TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA HỆ THỐNG THÔNG TIN KINH TẾ

NGUYỄN VĂN HUÂN
VŨ XUÂN NAM
NGUYỄN VĂN GIÁP
ĐỖ VĂN ĐẠI

BÀI GIẢNG

PHÂN TÍCH THIẾT KẾ GIẢI THUẬT VÀ CẤU TRÚC DỮ LIỆU

NGÀNH HỆ THỐNG THÔNG TIN QUẢN LÝ

THÁI NGUYÊN, NĂM 2012

MỤC LỤC

LỜI NÓI ĐẦU

Phân tích – thiết kế giải thuật và Cấu trúc dữ liệu là một trong những môn học cơ bản của sinh viên Công nghệ thông tin nói chung và ngành Hệ thống thông tin Kinh tế nói riêng. Các cấu trúc dữ liệu và các giải thuật được xem như là 2 yếu tố quan trọng nhất trong lập trình, đúng như câu nói nổi tiếng của Niklaus Wirth: Chương trình = Cấu trúc dữ liệu + Giải thuật (Programs = Data Structures + Algorithms). Nắm vững các cấu trúc dữ liệu và các giải thuật là cơ sở để sinh viên tiếp cận với việc thiết kế và xây dựng phần mềm cũng như sử dụng các công cụ lập trình hiện đại. Cấu trúc dữ liệu có thể được xem như là 1 phương pháp lưu trữ dữ liệu trong máy tính nhằm sử dụng một cách có hiệu quả các dữ liệu này. Và để sử dụng các dữ liệu một cách hiệu quả thì cần phải có các thuật toán áp dụng trên các dữ liệu đó. Do vậy, cấu trúc dữ liệu và phân tích – thiết kế giải thuật là 2 yếu tố không thể tách rời và có những liên quan chặt chẽ với nhau. Việc lựa chọn một cấu trúc dữ liệu có thể sẽ ảnh hưởng lớn tới việc lựa chọn áp dụng giải thuật nào.

Giáo trình gồm sáu chương: Chương 1 đi tìm hiểu các cấu trúc dữ liệu cơ bản; Chương 2 tác giả đi sâu tìm hiểu các thuật toán kinh điển nhằm giúp người đọc nắm được ý nghĩa của thuật toán; Chương 3, 4, 5, 6 đi sâu tìm hiểu các cách tổ chức dữ liệu và thuật toán trên kiểu dữ liệu đó.

Với mục đích cung cấp cho các em sinh viên một cái nhìn toàn thể và cơ bản. Tác giả kỳ vọng kết thúc môn học người học sẽ nắm được những cách tổ chức và cấu trúc dữ liệu. Từ đó áp dụng một phần kiến thức ấy vào nghiên cứu những mảng khác hiệu quả, tối ưu hơn.

Mặc dù đã cố gắng biên soạn, song giáo trình không tránh khỏi những thiếu sót. Rất mong nhận được ý kiến đóng góp từ phía người đọc.

Trân trọng cảm ơn!

Thái Nguyên, tháng 08 năm 2011

Biên soạn

Bộ môn Thương mại điện tử

Chương 1

CẤU TRÚC DỮ LIỆU CƠ BẢN

1.1. Mảng

1.1.1. Khái niệm

Mảng là 1 dãy có thứ tự (về mặt vị trí) các phần tử với 2 đặc điểm sau:

- Số lượng phần tử cố định

- Mọi phần tử đều có cùng kiểu dữ liệu (dữ liệu cơ sở của mảng)

Các đặc trưng cơ bản :

+ Cho phép truy cập ngẫu nhiên đến từng phần tử. Thời gian truy cập đến mọi phần tử đều bằng nhau.

