Ngôn ngữ lập trình C

Bài 8. Cấu trúc và Nhóm

Soạn bởi: TS. Nguyễn Bá Ngọc

2021

Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Nhập, xuất với kiểu cấu trúc và kiểu nhóm
- Hàm với kiểu cấu trúc và kiểu nhóm

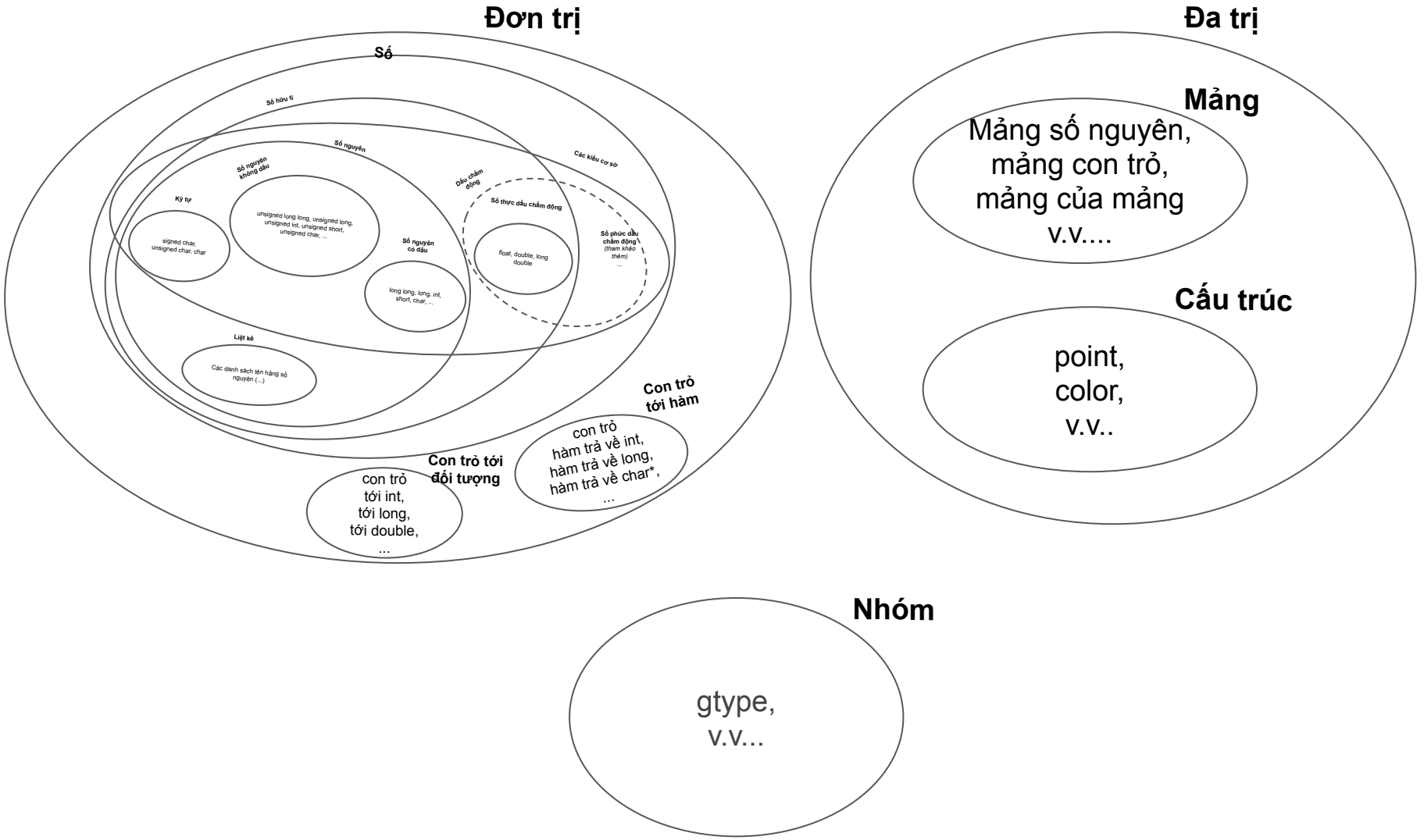
Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Nhập, xuất với kiểu cấu trúc và kiểu nhóm
- Hàm với kiểu cấu trúc và kiểu nhóm

