

Ngôn ngữ lập trình C

Bài 9. Cấu trúc và nhóm

Soạn bởi: TS. Nguyễn Bá Ngọc

Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Xử lý cấu trúc và nhóm

Nội dung

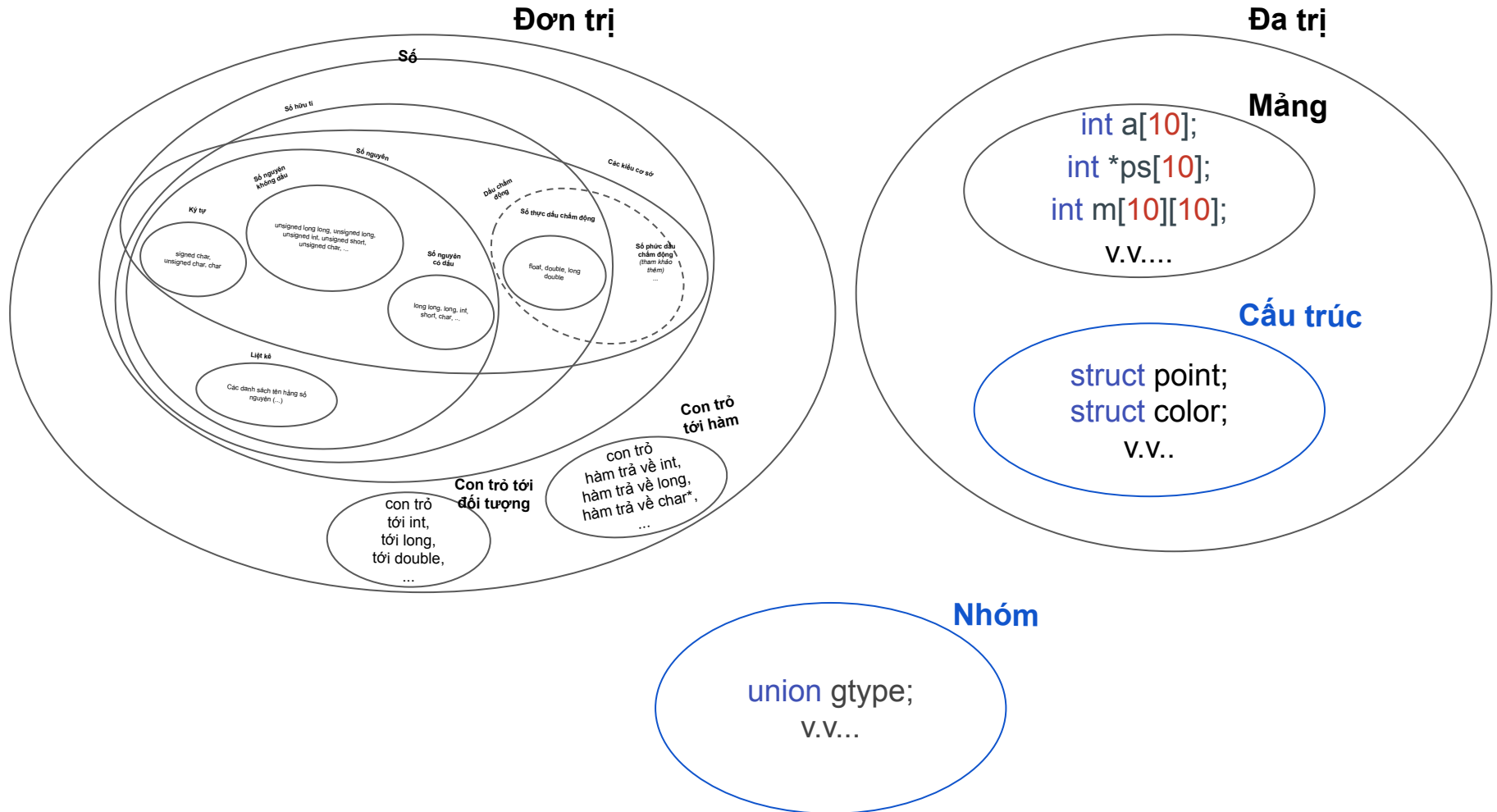
- Khái niệm & cú pháp
- Các toán tử
- Xử lý cấu trúc và nhóm