+ Số lượng phần tử của mảng là cố định. Việc bổ sung và loại bỏ phần tử là khó khăn (mất thời gian)

Các phép toán cơ bản :

Tạo mảng, duyệt mảng, tìm kiếm, sắp xếp, trộn mảng, tách mảng …

1.1.2. Mảng một chiều

Cấu trúc lưu trữ: Các phần tử được bố trí sát nhau trong bộ nhớ và theo thứ tự tăng dần của các chỉ số nên dễ dàng tìm được địa chỉ của 1 phần tử bất kỳ nếu biết chỉ số:

$Loc(a[i]) = a_0 + (i-1) * l$

a_0 là địa chỉ của phần tử thứ nhất ; l là độ dài 1 ô nhớ (byte)

1.1.3 Mảng hai chiều

***Cấu trúc lưu trữ*:** Có hai phương pháp lưu trữ

+ Phương pháp lưu trữ ưu tiên hàng

Với mảng A_{nm} (n hàng và m cột)

$Loc(a_{ij}) = L_0 + (i-1)*m + (j-1)$

+ Phương pháp lưu trữ ưu tiên cột

Với mảng A_{nm} (n hàng và m cột)

$$Loc(a_{ij}) = L_0 + (j-1)*n + (i-1)$$

1.2. Biến động và con trỏ

1.2.1. Biến động

Tất cả các biến có kiểu cấu trúc dữ liệu mà ta đã nghiên cứu như Array, Record, Set được gọi là biến tĩnh vì chúng được xác định một cách rõ ràng khi khai báo, sau đó chúng được dùng thông qua tên. Thời gian tồn tại của biến tĩnh cũng là thời gian tồn tại của khối chương trình có chứa khai báo biến này. Chẳng hạn, các biến được tĩnh khai báo trong chương trình (biến toàn cục) sẽ tồn tại từ khi chương trình được thực hiện cho đến khi kết thúc chương trình, còn các biến tĩnh được khai báo trong một chương trình con (biến địa phương) sẽ tồn tại từ khi chương trình con được thực hiện cho đến khi kết thúc chương trình con này.

Ngoài các biến tĩnh được xác định trước, người ta còn có thể tạo ra các biến trong lúc chạy chương trình, tuỳ theo nhu cầu. ***Việc tạo ra các biến theo kiểu này được gọi là cấp pháp bộ nhớ động, các biến được tạo ra được gọi là biến động.***

Các biến động không có tên. Để tạo ra biến động, người ta sử dụng một kiểu biến đặc biệt, gọi là con trỏ và thủ tục cấp phát bộ nhớ động (NEW) thông qua con trỏ. Khi không sử dụng biến động nữa, người ta có thể xoá nó khỏi bộ nhớ, việc này gọi là thu hồi bộ nhớ động. Để thu hồi bộ nhớ dành cho biến động, người ta dùng thủ tục DISPOSE và thông qua con trỏ đã sử dụng để tạo ra biến động.

So với biến tĩnh, việc sử dụng biến động có ưu điểm là tiết kiệm được bộ nhớ. Bởi vì, khi cần dùng biến động thì người ta sẽ tạo ra nó và khi không cần nữa người ta lại có thể xoá nó đi. Còn đối với các biến tĩnh, chúng được xác định và cấp phát bộ nhớ khi biên dịch, chúng sẽ chiếm giữ bộ nhớ trong suốt thời gian chương trình làm việc. Chẳng hạn, nếu cần sử dụng một mảng ta phải khai báo ngay ở phần đầu chương trình, ngay lúc này ta đã phải xác định kích thước của mảng và thường khai báo dôi ra gây lãng phí bộ nhớ.

1.2.2. Con trỏ

1.2.2.1. Kiểu con trỏ.

Kiểu con trỏ là một một kiểu dữ liệu đặc biệt để biểu diễn địa chỉ của các đối tượng (biến, mảng, bản ghi...) trong bộ nhớ. Có bao nhiêu kiểu đối tượng thì cũng có bấy nhiêu kiểu con trỏ tương ứng. Các giá trị thuộc kiểu con trỏ là địa chỉ (vị trí) trong bộ nhớ của máy tính để lưu giữ các đối tượng thuộc kiểu đối tượng. Ví dụ, kiểu con trỏ nguyên dùng để biểu thị địa chỉ của biến nguyên, các giá trị thuộc kiểu con trỏ nguyên là địa chỉ trong bộ nhớ để lưu trữ các số nguyên, kiểu con trỏ bản ghi dùng để biểu thị địa chỉ của bản ghi, các giá trị thuộc kiểu con trỏ bản ghi là địa chỉ trong bộ nhớ để lưu trữ các bản ghi v.v... Để định nghĩa kiểu con trỏ ta dùng mẫu sau:

```
TYPE
    Kiểu_con_trỏ = ^Kiểu_đối_tượng ;
```

Ví dụ 1:

```
TYPE
    Tro_nguyen = ^Integer ;
    Tro_hoc_sinh = ^Hoc_sinh;
    Hoc_sinh = Record
        Ho_ten : String[25];
        tuoi : Integer;
    End;
```

Chú ý: Khi định nghĩa kiểu con trỏ bản ghi có thể tiến hành theo một trong hai cách sau:

+ *Cách 1*: Định nghĩa kiểu bản ghi trước, rồi dùng nó định nghĩa kiểu con trỏ bản ghi tương ứng.

