KỸ THUẬT LẬP TRÌNH C/C++

Chương 6:
KỸ THUẬT LẬP TRÌNH VỚI MẢNG VÀ CON TRỎ

Nội dung

1. Kỹ thuật lập trình với mảng một chiều
2. Kỹ thuật lập trình với mảng hai chiều
3. Kỹ thuật lập trình với con trỏ
4. Sự tương quan giữa con trỏ và mảng

1. Kỹ thuật lập trình với mảng một chiều

1. Kỹ thuật lập trình với mảng một chiều

Khái niệm mảng:

Giả sử trong một chương trình cần lưu trữ 3 số nguyên để tính toán, đơn giản là khai báo 3 biến tương ứng cho 3 số nguyên, ví dụ: int s1, s2, s3;

Nhưng trong chương trình cần lưu trữ 100 số nguyên, thì cần khai báo 100 biến kiểu số nguyên? Muốn nhập và lưu trữ các số nguyên này thì phải thực hiện 100 lần. Cách xử lý như thế lại quá "cồng kềnh" và không hiệu quả trong việc viết chương trình C++.

=> Giải pháp: Cần có một cấu trúc dữ liệu cho phép lưu trữ một dãy các số nguyên và dễ dàng truy xuất, đó là mảng (array).

Mảng được sử dụng để lưu trữ nhiều giá trị (hay phần tử mảng) có cùng kiểu dữ liệu trong một biến duy nhất, thay vì khai báo các biến riêng biệt cho từng giá trị.

Mảng được cấp phát một vùng nhớ liên tiếp để lưu trữ các phần tử trong mảng.

Có thể chia mảng làm 2 loại: mảng một chiều và mảng nhiều chiều.

1. Kỹ thuật lập trình với mảng một chiều

Mảng một chiều là một dãy các phần tử có cùng một biến kiểu mảng, có một chỉ số để chỉ thứ tự của phần tử đó trong dãy. Cú pháp khai báo mảng một chiều trong C++ như sau:

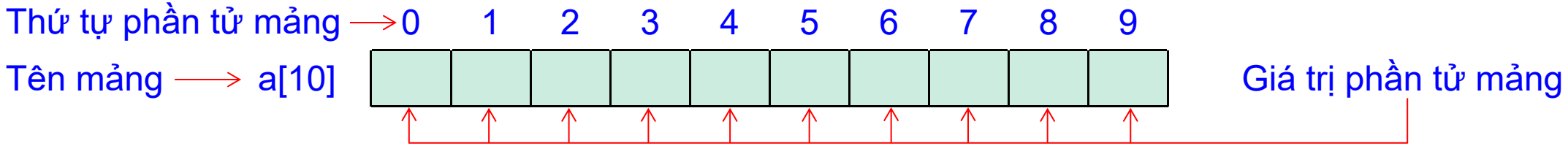
data_type array_name[array_size];

Trong đó:

- data_type: kiểu dữ liệu của biến mảng (int, float, char, string, ...)
- array_name: tên của mảng
- array_size: kích thước của mảng, chỉ định số phần tử mảng. Kích thước của mảng phải được định nghĩa tại thời điểm khai báo, không thể lưu số phần tử vượt quá kích thước mảng.

Thứ tự các phần tử trong mảng được đánh số từ 0 đến (array_size - 1).

Ví dụ 1: Khai báo mảng a có 10 phần tử với kiểu dữ liệu số nguyên: **int a[10];**



1. Kỹ thuật lập trình với mảng một chiều

❖ **Lưu ý khi khai báo mảng:**

- Phải xác định cụ thể số phần tử của mảng ngay lúc khai báo. Không được sử dụng biến để khai báo số phần tử của mảng.

Ví dụ 2: int n = 10;

```
int a[n]; //error: expected constant expression
```

- Nên sử dụng khai báo hằng số #define để định nghĩa số phần tử của mảng bằng biến.

Ví dụ 3:

```
#define n 10
int  a[n];  //tương đương int a[10];
```

1. Kỹ thuật lập trình với mảng một chiều

1. Khởi tạo mảng một chiều:

Cách 1: Khởi tạo giá trị cho mọi phần tử của mảng.

Ví dụ 4: int a[4] = {120, 174, 135, 199};

	0	1	2	3
a[4]	120	174	135	199

Cách 2: Khởi tạo giá trị cho một số phần tử đầu mảng.

Ví dụ 5: int a[4] = {120, 174};

	0	1	2	3
a[4]	120	174	0	0

Các phần tử trong mảng không được gán giá trị khởi tạo thì sẽ tự động nhận giá trị ngẫu nhiên. Trong mảng, giá trị ngẫu nhiên này thường là 0.

1. Kỹ thuật lập trình với mảng một chiều

1. Khởi tạo mảng một chiều (tt):

Cách 3: Khởi tạo giá trị 0 cho tất cả phần tử của mảng.

Ví dụ 6: int a[4] = {0};

	0	1	2	3
a[4]	0	0	0	0

Cách 4: Tự động xác định số lượng phần tử

Ví dụ 7: int a[ ] = {120, 174, 135, 199}

	0	1	2	3
a[ ]	120	174	135	199

Chương trình sẽ tự động hiểu số lượng các phần tử trong mảng là 4, không cần khai báo số phần tử (int a[4]) trong mảng.

