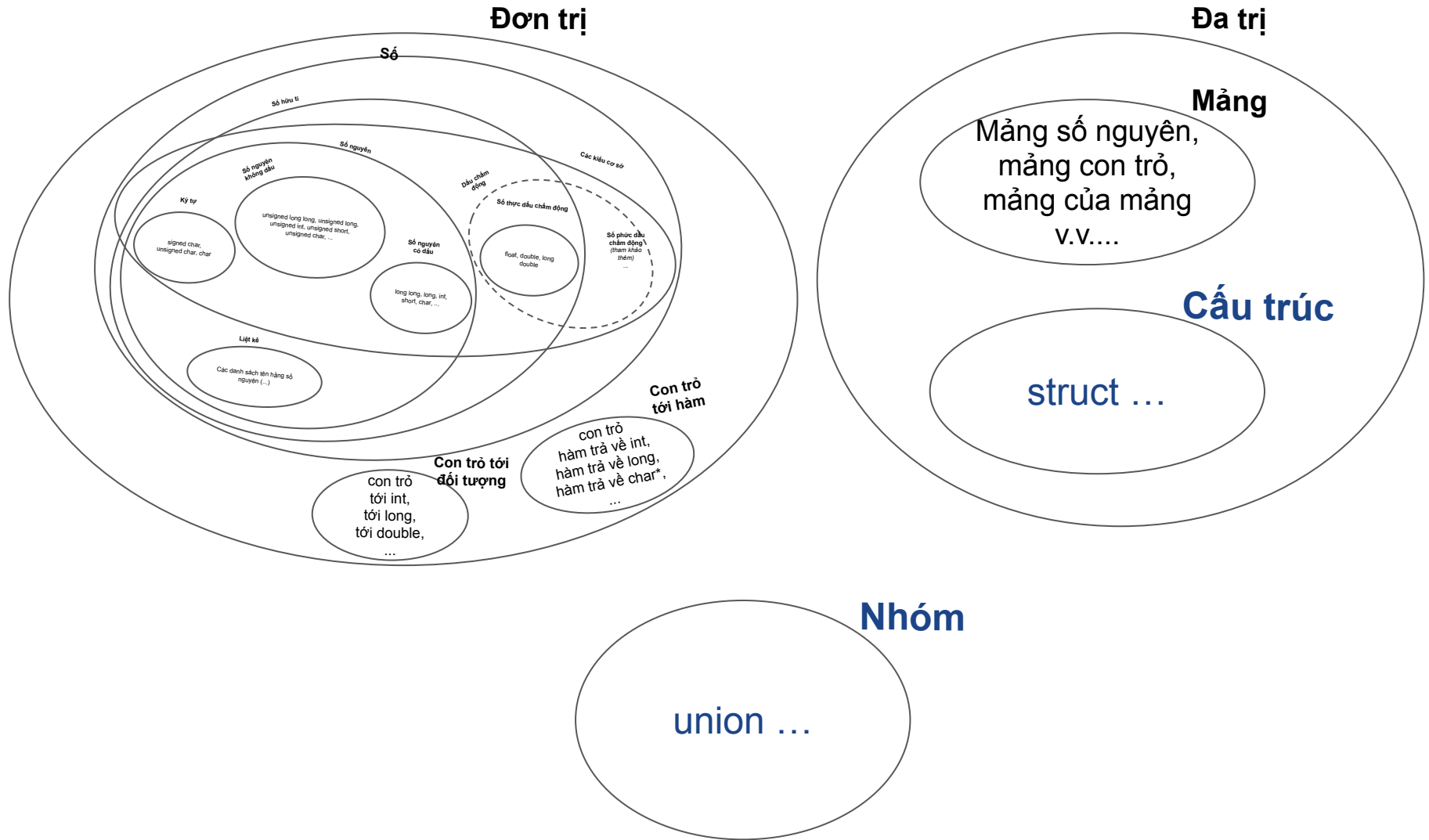


Ngôn ngữ lập trình C

Bài 8. Cấu trúc và Nhóm

Soạn bởi: TS. Nguyễn Bá Ngọc

Phân loại kiểu dữ liệu trong C99



Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Cấu trúc, nhóm, và hàm

Nội dung

- Khái niệm & cú pháp
- Các toán tử
- Cấu trúc, nhóm, và hàm

Các khái niệm

- Mỗi 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, trong vùng nhớ của đối tượng cấu trúc. Các thành viên có thể có kiểu khác nhau.
- Mỗi 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 rỗng và nhóm rỗng không hợp lệ trong quy chuẩn ISO, nhưng hợp lệ trong mở rộng GNU (GCC).

