IT001 - NHẬP MÔN LẬP TRÌNH

CHƯƠNG 9: CẤU TRÚC STRUCT

Ngoài những kiểu dữ liệu cơ bản, trong quá trình xây dựng chương trình, chúng ta cần phải định nghĩa kiểu dữ liệu mới, đòi hỏi độ phức tạp cao hơn. Trong chương này, chúng ta sẽ được cung cấp các kiến thức cơ bản về cách định nghĩa một kiểu dữ liệu mới và cách sử dụng chúng trong lập trình. Phần kiến thức này sẽ giúp chúng ta hiểu tốt hơn về các kiểu dữ liệu cấu trúc khác (như danh sách liên kết, cấu trúc cây) và phương pháp lập trình hướng đối tượng.

Khoa Khoa học Máy tính

NỘI DUNG

9.1 Đặt vấn đề

9.2 Khai báo và khởi tạo biến cấu trúc

9.3 Kích thước của cấu trúc

9.4 Truy xuất thuộc tính của kiểu cấu trúc

9.5 Phép Gán dữ liệu kiểu cấu trúc

9.6 Mảng cấu trúc, Cấu trúc phức hợp, Cấu trúc tự trỏ

9.7 Kiểu hợp nhất union

9.8 Cấu trúc và con trỏ

9.9 Truyền cấu trúc cho hàm

9.10 Ví dụ minh họa

Bài tập

9.1 Đặt vấn đề

9.1 Đặt vấn đề

- Thông tin 1 sinh viên (SV)
  - `MSSV: kiểu chuỗi`
  - `Tên SV: kiểu chuỗi`
  - `Ngày tháng năm sinh: kiểu chuỗi`
  - `Giới tính: ký tự`
  - `Điểm toán, lý, Anh văn: số thực`
- Yêu cầu
  - Lưu thông tin cho N sinh viên ?
  - Truyền thông tin N sinh viên vào một hàm ?

9.1 Đặt vấn đề

- Khai báo các biến để lưu trữ 1 SV

```
char mssv[8];          // Ví dụ: “07520045”
char hoten[30];        // Ví dụ: “Luong Bao Yen”
char ngaysinh[10];     // Ví dụ: “27/05/1989”
char gioitinh;         // Ví dụ:‘F’ → Nữ (‘M’ →Nam)
float toan, ly, anhvan;     // Ví dụ: Toán: 9.5, Lý:
9.25, Anh: 10
```

- Truyền thông tin 1 SV cho hàm:

```
void xuat(char mssv[], char hoten[], char ngaysinh[],
          char gioitinh, float toan, float ly, float
```

9.1 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

9.2 Khai báo và khởi tạo biến cấu trúc

Khai báo cấu trúc

- **Cú pháp 1:**

```
struct <tên_kiểu_cấu_trúc> {
    <kiểu_dữ_liệu>
<tên_thành_phần_1>;
    …
    <kiểu_dữ_liệu>
<tên_thành_phần_n>;
};
```

- Trong đó:
  - <tên_kiểu_cấu_trúc>: Đặt theo tên/định danh
  - <kiểu_dữ_liệu>: kiểu dữ liệu
  - <tên_thành_phần_1>: tên biến có <kiểu dữ liệu> tương ứng
  - <tên_biến>: biến có kiểu cấu trúc

- **Ví dụ:**

```
struct Point {
    float x, y;
};
struct Point A, B;
(lưu ý: C++ có thể bỏ từ khóa struct ở
khai báo biến A, B)

struct NgaySinh {
    int ngay, thang, nam;
};
NgaySinh date;
```

Sử dụng typedef

- **Cú pháp 2:**

```
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>;
```

- **Ví dụ:**

```
typedef struct {
    float x, y;
} Point;
Point A, B;
```

```
typedef struct {
    int ngay, thang,
nam;
} NgaySinh;
NgaySinh date;
```

Khởi tạo cho biến cấu trúc

- **Cú pháp 3:**

```
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_n>};
```

- **Ví dụ:**

```
struct Point {
    float x, y;
} A = { 20.5, 10 },
B;
```

```
struct NgaySinh {
    int ngay, thang,
nam;
} date = {27, 5,
1989};
```

Khởi tạo cho biến cấu trúc

- **Cú pháp 4:**

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

- **Lưu ý:** Cấu trúc chỉ được dùng để khai báo kiểu dữ liệu cho các <tên_biến>.
- **Ví dụ:**

```
struct {
    float x, y;
} A = { 20.5, 10 }, B;
// Với A, B là biến có cấu trúc gồm 2 thành phần float x, y;
```

```
struct {
    int ngay, thang, nam;
} date = {27, 5, 1989}, date1;
// Với date, date1 là 2 biến có cấu trúc gồm 3 thành phần int ngay, thang, nam;
```

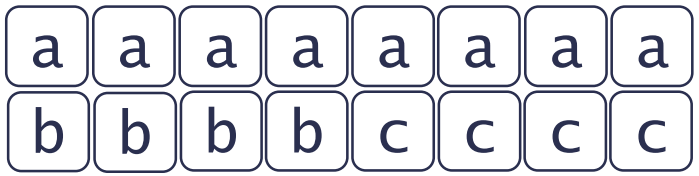
9.3 Kích thước của cấu trúc

9.3 Kích thước của cấu trúc

- Ví dụ:

