CHƯƠNG 5

DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP

Kiểu cấu trúc
Cấu trúc tự trỏ và danh sách liên kết
Kiểu hợp
Kiểu liệt kê

Để lưu trữ các giá trị gồm nhiều thành phần dữ liệu giống nhau ta có kiểu biến mảng. Thực tế rất nhiều dữ liệu là tập các kiểu dữ liệu khác nhau tập hợp lại, để quản lý dữ liệu kiểu này C++ đưa ra kiểu dữ liệu cấu trúc. Một ví dụ của dữ liệu kiểu cấu trúc là một bảng lý lịch trong đó mỗi nhân sự được lưu trong một bảng gồm nhiều kiểu dữ liệu khác nhau như họ tên, tuổi, giới tính, mức lương …

I. KIỂU CẤU TRÚC

1. Khai báo, khởi tạo

Để tạo ra một kiểu cấu trúc NSD cần phải khai báo tên của kiểu (là một tên gọi do NSD tự đặt), tên cùng với các thành phần dữ liệu có trong kiểu cấu trúc này. Một kiểu cấu trúc được khai báo theo mẫu sau:

```
struct <tên kiểu>
{
    các thành phần ;
} <danh sách biến>;
```

- Mỗi thành phần giống như một biến riêng của kiểu, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.
- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy (;).
- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.
- Một biến có kiểu cấu trúc sẽ được phân bố bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

– Khai báo biến kiểu cấu trúc cũng giống như khai báo các biến kiểu cơ sở dưới dạng:

struct <tên cấu trúc> <danh sách biến> ; // kiểu cũ trong C

hoặc

<tên cấu trúc> <danh sách biến> ; // trong C++

Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên cấu trúc> biến = { giá trị khởi tạo } ;

Ví dụ:

– Khai báo kiểu cấu trúc chứa phân số gồm 2 thành phần nguyên chứa tử số và mẫu số.

```
struct Phanso
{
        int tu ;
        int mau ;
} ;
```

hoặc:

```
struct Phanso { int tu, mau ; }
```

– Kiểu ngày tháng gồm 3 thành phần nguyên chứa ngày, tháng, năm.

```
struct Ngaythang {
        int ng ;
        int th ;
        int nam ;
} holiday = { 1,5,2000 } ;
```

một biến holiday cũng được khai báo kèm cùng kiểu này và được khởi tạo bởi bộ số 1. 5. 2000. Các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức ng = 1, th = 5 và nam = 2000.

– Kiểu Lop dùng chứa thông tin về một lớp học gồm tên lớp và sĩ số sinh viên. Các biến kiểu Lop được khai báo là daihoc và caodang, trong đó daihoc được khởi tạo bởi bộ giá trị {"K41T", 60} với ý nghĩa tên lớp đại học là K41T và sĩ số là 60 sinh viên.

```
struct Lop {
        char tenlop[10],
```

```
        int soluong;
    } ;
    struct Lop daihoc = {"K41T", 60}, caodang ;
```

hoặc:

```
    Lop daihoc = {"K41T", 60}, caodang ;
```

– Kiểu Sinhvien gồm có các trường hoten để lưu trữ họ và tên sinh viên, ns lưu trữ ngày sinh, gt lưu trữ giới tính dưới dạng số (qui ước 1: nam, 2: nữ) và cuối cùng trường diem lưu trữ điểm thi của sinh viên. Các trường trên đều có kiểu khác nhau.

```
    struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
    } x, *p, K41T[60];
    Sinhvien y = {"NVA", {1,1,1980}, 1} ;
```

Khai báo cùng với cấu trúc Sinhvien có các biến x, con trỏ p và mảng K41T với 60 phần tử kiểu Sinhvien. Một biến y được khai báo thêm và kèm theo khởi tạo giá trị {"NVA", {1,1,1980}, 1}, tức họ tên của sinh viên y là "NVA", ngày sinh là 1/1/1980, giới tính nam và điểm thi để trống. Đây là kiểu khởi tạo thiếu giá trị, giống như khởi tạo mảng, các giá trị để trống phải nằm ở cuối bộ giá trị khởi tạo (tức các thành phần bỏ khởi tạo không được nằm xen kẽ giữa những thành phần được khởi tạo).Ví dụ này còn minh hoạ cho các cấu trúc lồng nhau, cụ thể trong kiểu cấu trúc Sinhvien có một thành phần cũng kiểu cấu trúc là thành phần ns.