Cú pháp định nghĩa kiểu dữ liệu

- Định nghĩa kiểu cấu trúc:

struct định-danh { /* Các đối tượng thành viên */ };

- Định nghĩa kiểu nhóm:

union định-danh { /* Các đối tượng thành viên */};

- Ví dụ:

```
// Định nghĩa kiểu cấu trúc xy
struct xy {
    long x;
    double y;
};
// Kiểu cấu trúc rỗng
struct empt {};
```

```
// Định nghĩa kiểu nhóm xy
union xy {
    long x;
    double y;
};
// Kiểu nhóm rỗng
union empt {};
```

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

Cú pháp tạo đối tượng

- Ví dụ tạo đối tượng cấu trúc và đối tượng nhóm:
 - **struct xy** s1, s2; // s1 và s2 là các biến cấu trúc kiểu struct xy
 - **union xy** u1, u2; // u1 và u2 là các biến nhóm kiểu union xy
 - **const struct xy** cs1, cs2; // các biến cấu trúc chỉ đọc
 - **const union xy** us1, us2; // các biến nhóm chỉ đọc
- Biến cũng có thể được khai báo trong các mô tả kiểu, ví dụ:
 - **struct xy** {long x; double y;} s1, s2;
 - **union xy** {long x; double y;} u1, u2;

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

Cú pháp tạo con trỏ

Ví dụ tạo con trỏ tới kiểu cấu trúc và con trỏ tới nhóm:

- **struct xy *ps;**
 - ps là con trỏ tới cấu trúc xy, ps có kiểu struct xy * .
- **union xy *pu;**
 - pu là con trỏ tới nhóm xy, pu có kiểu union xy *
- **const struct xy *pcs;**
 - pcs là con trỏ tới cấu trúc chỉ đọc, có kiểu const struct xy*
- **const union xy *ucs;**
 - ucs là con trỏ tới nhóm chỉ đọc, có kiểu const union xy*
- **struct xy * const cps;**
 - cps là con trỏ chỉ đọc, có kiểu struct xy * const
- **union xy * const cpu;**
 - cpu là con trỏ chỉ đọc, có kiểu union xy * const

Cú pháp khởi tạo

- Có thể khởi tạo bằng đối tượng đã có, ví dụ:
 - `struct xy s = s1; // Tương đương: struct xy s; s = s1;`
 - `union xy u = u1; // - union xy u; u = u1;`
- Có thể khởi tạo với danh sách khởi tạo:
 - `struct xy s1 = {10, 9.5}; // Tuần tự: s1.x = 10; s1.y = 9.5;`
 - `struct xy s2 = {.y = 9.5, .x = 10}; // Chỉ định thành viên`
 - `struct xy s3 = {10}; // Tuần tự, 1 phần: s1.x = 10;`
 - `struct xy s4 = {.y = 9.5}; // Chỉ định, 1 phần: s1.y = 9.5;`
 - Nếu đối tượng cấu trúc được khởi tạo 1 phần, thì các thành viên còn lại được khởi tạo với giá trị mặc định (giá trị 0) được quy ước cho kiểu của các thành viên đó.
- Khởi tạo 1 thành viên của nhóm:
 - `union xy u1 = {.x = 10}; // union xy u1; u1.x = 10;`