+ *Cách 2*: (xem ví dụ trên) Định nghĩa kiểu con trỏ bản ghi thông qua kiểu bản ghi còn chưa được định nghĩa. Nhưng ngay sau đó phải định nghĩa kiểu bản ghi này.

1.2.2.2. Biến con trỏ.

Biến con trỏ là biến dùng để chứa địa chỉ của biến động trong bộ nhớ. Có thể khai báo biến con trỏ thông qua kiểu con trỏ đã định nghĩa trước hoặc khai báo một cách trực tiếp.

Ví dụ 2:

```
Var
    pn1, pn2 : Tro_nguyen;
    phs : Tro_hoc_sinh;
    pt1, pt2  : ^real;
```

Trong ví dụ này khai báo 5 biến con trỏ (hay còn gọi là con trỏ), trong đó:

pn1, pn2 là con trỏ kiểu nguyên.

phs là con trỏ kiểu Hoc_sinh (bản ghi).

pt là con trỏ kiểu thực

1.2.3. Sử dụng con trỏ

Để thâm nhập vào biến động có địa chỉ nằm trong biến con trỏ, chẳng hạn con trỏ Ptr ta dùng ký hiệu Ptr^

Ví dụ: thông qua con trỏ pn1 ta có biến động pn1^, thông qua con trỏ phs ta có biến động phs^.

Cũng giống như biến tĩnh, biến động được tạo ra để chứa dữ liệu. Do đó, các câu lệnh được viết dưới đây là hợp lệ.

```
pn1^ := 10;          {gán giá trị 10 cho biến động}
readln(pn2^);        {nhập dữ liệu vào biến động pn2^ từ bàn phím}
Readln(phs^.ho_ten); {Nhập họ tên cho học sinh từ bàn phím vào trường Ho_ten của biến động phs^}
phs^.tuoi := 16; {gán giá trị 16 cho trường tuoi biến động phs^}
```

a. Các thao tác với con trỏ.

+ Phép gán hai con trỏ cùng kiểu. Ví dụ: pn1 := pn2;

+ Phép so sánh hai con trỏ cùng kiểu gồm: so sánh = (bằng nhau) và phép sánh <> (khác nhau).

b. Hằng con trỏ NULL.

NULL là hằng con trỏ đặc biệt dành cho các biến con trỏ, nó được dùng để báo rằng con trỏ không trỏ vào đâu cả. Hằng NULL có thể được đem gán cho bất kỳ biến con trỏ nào. Đương nhiên khi đó việc thâm nhập vào biến động thông qua con trỏ có giá trị NULL là vô nghĩa. Thực chất NULL là con trỏ đặc biệt chứa giá trị 0.

c. Tạo lập và giải phóng biến động.

Trong ngôn ngữ pascal, thủ tục chuẩn NEW được dùng để tạo ra biến động, với tham số là biến con trỏ, trỏ tới biến động mà ta muốn lập ra.

Cách viết:

NEW(Biến_con_trỏ);

Ví dụ: để tạo ra biến động phs^ do con trỏ phs trỏ tới, ta viết:

```
NEW(phs);
```

Như vậy NEW(phs) sẽ sắp xếp bộ nhớ cho một biến động có kiểu là Hoc_sinh.

Trong một chương trình ta có thể dùng thủ tục NEW(phs) nhiều lần, mỗi lần sẽ tạo ra một biến động phs^, song con trỏ phs sẽ chỉ trỏ vào biến động được tạo ra lần cuối cùng.