2. Truy nhập các thành phần kiểu cấu trúc

Để truy nhập vào các thành phần kiểu cấu trúc ta sử dụng cú pháp: tên biến.tên thành phần hoặc tên biến → tên thành phần đối với biến con trỏ cấu trúc. Cụ thể:

– Đối với biến thường: **tên biến.tên thành phần**

Ví dụ:

```
    struct Lop {
        char tenlop[10];
        int siso;
```

```
} ;
Lop daihoc = "K41T", caodang ;
caodang.tenlop = daihoc.tenlop ;      // gán tên lớp cđẳng bởi tên lớp đhọc
caodang.siso++;                       // tăng sĩ số lớp caodang lên 1
```

– Đối với biến con trỏ: **tên biến → tên thành phần**

Ví dụ:

```
struct Sinhvien {
      char hoten[25] ;
      Ngaythang ns;
      int gt;
      float diem ;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1} ;
y.diem = 5.5 ;                        // gán điểm thi cho sinh viên y
p = new Sinhvien ;                    // cấp bộ nhớ chứa 1 sinh viên
strcpy(p→hoten, y.hoten) ;            // gán họ tên của y cho sv trỏ bởi p
cout << p→hoten << y.hoten;           // in hoten của y và con trỏ p
```

– Đối với biến mảng: truy nhập thành phần mảng rồi đến thành phần cấu trúc.

Ví dụ:

```
strcpy(K41T[1].hoten, p→hoten) ;      // gán họ tên cho sv đầu tiên của lớp
K41T[1].diem = 7.0 ;                  // gán điểm cho sv đầu tiên
```

– Đối với cấu trúc lồng nhau. Truy nhập thành phần ngoài rồi đến thành phần của cấu trúc bên trong, sử dụng các phép toán . hoặc → (các phép toán lấy thành phần) một cách thích hợp.

```
x.ngaysinh.ng = y.ngaysinh.ng ;       // gán ngày,
x.ngaysinh.th = y.ngaysinh.th ;       // tháng,
x.ngaysinh.nam = y.ngaysinh.nam ;     // năm sinh của y cho x.
```

3. Phép toán gán cấu trúc

Cũng giống các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Ví dụ vào/ra một biến cấu trúc phải viết

câu lệnh vào/ra từng cho từng thành phần. Nhận xét này được minh họa trong ví dụ sau:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
cin >> x.diem
cout << "Thông tin về sinh viên x là:" << endl ;
cout << "Họ và tên: " << x.hoten << endl;
cout << "Sinh ngày: " << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam ;
cout << "Giới tính: " << (x.gt == 1) ? "Nam": "Nữ ;
cout << x.diem
```

Tuy nhiên, khác với biến mảng, **đối với cấu trúc chúng ta có thể gán giá trị của 2 biến cho nhau**. Phép gán này cũng tương đương với việc gán từng thành phần của cấu trúc. Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y, *p ;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
```

```
cin >> x.diem

y = x ;       // Đối với biến mảng, phép gán này là không thực hiện được
p = new Sinhvien[1] ; *p = x ;

cout << "Thông tin về sinh viên y là:" << endl ;
cout << "Họ và tên: " << y.hoten << endl;
cout << "Sinh ngày: " << y.ns.ng << "/" << y.ns.th << "/" << y.ns.nam ;
cout << "Giới tính: " << (y.gt = 1) ? "Nam": "Nữ ;
cout << y.diem
```

Chú ý: không gán bộ giá trị cụ thể cho biến cấu trúc. Cách gán này chỉ thực hiện được khi khởi tạo. Ví dụ:

```
Sinhvien x = { "NVA", {1,1,1980}, 1, 7.0}, y ;          // được
y = { "NVA", {1,1,1980}, 1, 7.0};                       // không được
y = x;                                                  // được
```

4. Các ví dụ minh hoạ

Dưới đây chúng ta đưa ra một vài ví dụ minh hoạ cho việc sử dụng kiểu cấu trúc.