1. Kỹ thuật lập trình với mảng một chiều

2. Truy xuất đến một phần tử trong mảng một chiều:

Truy xuất mảng một chiều thông qua chỉ số thứ tự phần tử của mảng.

Cú pháp: **array_name[index]**

- array_name: tên biến mảng
- index: chỉ số thứ tự phần tử mảng

Ví dụ 8: Giả sử đã khởi tạo mảng một chiều như sau: int a[4] = {120, 174, 135, 199};

Truy xuất phần tử trong mảng a:

- Các truy xuất hợp lệ: a[0] có giá trị là 120
  a[1] có giá trị là 174
  a[2] có giá trị là 135
  a[3] có giá trị là 199
- Các truy xuất không hợp lệ: a[-1], a[4], a[5]

	0	1	2	3
Mảng a[4]	120	174	135	199
	a[0]	a[1]	a[2]	a[3]

1. Kỹ thuật lập trình với mảng một chiều

3. Xuất mảng một nhiều

Ví dụ 9 (a): Truy xuất mảng số nguyên.

```
#include <iostream>
using namespace std;
int main() {
        int a[4] = {120, 174, 135, 199};
        cout << a[0] <<endl;
        cout << a[1] <<endl;
        cout << a[2] <<endl;
        cout << a[3] <<endl;
        return 0;
}
```

➢ Kết quả:

Ví dụ 9(b): In tất cả giá trị phần tử trong mảng ra màn hình.

```
#include <iostream>
using namespace std;
int main() {
        int a[4] = {120, 174, 135, 199};
        for (int i = 0; i < 4; i++) {
            cout << "Phan tu thu " << i << " la: " << a[i] << endl;
        }
  return 0;
}
```

➢ Kết quả:

```
Phan tu thu 0 la: 120
Phan tu thu 1 la: 174
Phan tu thu 2 la: 135
Phan tu thu 3 la: 199
```

1. Kỹ thuật lập trình với mảng một chiều

3. Xuất mảng một nhiều (tt)

Ví dụ 10 (a): Truy xuất mảng số thực.

```
#include <iostream>
using namespace std;
int main() {
        float a[4] = {11.5, 34.7, 22.4, 15.6};
        cout << a[0] <<endl;
        cout << a[2] <<endl;
        return 0;
}
```

➢ Kết quả: 11.5
22.4

Ví dụ 10 (b): In tất cả giá trị phần tử trong mảng ra màn hình.

```
#include <iostream>
using namespace std;
int main() {
      float a[4] = {11.5, 34.7, 22.4, 15.6};
      for (int i = 0; i < 4; i++) {
          cout << "Phan tu thu " << i << " la: " << a[i] << endl;
      }
  return 0;
}
```

➢ Kết quả:

```
Phan tu thu 0 la: 11.5
Phan tu thu 1 la: 34.7
Phan tu thu 2 la: 22.4
Phan tu thu 3 la: 15.6
```

1. Kỹ thuật lập trình với mảng một chiều

3. Xuất mảng một nhiều (tt)

Ví dụ 11 (a): Truy xuất mảng ký tự.

```
#include <iostream>
using namespace std;
int main() {
    char c[8]={'G', 'i', 'a', ' ', 'D', 'i', 'n', 'h'};
    cout << "Ky tu dau tien la: " << c[0] << endl;
    cout << "Ky tu thu nam la: " << c[4] << endl;
    return 0;
}
```

➢ Kết quả: Ky tu dau tien la: G
Ky tu thu nam la: D

Ví dụ 11 (b): Truy xuất mảng ký tự.

```
#include <iostream>
using namespace std;
int main() {
   char c[8]={'G', 'i', 'a', ' ', 'D', 'i', 'n', 'h'};
    for (int i = 0; i < 8; i++) {
        cout << c[i];
    }
    return 0;
}
```

➢ Kết quả: Gia Dinh

1. Kỹ thuật lập trình với mảng một chiều

3. Xuất mảng một nhiều (tt)

Ví dụ 12 (a): Truy xuất mảng chuỗi ký tự.

```
#include <iostream>
using namespace std;
int main() {
	string cars[4] = {"Xe Toyota", "Xe Mazda", "Xe Honda", "Xe Nissan"};
	cout << cars[0] <<endl;
	cout << cars[2] <<endl;
	return 0;
}
```

➢ Kết quả: Xe Toyota
Xe Honda

1. Kỹ thuật lập trình với mảng một chiều

3. Xuất mảng một nhiều (tt)

Ví dụ 12 (b): Truy xuất mảng chuỗi ký tự.

```
#include <iostream>
using namespace std;
int main() {
  string cars[4] = {"Xe Toyota", "Xe Mazda", "Xe Honda", "Xe Nissan"};
  for (int i = 0; i < 4; i++) {
        cout << cars[i] <<endl;
    }
  return 0;
}
```

- Kết quả:

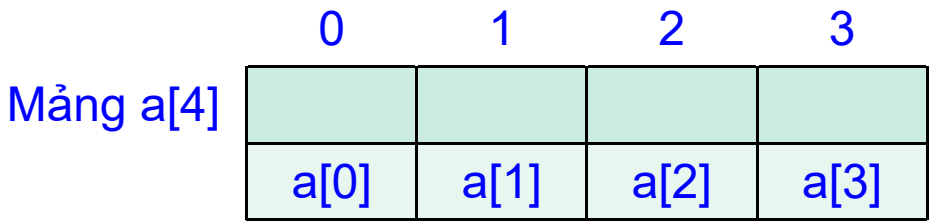
```
Xe Toyota
Xe Mazda
Xe Honda
Xe Nissan
```

