DANH SÁCH LIÊN KẾT
(LINKED LISTS)

Nội dung

- Giới thiệu
- Danh sách liên kết đơn (Single Linked List)
- Danh sách liên kết đôi (Double Linked List)
- Danh sách liên kết vòng (Circular Linked List)

Giới thiệu

- Kiểu dữ liệu tĩnh
  - Khái niệm: Một số đối tượng dữ liệu không thay thay đổi được kích thước, cấu trúc, … trong suốt quá trình sống. Các đối tượng dữ liệu thuộc những kiểu dữ liệu gọi là kiểu dữ liệu tĩnh.
  - Một số kiểu dữ liệu tĩnh: các cấu trúc dữ liệu được xây dựng từ các kiểu cơ sở như: kiểu thực, kiểu nguyên, kiểu ký tự ... hoặc từ các cấu trúc đơn giản như mẩu tin, tập hợp, mảng ...

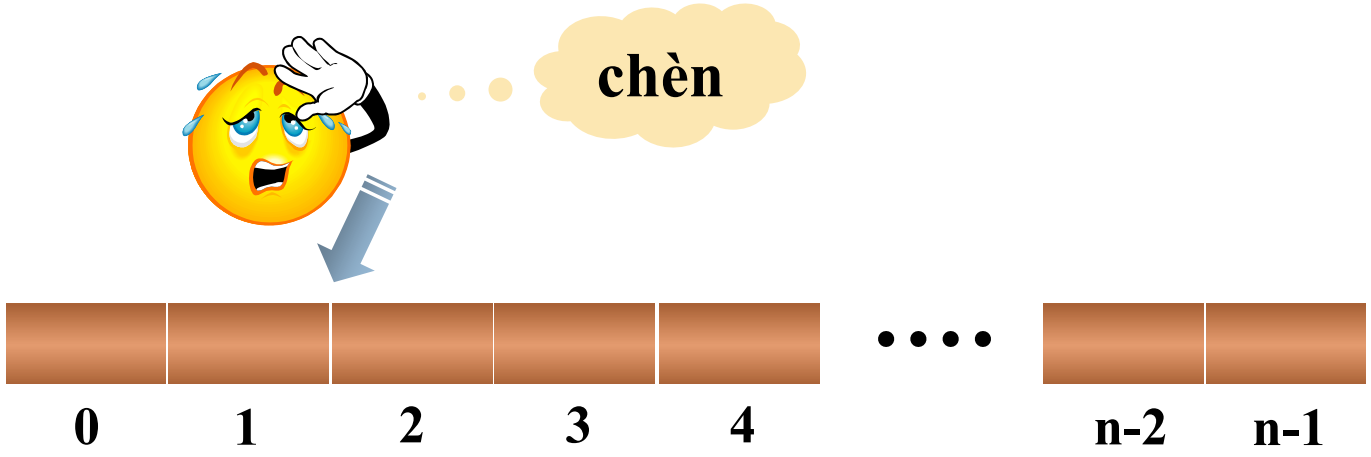
➔ Các đối tượng dữ liệu được xác định thuộc những kiểu dữ liệu này thường cứng ngắt, gò bó ➔ khó diễn tả được thực tế vốn sinh động, phong phú.

