```
struct <tên kiểu>
{
        các thành phần chứa thông tin … ;
        <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
struct Sinhvien
{
        char hoten[30] ;          // thành phần chứa thông tin
        float diem ;              // thành phần chứa thông tin
        Sinhvien *tiep ;          // con trỏ chứa địa chỉ của phần tử tiếp.
} ;
```

Trong các cách trên ta thấy 2 cách khai báo cuối cùng là đơn giản nhất. C++ quan niệm các tên gọi đứng sau các từ khoá struct, union, enum là các tên kiểu (dù không có từ khoá typedef), do vậy có thể sử dụng các tên này để khai báo.

2. Khái niệm danh sách liên kết

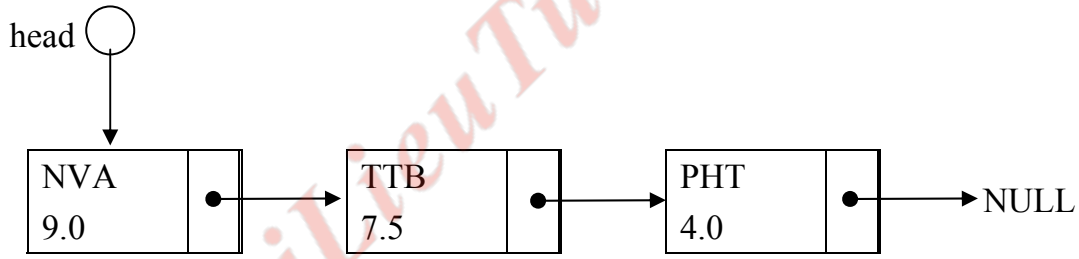
Danh sách liên kết là một cấu trúc dữ liệu cho phép thể hiện và quản lý danh sách bằng các cấu trúc liên kết với nhau thông qua các con trỏ trong cấu trúc. Có nhiều dạng danh sách liên kết phụ thuộc vào các kết nối, ví dụ:

- Danh sách liên kết đơn, mỗi cấu trúc chứa một con trỏ trỏ đến cấu trúc tiếp theo hoặc trước đó. Đối với danh sách con trỏ trỏ về trước, cấu trúc đầu tiên của danh sách sẽ trỏ về hằng con trỏ NULL, cấu trúc cuối cùng được đánh dấu bởi con trỏ last là con trỏ trỏ vào cấu trúc này. Đối với danh sách con trỏ trỏ về cấu trúc tiếp theo, cấu trúc đầu sẽ được đánh dấu bằng con trỏ head và cấu trúc cuối cùng chưa con trỏ NULL.
- Danh sách liên kết kép gồm 2 con trỏ, một trỏ đến cấu trúc trước và một trỏ đến cấu trúc sau, 2 đầu của danh sách được đánh dấu bởi các con trỏ head, last.
- Danh sách liên kết vòng gồm 1 con trỏ trỏ về sau (hoặc trước), hai đầu của danh sách được nối với nhau tạo thành vòng tròn. Chỉ cần một con trỏ head để đánh dấu đầu danh sách.

Do trong cấu trúc có chứa các con trỏ trỏ đến cấu trúc tiếp theo và/hoặc cấu trúc đứng trước nên từ một cấu trúc này chúng ta có thể truy cập đến một cấu trúc khác

(trước và/hoặc sau nó). Kết hợp với các con trỏ đánh dấu 2 đầu danh sách (head, last) chúng ta sẽ dễ dàng làm việc với bất kỳ phần tử nào của danh sách. Có thể kể một số công việc thường thực hiện trên một danh sách như: bổ sung phần tử vào cuối danh sách, chèn thêm một phần tử mới, xoá một phần tử của danh sách, tìm kiếm, sắp xếp danh sách, in danh sách …

Hình vẽ bên dưới minh hoạ một danh sách liên kết đơn quản lý sinh viên, thông tin chứa trong mỗi phần tử của danh sách gồm có họ tên sinh viên, điểm. Ngoài ra mỗi phần tử còn chứa con trỏ tiep để nối với phần tử tiếp theo của nó. Phần tử cuối cùng nối với cấu trúc rỗng (NULL).