Các khái niệm

- Kiểu cấu trúc mô tả *1 danh sách không rỗng* các đối tượng thành viên được lưu trong các phân vùng nhớ ***tuần tự*** theo thứ tự khai báo (nhưng không bắt buộc phải liên tục như mảng), trong vùng nhớ của đối tượng cấu trúc. Các thành viên có thể có kiểu khác nhau.
- Kiểu nhóm mô tả *1 tập không rỗng* các đối tượng thành viên được lưu ở ***cùng 1 địa chỉ***, đó cũng là địa chỉ của đối tượng nhóm. Các thành viên có thể có kiểu khác nhau.
- Cấu trúc và nhóm có thể có thành viên là cấu trúc và thành viên là nhóm, nhưng không được có thành viên là chính nó.

Cấu trúc rỗng và nhóm rỗng (không chứa thành viên nào) không hợp lệ trong quy chuẩn ISO, nhưng hợp lệ trong mở rộng GNU (GCC).

Một số (lớp) kiểu dữ liệu trong C99



**Tuy có thể có nhiều thành viên nhưng nhóm không thuộc lớp đa trị bởi vì các thành viên chồng lấn bộ nhớ - một cách tương đối ở 1 thời điểm 1 đối tượng nhóm chỉ lưu được 1 giá trị của 1 thành viên.*

Sơ lược cú pháp định nghĩa kiểu

- Cấu trúc: **struct** định-danh { /* Danh sách thành viên */ };
- Nhóm: **union** định-danh { /* Danh sách thành viên */ };
- Ví dụ:

```
// Thành viên y được cấp  
phát sau thành viên x
```

```
struct xy {  
    long x;  
    double y;  
};
```

```
// Kiểu cấu trúc rỗng
```

```
struct empty {};
```

```
// Các thành viên x và y  
chồng lẫn bộ nhớ
```

```
union xy {  
    long x;  
    double y;  
};
```

```
// Kiểu nhóm rỗng
```

```
union empty {};
```

struct empty và **union** empty trong ví dụ không hợp lệ trong quy chuẩn ISO, nhưng hợp lệ trong mở rộng GNU (GCC).

Sơ lược cú pháp tạo đối tượng

- Khai báo biến cấu trúc:
 - Định dạng: **struct** tên-cấu-trúc tên-biến;
 - **struct point** p, q; // p và q là các (biến) cấu trúc, có kiểu struct point
 - **const struct point** cp, cq; // cp và cq là các cấu trúc chỉ đọc
- Khai báo biến nhóm (tương tự cấu trúc):
 - Định dạng: **union** tên-nhóm tên-biến;
 - **union** gtype a, b; // a và b là các (biến) nhóm, có kiểu union gtype
 - **const** gtype ca, cb; // ca và cb là các nhóm chỉ đọc
- Cũng có thể khai báo các biến khi định nghĩa kiểu:
 - **struct point** {**double** x, y; } p, q, *pp = &p;
 - pp là con trỏ và được khởi tạo trỏ tới p.
 - **union** gtype { **long** l; **double** d; } a, b, *pg = &a;
 - pg là con trỏ và được khởi tạo trỏ tới a.

*!Lưu ý: Các từ khóa **struct** và **union** được kết hợp với định danh để xác định kiểu cấu trúc và kiểu nhóm.*

Sơ lược cú pháp tạo con trỏ

Có thể áp dụng cơ chế suy diễn kiểu con trỏ tương tự như đã thực hiện với các kiểu đơn trị.