Giới thiệu

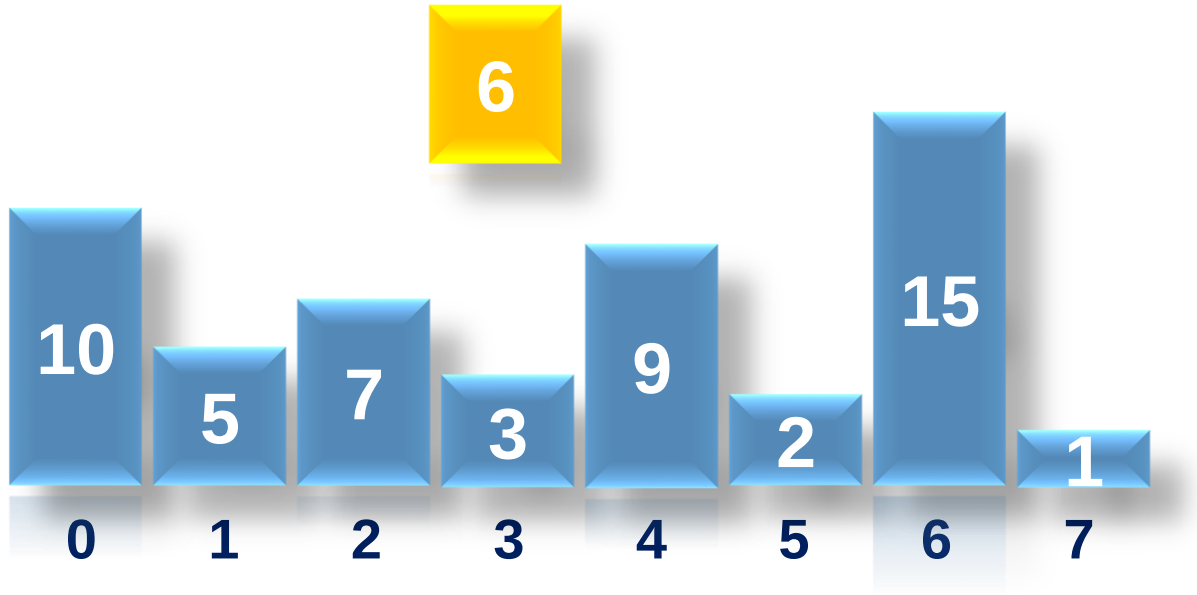
- Một số hạn chế của CTDL tĩnh
  - Một số đối tượng dữ liệu trong chu kỳ sống của nó có thể thay đổi về cấu trúc, độ lớn, như danh sách các học viên trong một lớp học có thể tăng thêm, giảm đi ... Nếu dùng những cấu trúc dữ liệu tĩnh đã biết như mảng để biểu diễn ➔ Những thao tác phức tạp, kém tự nhiên ➔ chương trình khó đọc, khó bảo trì và nhất là khó có thể sử dụng bộ nhớ một cách có hiệu quả
  - Dữ liệu tĩnh sẽ chiếm vùng nhớ đã dành cho chúng suốt quá trình hoạt động của chương trình ➔ sử dụng bộ nhớ kém hiệu quả

Giới thiệu

- **Cấu trúc dữ liệu tĩnh:** Ví dụ: Mảng 1 chiều (hay danh sách kề)
  - Kích thước cố định (fixed size)
  - Chèn 1 phần tử vào mảng rất khó
  - Các phần tử tuần tự theo chỉ số 0 ⇒ n-1
  - Truy cập ngẫu nhiên (random access)