Ví dụ: Nếu trong chương trình ta viết 2 lần như sau:

```
NEW(phs);
phs^.Ho_ten := ‘Mot’;
phs^.tuoi := 1;
NEW(phs);
phs^.Ho_ten := ‘Hai’;
phs^.tuoi := 2;
```

Khi đó con trỏ phs sẽ trỏ vào biến động phs^ được tạo ra lần 2 với nội dung là (Hai, 2). Tuy nhiên, biến động phs^ được tạo ra lần 1 với nội dung (Mot, 1) vẫn còn nằm trong bộ nhớ, nhưng không được con trỏ phs trỏ tới.

Một điều lý thú là khi một biến động không được dùng nữa, ta có thể thu hồi lại (giải phóng) vùng nhớ mà nó chiếm giữ để dùng vào việc khác (điều này giúp tiết kiệm bộ nhớ) nhờ sử dụng thủ tục chuẩn DISPOSE. Tham số cho thủ tục này là con trỏ trỏ tới biến động cần giải phóng. Cách viết như sau:

DISPOSE(Biến_con_trỏ);

Ví dụ: Để giải phóng vùng nhớ của biến động phs^ do con trỏ phs trỏ tới ta viết.

DISPOSE(phs);

Ví dụ: Về chương trình sử dụng con trỏ.

Chương trình sau minh hoạ cách dùng con trỏ bản ghi. Chương trình bao gồm việc đọc giá trị cho một bản ghi, hiển thị nội dung của bản ghi lên màn hình.

```
Program Con_tro;
Uses crt;
Type
    Hoc_sinh = Record
        ht : string[25];
        tuoi : integer;
        qq : string[40];
    End;
Var
    phs : ^Hoc_sinh;
Procedure Doc_so_lieu(Var hs : Hoc_sinh);
Begin
    With hs Do
```

```
        begin
            Write('Ho ten:'); readln(ht);
            Write('Tuoi:'); readln(tuoi);
            Write('Que quan:'); readln(qq);
        end;
    End;
    Procedure Hien_thi(hs : Hoc_sinh);
    Begin
        With hs Do
        begin
            Writeln('Ho ten:',ht);
            Writeln('Tuoi:',tuoi);
            Writeln('Que quan:',qq);
        end;
    End;
(*Thân chương trình*)
    BEGIN
        CLRSCR;
        NEW(phs); {tạo một bản ghi động}
        Doc_so_lieu(phs^); {đọc số liệu cho bản ghi này}
        Hien_thi(phs^); {hiển thị nội dung bản ghi lên màn hình}
        DISPOSE(phs); {giải phóng bản ghi động phs^}
        readln;
    END.
```

1.3. Danh sách (LIST)

1.3.1. Khái niệm

Về mặt toán học, danh sách là một dãy hữu hạn các phần tử thuộc cùng một lớp các đối tượng nào đó. Chẳng hạn, danh sách sinh viên của một lớp, danh sách các số nguyên, danh sách các báo xuất bản hàng ngày ở thủ đô v.v...

Giả sử L là danh sách có n ($n \geq 0$) phần tử

$$L = (a_1, a_2, ..., a_n)$$

Ta gọi số n là *độ dài* của của danh sách. Nếu $n \geq 1$ thì a_1 được gọi là *phần tử đầu tiên* của danh sách, còn a_n là *phần tử cuối cùng* của danh sách. Nếu n = 0 tức danh sách không có phần tử nào, thì danh sách được gọi là rỗng.

Một tính chất quan trọng của danh sách là các phần tử của nó được sắp tuyến tính : nếu n > 1 thì phần tử a_i "đi trước" phần tử a_{i+1} hay "đi sau" phần tử a_i với i = 1,2, ..., n-1. Ta sẽ nói a_i (i = 1,2, ..., n) là phần tử ở vị trí thứ i của danh sách.

Cần chú ý rằng, một đối tượng có thể xuất hiện nhiều lần trong một danh sách. Chẳng hạn như trong danh sách các số ngày của các tháng trong một năm

(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

**** Danh sách con***.

Nếu $L = (a_1, a_2, ..., a_n)$ là một danh sách và i, j là các vị trí, $1 \leq i \leq j \leq n$ thì danh sách $L' = (b_1, b_2, ..., b_{j-i+1})$ trong đó $b_1 = a_i$, $b_2 = a_{i+1})$ ... $b_{j-i+1}=a_j$, Như vậy, danh sách con L' gồm tất cả các phần tử từ a_i đến a_j của danh sách L. Danh sách rỗng được xem là danh sách con của một danh sách bất kỳ.