1. Kỹ thuật lập trình với mảng một chiều

4. Nhập giá trị cho mảng một nhiều

Ví dụ 13: Nhập giá trị số nguyên cho mảng một chiều từ bàn phím trong lúc chạy chương trình.

```
#include <iostream>
using namespace std;
int main() {
	int a[4]; //khai báo mãng rỗng chứa 4 phần tử
	//Nhập phần tử mảng 1 chiều
	cout << "Nhap vao gia tri phan tu mang (so nguyen): "<<endl;
	for(int i= 0; i < 4; i++){
		cin >> a[i];
	}
	//Xuất mảng 1 chiều
	cout << "Gia tri cua mang da nhap la: ";
	for(int i= 0; i < 4; i++){
		cout << a[i] << " ";
	}
	return 0;
}
```



➢ Kết quả:

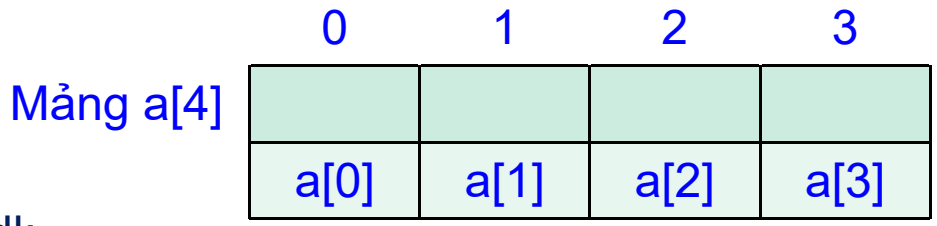
```
Nhap vao gia tri phan tu mang (so nguyen):
11
19
14
25
Gia tri cua mang da nhap la: 11 19 14 25
```

1. Kỹ thuật lập trình với mảng một chiều

4. Nhập giá trị cho mảng một nhiều (tt)

Ví dụ 14: Nhập giá trị cho mảng số thực từ bàn phím trong lúc chạy chương trình.

```
#include <iostream>
using namespace std;
int main() {
	float a[4];
	cout << "Nhap vao gia tri phan tu mang (so thuc): "<<endl;
	for(int i= 0; i < 4; i++){
		cin >> a[i];
	}
	cout<<"Gia tri cua mang da nhap la: ";
	for(int i= 0; i < 4; i++){
		cout<< a[i] << " ";
	}
	return 0;
}
```



➢ Kết quả:

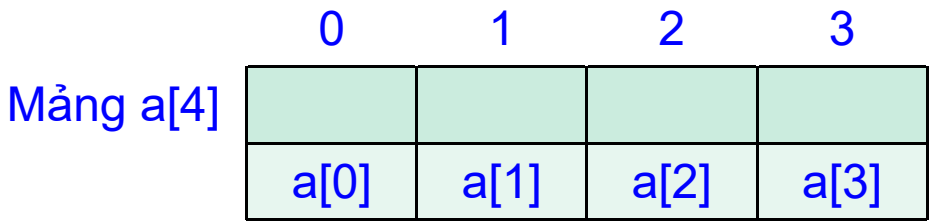
```
Nhap vao gia tri phan tu mang (so thuc):
5.7
6.8
7.2
4.9
Gia tri cua mang da nhap la: 5.7 6.8 7.2 4.9
```

1. Kỹ thuật lập trình với mảng một chiều

4. Nhập giá trị cho mảng một nhiều (tt)

Ví dụ 15: Nhập giá trị cho mảng ký tự từ bàn phím trong lúc chạy chương trình.

```
#include <iostream>
using namespace std;
int main() {
	char a[4];
	cout << "Nhap vao gia tri phan tu mang (ky tu): "<<endl;
	for(int i= 0; i < 4; i++){
		cin >> a[i];
	}
	cout<<"Gia tri cua mang da nhap la: ";
	for(int i= 0; i < 4; i++){
		cout<< a[i] << " ";
	}
	return 0;
}
```



➢ Kết quả:

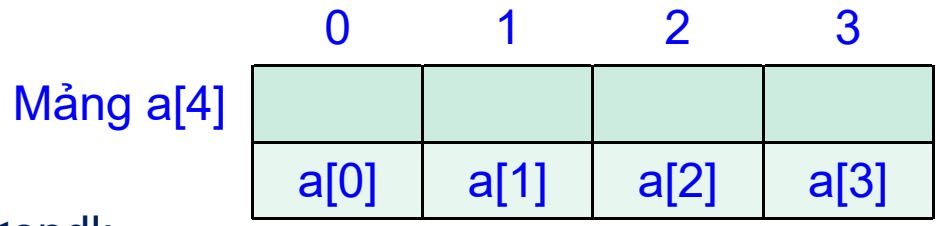
```
Nhap vao gia tri phan tu mang (ky tu):
A
B
C
D
Gia tri cua mang da nhap la: A B C D
```

