



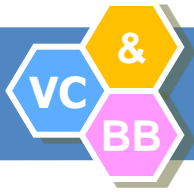
Bộ môn Công nghệ phần mềm  
Khoa Công nghệ thông tin  
Trường Đại học Khoa học Tự nhiên

# KỸ THUẬT LẬP TRÌNH

ThS. Đặng Bình Phương  
dbphuong@fit.hcmus.edu.vn

## DỮ LIỆU KIỂU CẤU TRÚC





# Nội dung

1

**Khái niệm kiểu cấu trúc (struct)**

2

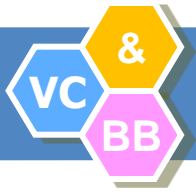
**Khai báo & truy xuất kiểu cấu trúc**

3

**Kiểu dữ liệu hợp nhất (union)**

4

**Bài tập**



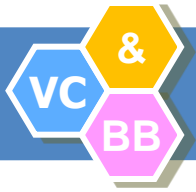
# Đặt vấn đề

## ❖ Thông tin 1 SV

- MSSV: kiểu chuỗi
- Tên SV: kiểu chuỗi
- NTNS: kiểu chuỗi
- Phái: kiểu ký tự
- Điểm Toán, Lý, Hóa: kiểu số thực

## ❖ Yêu cầu

- Lưu thông tin  $n$  SV?
- Truyền thông tin  $n$  SV vào hàm?



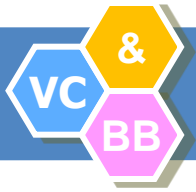
# Đặt vấn đề

## ❖ Khai báo các biến để lưu trữ 1 SV

- `char mssv[8]; // "0012078"`
- `char hoten[30]; // "Nguyen Van A"`
- `char ntns[9]; // "29/12/82"`
- `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);`



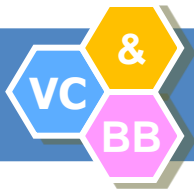
# Đặt vấn đề

## ❖ 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 DIEM
{
    int x;
    int y;
};
```



# Khai báo biến cấu trúc

## ❖ Cú pháp 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 DIEM
{
    int x;
    int y;
} diem1, diem2;
```



# Khai báo biến cấu trúc

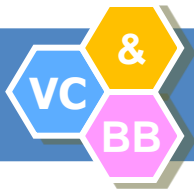
## ❖ Cú pháp không tường minh

```
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>;
```

## ❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
};
struct DIEM diem1, diem2; // C++ có thể bỏ struct
```





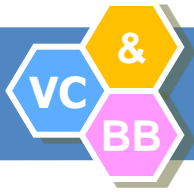
# Sử dụng typedef

## ❖ 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>;
```

## ❖ Ví dụ

```
typedef struct
{
    int x;
    int y;
} DIEM;
struct DIEM diem1, diem2;
```



# Khởi tạo cho biến cấu trúc

## ❖ Cú pháp tường minh

```
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 DIEM
{
    int x;
    int y;
} diem1 = {2912, 1706}, diem2;
```



# Truy xuất dữ liệu kiểu cấu trúc

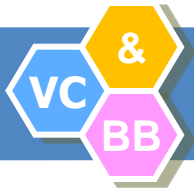
## ❖ Đặc điểm

- 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 thành phần>`

## ❖ Ví dụ

```
struct DIEM
{
    int x;
    int y;
} diem1;
printf("x = %d, y = %d", diem1.x, diem1.y);
```



# Gán dữ liệu kiểu cấu trúc

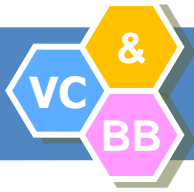
## ❖ Có 2 cách

```
<biến cấu trúc đích> = <biến cấu trúc nguồn>;
```

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

## ❖ Ví dụ

```
struct DIEM
{
    int x, y;
} diem1 = {2912, 1706}, diem2;
...
diem2 = diem1;
diem2.x = diem1.x;
diem2.y = diem1.y * 2;
```



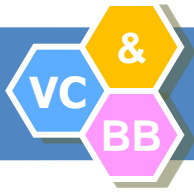
# Cấu trúc phức tạp

## ❖ Thành phần của cấu trúc là cấu trúc khác

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

struct HINHCHUNHAT
{
    struct DIEM traitren;
    struct DIEM phaiduai;
} hcn1;

...
hcn1.traitren.x = 2912;
hcn1.traitren.y = 1706;
```

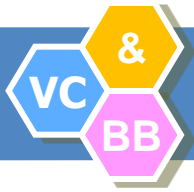


# Cấu trúc phức tạp

## ❖ Thành phần của cấu trúc là mảng

```
struct SINHVIEN
{
    char hoten[30];
    float toan, ly, hoa;
} sv1;

...
strcpy(sv1.hoten, "Nguyen Van A");
sv1.toan = 10;
sv1.ly = 6.5;
sv1.hoa = 9;
```



# Cấu trúc phức tạp

## ❖ Cấu trúc đệ quy (tự trỏ)

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

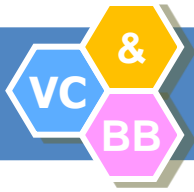
struct NODE
{
    int value;
    struct NODE *pNext;
};
```