Các đối tượng không tên

- Có thể tạo các đối tượng không tên bằng cách áp dụng toán tử ép kiểu cho danh sách khởi tạo:
 - Cấu trúc không tên: `(struct tên-kiểu){/*danh sách khởi tạo*/}`
 - Nhóm không tên: `(union tên-kiểu){/*danh sách khởi tạo*/}`
 - Có cú pháp tương tự biểu thức ép kiểu nhưng có thể cho kết quả là lvalue.
- Ví dụ:
 - `struct xy *ps = &((struct xy){10, 9.5});`
 - `union xy *pu = &((union xy){.x = 10});`
 - `ps->x += 5;`
 - `pu->x *= 2;`

Đặt tên tương đương với typedef

Cấu trúc

```
// Đặt xy_s là struct xy,  
// xy_sptr là struct xy*  
typedef struct xy xy_s, *xy_sptr;  
  
// Sử dụng trong khai báo  
xy_s s1; // struct xy s1;  
xy_sptr ps1; // struct xy *ps1;  
xy_s *ps2; // struct xy *ps2;
```

Nhóm

```
// Đặt xy_u là union xy,  
// xy_uptr là union xy*  
typedef union xy xy_u, *xy_uptr;  
  
// Sử dụng trong khai báo  
xy_u u1; // union xy u1;  
xy_uptr pu1; // union xy *pu1;  
xy_uptr pu2; // union xy *pu2;
```

Mảng cấu trúc và mảng nhóm

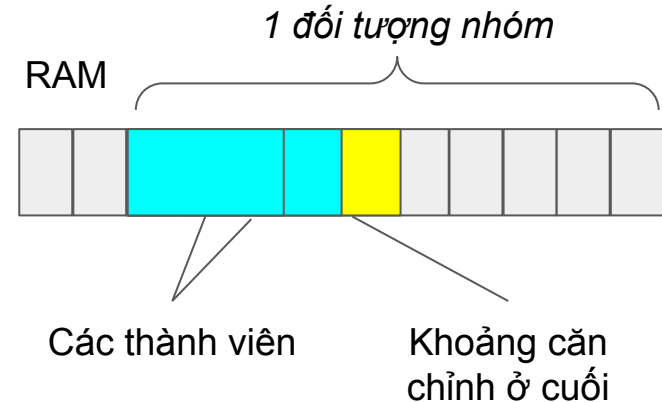
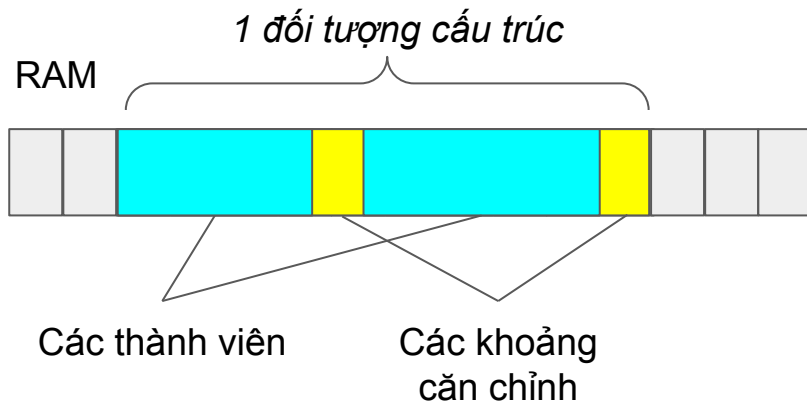
Tạo và khởi tạo mảng của cấu trúc và mảng của nhóm:

- `struct xy as[100];` // Mảng 100 phần tử kiểu `struct xy`
- `union xy au[100];` // Mảng 100 phần tử kiểu `union xy`
- `struct xy as1[] = {{10, 9.5}, {8, 8.5}};`
 - Mảng 1 chiều với kích thước được suy diễn = 2
 - `as1[0] = (struct xy){10, 9.5}; as1[1] = (struct xy){8, 8.5};`
- `union xy au1[] = {{.x = 10}, {.y = 8.5}};`
 - Kích thước được suy diễn tương tự `as1`.
 - `au1[0] = (union xy){.x = 10}; au1[1] = (union xy){.y = 9.5};`
- `struct xy *ps = as;` // `ps` trỏ tới `as[0]`;
- `union xy *pu = au;` // `pu` trỏ tới `au[0]`;

Mô hình bộ nhớ

Tuy giống nhau về cú pháp nhưng cấu trúc và nhóm có sự khác biệt rõ nét trong cách tổ chức lưu trữ các đối tượng thành viên.