Ví dụ 1 : Cộng, trừ, nhân chia hai phân số được cho dưới dạng cấu trúc.

```
#include <iostream.h>
#include <conio.h>
struct Phanso {
      int tu ;
      int mau ;
} a, b, c ;

void main()
{
      clrscr();
      cout << "Nhập phân số a:" << endl ;          // nhập a
      cout << "Tử:"; cin >> a.tu;
      cout << "Mẫu:"; cin >> a.mau;
```

```
        cout << "Nhập phân số b:" << endl ;              // nhập b
        cout << "Tử:"; cin >> b.tu;
        cout << "Mẫu:"; cin >> b.mau;
        c.tu = a.tu*b.mau + a.mau*b.tu;                 // tính và in a+b
        c.mau = a.mau*b.mau;
        cout << "a + b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.mau - a.mau*b.tu;                 // tính và in a-b
        c.mau = a.mau*b.mau;
        cout << "a - b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.tu;                               // tính và in axb
        c.mau = a.mau*b.mau;
        cout << "a + b = " << c.tu << "/" << c.mau;
        c.tu = a.tu*b.mau;                              // tính và in a/b
        c.mau = a.mau*b.tu;
        cout << "a + b = " << c.tu << "/" << c.mau;
        getch();
}
```

Ví dụ 2 : Nhập mảng K41T. Tính tuổi trung bình của sinh viên nam, nữ. Hiện danh sách của sinh viên có điểm thi cao nhất.

```
#include <iostream.h>
#include <conio.h>
void main()
{
        struct Sinhvien {
                char hoten[25] ;
                Ngaythang ns;
                int gt;
                float diem ;
        } x, K41T[60];
        int i, n;
```

```
        // nhập dữ liệu
        cout << "Cho biết số sinh viên: "; cin >> n;
        for (i=1, i<=n, i++)
        {
                cout << "Nhap sinh vien thu " << i);
                cout << "Ho ten: " ; cin.getline(x.hoten);
                cout << "Ngày sinh: " ; cin >> x.ns.ng >> x.ns.th >>x.ns.nam ;
                cout << "Giới tính: " ; cin >> x.gt ;
                cout << "Điểm: " ; cin >> x.diem ;
                cin.ignore();
                K41T[i] = x ;
        }
}

// Tính điểm trung bình
float tbnam = 0, tbnu = 0;
int sonam = 0, sonu = 0 ;
for (i=1; i<=n; i++)
        if (K41T[i].gt == 1) { sonam++ ; tbnam += K41T[1].diem ; }
else { sonu++ ; tbnu += K41T[1].diem ; }
cout << "Điểm trung bình của sinh viên nam là " << tbnam/sonam ;
cout << "Điểm trung bình của sinh viên nữ là " << tbnu/sonu ;

// In danh sách sinh viên có điểm cao nhất
float diemmax = 0;
for (i=1; i<=n; i++)                                    // Tìm điểm cao nhất
if (diemmax < K41T[i].diem) diemmax = K41T[i].diem ;

for (i=1; i<=n; i++)                                    // In danh sách
{
        if (K41T[i].diem < diemmax) continue ;
```

```
            x = K41T[1] ;
            cout << x.hoten << '\t' ;
            cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << '\t' ;
            cout << (x.gt == 1) ? "Nam": "Nữ" << '\t' ;
            cout << x.diem << endl;
        }
    }
```

5. Hàm với cấu trúc

a. Con trỏ và địa chỉ cấu trúc

Một con trỏ cấu trúc cũng giống như con trỏ trỏ đến các kiểu dữ liệu khác, có nghĩa nó chứa địa chỉ của một biến cấu trúc hoặc một vùng nhớ có kiểu cấu trúc nào đó. Một con trỏ cấu trúc được khởi tạo bởi:

- Gán địa chỉ của một biến cấu trúc, một thành phần của mảng, tương tự nếu địa chỉ của mảng (cũng là địa chỉ của phần tử đầu tiên của mảng) gán cho con trỏ thì ta cũng gọi là con trỏ mảng cấu trúc. Ví dụ:

```
struct Sinhvien {
  char hoten[25] ;
  Ngaythang ns;
  int gt;
  float diem ;
} x, y, *p, lop[60];

p = &x ;                    // cho con trỏ p trỏ tới biến cấu trúc x
p→diem = 5.0;               // gán giá trị 5.0 cho điểm của biến x
p = &lop[10] ;              // cho p trỏ tới sinh viên thứ 10 của lớp
cout << p→hoten;            // hiện họ tên của sinh viên này
*p = y ;                    // gán lại sinh viên thứ 10 là y
(*p).gt = 2;                // sửa lại giới tính của sinh viên thứ 10 là nữ
```

Chú ý: thông qua ví dụ này ta còn thấy một cách khác nữa để truy nhập các thành phần của x được trỏ bởi con trỏ p. Khi đó *p là tương đương với x, do vậy ta dùng lại cú

pháp sử dụng toán tử . sau *p để lấy thành phần như (*p).hoten, (*p).diem, …

- Con trỏ được khởi tạo do xin cấp phát bộ nhớ. Ví dụ:

```
Sinhvien *p, *q ;
p = new Sinhvien[1];
q = new Sinhvien[60];
```

trong ví dụ này *p có thể thay cho một biến kiểu sinh viên (tương đương biến x ở trên) còn q có thể được dùng để quản lý một danh sách có tối đa là 60 sinh viên (tương đương biến lop[60], ví dụ khi đó *(p+10) là sinh viên thứ 10 trong danh sách).

- Đối với con trỏ p trỏ đến mảng a, chúng ta có thể sử dụng một số cách sau để truy nhập đến các trường của các thành phần trong mảng, ví dụ để truy cập hoten của thành phần thứ i của mảng a ta có thể viết:
  - p[i].hoten
  - (p+i)→hoten
  - *(p+i).hoten

Nói chung các cách viết trên đều dễ nhớ do suy từ kiểu mảng và con trỏ mảng. Cụ thể trong đó p[i] là thành phần thứ i của mảng a, tức a[i]. (p+i) là con trỏ trỏ đến thành phần thứ i và *(p+i) chính là a[i]. Ví dụ sau gán giá trị cho thành phần thứ 10 của mảng sinh viên lop, sau đó in ra màn hình các thông tin này. Ví dụ dùng để minh hoạ các cách truy nhập trường dữ liệu của thành phần trong mảng lop.

Ví dụ 3 :

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60] ;

strcpy(lop[10].hoten, "NVA");
lop[10].gt = 1; lop[10].diem = 9.0 ;
Sinhvien *p ;                    // khai báo thêm biến con trỏ Sinh viên
p = &lop ;                       // cho con trỏ p trỏ tới mảng lop
```

```
cout << p[10].hoten ;          // in họ tên sinh viên thứ 10
cout << (p+10) → gt ;          // in giới tính của sinh viên thứ 10
cout << (*(p+10)).diem ;       // in điểm của sinh viên thứ 10
```

Chú ý: Độ ưu tiên của toán tử lấy thành phần (dấu chấm) là cao hơn các toán tử lấy địa chỉ (&) và lấy giá trị (*) nên cần phải viết các dấu ngoặc đúng cách.

b. Địa chỉ của các thành phần của cấu trúc

Các thành phần của một cấu trúc cũng giống như các biến, do vậy cách lấy địa chỉ của các thành phần này cũng tương tự như đối với biến bình thường. Chẳng hạn địa chỉ của thành phần giới tính của biến cấu trúc x là &x.gt (lưu ý độ ưu tiên của . cao hơn &, nên &x.gt là cũng tương đương với &(x.gt)), địa chỉ của trường hoten của thành phần thứ 10 trong mảng lớp là lop[10].hoten (hoten là xâu kí tự), tương tự địa chỉ của thành phần diem của biến được trỏ bởi p là &(p→diem).

Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60], *p, x = { "NVA", {1,1,1980}, 1, 9.0) };
        lop[10] = x; p = &lop[10] ;     // p trỏ đến sinh viên thứ 10 trong lop
        char *ht; int *gt; float *d;    // các con trỏ kiểu thành phần
        ht = x.ht ;                     // cho ht trỏ đến thành phần hoten của x
        gt = &(lop[10].gt) ;            // gt trỏ đến gt của sinh viên thứ 10
        d = &(p→diem) ;                 // p trỏ đến diem của sv p đang trỏ
        cout << ht ;                    // in họ tên sinh viên x
        cout << *gt ;                   // in giới tính của sinh viên thứ 10
        cout << *d ;                    // in điểm của sinh viên p đang trỏ.
```

c. Đối của hàm là cấu trúc

Một cấu trúc có thể được sử dụng để làm đối của hàm dưới các dạng sau đây:

– Là một biến cấu trúc, khi đó tham đối thực sự là một cấu trúc.