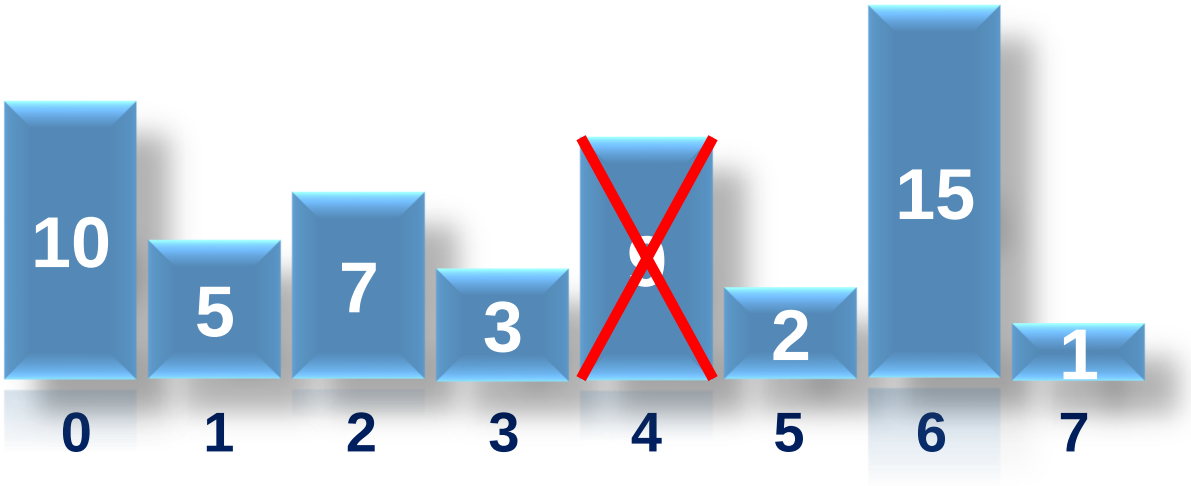
Chèn thêm phần tử vào mảng 1 chiều



? Làm sao để chèn thêm số 6 vào vị trí 4 của mảng

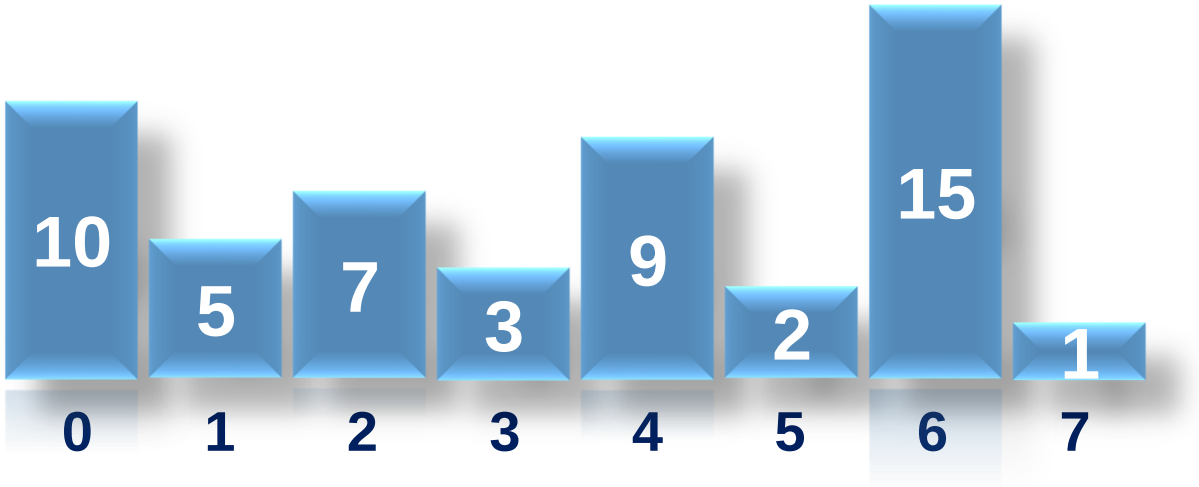
DS kề - chèn them 1 phân tử

Xóa phần tử khỏi mảng 1 chiều



? Làm sao để xóa phần tử 9

Danh sách kề - xóa phần tử

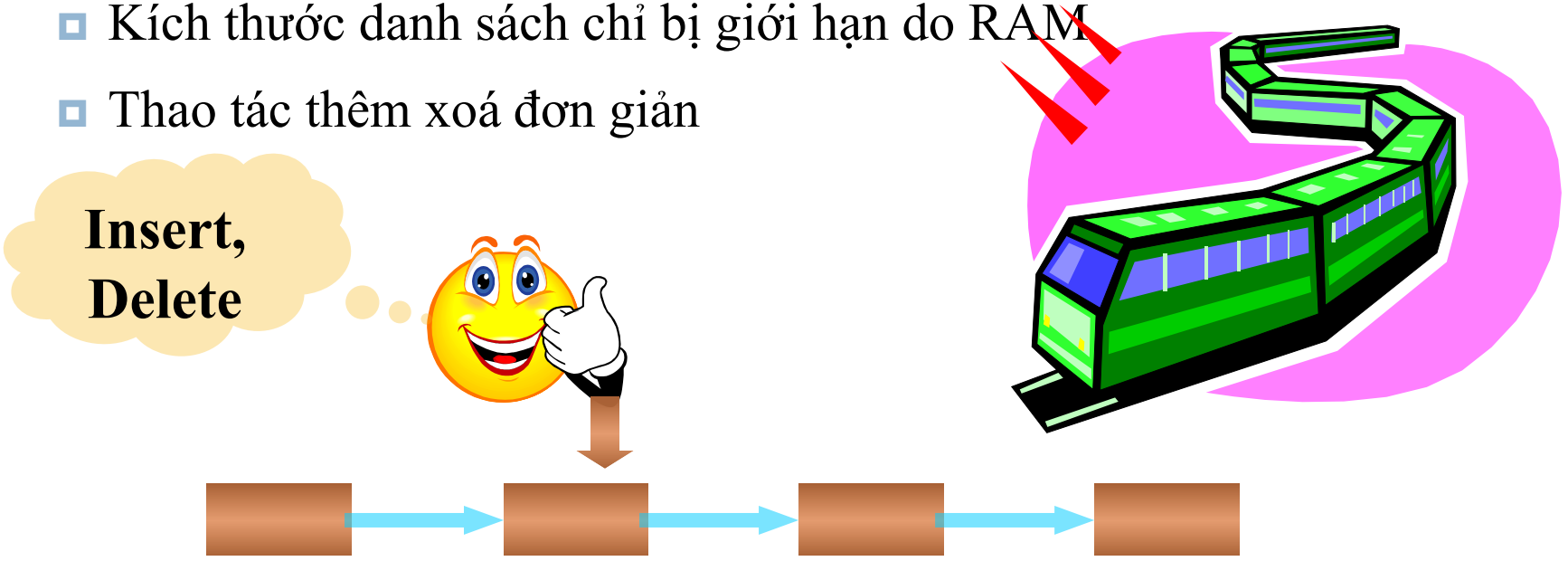


Bài tập

- Hãy cài đặt hàm chèn một phần tử có giá trị x vào vị trí vt trong mảng số nguyên a, kích thước n