Kiểu cấu trúc và kiểu nhóm

- Kiểu cấu trúc mô tả một *tập không rỗng* các biến thành viên được *cấp phát tuần tự* trong bộ nhớ. Mỗi thành viên có thể có kiểu riêng.
  - Ví dụ: struct point {float x; float y;};
    - x và y là các thành viên của point; y được cấp phát sau x.
  - struct empty {};
    - empty là cấu trúc rỗng, không hợp lệ trong quy chuẩn ISO, nhưng hợp lệ trong mở rộng GNU (với GCC).
- Kiểu nhóm mô tả một *tập không rỗng* các biến thành viên *chồng lấn bộ nhớ*, mỗi thành viên có thể có kiểu riêng.
  - Ví dụ: union gtype { long l; double d; };
    - l và d là các thành viên của gtype, có cùng địa chỉ và chồng lấn bộ nhớ.
  - union none {};
    - none là nhóm rỗng, không hợp lệ trong quy chuẩn ISO, nhưng hợp lệ trong mở rộng GNU (với GCC).

Một số (lớp) kiểu dữ liệu trong C99



Kích thước của cấu trúc và nhóm

- Kích thước cấu trúc >= Tổng kích thước các thành viên.
  - struct s1 {int x; char c; int y; };
    - c được cấp phát sau x, y được cấp phát sau c.
    - sizeof(s1) > sizeof(float) + sizeof(char) + sizeof(int);
    - *(Có thể có căn chỉnh)*
  - struct point3 {float x; float y; float z; }
    - sizeof(s2) == 3 * sizeof(float);
    - *(Có thể không có căn chỉnh)*
- Kích thước nhóm >= Kích thước thành viên lớn nhất.
  - union g1 { float x; double d; };
    - sizeof(g1) == sizeof(double);
    - *(Có thể không có căn chỉnh)*
  - union g2 { struct point3 p; double d; };
    - sizeof(g2) > sizeof(point3);
    - *(Có thể có căn chỉnh)*

Có thể kiểm tra vị trí bắt đầu của các thành viên (và phát hiện căn chỉnh) bằng macro offsetof (stddef.h)

Ví dụ 8.1. Kích thước của cấu trúc và nhóm

vd8-1.c

```c
#include <stdio.h>
#include <stddef.h>

struct s1 { int x; char c; int y; };
struct s2 { float x, y, z; };
union g1 { float x; double d; };
union g2 { struct s2 p; double d; };

int main() {
  printf("sizeof:\nchar: %zu, int: %zu, float: %zu, double: %zu\n",
         sizeof(char), sizeof(int), sizeof(float), sizeof(double));
  printf("sizeof(s1) = %zu\n", sizeof(struct s1));
  printf("offsetof(s1, c) = %zu\n", offsetof(struct s1, c));
  printf("offsetof(s1, y) = %zu\n", offsetof(struct s1, y));
  printf("sizeof(s2) = %zu\n", sizeof(struct s2));
  printf("sizeof(g1) = %zu\n", sizeof(union g1));
  printf("sizeof(g2) = %zu\n", sizeof(union g2));
  return 0;
}
```

```
bangoc:$gcc -o prog vd8-1.c
bangoc:$./prog
sizeof:
char: 1, int: 4, float: 4, double: 8
sizeof(s1) = 12
offsetof(s1, c) = 4
offsetof(s1, y) = 8
sizeof(s2) = 12
sizeof(g1) = 8
sizeof(g2) = 16
bangoc:$
```

Có thể thấy có 3 bytes căn chỉnh trước thành phần y và sau thành phần c trong s1.

Cú pháp khai báo cấu trúc và nhóm

- Sơ lược cú pháp định nghĩa:
  - **Cấu trúc:** struct tên-cấu-trúc { /* danh sách thành viên */ };
  - **Nhóm:** union tên-nhóm { /* danh sách thành viên */};
- Cấu trúc và nhóm không thể có thành viên là chính nó nhưng có thể có thành viên là con trỏ tới chính nó:
  - Ví dụ: struct node { long data; struct node *next; }; // Ok
  - struct node {long data; struct node nd; }; // NOK - Định nghĩa node chưa hoàn thiện ở vị trí khai báo nd.
- Khai báo biến:
  - Cấu trúc: struct tên-cấu-trúc tên-biến;
    - struct point {double x, y; } p0, *pp1; // p0, p1,p2 là các đối tượng point
    - struct point p1, p2;
    - struct point *pp2; // pp1, pp2 là các con trỏ tới point
  - Nhóm: union tên-nhóm tên-biến;
    - union gtype { long l; double d; } g0, *gp1; // g0, g1, g2 là các đối tượng gtype
    - union gtype g1, g2;
    - union gtype *gp2; // gp1, gp2 là các con trỏ tới gtype