1. Kỹ thuật lập trình với mảng một chiều

4. Nhập giá trị cho mảng một nhiều (tt)

Ví dụ 16: Nhập giá trị cho mảng chuỗi ký tự từ bàn phím trong lúc chạy chương trình.

```
#include <iostream>
using namespace std;
int main() {
	string a[4];
	cout << "Nhap vao gia tri phan tu mang (chuoi ky tu): "<<endl;
	for(int i= 0; i < 4; i++){
		getline(cin,a[i]);
	}
	cout<<"Gia tri cua mang da nhap la: ";
	for(int i= 0; i < 4; i++){
		cout<< a[i] << " ";
	}
	return 0;
}
```



➢ Kết quả:

```
Nhap vao gia tri phan tu mang (chuoi ky tu):
Mon Toan
Mon Van
Mon Anh
Mon Ly
Gia tri cua mang da nhap la: Mon Toan Mon Van Mon Anh Mon Ly
```

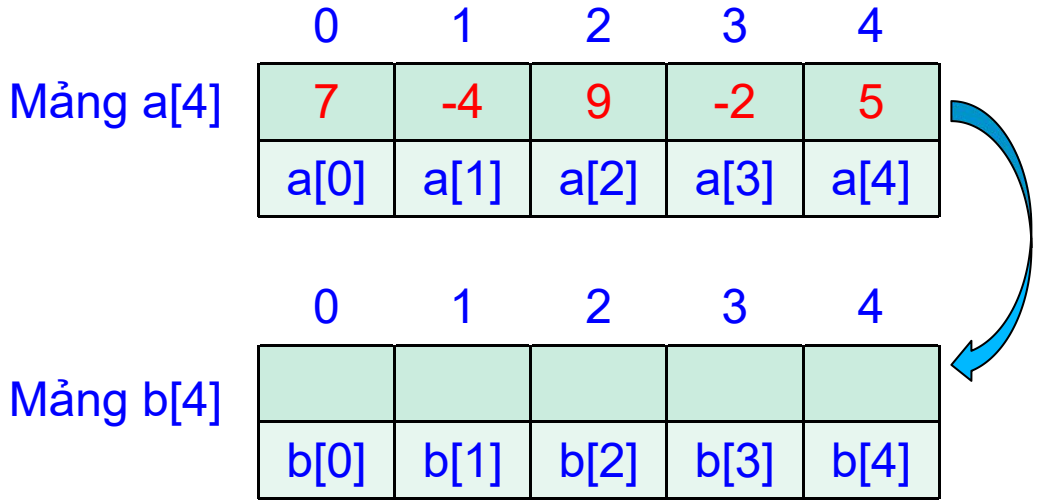
1. Kỹ thuật lập trình với mảng một chiều

5. Sao chép dữ liệu từ mảng này sang mảng khác

Ví dụ 17: Cho hai mảng số nguyên a và b có kích thước gồm 5 phần tử: int a[5] = {7, -4, 9, -2, 5}; int b[5];

Sao chép dữ liệu từ mảng a này sang mảng b.

```
#include <iostream>
using namespace std;
int main() {
	int a[5] = {7, -4, 9, -2, 5};
	int b[5];
	for(int i= 0; i < 5; i++){
		b[i] = a[i];
	}
	cout <<"Phan tu mang b la: ";
	for(int j= 0; j < 5; j++){
		cout << b[j] << " ";
	}
	return 0;
}
```



- Kết quả:

  Phan tu mang b la: 7 -4 9 -2 5

2. Kỹ thuật lập trình với mảng hai chiều

2. Kỹ thuật lập trình với mảng hai chiều

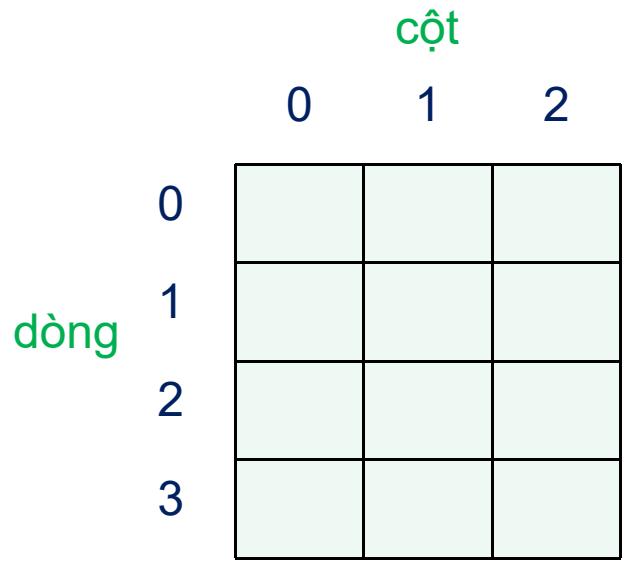
Trong lập trình C++, có những trường hợp cần lưu trữ dữ liệu dưới dạng ma trận gồm nhiều dòng và nhiều cột. Do đó, cần một cấu trúc dữ liệu để lưu trữ ma trận, đó là mảng hai chiều. Mảng 2 chiều lưu trữ các phần tử theo dạng bảng gồm dòng và cột. Mỗi dòng lưu trữ như mảng một chiều.

Cú pháp khai báo mảng hai chiều:

```
data_type  array_name[array_size1][array_size2];
```

Trong đó:

- data_type: kiểu dữ liệu của biến mảng (int, float, char, string, ...)
- array_name: tên của mảng
- array_size1: số dòng của mảng hai chiều
- array_size2: số cột của mảng hai chiều



2. Kỹ thuật lập trình với mảng hai chiều

1. Khởi tạo mảng hai chiều:

Cách 1: Khởi tạo mảng hai chiều tương tự như mảng một chiều nhưng có khai báo số dòng và số cột.