- Hãy cài đặt hàm xóa phần tử có giá trị x *(nếu có)* trong mảng số nguyên a, kích thước n *(giả sử giá trị các phần tử trong mảng không trùng nhau)*

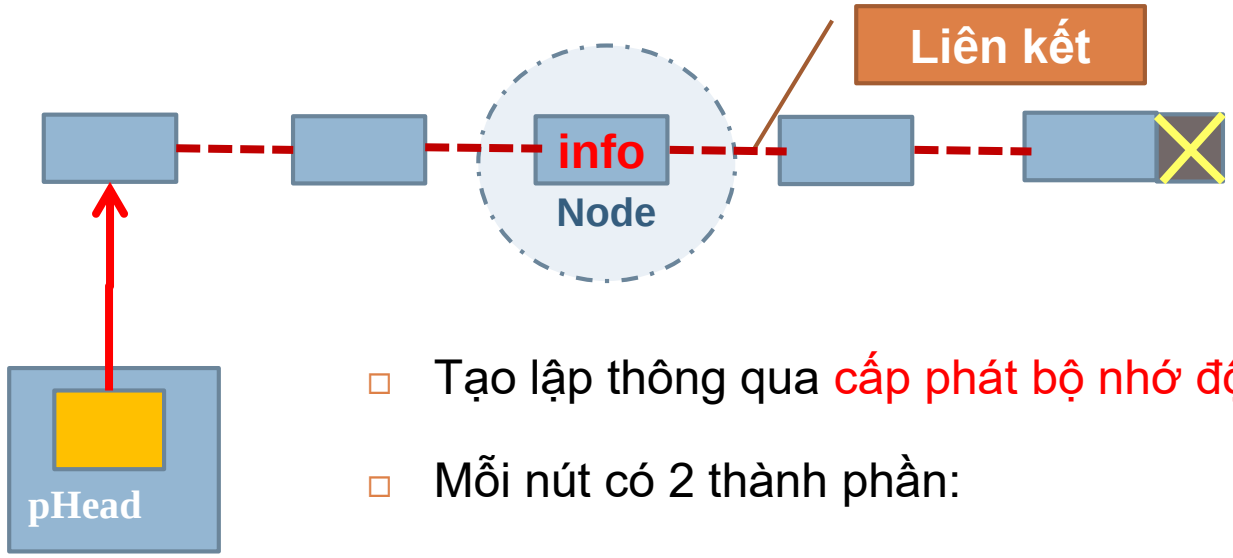
Giới thiệu

- **Cấu trúc dữ liệu động:** Ví dụ: Danh sách liên kết, cây
  - Cấp phát động lúc chạy chương trình
  - Các phần tử nằm rải rác ở nhiều nơi trong bộ nhớ
  - Kích thước danh sách chỉ bị giới hạn do RAM
  - Thao tác thêm xoá đơn giản



Cấu tạo Node

Danh sách liên kết: Mỗi phần tử của danh sách gọi là node (nút)