Danh sách con bất kỳ gồm các phần tử bắt đầu từ phần tử đầu tiên của danh sách L được gọi là *phần đầu* (prefix) của danh sách L. *Phần cuối* (postfix) của danh sách L là một danh sách con bất kỳ kết thúc ở phần tử cuối cùng của danh sách L.

**** Dãy con***

Một danh sách được tạo thành bằng cách loại bỏ một số (có thể bằng không) phần tử của danh sách L được gọi là *dãy con* của danh sách L.

Ví dụ. Xét danh sách

L = (black, blue, green, cyan, red, brown, yellow)

Khi đó danh sách (blue, green, cyan, red) là danh sách con của L. Danh sách (black, green, brown) là dãy con của L. Danh sách (black, blue, green) là phần đầu, còn danh sách (red, brown, yellow) là phần cuối của danh sách L.

**** Các phép toán trên danh sách.***

Chúng ta đã trình bày khái niệm toán học danh sách. Khi mô tả một mô tả một mô hình dữ liệu, chúng ta cần xác định các phép toán có thể thực hiện trên mô hình toán học được dùng làm cơ sở cho mô hình dữ liệu. Có rất nhiều phép toán trên danh sách. Trong các ứng dụng, thông thường chúng ta chỉ sử dụng một nhóm các phép toán nào đó. Sau đây là một số phép toán chính trên danh sách.

Giả sử L là một danh sách (List), các phần tử của nó có kiểu dữ liệu Item nào đó, p là một vị trí (position) trong danh sách. Các phép toán sẽ được mô tả bởi các thủ tục hoặc hàm.

1. Khởi tạo danh sách rỗng

    procedure Initialize (**var** L : List) ;

2. Xác định độ dài của danh sách.

    function Length (L : List) : **integer**

3. Loại phần tử ở vị trí thứ p của danh sách

    procedure Delete (p : position ; **var** L : List) ;

4. Xen phần tử x vào danh sách sau vị trí thứ p

    procedure Insert After (p : position ; x : Item ; **var** L: List) ;

5. Xen phần tử x vào danh sách trước vị trí thứ p

    procedure Insert Before (p : position ; x : Item ; **var** L:List);

6. Tìm xem trong danh sách có chứa phần tử x hay không ?

procedure Search (x : Item ; L : List : var found : **boolean**) ;

7. Kiểm tra danh sách có rỗng không ?

function Empty (L : List) : **boolean** ;

8. Kiểm tra danh sách có đầy không ?

function Full (L : List) : **boolean** ;

9. Đi qua danh sách. Trong nhiều áp dụng chúng ta cần phải đi qua danh sách, từ đầu đến hết danh sách, và thực hiện một nhóm hành động nào đó với mỗi phần tử của danh sách.

procedure Traverse (var L : List);

10. Các phép toán khác. Còn có thể kể ra nhiều phép toán khác. Chẳng hạn truy cập đến phần tử ở vị trí thứ i của danh sách (để tham khảo hoặc thay thế), kết hợp hai danh sách thành một danh sách, phân tích một danh sách thành nhiều danh sách, ...

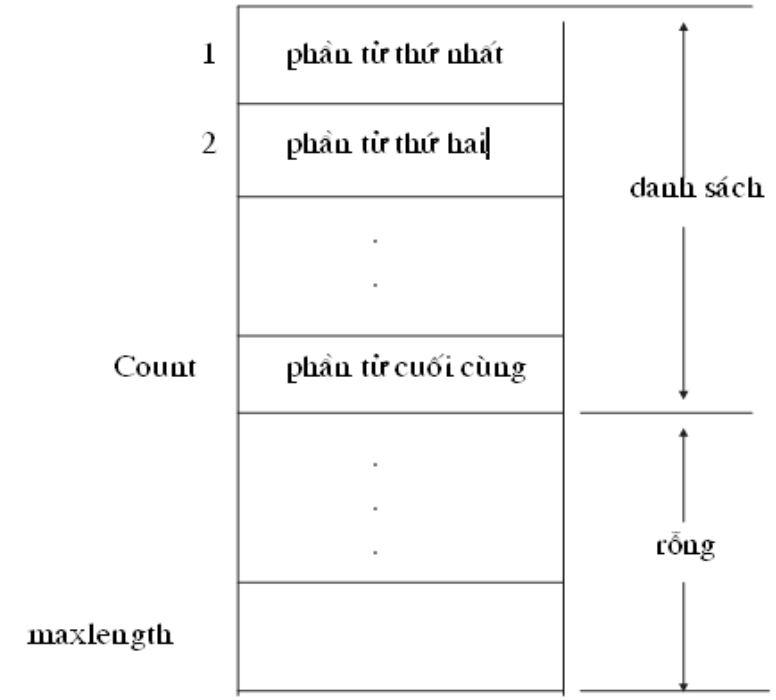
Ví dụ : Giả sử L là danh sách L = (3,2,1,5). Khi đó, thực hiện Delete (3,L) ta được danh sách (3,2,5). Kết quả của InsertBefor (1, 6, L) là danh sách (6, 3, 2, 1, 5).