- `struct point *pp;`
 - pp là con trỏ tới cấu trúc point, có kiểu `struct point *`
- `const struct point *pcp;`
 - - con trỏ tới cấu trúc point chỉ đọc, có kiểu `const struct point *`
- `struct point * const cpp; // biến chỉ đọc kiểu struct point *`
 - - con trỏ chỉ đọc tới cấu trúc point, có kiểu `struct point *const`
- `union gtype *pg;`
 - - con trỏ tới nhóm gtype, pg có kiểu `union xy *`
- `const union gtype *pcg;`
 - - con trỏ tới nhóm gtype chỉ đọc, có kiểu `const union gtype *`
- `union gtype * const cpg; // biến chỉ đọc kiểu union gtype *`
 - - con trỏ chỉ đọc tới nhóm gtype, có kiểu `union gtype *const`

Sơ lược cú pháp tạo con trỏ₍₂₎

- Có thể tạo con trỏ tới cấu trúc và con trỏ tới nhóm trong ngữ cảnh không có các định nghĩa kiểu cấu trúc và kiểu nhóm tương ứng.
- Cấu trúc và nhóm có thể có thành viên là con trỏ tới chính nó:
 - `struct node { long data; struct node *next; }; // Ok`
 - `struct node {long data; struct node nd; }; // Lỗi - Kiểu struct node chưa hoàn thiện ở thời điểm khai báo nd.`
 - `union gtype { long l; double d; union gtype *pg; }; // Ok`

Khởi tạo cấu trúc và nhóm

Có thể khởi tạo cấu trúc và nhóm bằng danh sách khởi tạo hoặc 1 đối tượng khác

- Danh sách khởi tạo:

- `struct point p = {20, 30};` // Khởi tạo tuần tự
- `struct point p = {.x = 100};` // Khởi tạo thành viên x
- Nếu cấu trúc được khởi tạo 1 phần thì các thành viên còn lại được khởi tạo với giá trị mặc định tương tự biến toàn cục (= 0 theo kiểu của thành viên).
- `union gtype g = {.i = 101};` // Khởi tạo thành viên i của g

- Khởi tạo bằng đối tượng khác:

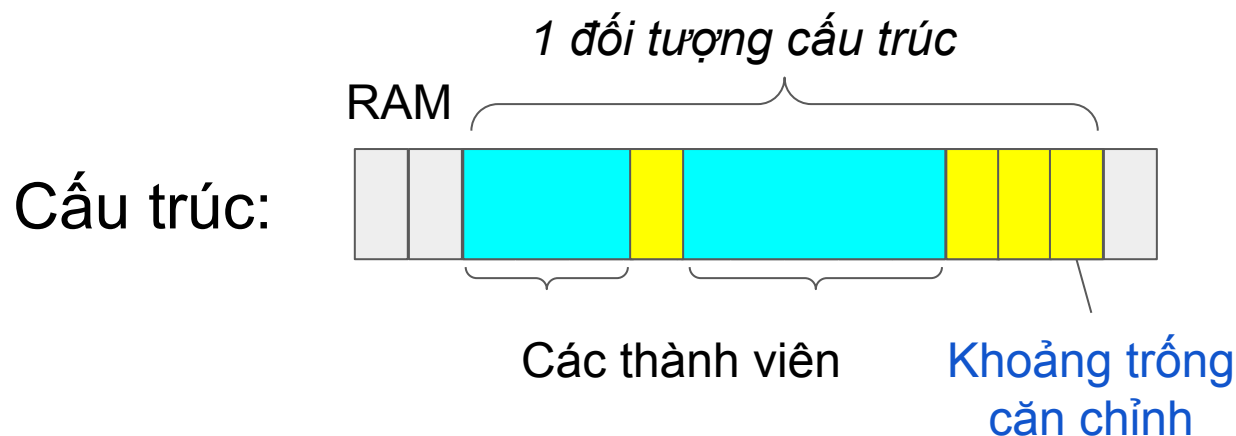
- `struct point p1 = p;` // Khởi tạo p1 giống như p
- `union gtype g1 = g;` // Khởi tạo g1 giống như g