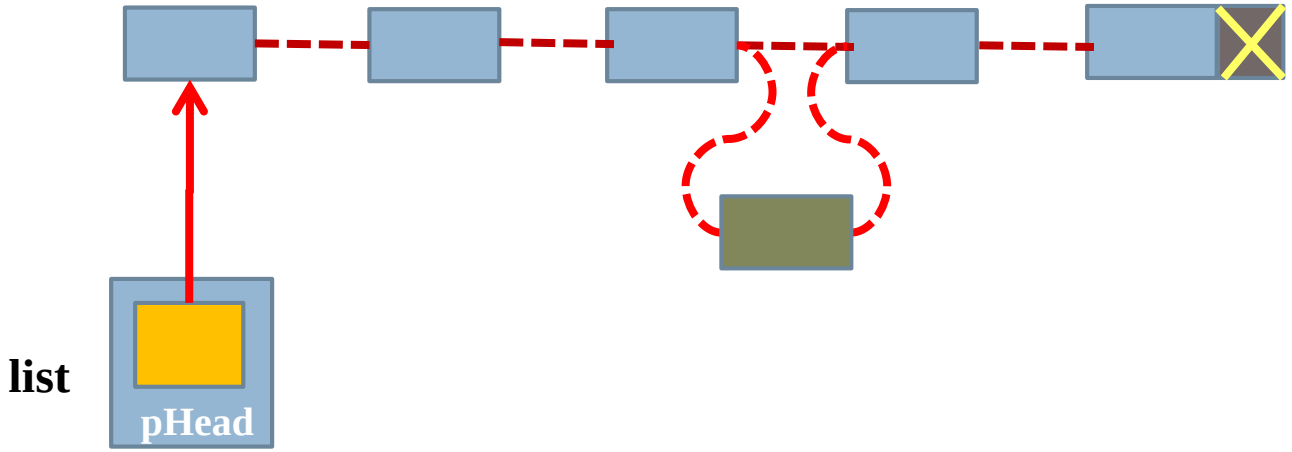
- Tạo lập thông qua cấp phát bộ nhớ động
- Mỗi nút có 2 thành phần:
  - *Chứa dữ liệu (info)*
  - *Con trỏ liên kết các phần tử với nhau (pointer link)*
- Nếu không trỏ đến phần tử nào thì liên kết = NULL

Giới thiệu

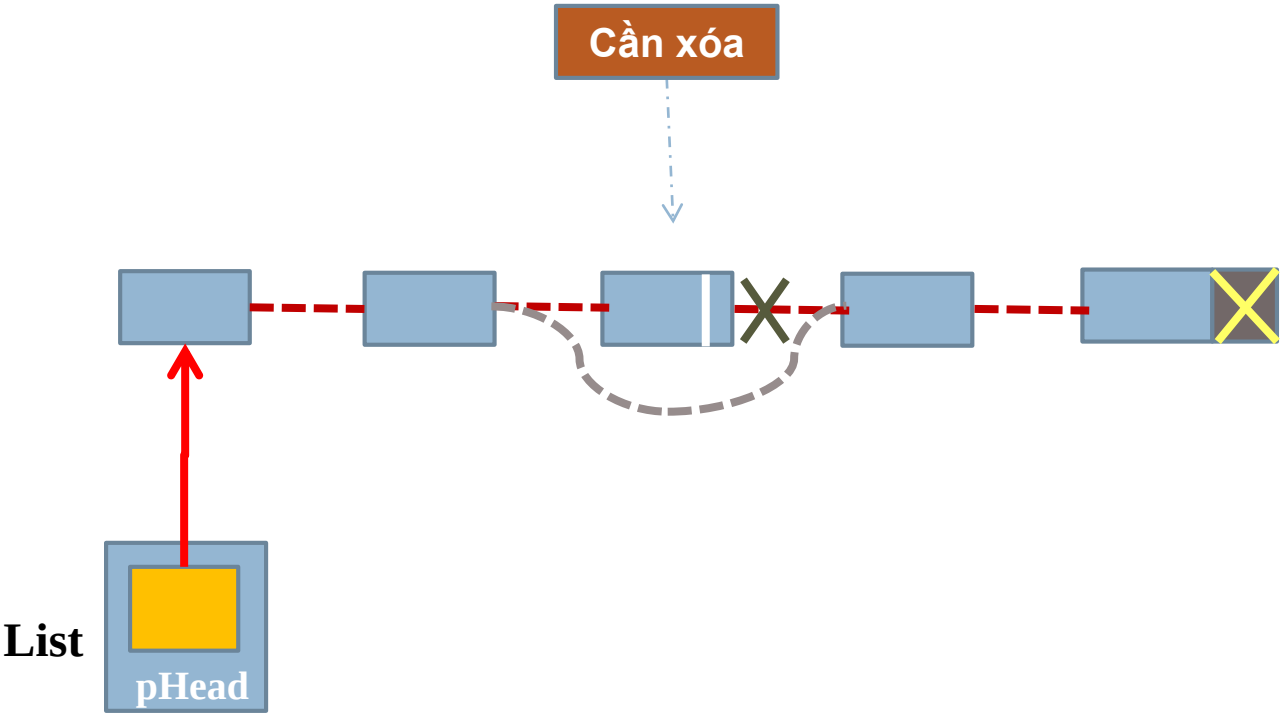
- Các thao tác cơ bản trên danh sách liên kết:
  - Thêm một phần tử mới
  - Xóa một phần tử
  - Tìm kiếm
  - …

