Giới thiệu tổng quan về lập trình

Nhập môn lập trình

Trình bày: …; Email: …@fit.hcmus.edu.vn

Nội dung

- Dữ liệu có cấu trúc
- Dữ liệu mảng với kích thước cố định
- Ứng dụng mảng trong lập trình
- Các vấn đề tìm hiểu mở rộng kiến thức nghề nghiệp
- Thuật ngữ và bài đọc thêm tiếng Anh

Dữ liệu có cấu trúc

Đặt vấn đề

- Khai báo các biến để lưu trữ 1 SV

```
char mssv[8];          // "0912345"
char hoten[30];        // "Nguyen Van A"
char ntns[9];          // "01/01/91"
char phai;             // 'n'
float toan, ly, hoa;   // 8.5 9.0 10.0
```

- Truyền thông tin 1 SV cho hàm

```
void xuat(char* mssv, char* hoten, char* ntns, char phai,
                                    float toan, float ly, float hoa);
```

Đặt vấn đề

- Nhận xét
  - Đặt tên biến khó khăn và khó quản lý
  - Truyền tham số cho hàm quá nhiều
  - Tìm kiếm, sắp xếp, sao chép,… khó khăn
  - Tốn nhiều bộ nhớ
  - …
- Ý tưởng
  - Gom những thông tin của cùng 1 SV thành một kiểu dữ liệu mới => Kiểu struct

Khai báo kiểu cấu trúc

- Cú pháp

```
struct <tên kiểu cấu trúc> {
    <kiểu dữ liệu> <tên thành phần 1>;
    …
    <kiểu dữ liệu> <tên thành phần n>;
};
```

- Ví dụ

```
struct Point2D {
    int x;
    int y;
};
```

Khai báo biến

- Cú pháp khai báo tường minh

```
struct <tên kiểu cấu trúc> {
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
} <tên biến 1>, <tên biến 2>;
```

- Ví dụ

```
struct Point2D {
    int x;
    int y;
} p1, p2;
```

Khai báo biến

- Cú pháp khai báo không tường minh

```
struct <tên kiểu cấu trúc> {
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
};
struct <tên kiểu cấu trúc> <tên biến 1>, <tên biến 2>;
```

- Ví dụ

```
struct Point2D {
    int x;
    int y;
};
struct Point2D p1, p2;      // C++ có thể bỏ từ khóa struct
```

Sử dụng typedef

- Cú pháp

```
typedef struct {
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
} <tên kiểu cấu trúc>;
<tên kiểu cấu trúc> <tên biến 1>, <tên biến 2>;
```

- Ví dụ

```
tyepdef struct {
    int x;
    int y;
} Point2D;
Point2D p1, p2;
```

Khởi tạo cho biến cấu trúc

- Cú pháp

```
struct <tên kiểu cấu trúc> {
        <kiểu dữ liệu> <tên thành phần 1>;
        …
        <kiểu dữ liệu> <tên thành phần n>;
} <tên biến> = {<giá trị 1>, <giá trị 2>, …, <giá trị n>};
```

- Ví dụ

```
struct Point2D {
        int x;
        int y;
} p1= {2912, 1706}, p2;
```

Truy xuất

- Đặc điểm
  - Không thể truy xuất trực tiếp.
  - Thông qua toán tử thành phần cấu trúc . Hay còn gọi là toán tử chấm (dot operation).

<tên biến cấu trúc>.<tên thàn phần>

- Ví dụ

```
struct Point2D {
      int x, y;
} p = {2912, 1706};
void show(Point2D p) { printf("x = %d, y = %d\n", p.x, py); }
```

Gán dữ liệu

- Có 2 cách

<biến cấu trúc đích> = biến cấu trúc nguồn

<biến cấu trúc đích>.<tên thành phần> = <giá trị>

- Ví dụ

```
struct Point2D {
	int x, y;
} p1 = {2912, 1706}, p2;
void main() {
	p2 = p1;
	p2.x = p1.x;
	p2.y = p1.y * 2;
}
```