Các giá trị gộp và đối tượng không tên

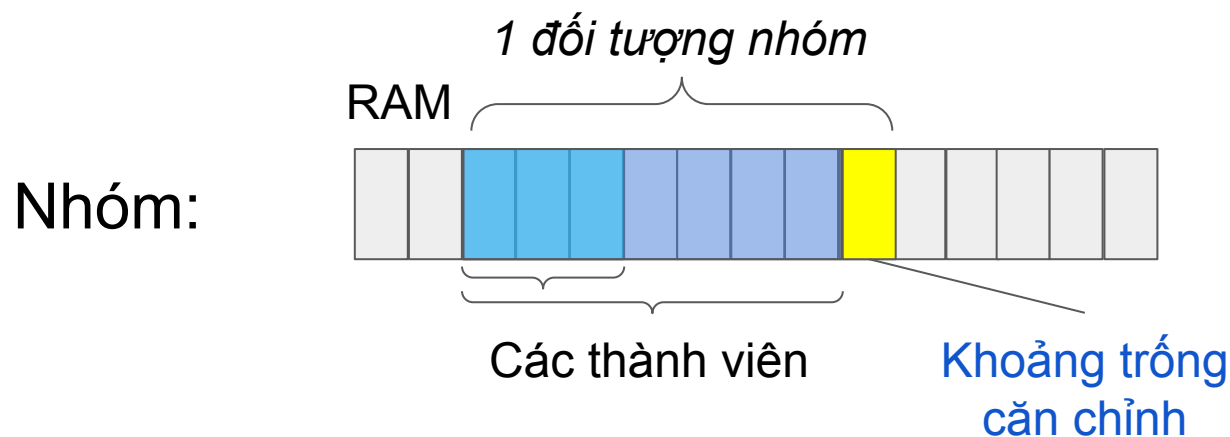
- Định dạng: (Kiểu-dữ-liệu){ */* danh sách khởi tạo */*}
 - (Thường sử dụng cho mảng, cấu trúc, và nhóm)
- Cấu trúc không tên:
 - (struct point){.x = 100, .y = 200}
 - (const struct point){10.0, 11.5} - Chỉ đọc
- Nhóm không tên:
 - (union gtype){.i = 1001}
 - (const union gtype){.d = 3.1415} - Chỉ đọc
- Mảng không tên:
 - (int []){1, 2, 3}

Các giá trị gộp (compound literals) có thể được lưu trong các đối tượng không tên, các đối tượng không tên chỉ đọc giống nhau có thể cùng là 1 đối tượng - được lưu trong 1 vùng nhớ.

Các mô hình bộ nhớ



Vùng nhớ của cấu trúc có thể có căn chỉnh ở giữa các thành viên hoặc ở cuối, nhưng không có căn chỉnh ở đầu.



Vùng nhớ của nhóm có thể có căn chỉnh ở cuối, nhưng không có ở đầu.

Hệ quả của mô hình bộ nhớ: Các kích thước

- Kích thước cấu trúc \geq Tổng kích thước các thành viên
 - `struct s1 { int x; char c; int y; };`
 - `sizeof(struct s1) \geq sizeof(int) + sizeof(char) + sizeof(int);`
 - `struct s2 { int x, y, z; };`
 - `sizeof(struct s2) \geq 3 * sizeof(int);`
 - (*> Nếu có căn chỉnh*)
- Kích thước nhóm \geq Kích thước thành viên lớn nhất
 - `union g1 { char s[9]; double d; };`
 - `sizeof(union g1) \geq 9;`
 - `union g2 { float f; double d; };`
 - `sizeof(union g2) \geq sizeof(double);`
 - (*> Nếu có căn chỉnh*)

Ở 1 thời điểm 1 đối tượng nhóm thường chỉ lưu được giá trị của 1 thành viên - lưu vào thành viên nào thì đọc từ thành viên đó

Có thể kiểm tra các khoảng trống dựa trên các kích thước và độ xê dịch của các thành viên tính từ địa chỉ của đối tượng chứa nó bằng macro `offsetof` (stddef.h)