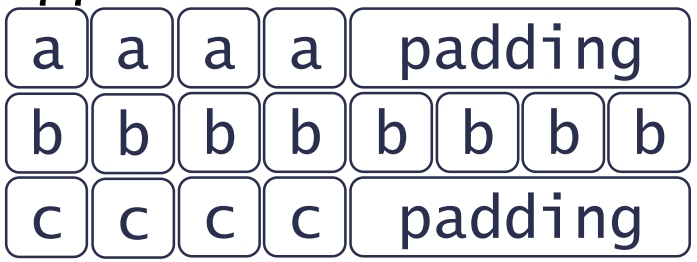
```
struct A {
    double a;
    int b;
    int c;
};
sizeof (A)
```

16 bytes



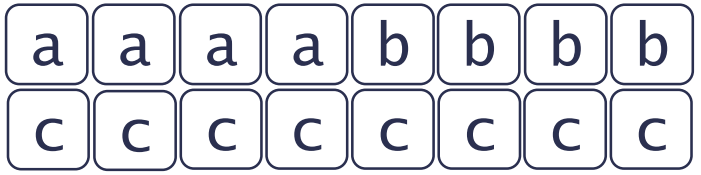

```
struct B {
    int a;
    double b;
    int c;
};
sizeof (B)
??
```

24 bytes




```
struct C {
    int a;
    int b;
    double c;
};
sizeof (C)
??
```

16 bytes



➔ Sự khác biệt đến từ **thứ tự khai báo các biến** và **biên kích thước** (tính theo byte) của cấu trúc.

9.3 Kích thước của cấu trúc

- Hoặc tối ưu biên cho cấu trúc (**alignment of struct**). Ví dụ trên nếu thay đổi biên cấu trúc thành 1 hoặc 4 thì **sizeof(B) = 16.**
- Điều chỉnh biên cấu trúc: **`Project settings → Compile Option C/C++ → Code Generation → Structure Alignment.`**
- Biên nhỏ mặc dù giúp giảm kích thước của cấu trúc nhưng làm tăng thời gian xử lý của tác vụ memory allocator ➔ Cần điều phối thích hợp giữa kích thước cấu trúc và tốc độ xử lý.
- Chương trình dùng nhiều cấu trúc có thành phần khác nhau ➔ điều chỉnh biên tốt nhất sẽ khó khăn
- **Ưu tiên: tối ưu bằng cách khai báo thứ tự các thành phần cấu trúc phù hợp với biên cấu trúc (tối ưu cục bộ trên cấu trúc)**

Chỉ thị #pragma pack

- Chỉ thị `#pragma pack(n)`
  - n = 1, 2, 4, 8, 16 (byte)
  - Biên lớn nhất của các thành phần trong `struct`
    - BC n mặc định là 1
    - VC++ n mặc định là 8
  - Canh biên cho 1 cấu trúc:

```
#pragma pack(push, 1)
struct MYSTRUCT  {
    …
};
#pragma pack(pop)
```

9.3 Kích thước của cấu trúc

```
// struct_not_pragm_pack.cpp
#include <iostream>
using namespace std;
// #pragma pack (1)
struct A { double x; int y; int z; } a;
struct B { int x; double y; int z; } b;
struct C { int x; int y; double z; } c;
int main() {
      cout << "sizeof: " << sizeof(a) << "\t" << sizeof(b) << "\t" <<
sizeof(c) << endl;
    cout << "Xuat dia chi:" << endl;
     cout << "a : " << &(a.x) << "\t" << &(a.y) << "\t" << &(a.z) <<
endl;
      cout << "b : " << &(b.x) << "\t" << &(b.y) << "\t" << &(b.z) <<
endl;
```

Kết quả thực thi:

```
sizeof: 16    24    16
Xuat dia chi:
a : 0x407030   0x407038     0x40703c
b : 0x407040   0x407048     0x407050
c : 0x407060   0x407064     0x407068
```

9.3 Kích thước của cấu trúc

```
// struct_pragma.cpp
#include <iostream>
using namespace std;
#pragma pack (1)
struct A { double x; int y; int z; } a;
struct B { int x; double y; int z; } b;
struct C { int x; int y; double z; } c;
int main() {
      cout << "sizeof: " << sizeof(a) << "\t" << sizeof(b) << "\t" <<
sizeof(c) << endl;
    cout << "Xuat dia chi:" << endl;
     cout << "a : " << &(a.x) << "\t" << &(a.y) << "\t" << &(a.z) <<
endl;
      cout << "b : " << &(b.x) << "\t" << &(b.y) << "\t" << &(b.z) <<
endl;
```

Kết quả thực thi:

sizeof: 16 **16** 16
Xuat dia chi:
a : 0x407030 0x407038 0x40703c
b : **0x407040** **0x407044** **0x40704c**
c : 0x407050 0x407054 0x407058

9.4 Truy xuất thuộc tính của kiểu cấu trúc

9.4 Truy xuất thuộc tính của kiểu cấu trúc

- **Đặc điểm**: Các trường dữ liệu trong cấu trúc Không thể truy xuất trực tiếp mà phải thông qua biến cấu trúc.
- Các biến cấu trúc truy xuất dữ liệu thông qua toán tử thành phần cấu trúc "." hay còn gọi là toán tử chấm (dot operation)

```
<tên_biến_cấu_trúc>.<tên_thành_phần>
```

- Ví dụ:

```
struct Point {
    float x, y;
} A;
cout << A.x;
cout << A.y;
```

9.4 Truy xuất thuộc tính của kiểu cấu trúc

- **Biến cấu trúc kiểu con trỏ**:
  - Thông qua toán tử thành phần cấu trúc "**->**" hay còn gọi là toán tử dấu mũi tên (arrow operation)

```
<tên_biến_cấu_trúc> ->
<tên_thành_phần>
```

- Ví dụ:

```
struct Point {
    float x, y;
} *A;
cout << A->x;
cout << A->y;
```

* Lưu ý biến con trỏ A cần xác định được vùng nhớ trỏ đến trước khi truy xuất các thuộc tính của A.