## ❖ Thành phần của cấu trúc có kích thước theo bit

```
struct bit_fields
{
    int bit_0 : 1;
    int bit_1_to_4 : 4;
    int bit_5 : 1;
    int bit_6_to_15 : 10;
};
```







# Kích thước của struct

## ❖ Ví dụ

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

```
struct B1
{
    int a;
    int b;
    double c;
};
sizeof(B1) = ???
```

```
struct B2
{
    int a;
    double c;
    int b;
};
sizeof(B2) = ???
```



# 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**
  - Project settings → Compile Option C/C++ → Code Generation → Structure Alignment
- Canh biên cho 1 cấu trúc

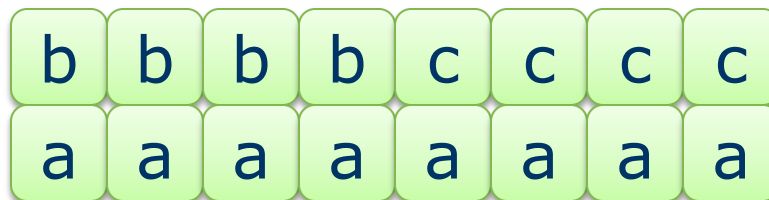
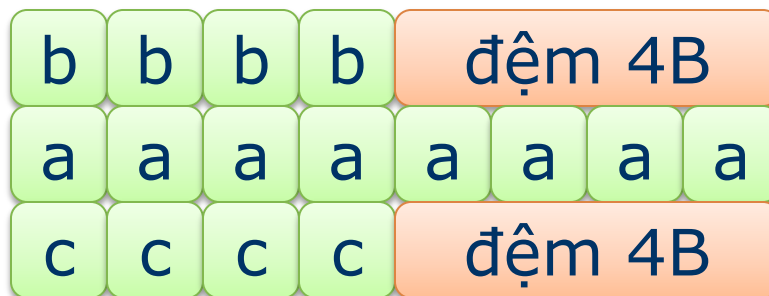
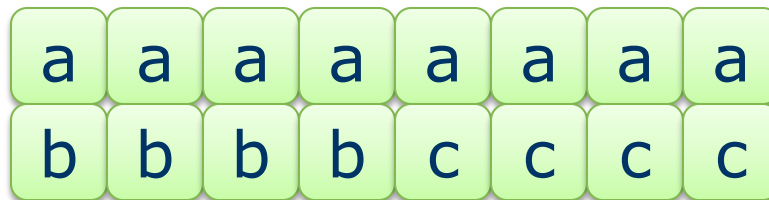
```
#pragma pack(push, 1)  
struct MYSTRUCT { ... };  
#pragma pack(pop)
```



# #pragma pack

## ❖ Ví dụ: không có #pragma pack (1)

```
struct A {  
    double a;  
    int b;  
    int c;  
};  
  
struct B {  
    int b;  
    double a;  
    int c;  
};  
  
struct C {  
    int b;  
    int c;  
    double a;  
};
```





# Các lưu ý về cấu trúc

## ❖ Lưu ý

- Kiểu cấu trúc được định nghĩa để làm khuôn dạng còn biến cấu trúc được khai báo để sử dụng khuôn dạng đã định nghĩa.
- Trong C++, có thể bỏ từ khóa struct khi khai báo biến (hoặc sử dụng typedef)
- Khi nhập các biến kiểu số thực trong cấu trúc phải nhập thông qua một biến trung gian.

```
struct DIEM { float x, y; } d1;  
float temp; scanf("%f", &temp); d1.x = temp;
```



# Mảng cấu trúc

## ❖ Mảng cấu trúc

- Tương tự như mảng với kiểu dữ liệu cơ sở (char, int, float, ...)

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

```
DIEM mang1[20];
```

```
DIEM mang2[10] = {{3, 2}, {4, 4}, {2, 7}};
```

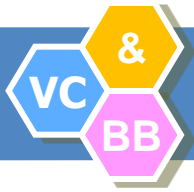


# Truyền cấu trúc cho hàm

## ❖ Truyền cấu trúc cho hàm

- Giống như truyền kiểu dữ liệu cơ sở
  - Tham trị (không thay đổi sau khi kết thúc hàm)
  - Tham chiếu
  - Con trỏ
- Ví dụ

```
struct DIEM { int x, y; };  
  
void xuat1(int x, int y) { ... };  
void xuat2(DIEM diem) { ... };  
void xuat3(DIEM &diem) { ... };  
void xuat4(DIEM *diem) { ... };
```