Thao tác chèn thêm node vào DSLK

- “Kết nối” lại sợi dây liên kết **theo trình tự**



Thao tác xóa node khỏi DSLK



Mảng vs DSLK

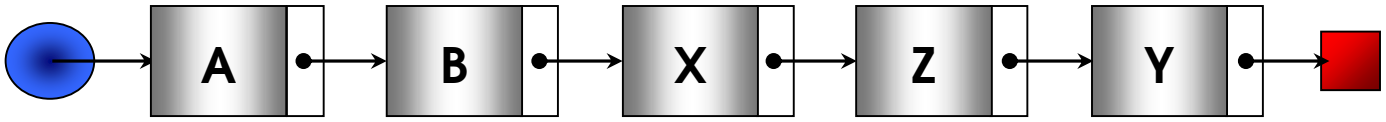
Nội dung	Mảng	Danh sách liên kết
Kích thước	•Kích thước cố định •Cần chỉ rõ kích thước trong khi khai báo	•*Kích thước thay đổi trong quá trình thêm/ xóa phần tử* •*Kích thước tối đa phụ thuộc vào bộ nhớ*
Cấp phát bộ nhớ	•Tĩnh: Bộ nhớ được cấp phát trong quá trình biên dịch	•*Động: Bộ nhớ được cấp phát trong quá trình chạy*
Thứ tự & sắp xếp	•Được lưu trữ trên một dãy ô nhớ liên tục	•*Được lưu trữ trên các ô nhớ ngẫu nhiên*
Truy cập	•*Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1)*	•Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n)
Tìm kiếm	•*Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân*	•Chỉ có thể tìm kiếm tuyến tính

CÁC LOẠI DSLK

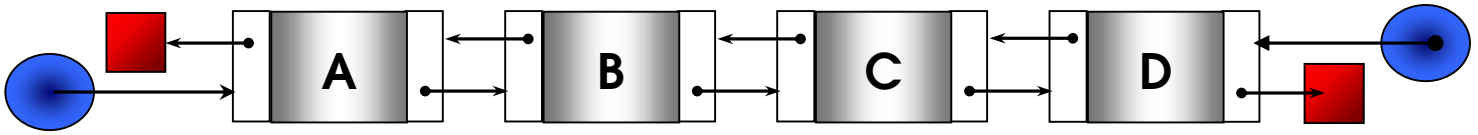
- Có nhiều kiểu tổ chức liên kết giữa các phần tử trong danh sách như:
  - Danh sách liên kết đơn
  - Danh sách liên kết kép
  - Danh sách liên kết vòng

Giới thiệu

- **Danh sách liên kết đơn:** mỗi phần tử liên kết với phần tử đứng sau nó trong danh sách:

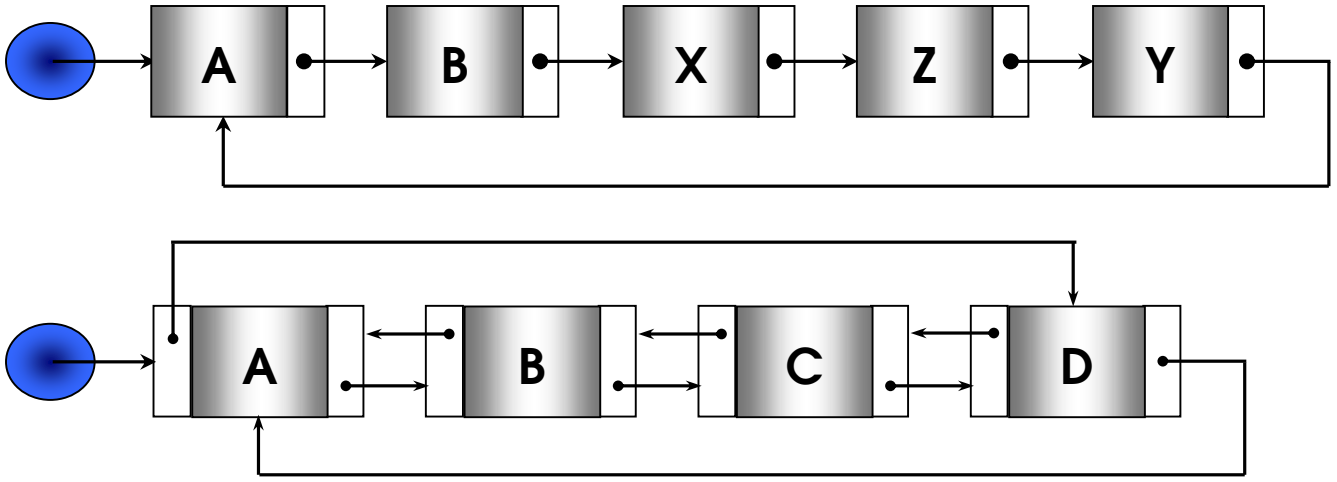


- **Danh sách liên kết đôi:** mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