- Vùng nhớ của các thành viên của cấu trúc không giao nhau và được *tổ chức tuần tự*, theo thứ tự khai báo.
 - Có thể có các khoảng căn chỉnh ở giữa các thành viên và ở cuối, nhưng không có ở đầu.
- Vùng nhớ của các thành viên của nhóm chồng lấp do có cùng địa chỉ = địa chỉ của đối tượng nhóm.
 - Có thể có các khoảng căn chỉnh ở cuối.



Các hệ quả của mô hình bộ nhớ: Kích thước

- Đối với cấu trúc:
 - Kích thước cấu trúc \geq Tổng kích thước các thành viên.
 - Có thể xác định khoảng cách tính bằng bytes của đối tượng thành viên tới đầu cấu trúc với macro `offsetof` (`stddef.h`)
 - `offsetof(struct xy, x)` cho kết quả = 0, kiểu `size_t`.
 - `offsetof(struct xy, y) \geq sizeof(long);` // x đứng trước y.
- Đối với nhóm:
 - Kích thước nhóm \geq Kích thước phần tử có kích thước lớn nhất trong nhóm.
 - Ở 1 thời điểm 1 đối tượng nhóm chỉ có thể lưu 1 giá trị của 1 thành viên.
 - *Thông thường khi lưu dữ liệu vào thành viên nào thì đọc dữ liệu từ thành viên đó.*

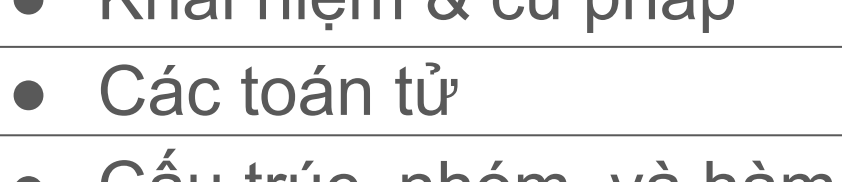
Ví dụ 8.1. Kích thước bộ nhớ

```
vd8-1.c x
5 #include <stdio.h>
6 #include <stddef.h>
7
8 struct s1 { int x; char c; int y; };
9 struct s2 { float x, y, z; };
10 union g1 { float x; double d; };
11 union g2 { struct s2 p; double d; };
12
13 int main() {
14     printf("sizeof:\nchar: %zu, int: %zu, float: %zu, double: %zu\n",
15         sizeof(char), sizeof(int), sizeof(float), sizeof(double));
16     printf("sizeof(struct s1) = %zu\n", sizeof(struct s1));
17     printf("offsetof(struct s1, c) = %zu\n", offsetof(struct s1, c));
18     printf("offsetof(struct s1, y) = %zu\n", offsetof(struct s1, y));
19     printf("sizeof(struct s2) = %zu\n", sizeof(struct s2));
20     printf("sizeof(union g1) = %zu\n", sizeof(union g1));
21     printf("sizeof(union g2) = %zu\n", sizeof(union g2));
22     return 0;
23 }
```

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

Có thể thấy có 3 bytes căn chỉnh giữa các thành phần c và y trong s1 và 4 bytes căn chỉnh trong g2 trong trường hợp này.

Nội dung

- Khái niệm & cú pháp
 - Các toán tử
 - Cấu trúc, nhóm, và hàm
- 

Toán tử thành viên

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

- Toán tử (dấu) chấm: . - được áp dụng cho đối tượng
- Toán tử mũi tên: -> - được áp dụng cho con trỏ
- Ví dụ:
 - struct xy s1, *ps; ps = &s1;
 - s1.y, ps->y và (*ps).y đều là thành viên y của s1 và có thể được sử dụng như 1 biến kiểu double.
 - union xy u1, *pu; pu = &u1;
 - u1.x, pu->x, và (*pu).x đều là thành viên x của u1 và có thể được sử dụng như 1 biến kiểu long.