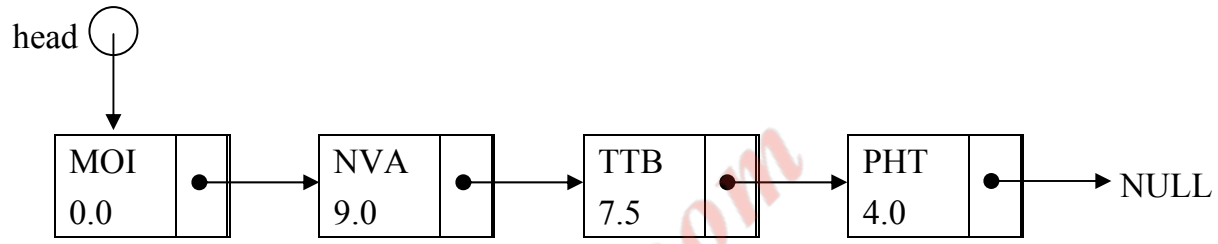
3. Các phép toán trên danh sách liên kết

Dưới đây chúng ta mô tả tóm tắt cách thức thực hiện một số thao tác trên danh sách liên kết đơn.

a. Tạo phần tử mới

Để tạo phần tử mới thông thường chúng ta thực hiện theo các bước sau đây:

- dùng toán tử new xin cấp phát một vùng nhớ đủ chứa một phần tử của danh sách.
- nhập thông tin cần lưu trữ vào phần tử mới. Con trỏ tiep được đặt bằng NULL.
- gắn phần tử vừa tạo được vào danh sách. Có hai cách:
  - hoặc gắn vào đầu danh sách, khi đó vị trí của con trỏ head (chỉ vào đầu danh sách) được điều chỉnh lại để chỉ vào phần tử mới.
  - hoặc gắn vào cuối danh sách bằng cách cho con trỏ tiep của phần tử cuối danh sách (đang trỏ vào NULL) trỏ vào phần tử mới. Nếu danh sách có con trỏ last để chỉ vào cuối danh sách thì last được điều chỉnh để trỏ vào phần tử mới. Nếu danh sách không có con trỏ last thì để tìm được phần tử cuối chương trình phải duyệt từ đầu, bắt đầu từ con trỏ head cho đến khi gặp phần tử trỏ vào NULL, đó là phần tử cuối của danh sách.

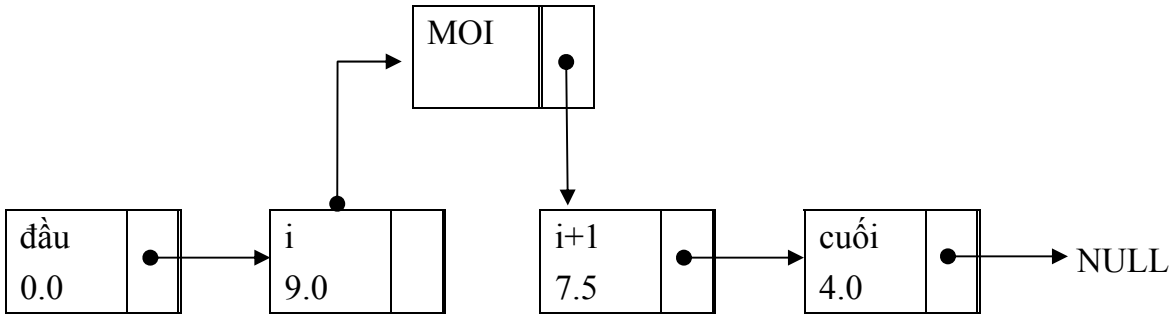


Gắn phần tử mới vào đầu danh sách

b. Chèn phần tử mới vào giữa

Giả sử phần tử mới được chèn vào giữa phần tử thứ i và i+1. Để chèn ta nối phần tử thứ i vào phần tử mới và phần tử mới nối vào phần tử thứ i+1. Thuật toán sẽ như sau:

- Cho con trỏ p chạy đến phần tử thứ i.
- Cho con trỏ tiep của phần tử mới trỏ vào phần tử thứ i+1 (tuc p->tiep).
- Cho con trỏ tiep của phần tử thứ i (hiện được trỏ bởi p) thay vì trỏ vào phần tử thứ i+1 bây giờ sẽ trỏ vào phần tử mới.