Ví dụ 18: Khởi tạo mảng số nguyên hai chiều với 4 dòng 3 cột.

```
int a[4][3] = {12, 27, 14, 39, 45, 52, 21, 63, 72, 15, 89, 34};
```

	0	1	2
0	12	27	14
1	39	45	52
2	21	63	72
3	15	89	34

Cách 2: Khởi tạo nhóm các dòng

Ví dụ 19: Khởi tạo mảng số nguyên hai chiều với 4 dòng 3 cột.

```
int a[4][3] = {{12, 27, 14}, {39, 45, 52}, {21, 63, 72}, {15, 89, 34}};
```

2. Kỹ thuật lập trình với mảng hai chiều

1. Khởi tạo mảng hai chiều (tt):

Cách 3: Khởi tạo một số phần tử đầu, các phần tử còn lại có giá trị ngẫu nhiên, thường là 0.

Ví dụ 20: Khởi tạo mảng số nguyên hai chiều với 6 dòng 4 cột.

```
int a[6][4] = {{12, 27, 14}, {39, 45, 52}, {21, 63, 72}, {15, 89, 34}};
```

	0	1	2	3
0	12	27	14	0
1	39	45	52	0
2	21	63	72	0
3	15	89	34	0
4	0	0	0	0
5	0	0	0	0

Cách 4: Tự động xác định số lượng dòng nhưng phải chỉ định trước số cột.

Ví dụ 21: Khởi tạo mảng số nguyên hai chiều với 4 dòng 3 cột.

```
int a[ ][3] = {{12, 27, 14}, {39, 45, 52}, {21, 63, 72}, {15, 89, 34}};
```

2. Kỹ thuật lập trình với mảng hai chiều

2. Truy xuất đến một phần tử trong mảng hai chiều:

Truy xuất đến một phần tử trong mảng hai chiều thông qua chỉ số hàng và chỉ số cột trong mảng.

Cú pháp: **array_name[index1][index2]**

- array_name: tên biến mảng
- index1: chỉ số hàng
- index2: chỉ số cột

	0	1	2
0	12	27	14
1	39	45	52
2	21	63	72
3	15	89	34

Ví dụ 22: Cho mảng số nguyên hai chiều với 4 dòng 3 cột.

int a[4][3] = {12, 27, 14, 39, 45, 52, 21, 63, 72, 15, 89, 34};

Truy xuất phần tử mảng ở dòng 2 và cột 1: a[2][1] => giá trị truy xuất được từ mảng là 63.

2. Kỹ thuật lập trình với mảng hai chiều

3. Xuất mảng hai nhiều

Ví dụ 23: Xuất giá trị số nguyên trong mảng hai chiều tại dòng 2 và cột 1.

```
#include <iostream>
using namespace std;
 int main() {
      int a[4][3] = {{12, 27, 14}, {39, 45, 52}, {21, 63, 72}, {15, 89, 34}};
      cout << "Gia tri phan tu mang o dong 2 cot 1 la: ";
      cout << a[2][1] << " ";
      return 0;
}
```

	0	1	2
0	12	27	14
1	39	45	52
2	21	63	72
3	15	89	34

- Kết quả:

  Gia tri phan tu mang o dong 2 cot 1 la: 63

2. Kỹ thuật lập trình với mảng hai chiều

3. Xuất mảng hai nhiều

Ví dụ 24: Xuất tất cả giá trị số nguyên trong mảng hai chiều ra màn hình.

```
#include <iostream>
using namespace std;
 int main() {
    int i, j;
    int a[4][3] = {{12, 27, 14}, {39, 45, 52}, {21, 63, 72}, {15, 89, 34}};
    for (i = 0; i < 4; i++) {
       for (j = 0; j < 3; j++) {
            cout << a[i][j] << " ";
       }
    cout << endl;
    }
    return 0;
}
```

- Kết quả:

```
12  27  14
39  45  52
21  63  72
15  89  34
```

2. Kỹ thuật lập trình với mảng hai chiều

3. Xuất mảng hai nhiều (tt)

Ví dụ 25: Xuất ký tự trong mảng hai chiều ra màn hình.

```
#include <iostream>
using namespace std;
int main() {
  string letters[2][4] = {{ "A", "B", "C", "D" },{ "E", "F", "G", "H" }};
  cout << letters[0][2] <<endl;
  cout << letters[1][3] <<endl;
  return 0;
}
```

- Kết quả:

```
C
H
```

2. Kỹ thuật lập trình với mảng hai chiều

4. Nhập mảng hai nhiều

Ví dụ 26: Nhập giá trị số nguyên cho mảng hai chiều (2 dòng và 3 cột), số nguyên được nhập từ bàn phím trong lúc chạy chương trình.

```
#include <iostream>
using namespace std;
int main() {
	int a[2][3];
	cout<< "Nhap gia tri cho mang hai chieu: "<<endl;
	for(int i=0;i<2;i++){
		for(int j=0;j<3;j++){
			cout << "a[" << i << "][" << j << "]=";
			cin>>a[i][j];
		}
		cout<< endl;
	}
	cout << "Mang hai chieu vua nhap la: " <<endl;
	for(int i=0;i<2;i++){
		for(int j=0;j<3;j++){
			cout<<a[i][j]<<" ";
		}
		cout<<endl;
	}
	system("pause");
}
```