Giới thiệu

- **Danh sách liên kết vòng :** phần tử cuối danh sách liên kết với phần tử đầu danh sách:



Giới thiệu

- Hướng giải quyết
- Cần xây dựng cấu trúc dữ liệu đáp ứng được các yêu cầu:
  - Linh động hơn
  - Có thể thay đổi kích thước, cấu trúc trong suốt thời gian sống

➔ *Cấu trúc dữ liệu động*

Biến không động

Biến không động (biến tĩnh, biến nửa tĩnh) là những biến thỏa:

- Được khai báo tường minh,
- Tồn tại khi vào phạm vi khai báo và chỉ mất khi ra khỏi phạm vi này,
- Được cấp phát vùng nhớ trong vùng dữ liệu (Data segment) hoặc là Stack (đối với biến nửa tĩnh - các biến cục bộ).
- Kích thước không thay đổi trong suốt quá trình sống.
- Do được khai báo tường minh, các biến không động có một định danh đã được kết nối với địa chỉ vùng nhớ lưu trữ biến và được truy xuất trực tiếp thông qua định danh đó.
- Ví dụ :

```
int     a;      // a, b là các biến không động
char    b[10];
```

Biến động

- Trong nhiều trường hợp, tại thời điểm biên dịch không thể xác định trước kích thước chính xác của một số đối tượng dữ liệu do sự tồn tại và tăng trưởng của chúng phụ thuộc vào ngữ cảnh của việc thực hiện chương trình.
- Các đối tượng dữ liệu có đặc điểm kể trên nên được khai báo như biến động. Biến động là những biến thỏa:
  - Biến không được khai báo tường minh.
  - Có thể được cấp phát hoặc giải phóng bộ nhớ khi người sử dụng yêu cầu.
  - Các biến này không theo qui tắc phạm vi (tĩnh).
  - Vùng nhớ của biến được cấp phát trong Heap.
  - Kích thước có thể thay đổi trong quá trình sống.

Biến động

- Do không được khai báo tường minh nên các biến động không có một định danh được kết buộc với địa chỉ vùng nhớ cấp phát cho nó, do đó gặp khó khăn khi truy xuất đến một biến động.
- Để giải quyết vấn đề, biến con trỏ (là biến không động) được sử dụng để trỏ đến biến động.
- Khi tạo ra một biến động, phải dùng một con trỏ để lưu địa chỉ của biến này và sau đó, truy xuất đến biến động thông qua biến con trỏ đã biết định danh.

Biến động

- Hai thao tác cơ bản trên biến động là tạo và hủy một biến động do biến con trỏ ‘p’ trỏ đến:
  - Tạo ra một biến động và cho con trỏ ‘p’ chỉ đến nó
  - Hủy một biến động do p chỉ đến

Biến động

- Tạo ra một biến động và cho con trỏ ‘p’ chỉ đến nó

  void* malloc(size); *// trả về con trỏ chỉ đến vùng nhớ*
  // size byte vừa được cấp phát.

  void* calloc(n,size); *// trả về con trỏ chỉ đến vùng nhớ*
  // vừa được cấp phát gồm n phần tử,
  // mỗi phần tử có kích thước size byte

  new *// toán tử cấp phát bộ nhớ trong C++*

- Hàm free(p) huỷ vùng nhớ cấp phát bởi hàm malloc hoặc calloc do p trỏ tới
- Toán tử **delete** p huỷ vùng nhớ cấp phát bởi toán tử **new** do p trỏ tới

Biến động – *Ví dụ*


```
int *p1, *p2;
// cấp phát vùng nhớ cho 1 biến động kiểu int
p1 = (int*)malloc(sizeof(int));
*p1 = 5; // đặt giá trị 5 cho biến động đang được p1
   quản lý
// cấp phát biến động kiểu mảng gồm 10 phần tử kiểu int
p2 = (int*)calloc(10, sizeof(int));
*(p2+3) = 0; // đặt giá trị 0 cho phần tử thứ 4 của
   mảng p2
free(p1);
free(p2);
```

Cấp phát bộ nhớ biến con trỏ (cách 2)