Ví dụ tìm trọng tâm tam giác

- Các khai báo cần thiết

```
#include <iostream>
using namespace std;
typedef struct {
	double x, y;
} Point2D;
typedef struct {
	Point2D ver[3];
} Triangle;
void inputPoint2D(Point2D& p);
void showPoint2D(Point2D p);
void gravCenter(Triangle t, Point2D& p);
void inputTriangle(Triangle& t);
```

Ví dụ tìm trọng tâm tam giác

- Các định nghĩa hàm

```
void inputPoint2D(Point2D& p) {
	cout << " + Coor X = "; cin >> p.x;
	cout << " + Coor Y = "; cin >> p.y;
}
void showPoint2D(Point2D p) {
	cout << "(" << p.x << ", " << p.y << "(";
}
void gravCenter(Triangle t, Point2D& p) {
	p.x = (t.ver[0].x + t.ver[1].x + t.ver[2].x) / 3;
	p.y = (t.ver[0].y + t.ver[1].y + t.ver[2].y) / 3;
}
```

Ví dụ tìm trọng tâm tam giác

- Các định nghĩa hàm

```
void inputTriangle(Point2D& p) {
    for (int i  = 0; i < 3; i++) {
        cout << "Vertex " << i + 1 << ": " << endl;
        inputPoint2D(t.ver[i]);
    }
}
void main() {
    Triangle t; Point2D p;
    inputTriangle(t);
    gravCenter(t, p);
    cout << "Gravity center: ";
    showPoint2D(p);
}
```

Ví dụ về phân số

- Các khai báo cần thiết

```
#include <iostream>
using namespace std;
typedef struct {
      long num, denom;
} Fraction;
void greatestDivisor(long a, long b);
void reduce(Fraction& p);
Fraction add(Fraction p, Fraction q);
Fraction sub(Fraction p, Fraction q);
void showFraction(Fraction p);
```

Ví dụ về phân số

- Các định nghĩa hàm

```
void greatestDivisor(long a, long b) {
    // Viết như các ví dụ trước…
}
void reduce(Fraction& p) {
    long gcd = greatestDivisor(p.num, p.denom);
    p.num /= gcd; p.denom /= gcd;
}
Fraction add(Fraction p, Fraction q) {
    Fraction r;
    r.num = p.num * q.denom + p.denom * q.num;
    r.denom = p.denom * q.denom;
    return r;
}
```

Ví dụ về phân số

- Các định nghĩa hàm

```
Fraction sub(Fraction p, Fraction q) {
        q.num = -q.num;
        return add(p, q);
}
void showFraction(Fraction p) {
        reduce(p);    // Tối giản trước khi in ra
        cout << p.num << "/" << p.denom;
}
```

Dữ liệu mảng với kích thước cố định

Dữ liệu kiểu mảng

- Khái niệm
  - Là một kiểu dữ liệu có cấu trúc do người lập trình định nghĩa.
  - Biểu diễn một dãy các biến có cùng kiểu. Ví dụ: dãy các số nguyên, dãy các ký tự…
  - Kích thước được xác định ngay khi khai báo và không bao giờ thay đổi.
  - NNLT C luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng.

Khai báo biến mảng 1 chiều

- Cú pháp tường minh

<kiểu cơ sở> <tên biến mảng>[<số phần tử>];

- Ví dụ

```
int a[100], b[200], c[100];
float d[50];
```

- Lưu ý
  - Phải xác định <số phần tử> cụ thể (hằng) khi khai báo.
  - Bộ nhớ sử dụng = <tổng số phần tử> * sizeof(<kiểu cơ sở>)
  - Là một dãy liên tục có chỉ số từ 0 đến <tổng số phần tử> - 1

Khai báo biến mảng 1 chiều

- Cú pháp (không tường minh)

typedef <kiểu cơ sở> <tên kiểu mảng>[<số lượng phần tử>];
<tên kiểu mảng> <tên biến mảng>;

- Ví dụ

```
typedef int Arr100int[100];
typedef int Arr200int[200];
typedef float Arr50float[50];
Arr100int a, c;      // int a[100], c[100];
Arr200int b;         // int b[200];
Arr50float d;        // float d[50];
```