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]

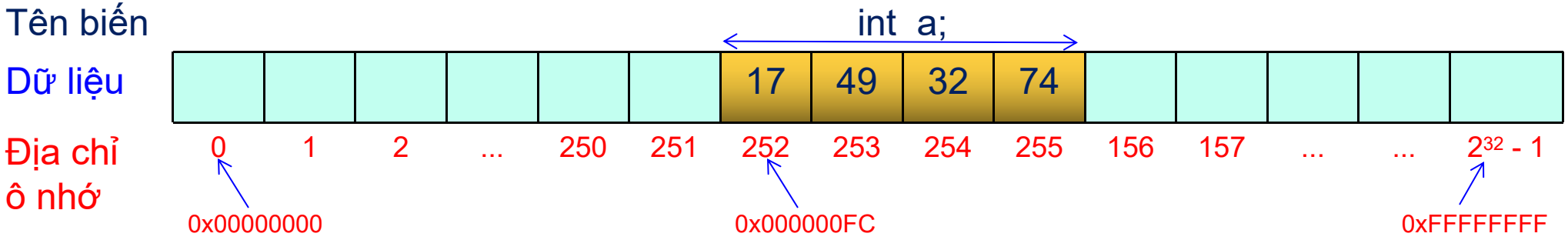
- Kết quả:

```
Nhap gia tri cho mang hai chieu:
a[0][0]=19
a[0][1]=82
a[0][2]=37

a[1][0]=25
a[1][1]=34
a[1][2]=55

Mang hai chieu vua nhap la:
19 82 37
25 34 55
```

3. Kỹ thuật lập trình với con trỏ

3. Kỹ thuật lập trình với con trỏ

Bộ nhớ RAM chứa rất nhiều ô nhớ, mỗi ô nhớ có kích thước 1 byte và có địa chỉ duy nhất, địa chỉ này được đánh số từ 0 trở đi. Đối với CPU 32 bit thì có 2^{32} địa chỉ được đánh số từ 0 đến 2^{32} - 1, thường được biểu diễn dưới dạng số thập lục phân từ 0x00000000 đến 0xFFFFFFFF



Khi khai báo biến, trình biên dịch dành riêng một vùng nhớ với địa chỉ duy nhất để lưu biến.

Trình biên dịch có nhiệm vụ liên kết địa chỉ ô nhớ đó với tên biến. Khi gọi tên biến, nó sẽ truy xuất tự động đến ô nhớ đã liên kết với tên biến để lấy dữ liệu. Trong lập trình C++ để lấy địa chỉ của một biến bằng cách sử dụng toán tử & đặt trước biến đó, ví dụ &a.

3. Kỹ thuật lập trình với con trỏ

Phân biệt giữa địa chỉ bộ nhớ và dữ liệu được lưu trong đó.

Ví dụ 27:

```
#include <iostream>
using namespace std;
int main() {
    int number = 1024;
    cout << "Gia tri bien: " << number << endl;
    cout << "Dia chi bien: " << &number << endl;
    return 0;
}
```

➢ Kết quả:

Gia tri bien: 1024

Dia chi bien: 0x78fe1c

3. Kỹ thuật lập trình với con trỏ

1. Khai báo biến con trỏ

Trong ngôn ngữ C++, con trỏ (pointer) là những biến lưu trữ địa chỉ bộ nhớ của những biến khác.

Cú pháp khai báo biến con trỏ:

kiểu_dữ_liệu *tên_biến;

Trong đó: kiểu_dữ_liệu là một trong các kiểu dữ liệu cơ bản (int, float, char, string, ...).

tên_biến là tên biến của con trỏ.

Ký tự * cho biết tên biến thuộc loại con trỏ.

Ví dụ 28:

```
int  *inp;    // inp là biến con trỏ, trỏ tới vùng nhớ kiểu int
float  *flop;  // flop là biến con trỏ,trỏ tới vùng nhớ kiểu float
double *dop;  // dop là biến con trỏ, trỏ tới vùng nhớ kiểu double
char  *chp     // chp là biến con trỏ, trỏ tới vùng nhớ kiểu char
```

3. Kỹ thuật lập trình với con trỏ

2. Khởi tạo biến con trỏ

Khi mới khai báo, biến con trỏ được đặt ở địa chỉ nào đó (không biết trước), chứa giá trị là một địa chỉ không xác định hoặc địa chỉ 0xCCCCCCCC (địa chỉ đặc biệt), cho biết con trỏ chưa được khởi tạo. Sử dụng toán tử & để đặt địa chỉ của một biến thông thường vào biến con trỏ.

Cú pháp khởi tạo biến con trỏ:

<biến_con_trỏ> = &<tên_biến>;

Ví dụ 29:

```
int a, b;  // khai báo biến a và b kiểu int, lúc này bộ nhớ RAM sẽ dành ra 2 vùng nhớ kiểu số
           // nguyên để lưu giá trị cho 2 biến này.
int *pa, *pb; // khai báo biến 2 biến con trỏ pa và pb, để trỏ tới 2 vùng nhớ kiểu int.
pa = &a; // khởi tạo biến con trỏ pa chứa địa chỉ của biến a.
pb = &b; // khởi tạo biến con trỏ pb chứa địa chỉ của biến b.
```

3. Kỹ thuật lập trình với con trỏ

3. Sử dụng biến con trỏ

Con trỏ được sử dụng để lưu địa chỉ của biến và lấy giá trị của biến do con trỏ trỏ đến.