9.5 Phép Gán dữ liệu kiểu cấu trúc

9.5 Phép Gán dữ liệu kiểu cấu trúc

- Có thể xét 2 cách gán sau:

```
<Biến_cấu_trúc_đích> =
<biến_cấu_trúc_nguồn>;
<Biến_cấu_trúc_đích>.<tên_thành_phần> =
<giá trị>;
```

- Ví dụ:

```
struct Point {
    int x, y;
} A = { 2912, 1706 }, B;
// C1:
B = A;
// C2
B.x = A.x;
B.y = A.y * 2;
```

- Lưu ý khi thành phần của cấu trúc có con trỏ.
- Ví dụ:

```
struct Point {
    int x, *y;
};
```

9.6 Mảng cấu trúc, cấu trúc phức hợp, cấu trúc tự trỏ

Mảng cấu trúc

- **Mảng cấu trúc**: Khai báo và hoạt động tương tự như mảng với kiểu dữ liệu cơ sở.
- Có thể khai báo Mảng cấu trúc 1 chiều, 2 chiều hoặc nhiều chiều.
- Ví dụ:

```
Point arr[10] = { { 3, 2 }, { 4, 4 }, { 2, 7 } };
cout << arr[0].x << arr[0].y;
cout << arr[1].x << arr[1].y;
cout << arr[2].x << arr[2].y;
```

Cấu trúc phức hợp

- Cấu trúc phức hợp có thành phần của cấu trúc là cấu trúc khác.
- Ví dụ:

```
struct Point {
    float x, y;
};
struct Rectangle {
    Point TraiTren;
    Point PhaiDuoi;
} Rec1;
Rec1.TraiTren.x = 2912;
Rec1.TraiTren.y = 1706;
```

Cấu trúc phức hợp

- Cấu trúc phức hợp có thành phần của cấu trúc là mảng.
- Ví dụ:

```
struct SINHVIEN {
    char HoTen[30];
    float DiemToan, DiemLy,
DiemHoa;
} SV1;

strcpy(SV1.HoTen, "Nguyen Van
A");
SV1.DiemToan = 10;
SV1.DiemHoa = 6.5;
SV1.DiemLy = 9;
```

Cấu trúc tự trỏ

- **Cấu trúc tự trỏ (tự trỏ):** Tồn tại một (hoặc nhiều) thành phần thông tin trong cấu trúc đang khai báo có kiểu dữ liệu chính là cấu trúc đó.
- Ví dụ:

```
struct PERSON {
    char hoten[30];
    PERSON *father,
*mother;
};

struct NODE {
    int value;
    NODE *pNext;
};
```

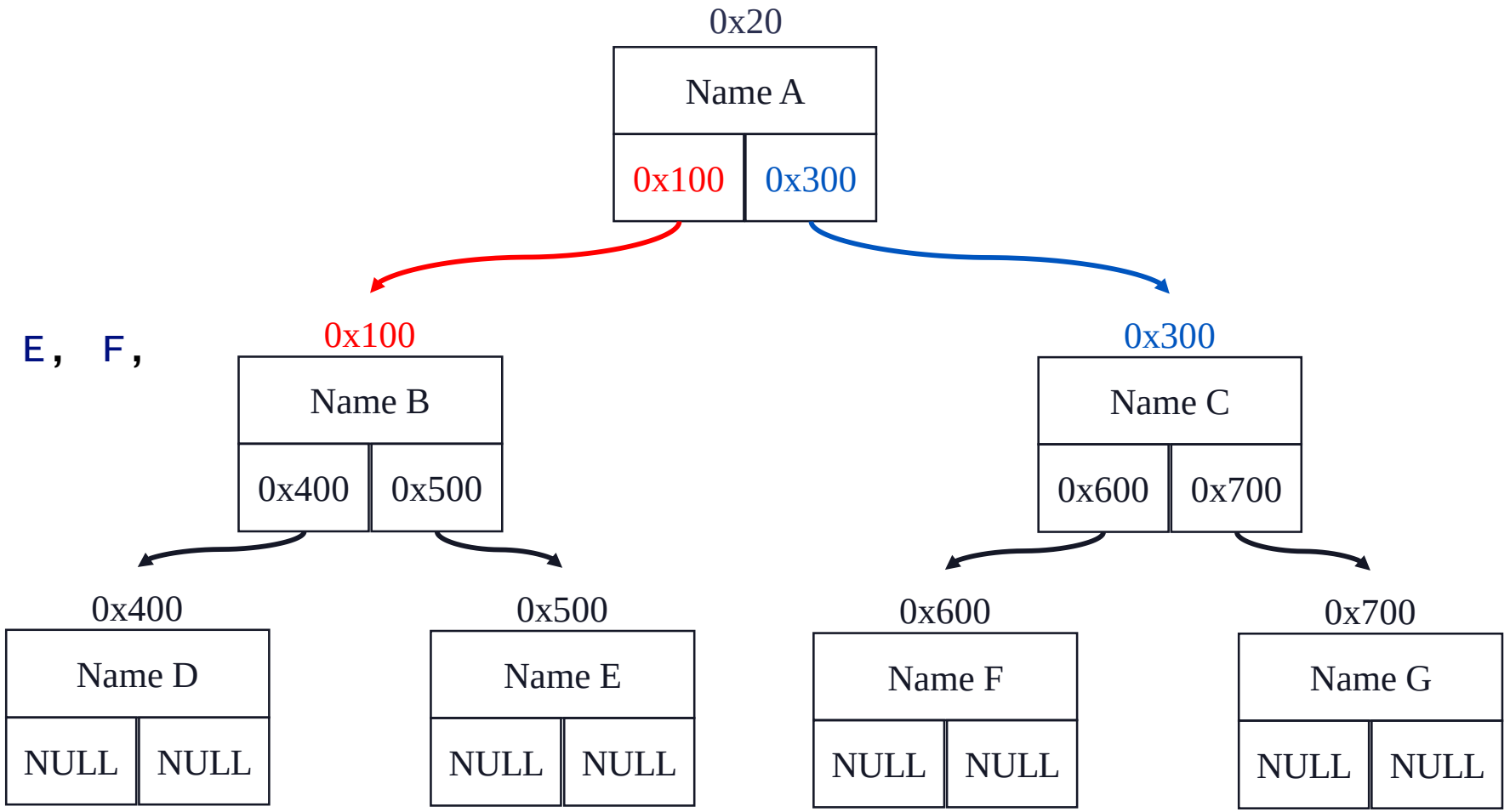
Cấu trúc tự trỏ

```
struct PERSON {
  char hoten[30];
  PERSON *father,
*mother;
} ;
PERSON A, B, C, D, E, F,
G;
A.father=&B;
A.mother=&C;

B.father=&D;
B.mother=&E;

C.father=&F;
C.mother=&G;

D.father=D.mother=NULL;
E.father=E.mother=NULL;
F.father=F.mother=NULL;
G.father=G.mother=NULL;
```