Ví dụ 9.1. Kích thước của cấu trúc và nhóm

```
1  #include <stdio.h>
2  #include <stddef.h>
3
4  struct s1 { int x; char c; int y; };
5  struct s2 { int x, y, z; };
6  union g1 { char s[9]; double d; };
7  union g2 { float f; double d; };
8
9  int main() {
10     printf("sizeof(struct s1) = %zu\n", sizeof(struct s1));
11     printf("offset: %zu %zu %zu\n", offsetof(struct s1, x),
12         offsetof(struct s1, c), offsetof(struct s1, y));
13     printf("sizeof(struct s2) = %zu\n", sizeof(struct s2));
14     printf("sizeof(union g1) = %zu\n", sizeof(union g1));
15     printf("offset: %zu %zu\n", offsetof(union g1, s),
16         offsetof(union g1, d));
17     printf("sizeof(union g2) = %zu\n", sizeof(union g2));
18 }
```

sizeof(struct s1) = 12 offset: 0 4 8 sizeof(struct s2) = 12 sizeof(union g1) = 16 offset: 0 0 sizeof(union g2) = 8

Có thể nhận thấy thành phần căn chỉnh trong s1 (giữa c và y) và trong g1 (ở cuối)

Đặt tên kiểu

- Sơ lược cú pháp

- `typedef T D;`
 - Nếu T D khai báo ident là 1 đối tượng thì `typedef T D` khai báo ident là kiểu của ident trong T D.
- Sau đó có thể sử dụng tên kiểu để tạo đối tượng:
 - `ident x;`
 - x là 1 đối tượng có kiểu như ident trong T D;


- Ví dụ:

- `typedef struct point {double x, y; } point_s, *point_ptr;`
 - `point_s` - kiểu cấu trúc `point`, `point_ptr` - kiểu con trỏ tới cấu trúc `point`;
- `point_s p2; // tương tự struct point {double x, y;} p2;`
- `point_ptr pp; // - struct point {double x, y;} *pp;`
- `typedef union gtype { long l; double d; } gtype_u, *gtype_ptr;`
 - `gtype_u` - kiểu nhóm `gtype`, `gtype_ptr` - kiểu con trỏ tới nhóm `gtype`;
- `gtype_u g2; // tương tự union gtype { long l; double d; } g2;`
- `gtype_ptr pp; // - union gtype { long l; double d; } *pp;`

Mảng cấu trúc và nhóm

- Tương tự như đã làm với các kiểu đơn trị chúng ta cũng có thể tạo mảng của cấu trúc và mảng của nhóm:
 - *(Đối chiếu và so sánh với struct point p;)*
 - `struct point points[100];` // Mảng 100 cấu trúc point
 - `struct point points2[] = {{100, 200}, {200, 300}};` // 1 chiều
 - `struct point triangles[][3] = {`
 - `{ {100, 100}, {200, 200}, {300, 300} },`
 - `};` // 2 chiều, kích thước 1 x 3
 - `points[i];` // Phần tử thứ i trong points
 - `struct point *pp = points;` // pp trỏ tới phần tử đầu tiên
 - `*(pp + i);` // Phần tử thứ i trong points
 - `union gtype values[100];` // Mảng 100 nhóm gtype
 - V.V..

Nội dung

- Khái niệm & cú pháp
 - Các toán tử
 - Xử lý cấu trúc và nhóm
- 

Các toán tử thành viên

- Định dạng:
 - Toán tử chấm $.$: $s.x$
 - Toán tử mũi tên \rightarrow : $p \rightarrow x$
- Điều kiện:
 - Vế trái của toán tử $.$ phải là đối tượng (có kiểu) cấu trúc hoặc đối tượng nhóm, vế trái của toán tử \rightarrow phải là con trỏ tới đối tượng cấu trúc hoặc con trỏ tới đối tượng nhóm.
- Ý nghĩa:
 - Toán tử $.$ (chấm) được sử dụng để truy cập tới thành viên với tên như vế phải của cấu trúc hoặc nhóm ở vế trái.
 - Toán tử \rightarrow (mũi tên) được sử dụng để truy cập tới thành viên với tên như vế phải của cấu trúc hoặc nhóm được trỏ tới bởi con trỏ ở vế trái.