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

Toán tử gán

Sao chép dữ liệu trong vùng nhớ của đối tượng ở vế phải sang vùng nhớ của đối tượng ở vế trái

- Ví dụ:
 - `struct xy s1, s2; /*...*/ s1 = s2;`
 - `union xy u1, u2; /* ... */ u1 = u2;`
 - Tương tự sử dụng `memcpy` (`string.h`)
 - `memcpy(&s1, &s2, sizeof(struct xy));`
 - `memcpy(&u1, &u2, sizeof(union xy));`
- Ví dụ đối tượng cấu trúc có thành viên là mảng:
 - `struct arr10 { int elems[10]; } a1, a2; /* ... */`
 - `a1 = a2;`
 - `memcpy(&a1, &a2, sizeof(struct ar10)); // OK`
 - `a1.data = a2.data; // NOK - Không hợp lệ với mảng`

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 s;`
 - `struct xy *ps = &s; long *px = (long*)ps; // Ok, px == &s.x;`
 - `long *px = &s.x; struct xy *ps = (struct xy*)px; // Ok, ps == &s;`
 - `double *py; py = (double*)ps; // NOK - y không phải phần tử đầu tiên`
 - Với các thành viên khác có thể sử dụng `offsetof`, ví dụ:
 - `struct xy s; double *py = &s.y;`
 - `struct xy *ps = (void*)((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 xy u;`
 - `union xy *pu = &u;`
`long *px = (long*)pu; double *py = (double *)pu; // Ok, px == &u.x`
 - `double *py = &u.y;`
`union xy *pu = (union xy*)py; // Ok, pu == &u;`

So sánh con trỏ

- 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;` // Vì x là phần tử đầu tiên
- `px < py;` // Vì y được khai báo sau x.

- Với

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

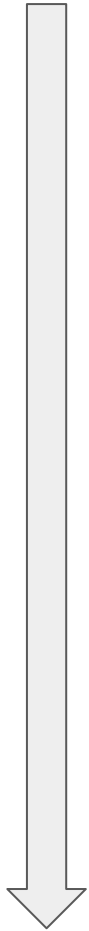
```
void *pu = &u, *px = &u.x, *py = &u.y;
```

các quan hệ so sánh sau cho kết quả đúng:

- `pu == px;` // Vì các thành viên của nhóm có cùng
- `px == py;` // địa chỉ đối tượng nhóm.

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

Thứ tự ưu tiên của các toán tử

- `struct xy s;`
 - `&s.x;` // Tương đương với `&(s.x)`
 - `s.x++;` // `(s.x)++;`
 - `++s.x;` // `++(s.x)`
- `struct arr10 { int elems[10]; } a;`
 - `a.elems[i];` // `(a.elems)[i]`
 - `&a.elems[i];` // `&((a.elems)[i])`
- `struct xyz {struct xy *p; double z; } s; /* ... */`
 - `s.p->x;` // `(s.p)->x;`
 - `&s.p->x;` // `&((s.p)->x);`


Tương tự với các đối tượng kiểu nhóm.

Các toán tử khác

Các kiểu cấu trúc và các kiểu nhóm được định nghĩa bởi người lập trình và trình biên dịch có thể không biết trước ý nghĩa của các giá trị. Các toán tử liên quan đến biểu diễn dữ liệu, phụ thuộc vào ý nghĩa của các giá trị, thường không áp dụng được với các kiểu do người dùng tự định nghĩa.

- Ví dụ:
 - Các toán tử đại số: Cộng (+), trừ (-), nhân (*), chia (/), v.v., và nhiều phép toán liên quan khác.
 - Các toán tử so sánh: Lớn hơn (>), nhỏ hơn (<), bằng nhau (==) v.v., và nhiều phép toán so sánh liên quan khác.
 - v.v., và còn nhiều toán tử khác nữa chưa được liệt kê.
- Có thể xử lý dữ liệu thuộc kiểu do người lập trình tự định nghĩa với hàm hoặc macro thay vì sử dụng các toán tử.