Cấu trúc tự trỏ

- Hãy cho biết làm sao để:
  - Truy xuất thông tin của B thông qua A?

    `B.hoten` tương đương `A.father->hoten`

    `B.father` tương đương `A.father->father`

    `B.mother` tương đương `A.father->mother`
  - Truy xuất thông tin của C thông qua A?

    `C.hoten` tương đương `A.mother->hoten`

    `C.father` tương đương `A.mother->father`

    `C.mother` tương đương `A.mother->mother`
  - Truy xuất họ tên của D thông qua A và B?

    `D.hoten` tương đương `B.father->hoten`

    tương đương `A.father->father->hoten`

9.7 Kiểu hợp nhất union

9.7 Kiểu hợp nhất union

- Khái niệm:
  - Được khai báo và sử dụng như cấu trúc `struct`
  - Các thành phần của union có chung địa chỉ đầu (nằm chồng lên nhau trong bộ nhớ)
- Khai báo:

```
union <tên kiểu union> {
    <kiểu dữ liệu> <tên thành phần 1>;
    …
    <kiểu dữ liệu> <tên thành phần 2>;
};
```

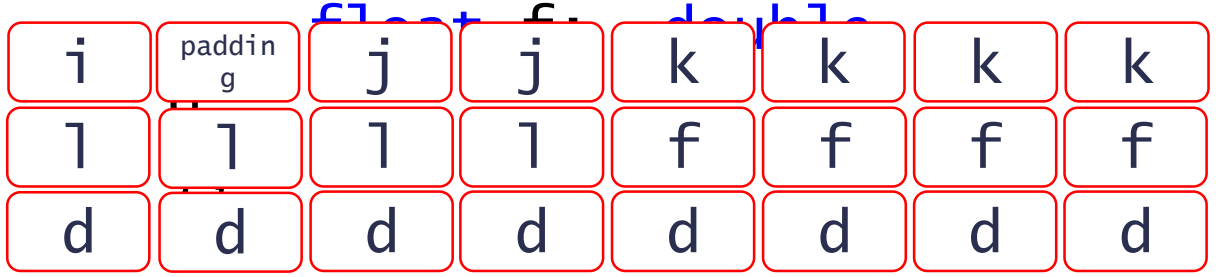
9.7 Kiểu hợp nhất union

- **Ví dụ**: Viết chương trình xử lý các loại dữ liệu khác nhau bằng cách sử dụng cùng một biến. Các kiểu dữ liệu cần xử lý là: `(char, short, int, long, float, double).`
  - **Tạo cấu trúc struct:** gồm tất cả các loại khác nhau cần lưu trữ, kích thước của cấu trúc này là 24 bytes
  - **Sử dụng cấu trúc union:** gồm tất cả các loại khác nhau cần lưu trữ, nhưng kích thước của union chỉ là 8 bytes