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

Ví dụ 9.2. Các thành viên

- `struct point { double x, y; } p, *pp = &p;`
 - `p.x`, `pp->x` đều là thành viên `x` của `p`, 1 biến kiểu `double`;
 - `p.y`, `pp->y` đều là thành viên `y` của `p`, 1 biến `double` khác.
- `struct line { struct point p1, p2; int color; } l;`
 - `l.p1.x` là thành viên `x` của thành viên `p1` của `l`;
 - `l.p2.y` là thành viên `y` của thành viên `p2` của `l`.
- `union gtype { char[8] s; struct point *pp;} g, *pg = &g;`
 - `g.s`, `pg->s` đều là thành viên `s` của `g`.
 - `g.s[1]`, `pg->s[1]` đều là ký tự với chỉ số 1 của thành viên `s`.
 - `g.pp->x`, `pg->pp->x` đều là thành viên `x` của đối tượng được trỏ tới bởi thành viên `pp` của `g`.

Toán tử gán với kiểu cấu trúc và kiểu nhóm

Biểu thức gán có hiệu ứng giống như sao chép dữ liệu trong vùng nhớ của đối tượng ở vế phải vào vùng nhớ của đối tượng ở vế trái, ví dụ:

- `struct xy s1, s2; /*...*/ s1 = s2;`
 - Có thể tạo hiệu ứng tương tự với `memcpy` (`string.h`)
 - `void *memcpy (void *dest, const void *src, size_t n);`
 - `memcpy(&s1, &s2, sizeof(struct xy));`
- `union xy u1, u2; /* ... */ u1 = u2;`
 - `memcpy(&u1, &u2, sizeof(union xy));`
- Có thể gán các đối tượng có thành viên là mảng (nhưng không gán được mảng cho mảng):
 - `struct arr10 { int elems[10]; } a1, a2;`
 - `a1 = a2; // memcpy(&a1, &a2, sizeof(struct arr10));`
 - `a1.elems = a2.elems; // Lỗi - không gán được mảng cho mảng`

Ví dụ 9.3. Biểu thức với nhiều toán tử

```
struct array {int n, *elems;} a, *pa = &a, b;
```

- `&a.n;` // Được hiểu là `(&a).n` hay `&(a.n)` ?
- `a.elems[10];` // `(a.elems)[10]` hay `a.(elems[10])`?
- `pa->elems[10];` // `(pa->elems)[10]` hay `pa->(elems[10])`?
- `a == b;` // Hợp lệ: `a.n == b.n && a.elems == b.elems` hay không hợp lệ?

Ví dụ 9.3. Biểu thức với nhiều toán tử₍₂₎

```
struct array {int n, *elems;} a, *pa = &a, b;
```

- `&a.n;` // Được hiểu là `(&a).n` hay `&(a.n)` ?
 - `&(a.n)` - địa chỉ của thành viên `n` của `a`, bởi vì toán tử hậu tố có thứ tự ưu tiên cao hơn toán tử tiền tố.
- `a.elems[10];` // `(a.elems)[10]` hay `a.(elems[10])`?
 - `(a.elems)[10]` - tương đương `*(a.elems + 10)`, bởi vì chuỗi toán tử hậu tố được thực hiện từ trái sang phải;
- `pa->elems[10];` // `(pa->elems)[10]` hay `pa->(elems[10])`?
 - `(pa->elems)[10]`
- `a == b;` // Hợp lệ: `a.n == b.n` && `a.elems == b.elems` hay không hợp lệ?
 - Không hợp lệ: Toán tử so sánh `==` chỉ áp dụng được với 1 số kiểu định sẵn như đã học, người dùng tự định nghĩa khái niệm `==` cho các đối tượng thuộc kiểu tự định nghĩa.

So sánh con trỏ tới thành viên

- Với

```
struct xy { long x; double y; } s;
```

```
void *ps = &s, *px = &s.x, *py = &s.y;
```

các quan hệ so sánh sau có kết quả đúng:

- `ps == px;` // Không có khoảng trống trước phần tử đầu tiên
- `px < py;` // Vì `y` được khai báo sau `x`.

- Với

```
union xy { long x; double y; } u;
```

```
void *pu = &u, *px = &u.x, *py = &u.y;
```

các quan hệ so sánh sau cho kết quả đúng:

- `px == py;` // Vì các thành viên của nhóm có cùng địa chỉ
- `px == pu;` // Không có khoảng trống ở đầu nhóm.