Định nghĩa với typedef

- Theo khai báo các biến
  - struct point {double x, y; } p, *pp;
    - p là đối tượng point, pp là con trỏ tới point.
  - union gtype {long l; double d;} g, *pg;
- Định nghĩa các kiểu cấu trúc với typedef
  - typedef struct point {double x, y; } point_s, *point_ptr;
    - point_s là kiểu cấu trúc point.
    - point_ptr là kiểu con trỏ tới cấu trúc point.
    - struct point p1; // p1 là đối tượng point
    - point_s p2; // Không yêu cầu từ khóa struct, p2 là đối tượng point
    - point_ptr pp; // pp là con trỏ tới point
- Định nghĩa kiểu nhóm với typedef:
  - typedef union gtype { long l; double d; } gtype_u, *gtype_ptr;
    - union gtype g1; // g1, g2 là các đối tượng union
    - gtype_u g2; // không yêu cầu từ khóa union
    - gtype_ptr pp; // pp là con trỏ tới gtype
    - *(Cú pháp tương tự như với struct)*

Khởi tạo cấu trúc và nhóm

- Có thể khởi tạo đối tượng cấu trúc và đối tượng nhóm bằng đối tượng đã có, danh sách khởi tạo, hoặc hằng giá trị:
- Khởi tạo bằng đối tượng đã có:
  - *(// struct point p; p.x = 10; p.y = 20;)*
  - *(// union gtype g; g.i = 100;)*
  - struct point p1 = p; // Khởi tạo đối tượng cấu trúc
  - union gtype g1 = g; // Khởi tạo đối tượng nhóm
- Khởi tạo với danh sách khởi tạo:
  - struct point p = {20, 30};
    - Các thành viên được khởi tạo theo thứ tự tuần tự tương ứng với các giá trị trong danh sách khởi tạo.
    - p.x = 20; p.y = 30;
    - Các thành viên còn lại (không có giá trị tương ứng trong danh sách khởi tạo), được khởi tạo với giá trị mặc định tương tự biến toàn cục.
  - union gtype g = {.i = 101};
    - Khởi tạo thành viên được chỉ định: g.i = 101;

Hằng cấu trúc và nhóm

- Biểu thức khởi tạo có thể được sử dụng để tạo các đối tượng không tên:
  - Cấu trúc: (struct tên-cấu-trúc){ /* danh sách khởi tạo */}
    - Ví dụ:
    - ((struct point){.x = 100, .y = 200});
    - (struct point){200, 300};
  - Nhóm không tên: (union tên-nhóm){ /* biểu thức khởi tạo */}
    - Ví dụ:
    - ((union gtype) {.i = 1001});
    - (union gtype){.d = 3.1415};

Có vai trò như hằng (đa trị) cho kiểu cấu trúc và kiểu nhóm.

Mảng cấu trúc và nhóm

- Tương tự như với các kiểu dữ liệu khác chúng ta cũng có thể tạo mảng của cấu trúc và nhóm:
  - *(Dựa trên khai báo biến, ví dụ* struct point p;*)*
  - struct point points[100]; // Mảng 100 cấu trúc point
  - union gtype values[100]; // Mảng 100 nhóm gtype
  - struct point points2[] = {{100, 200}, {200, 300}};
  - struct point triangles[][3] = {
    { {100, 100}, {200, 200}, {300, 300} },
    }; // => triangles[1][3]
  - points[i]; // Phần tử thứ i trong points
  - struct point *pp = points;
  - *(pp + i); // Phần tử thứ i trong points

Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Nhập, xuất với kiểu cấu trúc và kiểu nhóm
- Hàm với kiểu cấu trúc và kiểu nhóm

Toán tử thành viên

Được sử dụng để chỉ định thành viên của cấu trúc và nhóm

- Chỉ định thành viên với tên đối tượng: .
- Chỉ định thành viên với con trỏ tới đối tượng: ->
- Ví dụ:
  - struct point p1, *pp;
  - /* … */
  - p1.x và p1.y là các thành viên x và y của p1.
  - pp->x và pp->y là các thành viên x và y của đối tượng đang được trỏ tới bởi pp, tương đương với (*pp).x và (*pp).y
  - struct line { struct point p1, p2; int color; } l;
  - l.p1.x và l.p2.x lần lượt là thành viên x của thành viên p1 và thành viên p2 của l.