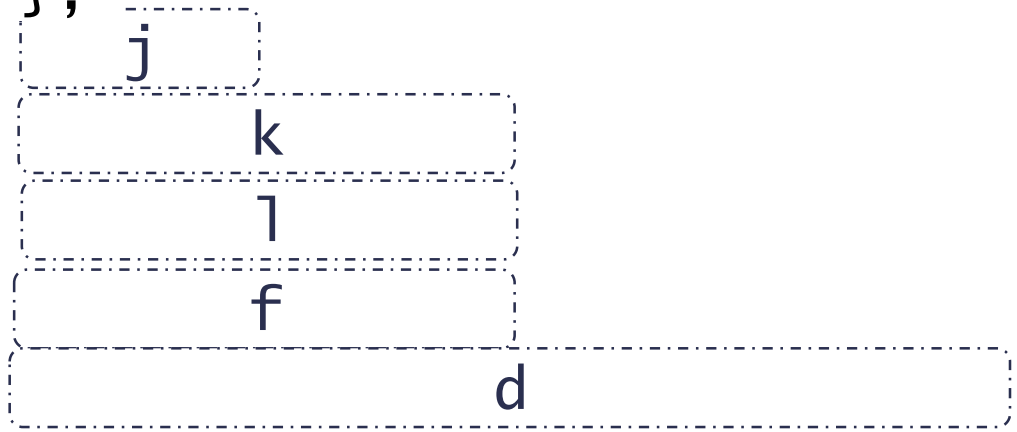
So sánh struct và union

- Ví dụ:

```
struct structPacked
{
    char i;   short
j;
    int k;    long
l;
    float f;  double
d;
};
```




```
union unionPacked {
    char i;   short
j;
    int k;    long
l;
    float f;  double
d;
};
```



9.7 Kiểu hợp nhất union

- **Lưu ý**: nếu gán một giá trị cho một biến trong union thì biến này có thể sử dụng giá trị được gán ➔ biến khác trong union sẽ không sử dụng được giá trị đó, trừ khi được gán trực tiếp một giá trị khác.

```
union unionPacked {
    char i;   short j;
    int k;    long l;
    float f;  double d;
};
unionPacked x;
x.f = 3.14159;
cout << x.f << endl; // Xuất ra 3.14159
cout << x.i << endl; // ➔ Xuất ra giá trị rác.
```

So sánh struct và union

```
#include <iostream>
using namespace std;
// The union will be the size of a
// double, since that's the largest element
union unionPacked {
    char i;    short j;
    int k;     long l;
    float f;   double d;
};
struct structPacked {
    char i;    short j;
    int k;     long l;
    float f;   double d;
};
```

```
int main() {
    cout << "sizeof(structPacked) = "
         << sizeof(structPacked) << endl;
    cout << "sizeof(unionPacked) = "
         << sizeof(unionPacked) << endl;
    structPacked x;
    cout << "Address of struct: ";
    cout << (void*)&x.i << "\t" << &x.j << "\t"
         << &x.k << "\t" << &x.l << "\t"
         << &x.f << "\t" << &x.d << endl;
    unionPacked y;
    cout << "Address of union:  ";
    cout << (void*)&y.i << "\t" << &y.j << "\t"
         << &y.k << "\t" << &y.l << "\t"
         << &y.f << "\t" << &y.d << endl;
}
```

Kết quả thực thi:

sizeof(structPacked) = 24 sizeof(unionPacked) = 8

Address of struct: 0x61fe00 0x61fe02 0x61fe04 0x61fe08 0x61fe0c 0x61fe10

Address of union: 0x61fdf8 0x61fdf8 0x61fdf8 0x61fdf8 0x61fdf8 0x61fdf8

Khai báo struct là thành phần của union

- Ví dụ:

```
#include <iostream>
#include <string.h>
using namespace std;
union date {
    char full_date[9];
    struct { int ngay, thang, nam; };
} ;
int main() {
    cout << "date = " << sizeof(date) << endl;
    date A, B;
    strcpy(A.full_date, "31/7/1989");
    cout << A.full_date << endl;

    B.ngay=31;  B.thang=7;   B.nam=1989;
    cout << B.ngay << "/" << B.thang << "/" << B.nam;

    cout << B.full_date << ".";
    return 0;
}
```

Kết quả thực thi:

```
date = 12
31/7/1989
31/7/1989
.
```

9.8 Cấu trúc và con trỏ

9.8 Cấu trúc và Con trỏ

- Con trỏ trỏ đến một vùng nhớ có kiểu cấu trúc (hoặc mảng cấu trúc):

```
struct StructType_Name {
        Các biến trường thông tin
};
StructType_Name <Variable_Name>;
StructType_Name     *     <PointerVariable_Name>     =
&<Variable_Name>;
```

- Truy cập các trường thông tin của cấu trúc thông qua con trỏ sử dụng toán tử `->` hoặc `.`

```
p->BienTruongThongTin
(*p).BienTruongThongTin
```