- Là một con trỏ cấu trúc, tham đối thực sự là địa chỉ của một cấu trúc.
- Là một tham chiếu cấu trúc, tham đối thực sự là một cấu trúc.
- Là một mảng cấu trúc hình thức hoặc con trỏ mảng, tham đối thực sự là tên mảng cấu trúc.

Ví dụ 4 : Ví dụ sau đây cho phép tính chính xác khoảng cách của 2 ngày tháng bất kỳ, từ đó có thể suy ra thứ của một ngày tháng bất kỳ. Đối của các hàm là một biến cấu trúc.

- Khai báo

```
struct DATE {                                   // Kiểu ngày tháng
  int ngay ;
  int thang;
  int nam ;
};
// Số ngày của mỗi tháng
int n[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
```

- Hàm tính năm nhuận hay không nhuận, trả lại 1 nếu năm nhuận, ngược lại trả 0.

```
int Nhuan(int nm)
{
  return (nam%4==0 && nam%100!=0 || nam%400==0)? 1: 0;
}
```

- Hàm trả lại số ngày của một tháng bất kỳ. Nếu năm nhuận và là tháng hai số ngày của tháng hai (28) được cộng thêm 1.

```
int Ngayct(int thang, int nam)
{
  return n[thang] + ((thang==2) ? Nhuan(nam): 0);
}
```

- Hàm trả lại số ngày tính từ ngày 1 tháng 1 năm 1 bằng cách cộng dồn số ngày của từng năm từ năm 1 đến năm hiện tại, tiếp theo cộng dồn số ngày từng tháng của năm hiện tại cho đến tháng hiện tại và cuối cùng cộng thêm số ngày hiện tại.

```
long Tongngay(DATE d)
{
  long i, kq = 0;
  for (i=1; i<d.nam; i++) kq += 365 + Nhuan(i);
  for (i=1; i<d.thang; i++) kq += Ngayct(i,d.nam);
  kq += d.ngay;
  return kq;
}
```

- Hàm trả lại khoảng cách giữa 2 ngày bất kỳ.

```
long Khoangcach(DATE d1, DATE d2)
{
  return Tongngay(d1)-Tongngay(d2);
}
```

- Hàm trả lại thứ của một ngày bất kỳ. Qui ước 1 là chủ nhật, 2 là thứ hai, … Để tính thứ hàm dựa trên một ngày chuẩn nào đó (ở đây là ngày 1/1/2000, được biết là thứ bảy). Từ ngày chuẩn này nếu cộng hoặc trừ 7 sẽ cho ra ngày mới cũng là thứ bảy. Từ đó, tính khoảng cách giữa ngày cần tính thứ và ngày chuẩn. Tìm phần dư của phép chia khoảng cách này với 7, nếu phần dư là 0 thì thứ là bảy, phần dư là 1 thì thứ là chủ nhật …

```
int Thu(DATE d)
{
  DATE curdate = {1,1,2000};                    // ngày 1/1/2000 là thứ bảy
  long kc = Khoangcach(d, curdate);
  int du = kc % 7; if (du < 0) du += 7;
  return du;
}
```

- Hàm dịch một số dư sang thứ

```
char* Dich(int t)
{
  char* kq = new char[10];
  switch (t) {
  case 0: strcpy(kq, "thứ bảy"); break;
        case 1: strcpy(kq, "chủ nhật"); break;
  case 2: strcpy(kq, "thứ hai"); break;
        case 3: strcpy(kq, "thứ ba"); break;
  case 4: strcpy(kq, "thứ tư"); break;
  case 5: strcpy(kq, "thứ năm"); break;
        case 6: strcpy(kq, "thứ sáu"); break;
  }
  return kq;
}
```

- Hàm main()

```
void main()
{
  DATE d;
  cout << "Nhap ngay thang nam: " ;
  cin >> d.ngay >> d.thang >> d.nam ;
  cout << "Ngày " << d.ngay << "/" << d.thang << "/" << d.nam ;
  cout << " là " << Dich(Thu(d));
}
```

Ví dụ 5 : Chương trình đơn giản về quản lý sinh viên.