Nội dung

- Khái niệm & cú pháp
 - Các toán tử
 - Cấu trúc, nhóm, và hàm
- 

Hàm, cấu trúc và nhóm

Hàm có thể nhận tham số là cấu trúc, nhóm và trả về cấu trúc, nhóm tương tự như với các kiểu dữ liệu có sẵn.

- Có thể sử dụng hàm để định nghĩa các thao tác xử lý với cấu trúc và nhóm.
- Cấu trúc có thể được sử dụng như giá trị trả về cho các hàm trả về nhiều giá trị.
- Khi trao đổi các đối tượng có kích thước lớn như trong trường hợp truyền tham số và nhận giá trị trả về của hàm, có thể sử dụng các con trỏ để giảm chi phí xử lý.

Cấu trúc point

Các minh họa tiếp theo về sử dụng hàm để xử lý dữ liệu kiểu cấu trúc sẽ được thực hiện với 1 cấu trúc thực tế hơn.

- Cấu trúc point được định nghĩa như sau:

```
struct point { double x; double y; };
```

- Mỗi đối tượng kiểu struct point lưu các giá trị tọa độ của 1 điểm trong mặt phẳng.
- Các phép + và phép / có những ý nghĩa nhất định với kiểu struct point.
 - Trung điểm C của 1 đoạn thẳng AB trong mặt phẳng có thể được xác định theo công thức $c = (a + b) / 2$; với a, b, c có kiểu struct point lần lượt là các đối tượng tương ứng với các điểm A, B và C.
 - Tuy biểu thức $c = (a + b) / 2$; không hợp lệ trong C, nhưng các tính toán tương đương có thể được thực hiện với hàm.

Nhập, xuất theo định dạng

- Ví dụ nhập 1 điểm với scanf:
 - `struct point p;`
 - `printf("Nhập các tọa độ x và y: ");`
 - `scanf("%lf%lf", &p.x, &p.y);`
- Ví dụ xuất 1 điểm với printf:
 - `struct point p; /*... */`
 - `printf("Tọa độ của p là (%.2f, %.2f)\n", p.x, p.y);`
- Có thể đóng gói các thao tác nhập xuất với đối tượng kiểu cấu trúc bằng hàm để tiện sử dụng nhiều lần, ví dụ:
 - `scan_point(&p);`
 - `print_point(&p);`

Ví dụ 8.2. Nhập/xuất cấu trúc

```
vd8-2.c x
6  #include <stdio.h>
7  struct point {double x, y;};
8  void scan_point(struct point *p) {
9      printf("Nhập tọa độ 1 điểm: ");
10     scanf("%lf%lf", &p->x, &p->y);
11 }
12 void print_point(const struct point *p) {
13     printf("Tọa độ của điểm là (%.2lf, %.2lf)\n",
14         p->x, p->y);
15 }
16 int main() {
17     struct point p;
18     scan_point(&p);
19     print_point(&p);
20     return 0;
21 }
```

bangoc:\$gcc -o prog vd8-2.c
bangoc:\$./prog
Nhập tọa độ 1 điểm: 3.5 7.5
Tọa độ của điểm là (3.50, 7.50)
bangoc:\$

Các phép toán đại số

- Ví dụ tính trung điểm của đoạn thẳng:
 - `struct point a, b, c; /* ... */`
 - `c.x = (a.x + b.x) / 2;`
 - `c.y = (a.y + b.y) / 2;`
- Có thể đóng gói các thao tác cộng các đối tượng struct point và chia đối tượng struct point với 1 giá trị số bằng hàm, ví dụ:
 - `sum_point(&a, &b, &c);`
 - `div_point(&c, 2);`
 - Hoặc kết hợp thành: `div_point(sum_point(&a, &b, &c), 2);`