9.8 Cấu trúc và Con trỏ

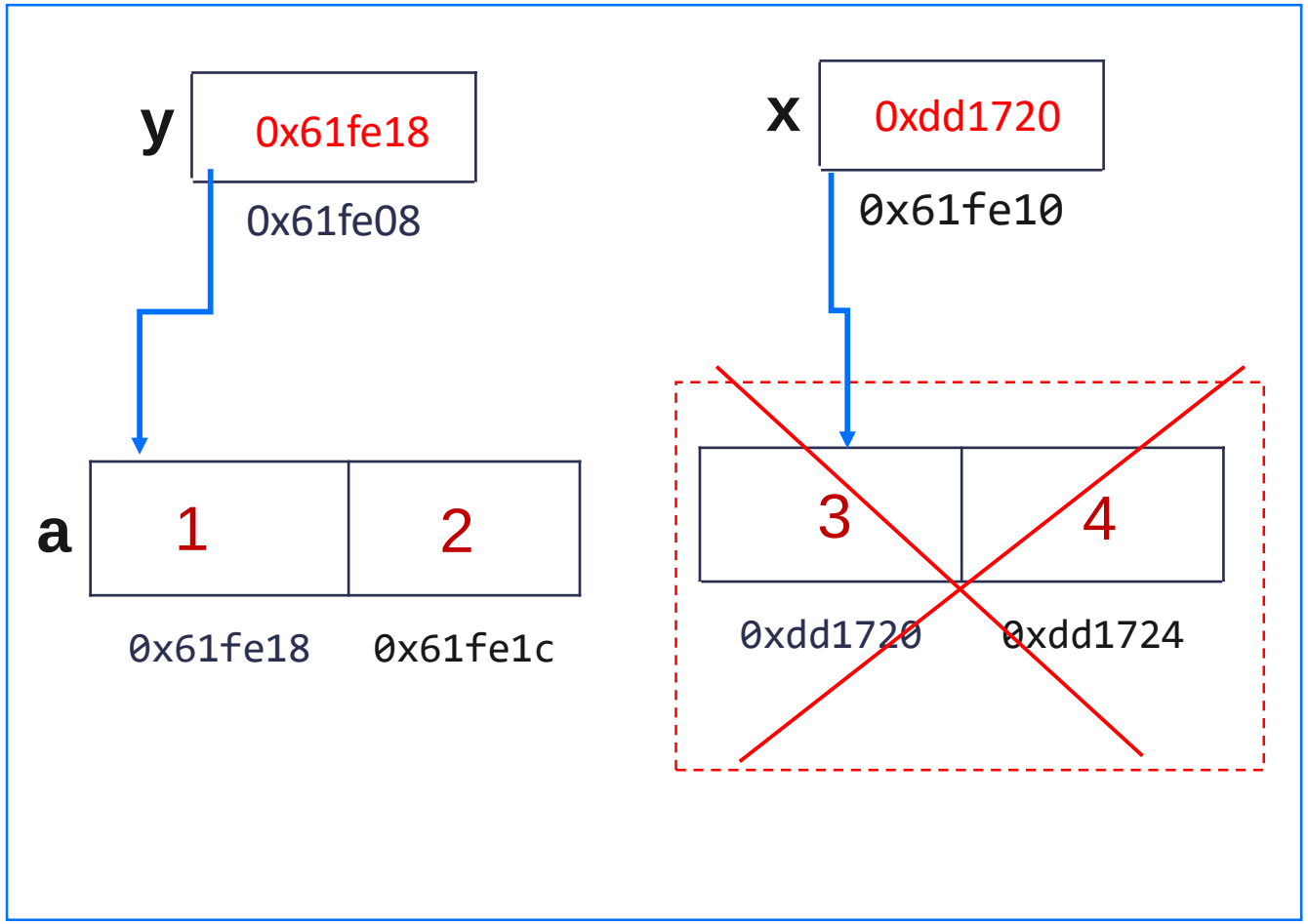
- Cấp phát động cho biến con trỏ có kiểu cấu trúc:

```
struct StructType_Name {
        Các biến trường thông tin
};
StructType_Name * p = new StructType_Name;
StructType_Name * p1 = new StructType_Name[n];
```

Minh họa cấu trúc và con trỏ

```
#include <iostream>
struct POINT { int A, B; };
int main() {
    POINT a{1, 2}, *x, *y;

    y=&a;
    std::cout << y->A << " "
              << y->B <<
std::endl;
    x =    new POINT;
    x->A=3;
    x->B=4;
    std::cout << x->A << " "
              << x->B;
    delete x;
    return 0;
}
```



Memory Layout

Code minh họa hiển thị giá trị các biến

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

struct POINT { int A, B; };
int main() {
    POINT a{1, 2}, *x, *y;
    y=&a;
    cout << "&a    : " << &a     << endl
         << "&a.A  : " << &a.A   << endl
         << "&a.B  : " << &a.B   << endl;
    cout << "&y    : " << &y     << endl
         << "&y->A : " << &y->A  << endl
         << "(*y).B: " << (*y).B << endl;
    cout << "y->A  : " << y->A   << endl
         << "y->B  : " << y->B   << endl << endl;

    x=new POINT;
    x->A=3;
    x->B=4;
    cout << "&x    : " << &x     << endl
         << "&x->A : " << &x->A  << endl
         << "(*x).B: " << (*x).B << endl;
    cout << "x->A  : " << x->A   << endl
         << "x->B  : " << x->B   << endl << endl;
    return 0;
}
```

Kết quả thực thi:

```
&a  : 0x61fe18
&a.A : 0x61fe18
&a.B : 0x61fe1c
&y  : 0x61fe08
&y->A : 0x61fe18
(*y).B: 2
y->A : 1
y->B : 2

&x  : 0x61fe10
&x->A : 0xf21720
(*x).B: 4
x->A : 3
x->B : 4
```

Câu hỏi 1:

- Tìm chỗ sai trong đoạn code sau:

```
struct SINHVIEN {
        char HoTen[30];
        float DiemToan, DiemLy, DiemHoa;
};
int main() {
        SINHVIEN *sv2;
        strcpy(sv2->HoTen, "Nguyen Van A");
        sv2->DiemToan = 10;
        sv2->DiemHoa = 6.5;
        sv2->DiemLy = 9;
}
```

➢Trả lời:

```
SINHVIEN *sv2=new SINHVIEN;
strcpy(sv2->HoTen, "Nguyen Van A");
```

Câu hỏi 2:

- Tìm chỗ sai:

```
struct SINHVIEN {
	char HoTen[30];
	float DiemToan, DiemLy, DiemHoa;
} sv1;

int main() {
	sv1.HoTen = "Nguyen Van A";
	sv1.DiemToan = 10;
	sv1.DiemHoa = 6.5;
	sv1.DiemLy = 9;
}
```

➢ Trả lời:

```
struct SINHVIEN {
	char HoTen[30];
	float DiemToan, DiemLy, DiemHoa;
} sv1;

int main() {
	strcpy(sv1.HoTen, "Nguyen Van A");
	sv1.DiemToan = 10;
	sv1.DiemHoa = 6.5;
	sv1.DiemLy = 9;
}
```

9.9 Truyền cấu trúc cho hàm

9.9 Truyền cấu trúc cho hàm

- Truyền cấu trúc cho hàm cũng giống như cách truyền kiểu dữ liệu cơ sở cho hàm:
    - Truyền Tham trị (không thay đổi sau khi kết thúc hàm)
    - Truyền Tham chiếu
    - Trả về kiểu cấu trúc (hoặc con trỏ kiểu cấu trúc)

Ví dụ 1:

- Hoàn thành đoạn mã bên để nhập, xuất các biến lưu giá trị kiểu Point.
- Cấu trúc Point như sau:

```
struct Point {
    int x, y;
};
```

```
int main() {
    Point A, B, *C, *D;

    Nhap(A);
    B=Nhap();
    NhapConTro(C);
    D=NhapConTro();

    Xuat(A);
    XuatConTro(&B);
    XuatConTro(C);
    Xuat(*D);

    return 0;
}
```

Ví dụ 1:

```
struct Point {
    int x, y;
};
int main() {
    Point A, B, *C,
*D;

    Nhap(A);
    B=Nhap();
    NhapConTro(C);
    D=NhapConTro();

    Xuat(A);
    XuatConTro(&B);
    XuatConTro(C);
    Xuat(*D);

    return 0;
}
```

```
void Xuat(const Point &D) {
    cout << "(" << D.x << ", " << D.y << ")" <<
endl;
}
void XuatConTro(const Point *D) {
    cout << "(" << D->x << ", " << D->y << ")"
<< endl;
}
```

```
void Nhap(Point &D)
{
    cin >> D.x >>
D.y;
}
Point Nhap() {
    Point D;
    cin >> D.x >>
D.y;
    return D;
}
```

```
void NhapConTro(Point
*&D) {
    D=new Point;
    cin >> D->x >> D->y;
}
Point* NhapConTro() {
    Point *D = new
Point;
    cin >> D->x >> D->y;
    return D;
}
```

Ví dụ 2: Nhập và xuất Danh sách điểm

```
struct Point {
    int x, y;
};

int main() {
    Point *points;
    int n;
    Nhap(points,
n);
    Xuat(points,
n);
    return 0;
}
```

```
void Xuat(Point *a, const int
&n) {
    for(int i=0; i<n; i++){
        Xuat(a[i]);
        cout << endl;
    }
}
```

```
void Nhap(Point *&a, int &n) {
    cin >> n;
    a=new Point[n];
    for(int i=0; i<n; i++)
        Nhap(a[i]);
}
```

9.10 Ví dụ minh họa

Ví dụ 1: Hàm sắp xếp mảng danh sách điểm

```
void HoanVi(Point &A, Point &B) {
    Point temp=A;
    A=B;
    B=temp;
}
void SapXepGiamDanTheoTungDo(Point Diem[100], const int &n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (Diem[i].y < Diem[j].y)
                HoanVi(Diem[i], Diem[j]);
}
```

Ví dụ 2:

- Yêu cầu: Hoàn thành đoạn chương trình nhập danh sách sinh viên dưới đây:

```
#include <iostream>
using namespace std;
#define MAX1 10
#define MAX2 100
struct SinhVien {
    char *MASV; char HoTen[MAX2]; float DiemTin;
};
//### INSERT CODE HERE -
int main() {
    SinhVien *A;
    int n;
    Nhap(A, n);
    Xuat(A, n);
    return 0;
}
```

Ví dụ 2: Hàm Nhập sinh viên

```
void Nhap(SinhVien *&a, int &n) {
    cout << "Nhap n: "; cin >> n;
    a = new SinhVien[n];
    for(int i=0; i<n; i++)
        Nhap(a[i]);
}
void Nhap(SinhVien &a) {
    a.MASV = new char[MAX1];
    cin.getline(a.MASV, MAX1);
    cin.getline(a.HoTen, MAX2);
    cin >> a.DiemTin;
}
```

```
struct SinhVien {
    char *MASV; char HoTen[MAX2]; float
DiemTin;
};
```

Ví dụ 2: Hàm xuất sinh viên

```
struct SinhVien {
    char *MASV; char HoTen[MAX2]; float
DiemTin;
```

```
void Xuat(SinhVien a) {
    cout << a.MASV << "\t";
    cout << a.HoTen << "\t";};
    cout << a.DiemTin << "\t";
}

void Xuat(SinhVien *a, const int &n) {
    cout << "\nXuat: \n";
    for (int i = 0; i < n; i++) {
        Xuat(a[i]);
        cout << "\n";
    }
}
```

Ví dụ 2: Code minh họa