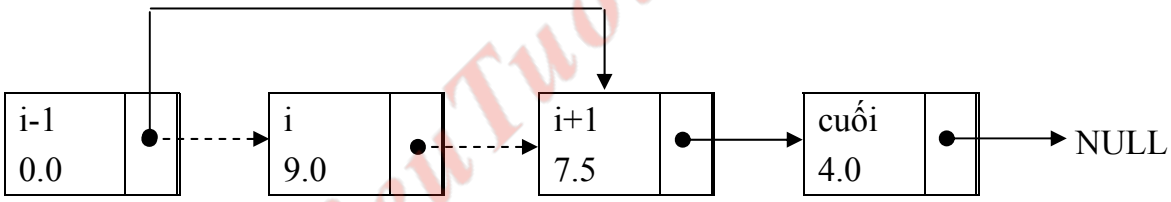
Chèn phần tử mới vào giữa phần tử i và i+1

a. Xoá phần tử thứ i khỏi danh sách

Việc xoá một phần tử ra khỏi danh sách rất đơn giản bởi chỉ việc thay đổi các con trỏ. Cụ thể giả sử cần xoá phần tử thứ i ta chỉ cần cho con trỏ tiep của phần tử thứ i-1 trỏ ("vòng qua" phần tử thứ i) vào phần tử thứ i+1. Như vậy bây giờ khi chạy trên danh sách đến phần tử thứ i-1, phần tử tiếp theo là phần tử thứ i+1 chứ không còn là phần tử thư i. Nói cách khác phần tử thứ i không được nối bởi bất kỳ phần tử nào nên nó sẽ

không thuộc danh sách. Có thể thực hiện các bước như sau:

- Cho con trỏ p chạy đến phần tử thứ i-1.
- Đặt phần tử thứ i vào biến x.
- Cho con trỏ tiep của phần tử thứ i-1 trỏ vào phần tử thứ i+1 bằng cách đặt tiep = x.tiep.
- Giải phóng bộ nhớ được trỏ bởi x bằng câu lệnh delete x.



Xóa phần tử thứ i

c. Duyệt danh sách

Duyệt là thao tác đi qua từng phần tử của danh sách, tại mỗi phần tử chương trình thực hiện một công việc gì đó trên phần tử mà ta gọi là thăm phần tử đó. Một phép thăm có thể đơn giản là hiện nội dung thông tin của phần tử đó ra màn hình chẳng hạn. Để duyệt danh sách ta chỉ cần cho một con trỏ p chạy từ đầu đến cuối danh sách đến khi phần tử cuối có con trỏ tiep = NULL thì dừng. Câu lệnh cho con trỏ p chuyển đến phần tử tiếp theo của nó là:

p = p → tiep ;

d. Tìm kiếm

Cho một danh sách trong đó mỗi phần tử của danh sách đều chứa một trường gọi là trường khoá, thường là các trường có kiểu cơ sở hoặc kết hợp của một số trường như vậy. Bài toán đặt ra là tìm trên danh sách phần tử có giá trị của trường khoá bằng với một giá trị cho trước. Tiến trình thực hiện nhiệm vụ thựcchất cũng là bài toán duyệt, trong đó thao tác "thăm" chính là so sánh trường khoá của phần tử với giá trị cho trước, nếu trùng nhau ta in kết quả và dừng, Nếu đã duyệt hết mà không có phần tử nào có trường khoá trùng với giá trị cho trước thì xem danh sách không chứa giá trị này.