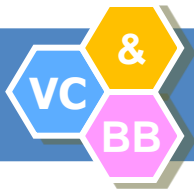
# Hợp nhất – union

## ❖ Khái niệm

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

## ❖ Khai báo

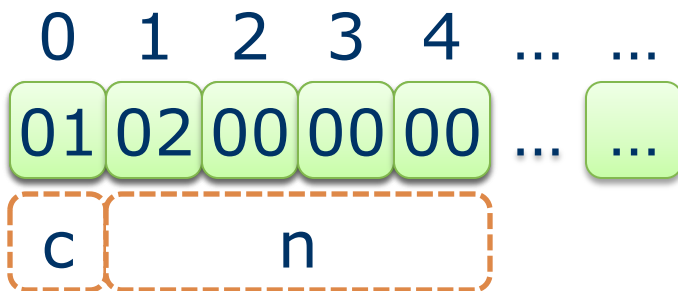
```
union <tên kiểu union>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần 2>;
};
```



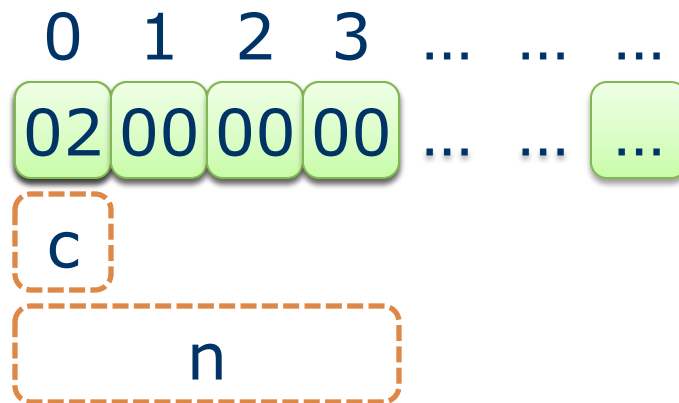
# So sánh struct và union

## ❖ Ví dụ

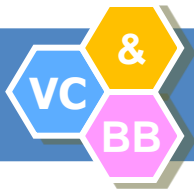
```
struct MYSTRUCT  
{  
    char c;  
    int n;  
} s;  
  
s.c = 1; s.n = 2;
```



```
union MYUNION  
{  
    char c;  
    int n;  
} u;  
  
u.c = 1; u.n = 2;
```



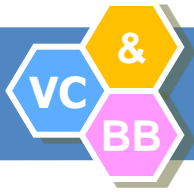




# Ví dụ

## ❖ struct trong union

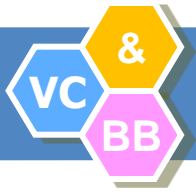
```
union date_tag
{
    char full_date[9];
    struct
    {
        char month[2];
        char break_value1;
        char day[2];
        char break_value2;
        char year[2];
    } part_date_tag;
} date = {"29/12/82"};
```



# Ví dụ

## ❖ union trong struct

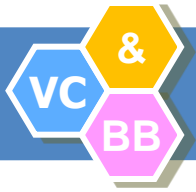
```
struct generic_tag
{
    char type;
    union
    {
        char c;
        int i;
        float f;
    } share_tag;
};
```



# Bài tập

## ❖ Phân số

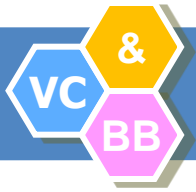
- Khai báo kiểu dữ liệu phân số (PHANSO)
- Nhập/Xuất phân số
- Rút gọn phân số
- Tính tổng, hiệu, tích, thương hai phân số
- Kiểm tra phân số tối giản
- Quy đồng hai phân số
- Kiểm tra phân số âm hay dương
- So sánh hai phân số



# Bài tập

## ❖ 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 đạo hàm cấp k của đơn thức
- Tính giá trị đơn thức tại  $x = x_0$

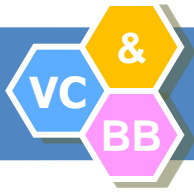


# Bài tập

## ❖ Đa thức

- Khai báo kiểu dữ liệu đa thức (DATHUC)
- Nhập/Xuất đa thức
- Tính tổng, hiệu, tích hai đa thức
- Tính đạo hàm cấp 1 của đa thức
- Tính đạo hàm cấp k của đa thức
- Tính giá trị đơn thức tại  $x = x_0$





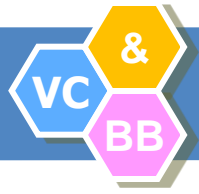
# Bài tập

## ❖ 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 tọa độ/trục Ox/Oy
- Kiểm tra điểm thuộc phần tư nào?

## ❖ Tam giác

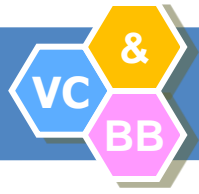
- Khai báo kiểu dữ liệu tam giác (TAMGIAC)
- Nhập/Xuất tam giác
- Tính chu vi, diện tích tam giác



# Bài tập

## ❖ 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

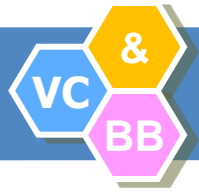


# Bài tập

## ❖ 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

## ❖ Mạng 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