Ví dụ 30 (a):

```
#include <iostream>
using namespace std;
int main() {
	int a = 5;
	int *pa;
	pa = &a;
	cout << "Gia tri cua bien con tro pa la dia chi cua bien a: " << pa <<endl;
	cout << "Dia chi cua bien a ma bien con tro pa tro den: " << &a <<endl;
	cout << "Gia tri vung nho ma bien con tro pa tro den: " << *pa <<endl;
	cout << "Gia tri cua bien a ma bien con tro pa tro den: " << a <<endl;
	cout << "Dia chi cua bien con tro pa: " << &pa <<endl;
	system("pause");
}
```

Kết quả:
```
Gia tri cua bien con tro pa la dia chi cua bien a: 0x78fe1c
Dia chi cua bien a ma bien con tro pa tro den: 0x78fe1c
Gia tri vung nho ma bien con tro pa tro den: 5
Gia tri cua bien a ma bien con tro pa tro den: 5
Dia chi cua bien con tro pa: 0x78fe10
```

3. Kỹ thuật lập trình với con trỏ

3. Sử dụng biến con trỏ

Con trỏ được sử dụng để lưu địa chỉ của biến và lấy giá trị của biến do con trỏ trỏ đến.

Ví dụ 30 (b):

```
#include <iostream>
using namespace std;
int main() {
	int *pa;
	int a=5;
	pa=&a;
	*pa=150;
	cout<<"Gia tri cua a la:"<<a;
	system("pause");
}
```

Kết quả:

Gia tri cua a: ??

3. Kỹ thuật lập trình với con trỏ

4. Kích thước của con trỏ

Con trỏ chỉ lưu địa chỉ nên kích thước của mọi con trỏ là như nhau. Kích thước này phụ thuộc vào hệ điều hành máy tính: Windows 32 bit là 4 bytes, Windows 64 bit là 8 bytes.

Ví dụ 31: Chương trình xem kích thước của con trỏ.

```
#include <iostream>
using namespace std;
int main() {
	char *p1;
	int *p2;
	float *p3;
	double *p4;
	cout <<"Size of char type pointer: " << sizeof(char *) <<" bytes" <<endl;
	cout <<"Size of int type pointer: " << sizeof(int *) <<" bytes" <<endl;
	cout <<"Size of float type pointer: " << sizeof(float *) <<" bytes" <<endl;
	cout <<"Size of double type pointer: " << sizeof(double *) <<" bytes" <<endl;
	system("pause");
}
```

Kết quả:

```
Size of char type pointer: 8 bytes
Size of int type pointer: 8 bytes
Size of float type pointer: 8 bytes
Size of double type pointer: 8 bytes
```

4. Sự tương quan giữa con trỏ và mảng

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 1 chiều

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác định qua biến con trỏ.

Xem xét một mảng có 5 phần tử được đánh vị trí phần tử trong mảng từ 0 đến 4. Các phần tử được lưu trữ trong một dãy các ô nhớ liên tục trên bộ nhớ. Do đó chỉ cần xác định được địa chỉ ô nhớ của phần tử đầu tiên thì sẽ suy ra được các phần tử còn lại.

Ví dụ 32: Cho mảng int a[5] = {120, 174, 135, 199, 231};

Tên mảng a bản chất là con trỏ lưu địa chỉ của phần tử đầu tiên trong mảng (a = &a[0]).

Có thể sử dụng một biến con trỏ để lưu trữ địa chỉ của phần tử đầu tiên trong mảng để thay cho tên của mảng. Lúc này, biến con trỏ tương đương với mảng 1 chiều.

Ví dụ 33: int a[5], *pa; //khai báo mảng và biến con trỏ

pa = a; hoặc pa = &a[0]; //biến con trỏ lưu trữ địa chỉ của phần tử đầu tiên trong mảng.

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 1 chiều

Sử dụng con trỏ để truy xuất các phần tử trong mảng 1 chiều.

Ví dụ 34: Truy xuất theo địa chỉ của những phần tử trong mảng

```
#include <iostream>
using namespace std;
int main() {
	int a[5] = {120, 174, 135, 199, 231};
	int *pa;
	pa = &a[0];
	cout << pa << endl;  //pa tương đương &a[0]
	cout << pa + 1 << endl; //pa + 1 tương đương &a[1]
	cout << pa + 2 << endl; //pa + 2 tương đương &a[2]
	cout << pa + 3 << endl; //pa + 3 tương đương &a[3]
	cout << pa + 4 << endl; //pa + 4 tương đương &a[4]
	return 0;
}
```

Kết quả:

```
0x78fe00
0x78fe04
0x78fe08
0x78fe0c
0x78fe10
```

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 1 chiều

Ví dụ 35: Truy xuất giá trị của những phần tử trong mảng

```
#include <iostream>
using namespace std;
int main() {
	int a[5] = {120, 174, 135, 199, 231};
	int *pa;
	pa = &a[0];
	cout << "Gia tri cua phan tu thu nhat: " << *pa << endl;
	cout << "Gia tri cua phan tu thu hai: " << *(pa + 1) << endl;
	cout << "Gia tri cua phan tu thu ba: " << *(pa + 2) << endl;
	cout << "Gia tri cua phan tu thu tu: " << *(pa + 3) << endl;
	cout << "Gia tri cua phan tu thu nam: " << *(pa + 4) << endl;
	return 0;
}
```