- Khai báo.

```
struct Sinhvien {                    // cấu trúc sinh viên
  char hoten[25] ;
  Ngaythang ns;
  int gt;
```

```
    float diem ;
} ;
Sinhvien lop[3];                    // lớp chứa tối đa 3 sinh viên
```

- Hàm in thông tin về sinh viên sử dụng biến cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void in(Sinhvien x)
{
    cout << x.hoten << "\t" ;
    cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << "\t" ;
    cout << x.gt << "\t";
    cout << x.diem << endl;
}
```

- Hàm nhập thông tin về sinh viên sử dụng con trỏ sinh viên làm đối. Trong lời gọi sử dụng địa chỉ của một cấu trúc để truyền cho hàm.

```
void nhap(Sinhvien *p)
{
    cin.ignore();
    cout << "Họ tên: "; cin.getline(p→hoten, 25) ;
    cout << "Ngày sinh: ";
    cin >> (p→ns).ng >> (p→ns).th >> (p→ns).nam ;
    cout << "Giới tính: "; cin >> (p→gt) ;
    cout << "Điểm: "; cin >> (p→diem) ;
}
```

- Hàm sửa thông tin về sinh viên sử dụng tham chiếu cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void sua(Sinhvien &r)
{
    int chon;
    do {
        cout << "1: Sửa họ tên" << endl ;
        cout << "2: Sửa ngày sinh" << endl ;
```

```
            cout << "3: Sửa giới tính" << endl ;
            cout << "4: Sửa điểm" << endl ;
            cout << "0: Thôi" << endl ;
            cout << "Sửa (0/1/2/3/4) ? ; cin >> chon ; cin.ignore();
            switch (chon) {
                case 1: cin.getline(r.hoten, 25) ; break;
                case 2: cin >> r.ns.ng >> r.ns.th >> r.ns.nam ; break;
                case 3: cin >> r.gt ; break;
                case 4: cin >> r.diem ; break;
            }
        } while (chon) ;
    }
```

- Hàm nhapds nhập thông tin của mọi sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void nhapds(Sinhvien *a)
{
    int sosv = sizeof(lop) / sizeof(Sinhvien) -1;        // bỏ phần tử 0
    for (int i=1; i<=sosv; i++) nhap(&a[i]) ;
}
```

- Hàm suads cho phép sửa thông tin của sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void suads(Sinhvien *a)
{
    int chon;
    cout << "Chọn sinh viên cần sửa: " ; cin >> chon ; cin.ignore();
    sua(a[chon]) ;
}
```

- Hàm inds hiện thông tin của mọi sinh viên trong mảng, sử dụng hằng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void hien(const Sinhvien *a)
{
        int sosv = sizeof(lop) / sizeof(Sinhvien) -1;        // bỏ phần tử 0
        for (int i=1; i<=sosv; i++) in(a[i]) ;
}
```

- Hàm main() gọi chạy các hàm trên để nhập, in, sửa danh sách sinh viên.

```
void main()
{
    nhapds(lop) ;
    inds(lop);
    suads(lop);
    inds(lop);
}
```

d. Giá trị hàm là cấu trúc

Cũng tương tự như các kiểu dữ liệu cơ bản, giá trị trả lại của một hàm cũng có thể là các cấu trúc dưới các dạng sau:

- là một biến cấu trúc.
- là một con trỏ cấu trúc.
- là một tham chiếu cấu trúc.

Sau đây là các ví dụ minh hoạ giá trị cấu trúc của hàm.

Ví dụ 6 : Đối và giá trị của hàm là cấu trúc: Cộng, trừ hai số phức.

– Khai báo kiểu số phức