Ngoài các thao tác trên, nói chung còn nhiều các thao tác quen thuộc khác tuy nhiên chúng ta không trình bày ở đây vì nó không thuộc phạm vi của giáo trình này.

Dưới đây là một ví dụ minh hoạ cho cấc cấu trúc tự trỏ, danh sách liên kết và một vài thao tác trên danh sách liên kết thông qua bài toán quản lý sinh viên.

- Khai báo

```
struct DATE
{
        int day, month, year;                   // ngày, tháng, năm
};
struct Sinhvien {                               // cấu trúc tự trỏ
        char hoten[31];
        DATE ns;
        float diem;
        Sinhvien *tiep ;
};
Sinhvien *dau = NULL, *cuoi = NULL;     // Các con trỏ tới đầu và cuối ds
Sinhvien *cur = NULL;                   // Con trỏ tới sv hiện tại
int sosv = 0;                           // Số sv của danh sách
```

- Tạo sinh viên mới và nhập thông tin, trả lại con trỏ trỏ đến sinh viên mới.

```
Sinhvien* Nhap1sv()                     // Tạo 1 khối dữ liệu cho sv mới
{
        Sinhvien *kq = new Sinhvien[1] ;        // Cấp phát bộ nhớ cho kq
        cout << "\nSinh vien thu ", sosv+1 ;
        cout << "Ho ten = " ; cin.getline(kq->hoten);
        cout << "Ns = " ; cin >> kq->ns.day >> kq->ns.month >> kq->ns.year;
        cout << "Diem = " ; cin >> kq->diem ; cin.ignore() ;
        kq->tiep = NULL;
        return kq ;
}
```

- Bổ sung sinh viên mới vào cuối danh sách.

```
void Bosung()                           // Bổ sung sv mới vào cuối ds
{
        cur = Nhap1sv();
```

```
        if (sosv == 0) {dau = cuoi = cur;}
        else { cuoi->tiep = cur; cuoi = cur; }
        sosv++;
    }
```

- Chèn sv mới vào trước sinh viên thứ n.

```
    void Chentruoc(int n)                    // Chèn sv mới vào trước sv thứ n
    {
        cur = Nhap1sv();
        if (sosv==0) { dau = cuoi = cur; sosv++; return; }
        if (sosv==1 || n==1) {cur->tiep = dau; dau = cur; sosv++; return;}
        Sinhvien *truoc, *sau;
        truoc = dau;
        sau = dau -> tiep;
        for (int i=1; i<n-1; i++) truoc = truoc->tiep;
        sau = truoc->tiep;
        truoc->tiep = cur;
        cur -> tiep = sau;
        sosv ++;
    }
```

- Chèn sv mới vào sau sinh viên thứ n.

```
    void Chensau(int n)                      // Chèn sv mới vào sau sv thứ n
    {
        cur = Nhap1sv();
        if (sosv==0 || sosv<n) { dau = cuoi = cur; sosv++; return; }
        Sinhvien *truoc, *sau;
        truoc = dau; sau = dau -> tiep;
        for (int i=1; i<n; i++) truoc = truoc->tiep;
        sau = truoc->tiep;
        truoc->tiep = cur;
```

```
        cur -> tiep = sau;
        sosv ++;
    }
```

- Xoá sinh viên thứ n.

```
    void Xoa(int n)                                    // Xoá sinh viên thứ n
    {
        if (sosv==1&&n==1) { delete dau ; dau = cuoi = NULL; sosv--; return; }
        if (n==1) { cur = dau; dau = cur->tiep; delete cur; sosv--; return; }
        Sinhvien *truoc, *sau;
        truoc = dau;
        sau = dau -> tiep;
        for (int i=1; i<n-1; i++) truoc = truoc->tiep;
        cur = truoc->tiep; sau = cur->tiep; truoc->tiep = sau;
        delete cur ;
        sosv --;
    }
```

- Tạo danh sách sinh viên.

```
    void Taods()                                       // Tạo danh sách
    {
        int tiep = 1;
        while (tiep) {
            Bosung();
            cout << "Tiep (0/1) ? " ; cin >> tiep ;
        }
    }
```