```
#include <iostream>
using namespace std;
#define MAX1 10
#define MAX2 100
struct SinhVien {
    char *MASV; char
HoTen[MAX2];
    float DiemTin;
};
void Nhap(SinhVien &a) {
    a.MASV=new char[MAX1];
    cin.getline(a.MASV,
MAX1);
    cin.getline(a.HoTen,
MAX2);
    cin >> a.DiemTin;
}
void Xuat(SinhVien a) {
    cout << a.MASV << "\t";
    cout << a.HoTen << "\t";
    cout << a.DiemTin <<
```

```
void Nhap(SinhVien *&a, int &n)
{
    cout << "Nhap n: "; cin >>
n;
    a=new SinhVien[n];
    for(int i=0; i<n; i++) {
        cin.ignore();   Nhap(a[
i]);
    }
}
void Xuat(SinhVien *a, const
int &n) {
    cout << "\nXuat: \n";
    for (int i = 0; i < n; i++)
{
        Xuat(a[i]);   cout <<
"\n";
    }
}
int main() {
```

Kết quả thực thi:

```
Nhap n: 3
07520045
Nguyen Van A
9
07520491
Nguyen Van B
10
07520015
Nguyen Van C
7

Xuat:
07520045    Nguyen Van A    9
07520491    Nguyen Van B    10
07520015    Nguyen Van C    7
```

Ví dụ 3:

- **Yêu cầu**: Viết chương trình nhập một đa giác (`Polygon`) có n điểm (`Point`), với 3 ≤ n ≤ 10.
- Lưu ý chương trình phải sử dụng các hàm thao tác nhập xuất 1 điểm.

```
#include<iostream>
using namespace std;
#define MAXN 10
struct Point { int x, y; };
struct Polygon {
    int n;
    Point *p;
};
//### INSERT CODE HERE -
int main() {
    Polygon B;
    B = NhapMotDaGiac();
    XuatMotDaGiac(B);
    return 0;
}
```

Ví dụ 3:

```
#include<iostream>
using namespace std;
#define MAXN 10
struct Point { int x, y;
};
struct Polygon {
    int n;
    Point *p;
};
//### INSERT CODE HERE -
int main() {
    Polygon B;
    B = NhapMotDaGiac();
    XuatMotDaGiac(B);
    return 0;
}
```

```
Polygon NhapMotDaGiac() {
    Polygon d;
    cin >> d.n;
    d.p = new Point[d.n];
    for(int i=0; i<d.n; i++)
        NhapMotDiem(d.p[i]);
    return d;
}
```

```
void XuatMotDaGiac(const
Polygon &d) {
    for(int i=0; i<d.n; i++)
        XuatMotDiem(d.p[i]);
}
```

Bài tập

Bài tập

- Câu 1: Phân số
  - Khai báo kiểu dữ liệu phân số (PHANSO)
  - Nhập/Xuất phân số
  - Kiểm tra phân số âm hay dương
  - Tính tổng, hiệu, tích, thương hai phân số
  - Rút gọn phân số
  - Kiểm tra phân số tối giản
  - Quy đồng hai phân số
  - So sánh hai phân số

Bài tập

- Câu 2: Đơn thức
  - Khai báo kiểu dữ liệu đơn thức (DONTHUC)
  - Nhập/Xuất đơn thức
  - Tính tích, thương hai đơn thức
  - Tính đạo hàm cấp 1 của đơn thức
  - Tính giá trị đơn thức tại $x = x_0$
- Câu 3: Đa thức một biến (gọi tắt là Đa thức)
  - Khai báo kiểu dữ liệu đa thức (DATHUC)
  - Nhập/Xuất đa thức
  - Tính tổng, hiệu, tích đa thức
  - Tính giá trị đa thức tại $x = x_0$

Bài tập

- Câu 4: Điểm trong mặt phẳng Oxy
  - Khai báo kiểu dữ liệu điểm (DIEM)
  - Nhập/Xuất tọa độ điểm
  - Tính khoảng cách giữa hai điểm
  - Tìm điểm đối xứng qua gốc toạ độ/trục Ox/Oy
  - Kiểm tra điểm thuộc phần tư nào?
- Câu 5: Tam giác
  - Khai báo kiểu dữ liệu tam giác (TAMGIAC)
  - Nhập/Xuất tam giác
  - Tính chu vi, diện tích tam giác

Bài tập

- Câu 6: Ngày
  - Khai báo kiểu dữ liệu ngày (NGAY)
  - Nhập/Xuất ngày (ngày, tháng, năm)
  - Kiểm tra năm nhuận
  - Tính số thứ tự ngày trong năm
  - Tính số thứ tự ngày kể từ ngày 1/1/1
  - Tìm ngày trước đó, sau đó k ngày
  - Tính khoảng cách giữa hai ngày
  - So sánh hai ngày

Bài tập

- Câu 7. Mảng phân số
  - Nhập/Xuất n phân số
  - Rút gọn mọi phân số
  - Đếm số lượng phân số âm/dương trong mảng
  - Tìm phân số dương đầu tiên trong mảng
  - Tìm phân số nhỏ nhất/lớn nhất trong mảng
  - Sắp xếp mảng tăng dần/giảm dần

Bài tập

- Câu 8: Mảng điểm
  - Nhập/Xuất n điểm
  - Đếm số lượng điểm có hoành độ dương
  - Đếm số lượng điểm không trùng với các điểm khác trong mảng
  - Tìm điểm có hoành độ lớn nhất/nhỏ nhất
  - Tìm điểm gần gốc tọa độ nhất

Chúc các em học tốt!