Các toán tử khác

Các kiểu cấu trúc và các kiểu nhóm được định nghĩa bởi người lập trình với ý nghĩa riêng của các giá trị. Vì vậy có nhiều toán tử không áp dụng được cho các kiểu do người dùng tự định nghĩa.

- Các toán tử sau không áp dụng được cho các đối tượng cấu trúc và các đối tượng nhóm:
 - Các toán tử đại số: Cộng (+), trừ (-), nhân (*), chia (/), các phép gán kết hợp, v.v..
 - Các toán tử so sánh: Lớn hơn (>), nhỏ hơn (<), bằng nhau (==) v.v., và nhiều phép toán so sánh liên quan khác.
 - V.v...

Có thể đóng gói các xử lý tương tự toán tử cho các đối tượng thuộc kiểu tự định nghĩa với hàm hoặc macro

Ép kiểu con trỏ

- 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 xy { double x, y; } s, *ps = &s;`
 - `double *px = (double*)ps; // Ok, px == &s.x;`
 - `ps = (struct xy*)px; // Ok, tmp == &s;`
 - Có thể lấy con trỏ tới đối tượng từ con trỏ tới thành viên không phải thành viên đầu tiên dựa trên `offsetof`, ví dụ:
 - `double *py = &s.y;`
 - `ps = (struct xy *)((char*)py - offsetof(struct xy, y)); // OK, ps == &s;`
- Đối với nhóm có thể ép kiểu con trỏ tới nhóm thành con trỏ tới thành viên bất kỳ của nó và ngược lại, ví dụ:
 - `union gtype {long l; double d; } g, *pg = &g;`
 - `long *pl = (long*)pg; // Tương đương với pl = &g.l;`
 - `double *pd = (double*)pg; // pd = &g.d;`
 - `pg = (union gtype *)pl; // OK - pg == &g`
 - `pg = (union gtype *)pd; // Ok - pg == &g`

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	Ví dụ	Chuỗi
Chỉ số Gọi hàm Thành viên Tăng, giảm 1 (hậu tố)	[] () ., -> ++, --	a[9] f(3, 5) o.x, pp->x x++, x--	⇒
Tăng, giảm 1 (tiền tố) Địa chỉ, truy cập Dấu (tiền tố) Đảo dãy bit, phủ định Dung lượng	++, -- &, * +, - ~, ! sizeof	++x, --x &x, *p +x, -x ~x, !x sizeof(int)	⇐
Ép kiểu	()	(double)n	⇐
Nhân, Chia, Phần dư	*, /, %	x * y, x / y, x % y	⇒
Cộng, trừ	+, -	x + y, x - y	⇒
Dịch trái, dịch phải	<<, >>	x << y, x >> y	⇒
So sánh thứ tự	<, >, <=, >=	x < y	⇒
So sánh bằng	==, !=	x == y	⇒
AND theo bit	&	x & y	⇒
XOR theo bit	^	x ^ y	⇒
OR theo bit		x y	⇒
AND lô-gic	&&	x && y	⇒
OR lô-gic		x y	⇒
Lựa chọn	?:	x? y: z	⇐
Gán đơn giản Gán kết hợp	= *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, =	x = y x *= y, x /= y, ...	⇐

Nội dung

- Khái niệm & cú pháp
 - Các toán tử
 - Xử lý cấu trúc và nhóm
- 

Nhập, xuất với kiểu cấu trúc và kiểu nhóm

- Có thể nhập, xuất từng thành viên của cấu trúc, nhóm
 - *(Có thể áp dụng đệ quy)*
- Ví dụ nhập và xuất cấu trúc:
 - `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
- Ví dụ tương tự với nhóm:
 - `union gtype {long l; double d; } g;`
 - `scanf("%ld", &g.l);` // Sử dụng g như 1 biến kiểu long
 - `printf("%ld", g.l);`

Hàm, cấu trúc và nhóm

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

- Ví dụ hàm với tham số và giá trị trả về là các cấu trúc:

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

- Ví dụ tương tự với nhóm:

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

Ví dụ 9.4. Hàm với tham số con trỏ cấu trúc

Tính khoảng cách Euclide giữa 2 điểm trên mặt phẳng

```
1  #include <stdio.h>
2  #include <math.h>
3  struct point {
4      double x, y; Con trỏ thường được sử dụng để truyền đối tượng cấu
5  }; trúc cho hàm, đặc biệt là các cấu trúc có kích thước lớn.
6  double distance(struct point *p1, struct point *p2)
7  {
8      double x = (p1->x - p2->x), y = (p1->y - p2->y);
9      return sqrt(x*x + y*y);
10 }
11 int main() {
12     struct point p1, p2;
13     printf("Nhập tọa độ 2 điểm: ");
14     scanf("%lf%lf%lf%lf",
15           &p1.x, &p1.y, &p2.x, &p2.y);
16     printf("Khoảng cách = %.3lf\n",
17           distance(&p1, &p2));
18 }
```

Nhập tọa độ 2 điểm: 1 1 3
3

Khoảng cách = 2.828

Ví dụ 9.5. Mạng cấu trúc

Nhập n điểm và tìm 1 điểm bất kỳ gần gốc tọa độ nhất

```
1  #include <stdio.h>
2  #include <math.h>
3  /* Định nghĩa point và distance */
4  int main() {
5      int n, j = 0;
6      scanf("%d", &n);
7      struct point a[N];
8      for (int i = 0; i < n; ++i) {
9          scanf("%lf%lf", &a[i].x, &a[i].y);
10     }
11     struct point p0 = {0, 0};
12     double min = distance(&p0, &a[0]);
13     for (int i = 1; i < n; ++i) {
14         double dist = distance(&p0, &a[i]);
15         if (dist < min) {
16             min = dist;
17             j = i;
18         }
19     }
20     printf("(%.3lf, %.3lf)\n", a[j].x, a[j].y);
21 }
```

5	
3	3
1	1
2	2
5	5
0.5	6
(1.000, 1.000)	

Ví dụ 9.6. Sử dụng nhóm như 1 tập kiểu

```
1  #include <stdio.h> gtype đại diện cho 1 tập kiểu
2  typedef union generic_type {
3      long l;
4      double d;
5  } gtype;
6  gtype sum_d(gtype g1, gtype g2) {
7      return (gtype){.d = g1.d + g2.d};
8  }
9  gtype sum_l(gtype g1, gtype g2) {
10     return (gtype){.l = g1.l + g2.l};
11 }
12 int main() {
13     gtype g1, g2;
14     printf("Nhập vào 2 số thực: ");
15     scanf("%lf%lf", &g1.d, &g2.d);
16     printf("%.3f + %.3f = %.3f\n",
17           g1.d, g2.d, sum_d(g1, g2).d);
18     printf("Nhập vào 2 số nguyên: ");
19     scanf("%ld%ld", &g1.l, &g2.l);
20     printf("%ld + %ld = %ld\n",
21           g1.l, g2.l, sum_l(g1, g2).l);
22 }
```

Nhập vào 2 số thực: 1.5 3.7
1.500 + 3.700 = 5.200
Nhập vào 2 số nguyên: 10 20
10 + 20 = 30

Ví dụ 9.7. Thêm thông tin kiểu cho đối tượng

Thử nghiệm giản lược định dạng hàm xuất dữ liệu

```
1  #include <stdio.h>
2  typedef union generic_type {
3      long l;
4      double d;
5  } gtype;
6  typedef struct tagged_gtype {
7      char tag;
8      gtype val;
9  } tgt;
10 void print_tgt(tgt x) {
11     if (x.tag == 'l') {
12         printf("%ld: long\n", x.val.l);
13     } else if (x.tag == 'd') {
14         printf("%g: double\n", x.val.d);
15     }
16 }
17 int main() {
18     tgt v1 = {'l', {.l = 10}},
19             v2 = {'d', {.d = 3.5}};
20     print_tgt(v1);
21     print_tgt(v2);
22 }
```

10: long 3.5: double