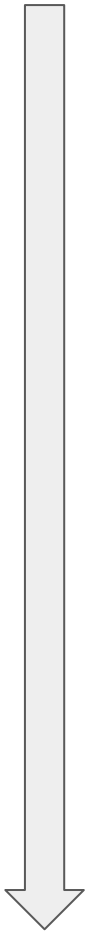
Chuỗi toán tử thành viên được thực hiện từ trái sang phải

Toán tử gán với kiểu cấu trúc và kiểu nhóm

- Chúng ta có thể sử dụng phép gán với các toán hạng là các cấu trúc cùng kiểu hoặc các nhóm cùng kiểu.
- Gán đối tượng cấu trúc: Với biểu thức gán giá trị p1 = p2; p1 có giá trị tương tự như gán tuần tự các thành viên của p1 = thành viên tương ứng của p2. Kết quả biểu thức là p1.
  - Ví dụ: struct point p1, p2;
  - p1 = p2; // *Tương đương với* p1.x = p2.x; p1.y = p2.y;
  - int a[] = {1, 2, 3}, n = 100;
  - struct array {int n, *elem; } arr1 = {n, a};
  - struct array arr2; arr2 = arr1;
    - Cả arr1.elem và arr2.elem cùng trỏ tới 1 vùng nhớ của mảng a.
    - a[1] = 200; => arr1.elem[1] và arr2.elem[1] đều == 200.
- Gán đối tượng nhóm: Với biểu thức g1 = g2; g1 có giá trị giống như g1, kết quả biểu thức là g1.
  - Ví dụ: union gtype g1 = {.i = 100}, g2; g2 = g1; // g2.i == 100

Thứ tự ưu tiên và chiều thực hiện chuỗi toán tử

Ưu tiên cao
(thực hiện trước)



Ưu tiên thấp
(thực hiện sau)

Tên toán tử	Ký hiệu	Thứ tự
Chỉ số Gọi hàm Thành viên Tăng 1, giảm 1 (hậu tố)	[] a[10] () f(3, 5) ., -> p.x, pp->x ++, -- (x++, x--)	Trái -> Phải
Địa chỉ, chỉ định Tăng, giảm 1 (tiền tố) Dấu, phủ định lô-gic sizeof	&, * (&x, *p) ++, -- (++x, --x) +, -, ! (+x, -x, !e) sizeof (sizeof(int))	Phải -> Trái
Ép kiểu	() (double)5	Phải -> Trái
Nhân, Chia, phần dư	*, /, % (x * y, x / y, x%y)	Trái -> Phải
Cộng Trừ	+ (x + y) - (x - y)	Trái -> Phải
Dịch sang trái Dịch sang phải	<< (x << y) >> (x >> y)	Trái -> Phải
So sánh thứ tự	<, >, <=, >=	Trái -> Phải
So sánh bằng	==, !=	Trái -> Phải
AND theo bit	& (x & y)	Trái -> Phải
XOR theo bit	^ (x ^ y)	Trái -> Phải
OR theo bit	| (x | y)	Trái -> Phải
AND lô-gic	&&	Trái -> Phải
OR lô-gic	||	Trái -> Phải
Lựa chọn	?:	Phải -> Trái (Rẽ nhánh)
Gán đơn giản & kết hợp	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=	Phải -> Trái

Kiểu cấu trúc, kiểu nhóm và 1 số toán tử

struct array {int n, *elem;} arr, *pa;

- &arr.n; // Được hiểu là (&arr).n hay &(arr.n) ?
  - Toán tử . được ưu tiên hơn toán tử &, vì vậy
  - &arr.x được hiểu là địa chỉ của thành viên x của arr, &(arr.n).
- arr.elem[10]; // (arr.elem)[10] hay arr.(elem[10])?
  - Toán tử . và [] có cùng mức ưu tiên, chuỗi toán tử được thực hiện từ trái sang phải.
  - Vì vậy arr.elem[10] tương đương với *((arr.elem) + 10));
- pa = &arr; pa->elem[10]; // (pa->elem)[10] hay lỗi cú pháp?
  - Thực hiện từ trái qua phải, tương đương với arr.elem[10];
- struct array arr1, arr2;
  - arr1 == arr2; // arr1.n == arr2.n && arr1.elem == arr2.elem hay đây là biểu thức không hợp lệ?