```
struct Sophuc                        // Khai báo kiểu số phức dùng chung
{
        float thuc;
        float ao;
};
```

- Hàm cộng 2 số phức, trả lại một số phức

```
Sophuc Cong(Sophuc x, Sophuc y)
{
```

```
        Sophuc kq;
        kq.thuc = x.thuc + y.thuc ;
        kq.ao = x.ao + y.ao ;
        return kq;
}
```

- Hàm trừ 2 số phức, trả lại một số phức

```
Sophuc Tru(Sophuc x, Sophuc y)
{
        Sophuc kq;
        kq.thuc = x.thuc + y.thuc ;
        kq.ao = x.ao + y.ao ;
        return kq;
}
```

- Hàm in một số phức dạng (r + im)

```
void In(Sophuc x)
{
        cout << "(" << x.thuc << "," << x.ao << ")" << endl ;
}
```

- Hàm chính

```
void main()
{
        Sophuc x, y, z ;
        cout << "x = " ; cin >> x.thuc >> x.ao ;
        cout << "y = " ; cin >> y.thuc >> y.ao ;
        cout << "x + y = " ; In(Cong(x,y)) ;
        cout << "x - y = " ; In(Tru(x,y)) ;
}
```

Ví dụ 7 : Chương trình nhập và in thông tin về một lớp cùng sinh viên có điểm cao nhất lớp.

- Khai báo kiểu dữ liệu Sinh viên và biến mảng lop.

```
struct Sinhvien {
        char *hoten ;
        float diem ;
} lop[4] ;
```

- Hàm nhập sinh viên, giá trị trả lại là một con trỏ trỏ đến dữ liệu vừa nhập.

```
Sinhvien* nhap()
{
        Sinhvien* kq = new Sinhvien[1];          // nhớ cấp phát vùng nhớ
        kq->hoten = new char[15];                // cho cả con trỏ hoten
        cout << "Họ tên: "; cin.getline(kq->hoten,30);
        cout << "Điểm: "; cin >> kq->diem; cin.ignore();
        return kq;                               // trả lại con trỏ kq
}
```

- Hàm tìm sinh viên có điểm cao nhất, giá trị trả lại là một tham chiếu đến sinh viên tìm được.

```
Sinhvien& svmax()
{
        int sosv = sizeof(lop)/sizeof(Sinhvien)-1;   // bỏ thành phần thứ 0
        float maxdiem = 0;
        int kmax;                                    // chỉ số sv có điểm max
        for (int i=1; i<sosv; i++)
        if (maxdiem < lop[i].diem)
        {
                maxdiem = lop[i].diem ;
                kmax = i;
        }
        return lop[kmax];                            // trả lại sv có điểm max
```

```
}
```

- Hàm in thông tin của một sinh viên x

```
void in(Sinhvien x)
{
        cout << x.hoten << "\t";
        cout << x.diem << endl;
}
```

- Hàm chính

```
void main()
{
        clrscr();
        int i;
        int sosv = sizeof(lop)/sizeof(Sinhvien)-1;    // bỏ thành phần thứ 0
        for (i=1; i<=sosv; i++) lop[i] = *nhap();       // nhập danh sách lớp
        for (i=1; i<=sosv; i++) in(lop[i]);             // in danh sách lớp
        Sinhvien &b = svmax();          // khai báo tham chiếu b và cho
                                        // tham chiếu đến sv có điểm max
        in(b);                          // in sinh viên có điểm max
        getch();
}
```

6. Cấu trúc với thành phần kiểu bit

a. Trường bit

Thông thường các trường trong một cấu trúc thường sử dụng ít nhất là 2 byte tức 16 bit. Trong nhiều trường hợp một số trường có thể chỉ cần đến số bit ít hơn, ví dụ trường gioitinh thông thường chỉ cần đến 1 bit để lưu trữ. Những trường hợp như vậy ta có thể khai báo kiểu bit cho các trường này để tiết kiệm bộ nhớ. Tuy nhiên, cách khai báo này ít được sử dụng trừ khi cần thiết phải truy nhập đến mức bit của dữ liệu trong các chương trình liên quan đến hệ thống.

Một trường bit là một khai báo trường int và thêm dấu: cùng số bit n theo sau,