- In danh sách sinh viên.

```
    void Inds()                                        // In danh sách
    {
```

```
        cur = dau;          int i=1;
        while (cur != NULL) {
            cout << "\nSinh vien thu " << i << " ----------\n") ;
            cout << "Hoten:" << cur->hoten ;
            cout << "Ngay sinh: "
            cout << cur -> ns.day << "/" ;
            cout << cur -> ns.month << "/" ;
            cout << cur -> ns.year ;
            cout << "Diem: " << cur->diem ;
            cur = cur->tiep; i++;
        }
    }
```

- Hàm chính.

```
    void main()
    {
        clrscr();
        Taods();
        Inds();
        getch();
    }
```

III. KIỂU HỢP

1. Khai báo

Giống như cấu trúc, kiểu hợp cũng có nhiều thành phần nhưng các thành phần của chúng sử dụng chung nhau một vùng nhớ. Do vậy kích thước của một kiểu hợp là độ dài của trường lớn nhất và việc thay đổi một thành phần sẽ ảnh hưởng đến tất cả các thành phần còn lại.

```
union <tên kiểu> {
    Danh sách các thành phần;
};
```

2. Truy cập

Cú pháp truy cập đến các thành phần của hợp cũng tương tự như kiểu cấu trúc, tức cũng sử dụng toán tử lấy thành phần (dấu chấm . hoặc → cho biến con trỏ kiểu hợp).

Dưới đây là một ví dụ minh hoạ việc sử dụng khai báo kiểu hợp để tách byte thấp, byte cao của một số nguyên.

Ví dụ 1 :

```
void main()
{
    union songuyen {
        int n;
        unsigned char c[2];
    } x;
    cout << "Nhập số nguyên: " ; cin >> x.n ;
    cout << "Byte thấp của x = " << x.c[0] << endl ;
    cout << "Byte cao của x = " << x.c[1] << endl;
}
```

Ví dụ 2 : Kết hợp cùng kiểu nhóm bit trong cấu trúc, chúng ta có thể tìm được các bit của một số như chương trình sau. Trong chương trình ta sử dụng một biến u có kiểu hợp. Trong kiểu hợp này có 2 thành phần là 2 cấu trúc lần lượt có tên s và f.

```
union {
    struct { unsigned a, b ; } s;
    struct {
        unsigned n1: 1;
        unsigned: 15;
        unsigned n2: 1;
        unsigned: 7;
        unsigned n3: 8;
    } t ;
} u;
```

với khai báo trên đây khi nhập u.s thì nó cũng ảnh hưởng đến u.t, cụ thể

- u.t.n1 là bit đầu tiên (0) của thành phần u.s.a
- u.t.n2 là bit 0 của thành phần u.s.b
- u.t.n3 là byte cao của u.s.b

IV. KIỂU LIỆT KÊ

Có thể gán các giá trị nguyên liên tiếp (tính từ 0) cho các tên gọi cụ thể bằng kiểu liệt kê theo khai báo sau đây:

enum tên_kiểu { d/s tên các giá trị };

Ví dụ:

```
enum Bool {false, true};
```

khai báo kiểu mới đặt tên Bool chỉ nhận 1 trong 2 giá trị đặt tên false và true, trong đó false ứng với giá trị 0 và true ứng với giá trị 1. Cách khai báo kiểu enum trên cũng tương đương với dãy các macro sau:

```
#define false 0
#define true 1
```

Với kiểu Bool ta có thể khai báo một số biến như sau:

```
Bool Ok, found;
```

hai biến Ok và found sẽ chỉ nhận 1 trong 2 giá trị false (thay cho 0) hoặc true (thay cho 1). Có nghĩa có thể gán:

```
Ok = true;
```

hoặc: `found = false;`

Tuy nhiên không thể gán các giá trị nguyên trực tiếp cho các biến enum mà phải thông qua ép kiểu. Ví dụ:

```
Ok = 0;              // sai
Ok = Bool(0) ;       // đúng
hoặc Ok = false ;    // đúng
```