1.3.2. Danh sách cài đặt bởi mảng

Phương pháp tự nhiên nhất để cài đặt một danh sách là sử dụng mảng, trong đó mỗi thành phần của mảng sẽ lưu giữ một phần tử nào đó của danh sách, và các phần tử kế nhau của danh sách được lưu giữ trong các thành phần kế nhau của mảng.

Giả sử độ dài tối đa của danh sách (maxlength) là một số N nào đó, các phần tử của danh sách có kiểu dữ liệu là Item. Item có thể là các kiểu dữ liệu đơn, hoặc các dữ liệu có cấu trúc, thông thường Item là bản ghi. Chúng ta biểu diễn danh sách (List) bởi bản ghi gồm hai trường. Trường thứ nhất là mảng các Item phần tử thứ i của danh sách được lưu giữ trong thành phần thứ i của mảng. Trường thứ hai ghi chỉ số của thành phần mảng lưu giữ phần tử cuối cùng của danh sách (xem hình 3.1). Chúng ta có các khai báo như sau:

```
const  maxlength = N;
       type   List = record
                     element : array [1 ... maxlength]
                                  of  Item ; count : 0 ... maxlength ;
                     end ;
       var L : List ;
```



Hình 3.1. Mảng biểu diễn danh sách

rên danh sách được t gần một lệnh gán :

```
L.count : = 0 ;
```

Độ dài của danh sách là L.count. Danh sách đầy, nếu L.count = maxlength.

Sau đây là các thủ tục thực hiện các phép toán xen một phần tử mới vào danh sách và loại một phần tử khỏi danh sách.

Thủ tục loại bỏ.

```
procedure Delete (p : 1 ... maxlength ; var L : List ;
                  var OK : boolean) ;
```

```
        var i : 1 ... maxlength ;
    begin
        OK : = false ;
        with L do
            if p < = count then
    begin
            i : = p;
        while i < count do
        begin
            element [i] : = element [i + 1] ;
            i: = i + 1
        end ;
        count : = count -1 ;
        OK : = true ;
     end ;
  end ;
```

Thủ tục trên thực hiện phép loại bỏ phần tử ở vị trí p khỏi danh sách. Phép toán chỉ được thực hiện khi danh sách không rỗng và p chỉ vào một phần tử trong danh sách. Tham biến OK ghi lại phép toán có được thực hiện thành công hay không. Khi loại bỏ, chúng ta phải dồn các phần tử các vị trí p+1, p + 2, ... lên trên một vị trí.

Thủ tục xen vào.

```
    procedure InsertBefore (p : 1 ... maxlength ; x : Item ;
            var L : List ; var OK : boolean) ;
        var i : 1... maxlength ;
```

```
begin
        OK: = false ;
with L do
if (count < maxlength) and ( p <= count) then
begin
        i: = count + 1 ;
        while i > p do
                begin
                        element[i]:= element[i-1] ;
                        i:=i-1 ;
                end ;
element [p] : = x ;
count : = count + 1 ;
OK : = true ;
   end ;
end ;
```

Thủ tục trên thực hiện việc xen phần tử mới x vào trước phần tử ở vị trí p trong danh sách. Phép toán này chỉ được thực hiện khi danh sách chưa đầy (count < maxlength) và p chỉ vào một phần tử trong danh sách (p <= count). Chúng ta phải dồn các phần tử ở các vị trí p, p+1, ... xuống dưới một vị trí để lấy chỗ cho x.

