- Ví dụ:

new <kiểu_dữ_liệu_của_mỗi_phần_tử>[size];

int *myArr = new int[100];

- Việc thao tác với mảng động thực hiện qua con trỏ cũng tương tự như đối với mảng thông thường như sau:

myArr[0] = 1;

myArr[1] = 2;

- Hoặc thao tác theo cách dung con trỏ như sau:

*(myArr + 0) = 1;

*(myArr + 1) = 2;

3.7 Cấp phát tĩnh & động

- Ví dụ:

```
int main(){
    int length;
    cin >> length;

    // kích thước mảng có thể là biến số
    int *array = new int[length];

    // sử dụng mảng
    // ...

    delete[] array; // trả lại vùng nhớ mảng array cho hệ điều hành

    return 0;
}
```

3.7 Cấp phát tĩnh & động

- Ưu điểm:
  - Có thể sử dụng biến, hoặc giá trị trả về từ hàm làm tham số trong cặp ngoặc vuông [ ]
  - Cấp phát động lưu trữ vùng nhớ trong Heap, tùy nhu cầu lúc đang thực thi mà cấp phát (lvalue, rvalue: hằng số, truyền biến, giá trị lấy từ hàm), có thể cấp phát cho đến khi hết bộ nhớ.
- Nhược điểm:
  - Phải hoàn toàn kiểm soát được vùng nhớ động, chủ động thu hồi, tránh rò rỉ bộ nhớ.
  - Để làm được điều này chúng ta phải sử dụng đến từ khóa delete (C++) hoặc hàm free (C)

4. Vector

- Vector trong C++ là một container kiểu mảng động, tương tự như mảng nhưng linh hoạt hơn và có khả năng tự động thay đổi kích thước khi thêm hoặc xóa phần tử.
- Ngoài ra nó hỗ trợ truy cập các phần tử trong mảng thông qua chỉ số như mảng 1 chiều.

- Vector có thể lưu các kiểu dữ liệu như int, long long, float, char, pair, hoặc thậm chí là một vector khác. Khi sử dụng vector bạn cần thêm thư viện "vector" vào chương trình của mình.
- Cú pháp: vector<data_type> vector_name;

4. Vector

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main(){
    //Khai báo vector rỗng
    vector<int> v1;

    //Khai báo vector với các phần tử
    vector<int> v2 = {1, 2, 3, 4, 5};

    //Khai báo vector có sẵn n phần tử
    int n = 20;
    vector<int> v3(n);

    //Khai báo vector có sẵn n phần tử có cùng giá trị val
    int val = 0;
    vector<int> v4(n, val);
    return 0;
}
```

4. Vector

Các hàm hỗ trợ

- size() : Trả về số lượng phần tử trong vector. Độ phức tạp O(1)
- length() : Trả về số lượng phần tử trong vector. Độ phức tạp O(1)
- push_back() : Thêm phần tử vào cuối vector. Độ phức tạp O(1)
- pop_back() : Xóa phần tử cuối cùng trong vector. Độ phức tạp O(1)
- assign(): được sử dụng để gán các nội dung mới cho vector, thay thế các nội dung hiện tại và có thể thay đổi kích thước của vector.
- Để duyệt vector bạn có thể duyệt vòng for thông qua chỉ số

4. Vector

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main(){
    vector<int> v;
    v.push_back(1); // {1}
    v.push_back(2); // {1, 2}
    v.push_back(3); // {1, 2, 3}
    v.push_back(4); // {1, 2, 3, 4}
    cout << "Kich thuoc vector : " << v.size() << endl;
    cout << "Duyet vector bang chi so : \n";
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << ' ';
    }
    cout << endl;

    cout << "Duyet vector bang ranged-base for loop :\n";
    for(int x : v){
        cout << x << ' ';
    }
    return 0;
}
```

4. Vector

Thay thế vector cho mảng 1 chiều

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

int main(){
    vector<int> v;
    int n, tmp;
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> tmp;
        v.push_back(tmp);
    }

    cout << "Day so vua nhap : \n";
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << " ";
    }
    return 0;
}
```

4. Vector

Cách 2:

Khai báo vector có sẵn số lượng phần tử được nhập từ bàn phím

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

int main(){
    int n, tmp;
    cin >> n;
    vector<int> v(n);
    for(int i = 0; i < n; i++){
        cin >> v[i];
    }

    for(int i = 0; i < v.size(); i++){
        cout << v[i] << " ";
    }
    return 0;
}
```

vector<int> v;
v.assign(n)

4. Vector và mảng 2 chiều

Cách 1: Nhập từng dòng của mảng 2 chiều như 1 vector

```
int main(){
    int n, m;
    cout << "Nhap hang, cot : ";
    cin >> n >> m;
    vector<vector<int>> v;
    for(int i = 0; i < n; i++){
        vector<int> row;
        for(int j = 0; j < m; j++){
            int tmp;
            cin >> tmp;
            row.push_back(tmp);
        }
        v.push_back(row);
    }
    cout << "Mang 2 chieu vua nhap : \n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << v[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

4. Vector và mảng 2 chiều

Cách 2: Khai báo sẵn vector với kích thước nhập vào

```
int main(){
    int n, m;
    cout << "Nhap hang, cot : ";
    cin >> n >> m;
    vector<vector<int>> v(n, vector<int>(m));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> v[i][j];
        }
    }

    cout << "Mang 2 chieu vua nhap : \n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << v[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

4. So sánh mảng và vector

Mảng:

- Các chỉ số có thể bắt đầu từ 0 hoặc 1 hoặc bất kì
- Tốc độ truy xuất nhanh hơn (kiểu dữ liệu nguyên thủy)

Vector:

- Phần tử đầu tiên luôn là v[0].
- Tốc độ truy xuất chậm hơn 1 chút, không đáng kể lắm nếu số lượng truy xuất không quá nhiều.

Bài tập

- CPP0115, CPP0124
- Phân tích một số ra thừa số nguyên tố:

N = 40

Out

5 1

2 3

Giới hạn:

- $N <= 10^9$
- **In ra các ước theo thứ tự tăng dần của số mũ**

QUESTIONS & ANSWERS