Không sử dụng được các toán tử so sánh (và một số toán tử khác) với kiểu cấu trúc và kiểu nhóm, tuy nhiên có thể xử lý bằng hàm.

Ép kiểu con trỏ tới cấu trúc và nhóm

- Chúng ta có thể ép kiểu con trỏ tới cấu trúc thành con trỏ tới phần tử đầu tiên của nó và ngược lại, ví dụ:
  - struct b {int x; float f; };
  - struct c {struct b b; double d;} c, *pc = &c;
  - struct b *pb = (struct b*)pc;
  - pb->x; // Tương đương với pc->b.x; (và c.b.x).
  - *(Chỉ đúng với thành viên đầu tiên của cấu trúc.)*
- Chúng ta có thể ép kiểu con trỏ tới nhóm thành con trỏ tới phần tử bất kỳ của nó và ngược lại, ví dụ:
  - union gtype {long l; double d; } g, *pg = &g;
  - long *pl = (long*)g; // Tương đương với pl = &g.l;
  - double *pd = (double*)g; // pd = &g.d;
  - *(Cho phép sử dụng kiểu nhóm thay thế cho 1 tập kiểu)*

Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Nhập, xuất với kiểu cấu trúc và kiểu nhóm
- Hàm với kiểu cấu trúc và kiểu nhóm

Nhập, xuất với kiểu cấu trúc và kiểu nhóm

Chúng ta có thể nhập, xuất (lần lượt từng) thành phần của cấu trúc, nhóm:

- Nhập xuất với kiểu cấu trúc:
  - Ví dụ: struct point {double x, y; } p;
  - scanf("%lf%lf", &p.x, &p.y); // Nhập các thành phần p.x, p.y
  - printf("x = %f, y = %f\n", p.x, p.y); // Xuất p.x và p.y
- Nhập xuất với kiểu nhóm:
  - Ví dụ: union gtype {long l; double d; } g;
  - scanf("%ld”, &g.l); // Sử dụng g như 1 biến kiểu long
  - printf("%ld", g.l);

Ví dụ 8.2. Nhập/xuất với cấu trúc và nhóm

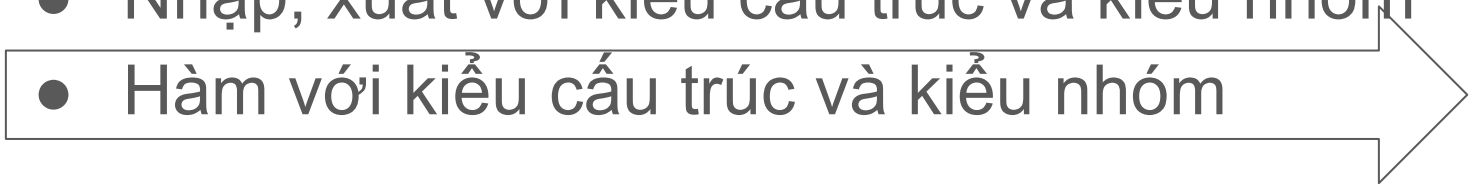
vd8-2.c

```c
#include <stdio.h>
struct point {float x; float y;};
union gtype {long l; double d;};
int main() {
    struct point p;
    union gtype g;
    printf("Nhập tọa độ 1 điểm: ");
    scanf("%f%f", &p.x, &p.y);
    printf("Bạn đã điểm: (%.3f, %.3f)\n", p.x, p.y);
    printf("Nhập 1 giá trị kiểu long: ");
    scanf("%ld", &g.l);
    printf("Bạn đã nhập %ld\n", g.l);
    return 0;
}
```

```
bangoc:$gcc -o prog vd8-2.c
bangoc:$./prog
Nhập tọa độ 1 điểm: 3 5
Bạn đã điểm: (3.000, 5.000)
Nhập 1 giá trị kiểu long: 101
Bạn đã nhập 101
bangoc:$
```

Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Nhập, xuất với kiểu cấu trúc và kiểu nhóm
- Hàm với kiểu cấu trúc và kiểu nhóm



Hàm, cấu trúc và nhóm

Hàm có thể nhận tham số là cấu trúc, nhóm và trả về cấu trúc, nhóm thông qua đó có thể triển khai các thao tác với cấu trúc và nhóm, trả về nhiều giá trị, v.v...