Khởi tạo mảng 1 chiều

- Sử dụng một trong 4 cách sau:
  - Khởi tạo giá trị cho mọi phần tử của mảng

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

  - Khởi tạo giá trị cho một số phần tử đầu mảng

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

  - Khởi tạo giá trị 0 cho mọi phần tử của mảng

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

  - Tự động xác định số lượng phần tử

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

Truy xuất mảng 1 chiều

- Thông qua chỉ số:

<tên biến mảng>[<chỉ số>]

- Ví dụ cho mảng int a[4];
  - Các truy xuất hợp lệ: a[0], a[1], a[2], a[3]
  - Các truy xuất không hợp lệ: a[-1], a[4], a[5]

Gán dữ liệu mảng 1 chiều

- Không được sử dụng phép gán thông thường mà phải gán trực tiếp giữa các phần tử tương ứng
- Ví dụ

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

void main() {
	b = a;		// sai
	for (int i = 0; i < 3; i++)
		b[i] = a[i];
}
```

Truyền mảng 1 chiều cho hàm

- Tham số kiểu mảng truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng:
  - Có thể bỏ số lượng phần tử (hoặc sử dụng con trỏ), số lượng phần tử thực sự truyền kèm theo.
  - Mảng có thể thay đổi nội dung sau khi thực hiện hàm.
- Ví dụ

```
void sort(int a[], int n);
```

Xử lý mảng 1 chiều

- Một số thao tác cơ bản
  - Nhập/xuất mảng
  - Tìm kiếm một phần tử trong mảng
  - Kiểm tra tính chất của mảng
  - Chia/gộp mảng
  - Tìm giá trị nhỏ nhất/lớn nhất trong mảng
  - Sắp xếp mảng
  - Thêm/xóa/sửa một phần tử trong mảng

Mảng 2 chiều

- Mảng 2 chiều giống như một ma trận gồm nhiều dòng và nhiều cột giao nhau tạo thành các ô, mỗi ô là một phần tử mảng.
- Mọi thao tác xử lý trên mảng 2 chiều hoàn toàn tương tự trên mảng 1 chiều.
- Tạm thời giới hạn trong phạm vi mảng 2 chiều tĩnh (số dòng và cột cố định).

(Xem trong giáo trình NMLT trang 203-221)

Ứng dụng mảng trong lập trình

Một số ứng dụng

- Kỹ thuật dùng bảng tra cứu trong bộ nhớ để cải tiến tính toán và xử lý.
- Kỹ thuật dùng cờ hiệu khi xử lý mảng.
- Thuật toán tìm kiếm và tính toán trên mảng.
- Thuật toán xáo trộn, sắp xếp các phần tử của mảng.

Các vấn đề tìm hiểu mở rộng kiến thức nghề nghiệp

Tìm hiểu thêm

- Sử dụng mảng kích thước biến động.
- Qui hoạch động và ứng dụng để giải các bài toán tối ưu.
- Các thuật toán chia để trị.

Thuật ngữ và bài đọc thêm tiếng Anh

Thuật ngữ tiếng Anh

- ***array parameter(s), array argument(s)***: tham số mảng
- ***array size***: kích thước mảng
- ***column***: cột
- ***copy***: sao chép
- ***data type declaration, data type definition***: khai báo kiểu dữ liệu
- ***dynamic array***: mảng động
- ***element***: phần tử
- ***implementation***: cài đặt (viết mã nguồn)
- ***index***: chỉ số
- ***insert***: chèn vào
- ***one-dimension array***: mảng một chiều
- ***two-dimension array***: mảng hai chiều
- ***merge***: trộn lại

Thuật ngữ tiếng Anh

- ***remove, delete***: xóa đi
- ***row***: dòng
- ***split***: tách ra
- ***static array***: mảng tĩnh
- ***structured data***: dữ liệu có cấu trúc nói chung

Bài đọc thêm tiếng Anh

- **Thinking in C**, Bruce Eckel, E-book, 2006.
- **Theory and Problems of Fundamentals of Computing with C++**, John R.Hubbard, Schaum's Outlines Series, McGraw-Hill, 1998.