Kết quả:

```
Gia tri cua phan tu thu nhat: 120
Gia tri cua phan tu thu hai: 174
Gia tri cua phan tu thu ba: 135
Gia tri cua phan tu thu tu: 199
Gia tri cua phan tu thu nam: 231
```

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 1 chiều

Ví dụ 36: Nhập xuất mảng 1 chiều với con trỏ

```
#include <iostream>
using namespace std;
int main() {
	int a[5];
	int *pa;
	pa = &a[0];
	cout << "Nhap vao gia tri mang (5 phan tu): "<<endl;
	for(int i=0;i<5;i++){
		cin>>pa[i];
	}
	cout << "Gia tri cua mang da nhap la: ";
	for(int i=0;i<5;i++){
		cout<<*(pa + i)<<" ";
	}
	return 0;
}
```

Kết quả:

```
Nhap vao gia tri mang (5 phan tu):
5
12
9
22
15
Gia tri cua mang da nhap la:
5 12 9 22 15
```

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 2 chiều

Trong mảng 2 chiều thì các phần tử cũng được cấp phát một dãy các ô nhớ liên tục trên bộ nhớ.

Ví dụ 37: Khai báo mảng 2 chiều, gồm 2 dòng, 3 cột: int a[2][3] = {{12, 27, 14}, {39, 45, 52}};

Có thể xem mảng a như mảng 1 chiều có 2 phần tử, mỗi phần tử là một mảng 1 chiều chứa 3 số nguyên.

- Phần tử đầu tiên: a[0] = {12, 27, 14}
- Phần tử thứ hai: a[1] = {39, 45, 52}

Tên mảng a chứa địa chỉ của phần tử đầu tiên trong mảng 2 chiều (a = &a[0][0]).

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 2 chiều

Có thể sử dụng một biến con trỏ để lưu trữ mảng 2 chiều để thay cho tên của mảng 2 chiều.

Ví dụ 38:

```
int a[2][3] = {{12, 27, 14}, {39, 45, 52}};
int *pa;
pa = (int *)a;
```

Chú ý: Cần phải khai báo pa = (int *)a

Vì biến pa là biến con trỏ kiểu int, a = &a[0][0], tức là a lưu địa chỉ phần tử đầu tiên của mảng a[0].

Nên phải ép kiểu pa = (int *)a thì mới lấy địa chỉ của a[0][0] (&arr[0][0]) gán cho con trỏ pa được.

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 2 chiều

Sử dụng con trỏ để truy xuất địa chỉ của các phần tử trong mảng 2 chiều.

Ví dụ 39: Truy xuất theo địa chỉ của những phần tử trong mảng 2 chiều.

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

int main() {
	int a[2][3]={{12, 27, 14},{39, 45, 52}};
	int *pa;
	pa = (int *)a;
	cout <<"Dia chi phan tu a[0][0]: " <<pa << endl;
	cout <<"Dia chi phan tu a[0][1]: " <<pa + 1 << endl;
	cout <<"Dia chi phan tu a[0][2]: " <<pa + 2 << endl;
	cout <<"Dia chi phan tu a[1][0]: " <<pa + 3 << endl;
	cout <<"Dia chi phan tu a[1][1]: " <<pa + 4 << endl;
	cout <<"Dia chi phan tu a[1][2]: " <<pa + 5 << endl;
	return 0;
}
```

Kết quả:

```
Dia chi phan tu a[0][0]: 0x78fe00
Dia chi phan tu a[0][1]: 0x78fe04
Dia chi phan tu a[0][2]: 0x78fe08
Dia chi phan tu a[1][0]: 0x78fe0c
Dia chi phan tu a[1][1]: 0x78fe10
Dia chi phan tu a[1][2]: 0x78fe14
```

4. Sự tương quan giữa con trỏ và mảng

1. Con trỏ và mảng 2 chiều

Sử dụng con trỏ để truy xuất giá trị của các phần tử trong mảng 2 chiều.

Ví dụ 40: Truy xuất giá trị của những phần tử trong mảng 2 chiều.

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

int main() {
	int a[2][3]={{12, 27, 14},{39, 45, 52}};
	int *pa;
	pa = (int *)a;
	cout <<"Gia tri phan tu a[0][0]: " <<*pa << endl;
	cout <<"Gia tri phan tu a[0][1]: " <<*(pa + 1) << endl;
	cout <<"Dia chi phan tu a[0][2]: " <<*(pa + 2) << endl;
	cout <<"Dia chi phan tu a[1][0]: " <<*(pa + 3) << endl;
	cout <<"Dia chi phan tu a[1][1]: " <<*(pa + 4) << endl;
	cout <<"Dia chi phan tu a[1][2]: " <<*(pa + 5) << endl;
	return 0;
}
```

Kết quả:

```
Gia tri phan tu a[0][0]: 12
Gia tri phan tu a[0][1]: 27
Dia chi phan tu a[0][2]: 14
Dia chi phan tu a[1][0]: 39
Dia chi phan tu a[1][1]: 45
Dia chi phan tu a[1][2]: 52
```