- Cấu trúc có thể là tham số và giá trị trả về của hàm, ví dụ:

```
struct point scale(struct point p0, float factor) {
    p0.x *= factor; p0.y *= factor;
    return p0;
}
```

- Nhóm có thể là tham số và giá trị trả về của hàm, ví dụ:

```
union gtype sum_d(union gtype g1, union gtype g2) {
    union gtype s = {.d = g1.d + g2.d};
    return s;
}
```

Ví dụ 8.3. Tính khoảng cách Euclide

vd8-3.c

```c
#include <stdio.h>
#include <math.h>
struct point {
  double x, y;
};
double distance(struct point *p1, struct point *p2) {
  double x = (p1->x - p2->x), y = (p1->y - p2->y);
  return sqrt(x*x + y*y);
}
int main() {
   struct point p1, p2;
   printf("Nhập tọa độ 2 điểm: ");
   scanf("%lf%lf%lf%lf", &p1.x, &p1.y, &p2.x, &p2.y);
   printf("Khoảng cách = %.3lf\n", distance(&p1, &p2));
   return 0;
}
```

```
bangoc:$./prog
Nhập tọa độ 2 điểm: 1 1 3 3
Khoảng cách = 2.828
bangoc:$
```

Con trỏ thường được sử dụng để truyền đối tượng cấu trúc cho hàm, đặc biệt là các cấu trúc có kích thước lớn.

Ví dụ 8.4. Xử lý mảng cấu trúc

Chương trình hỏi người dùng nhập n điểm và tìm 1 điểm bất kỳ gần gốc tọa độ nhất (theo khoảng cách Euclide)

vd8-4.c

```c
#include <stdio.h>
#include <math.h>
struct point {
  double x, y;
};
double distance(struct point *p1, struct point *p2) {
  double x = (p1->x - p2->x), y = (p1->y - p2->y);
  return sqrt(x*x + y*y);
}
#define N 100
int main() {
   int n, j = 0;
   printf("Nhập số n (<= %d) = ", N);
   scanf("%d", &n);

   // a[100], có thể cải tiến với mảng cấp phát động.
   struct point a[N];
   for (int i = 0; i < n; ++i) {
     printf("Nhập điểm %d: ", i);
     scanf("%lf%lf", &a[i].x, &a[i].y);
   }
   struct point p0 = {0, 0};
   double min = distance(&p0, &a[0]);
   for (int i = 1; i < n; ++i) {
     double dist = distance(&p0, &a[i]);
     if (dist < min) {
       min = dist;
       j = i;
     }
   }
   printf("Điểm gần nhất là: (%.3lf, %.3lf)\n",  a[j].x, a[j].y);
   return 0;
}
```

```
bangoc:$gcc -o prog vd8-4.c -lm
bangoc:$./prog
Nhập số n (<= 100) = 3
Nhập điểm 0: 3 3
Nhập điểm 1: 1 1
Nhập điểm 2: 0.5 1.5
Điểm gần nhất là: (1.000, 1.000)
bangoc:$
```

Ví dụ 8.5. Sử dụng nhóm như 1 tập kiểu

vd8-5.c

```c
#include <stdio.h>
typedef union {long l; double d;} gtype;
gtype sum_d(gtype g1, gtype g2) {
  return (gtype){.d = g1.d + g2.d};
}
gtype sum_l(gtype g1, gtype g2) {
  return (gtype){.l = g1.l + g2.l};
}
int main() {
  gtype g1, g2;
  printf("Nhập vào 2 số thực: ");
  scanf("%lf%lf", &g1.d, &g2.d);
  printf("%.3f + %.3f = %.3f\n", g1.d, g2.d, sum_d(g1, g2).d);
  printf("Nhập vào 2 số nguyên: ");
  scanf("%ld%ld", &g1.l, &g2.l);
  printf("%ld + %ld = %ld\n", g1.l, g2.l, sum_l(g1, g2).l);
  return 0;
}
```

```
bangoc:$gcc -o prog vd8-5.c
bangoc:$./prog
Nhập vào 2 số thực: 3.15 6.35
3.150 + 6.350 = 9.500
Nhập vào 2 số nguyên: 101 202
101 + 202 = 303
bangoc:$
```

Kiểu nhóm có thể thay thế 1 tập kiểu