Nếu n là độ dài của danh sách ; dễ dàng thấy rằng, cả hai phép toán loại bỏ và xen vào được thực hiện trong thời gian O(n).

Việc tìm kiếm trong danh sách là một phép toán được sử dụng thường xuyên trong các ứng dụng. Chúng ta sẽ xét riêng phép toán này trong mục sau.

**** Nhận xét về phương pháp biểu diễn danh sách bởi mảng.***

Chúng ta đã cài đặt danh sách bởi mảng, tức là dùng mảng để lưu giữ các phần tử của danh sách. Do tính chất của mảng, phương pháp này cho phép ta truy cập trực tiếp đến phần tử ở vị trí bất kỳ trong danh sách. Các phép toán khác đều được thực hiện rất dễ dàng. Tuy nhiên phương pháp này không thuận tiện để thực hiện phép toán xen vào và loại bỏ. Như đã chỉ ra ở trên, mỗi lần cần xen phần tử mới vào danh sách ở vị trí p (hoặc loại bỏ phần tử ở vị trí p) ta phải đẩy xuống dưới (hoặc lên trên) một vị trí tất cả các phần từ đi sau phần tử thứ p. Nhưng hạn chế chủ yếu của cách cài đặt này là ở không gian nhớ cố định giành để lưu giữ các phần tử của danh sách. Không gian nhớ này bị quy định bởi cỡ của mảng. Do đó danh sách không thể phát triển quá cỡ của mảng, phép toán xen vào sẽ không được thực hiện khi mảng đã đầy.

1.3.3. Danh sách liên kết

Trong mục này chúng ta sẽ biểu diễn danh sách bởi cấu trúc dữ liệu khác, đó là danh sách liên kết. Trong cách cài đặt này, danh sách liên kết được tạo nên từ các tế bào mỗi tế bào là một bản ghi gồm hai trường, trường infor "chứa" phần tử của danh sách, trường next là con trỏ trỏ đến phần tử đi sau trong danh sách. Chúng ta sẽ sử dụng con trỏ head trỏ tới đầu danh sách. Như vậy một danh sách (a_1, a_2, ...a_n) có thể biểu diễn bởi cấu trúc dữ liệu danh sách liên kết được minh hoạ trong hình 3.2.

Hình 3.2. Danh sách liên kết biểu diễn danh sách (a_1, a_2, ...a_n)

l trỏ tới đầu danh sách, do đó, ta có thể khai báo như sau.

```
type   pointer = ^ cell
       cell    = record
                    infor : Item ;
                    next : pointer
```

end ;

var head : pointer ;

Chú ý : Không nên nhầm lẫn danh sách và danh sách liên kết. Danh sách và danh sách liên kết là hai khái niệm hoàn toàn khác nhau. Danh sách là một mô hình dữ liệu, nó có thể được cài đặt bởi các cấu trúc dữ liệu khác nhau. Còn danh sách liên kết là một cấu trúc dữ liệu, ở đây nó được sử dụng để biểu diễn danh sách.

**** Các phép toán trên danh sách liên kết.***

Sau đây chúng ta sẽ xét xem các phép toán trên danh sách được thực hiện như thế nào khi mà danh sách được cài đặt bởi danh sách liên kết.

Điều kiện để một danh sách liên kết rỗng là

head = **NULL**

Do đó, muốn khởi tạo một danh sách rỗng, ta chỉ cần lệnh gán :

head : = **NULL**

Danh sách liên kết chỉ đầy khi không còn không gian nhớ để cấp phát cho các thành phần mới của danh sách. Chúng ta sẽ giả thiết điều này không xẩy ra, tức là danh sách liên kết không khi nào đầy. Do đó phép toán xen một phần tử mới vào danh sách sẽ luôn luôn được thực hiện.

**** Phép toán xen vào.***

Giả sử Q là một con trỏ trỏ vào một thành phần của danh sách liên kết, và trong trường hợp danh sách rỗng (head = NULL) thì Q = NULL. Chúng ta cần xen một thành phần mới với infor là x vào sau thành phần của danh sách được trỏ bởi Q. Phép toán này được thực hiện bởi thủ tục sau :

```
procedure  InsertAfter (x : Item ; Q : pointer ; var head : pointer) ;
    var P : pointer ;
    begin
        new (P) ;
        P^ . infor : = x ;
```