Ví dụ 8.3. Các phép toán đại số với cấu trúc

```
vd8-3.c
15 struct point *sum_point(const struct point *a,
16     const struct point *b, struct point *c) {
17     c->x = a->x + b->x;
18     c->y = a->y + b->y;
19     return c;
20 }
21 struct point *div_point(struct point *p, double k) {
22     p->x /= k;
23     p->y /= k;
24     return p;
25 }
26 int main() {
27     struct point a, b, c;
28     scan_point(&a);
29     scan_point(&b);
30
31     // c = (a + b)/2
32     div_point(sum_point(&a, &b, &c), 2);
33     print_point(&c);
34     return 0;
35 }
```

```
bangoc:$gcc -o prog vd8-3.c
bangoc:$./prog
Nhập tọa độ 1 điểm: 1 3
Nhập tọa độ 1 điểm: 5 8
Tọa độ của điểm là (3.00, 5.50)
bangoc:$
```

Quan hệ thứ tự

- Ví dụ so sánh 2 điểm theo khoảng cách tới gốc tọa độ, điểm gần gốc tọa độ hơn được coi là nhỏ hơn.
 - `struct point p1, p2; /* ... */`
 - $p1.x * p1.x + p1.y * p1.y < p2.x * p2.x + p2.y * p2.y;$
- Có thể đóng gói các thao tác so sánh các điểm theo khoảng cách tới gốc tọa độ bằng hàm, ví dụ:
 - `less_point(&p1, &p2);`

Ví dụ 8.4. Tìm min trong 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 x
16 #define MAXN 100
17 int less_point(const struct point*p1,
18               const struct point *p2) {
19     return p1->x * p1->x + p1->y * p1->y <
20           p2->x * p2->x + p2->y * p2->y;
21 }
22 int main() {
23     struct point a[MAXN];
24     printf("Nhập số lượng phần tử (<=100): ");
25     int n;
26     scanf("%d", &n);
27     printf("Nhập %d điểm: \n", n);
28     for (int i = 0; i < n; ++i) {
29         scan_point(a + i);
30     }
31     struct point min = a[0];
32     for (int i = 1; i < n; ++i) {
33         if (less_point(&a[i], &min)) {
34             min = a[i];
35         }
36     }
37     print_point(&min);
38     return 0;
39 }
```

```
bangoc:$gcc -o prog vd8-4.c
bangoc:$./prog
Nhập số lượng phần tử (<=100): 5
Nhập 5 điểm:
Nhập tọa độ 1 điểm: 3 3
Nhập tọa độ 1 điểm: 5 5
Nhập tọa độ 1 điểm: 2 2
Nhập tọa độ 1 điểm: 6 6
Nhập tọa độ 1 điểm: 1 1
Tọa độ của điểm là (1.00, 1.00)
bangoc:$
```


Biểu diễn nhóm kiểu

- Ví dụ nhóm kiểu:
 - `typedef union generic_type{long l; double d;} gtype;`
- Ví dụ thiết kế hàm xuất dữ liệu với giao diện đơn giản:
 - `typedef struct tagged_gtypes { char tag; gtype value; } tgt;`
 - `tgt v1 = {'l', {.l = 10}}, v2 = {'d', {.d = 3.5}}; /* ... */`
 - `print_tgt(v1); // 10: long`
 - `print_tgt(v2); // 3.5: double`

Ví dụ 8.5. Hàm khái quát với cấu trúc và nhóm

```
vd8-5.c x
6  #include <stdio.h>
7  typedef union generic_type {long l; double d;} gtype;
8  typedef struct tagged_gtype {char tag; gtype val;} tgt;
9  void print_tgt(tgt x) {
10     if (x.tag == 'l') {
11         printf("%ld: long\n", x.val.l);
12     } else if (x.tag == 'd') {
13         printf("%g: double\n", x.val.d);
14     }
15 }
16 int main() {
17     tgt v1 = {'l', {.l = 10}},
18     v2 = {'d', {.d = 3.5}};
19     print_tgt(v1);
20     print_tgt(v2);
21     return 0;
22 }
```

```
bangoc:$gcc -o prog vd8-5.c
bangoc:$./prog
10: long
3.5: double
bangoc:$
```