- Biến con trỏ
  - ✓ Cấp phát bộ nhớ: new kdl[kích thước]
  - ✓ Giải phóng bộ nhớ: delete[] tên_biến_con_trỏ
- Ví dụ:

```
int *a;
a=new int[10];        // Cấp phát 10 phần tử
delete[] a;           // Giải phóng
```

Nếu chỉ lưu 1 giá trị thì bỏ cặp dấu ngoặc []
VD: a = new int hay delete a

Kiểu dữ liệu Con trỏ

- Kiểu con trỏ là kiểu cơ sở dùng lưu địa chỉ của một đối tượng dữ liệu khác.
- Biến thuộc kiểu con trỏ Tp là biến mà giá trị của nó là địa chỉ cuả một vùng nhớ ứng với một biến kiểu T, hoặc là giá trị NULL.

Con trỏ – Khai báo

- Cú pháp định nghĩa một kiểu con trỏ trong ngôn ngữ C :

  typedef <kiểu cơ sở> * < kiểu con trỏ>;

- Ví dụ :

```
typedef     int     *intpointer;
intpointer  p;
```

hoặc

```
int     *p;
```

là những khai báo hợp lệ.

Con trỏ – *Thao tác căn bản*

- Các thao tác cơ bản trên kiểu con trỏ:(minh họa bằng C)
  - Khi 1 biến con trỏ p lưu địa chỉ của đối tượng x, ta nói ‘p trỏ đến x’.
  - Gán địa chỉ của một vùng nhớ con trỏ p:

    p = <địa chỉ>;

    ví dụ :

    ```
    int  i,*p;
    p=&i;
    ```
  - Truy xuất nội dung của đối tượng do p trỏ đến (*p)

Nội dung

- Giới thiệu
- Danh sách liên kết đơn (Single Linked List)
- Danh sách liên kết kép (Doule Linked List)
- Danh sách liên kết vòng (Circular Linked List)

Danh sách liên kết đơn (DSLK đơn)

- Khai báo
- Các thao tác cơ bản trên DSLK đơn
- Sắp xếp trên DSLK đơn

DSLK đơn – Khai báo

- Là danh sách các node mà mỗi node có 2 thành phần:
  - Thành phần dữ liệu: lưu trữ các thông tin về bản thân phần tử
  - Thành phần mối liên kết: lưu trữ địa chỉ của phần tử kế tiếp trong danh sách, hoặc lưu trữ giá trị NULL nếu là phần tử cuối danh sách

  - Khai báo node

```
struct  Node
{
    DataType data; // DataType là kiểu đã định nghĩa trước
    Node *next;    // con trỏ chỉ đến cấu trúc Node
};
typedef struct Node *pNode
```

DSLK đơn – Khai báo

- Ví dụ 1: Khai báo node lưu số nguyên:

```
struct  Node
{
        int     data;
        Node *next;
};
typedef struct Node *pNode
```

- Ví dụ 2: Định nghĩa một phần tử trong danh sách đơn lưu trữ hồ sơ sinh viên:

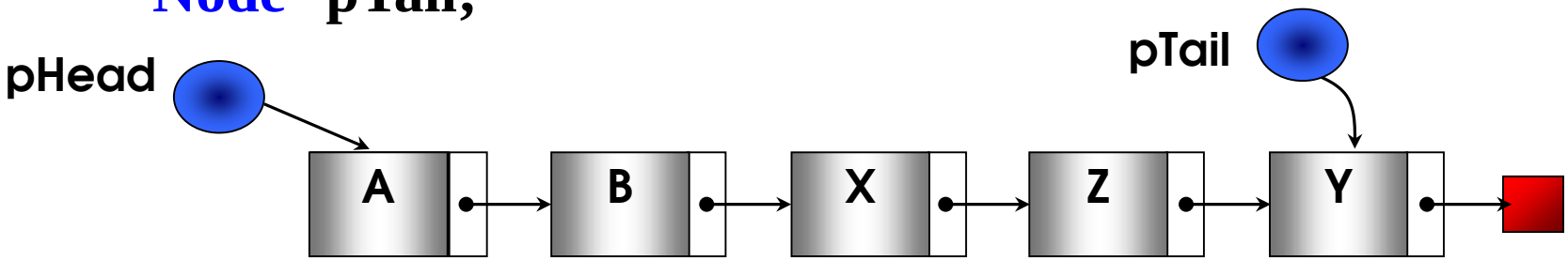
```
struct SinhVien {
        char  Ten[30];
        int     MaSV;
};
struct SVNode {
        SinhVien data;
        SVNode  *next;
};
typedef struct SVNode *pNode
```

DSLK đơn – Khai báo

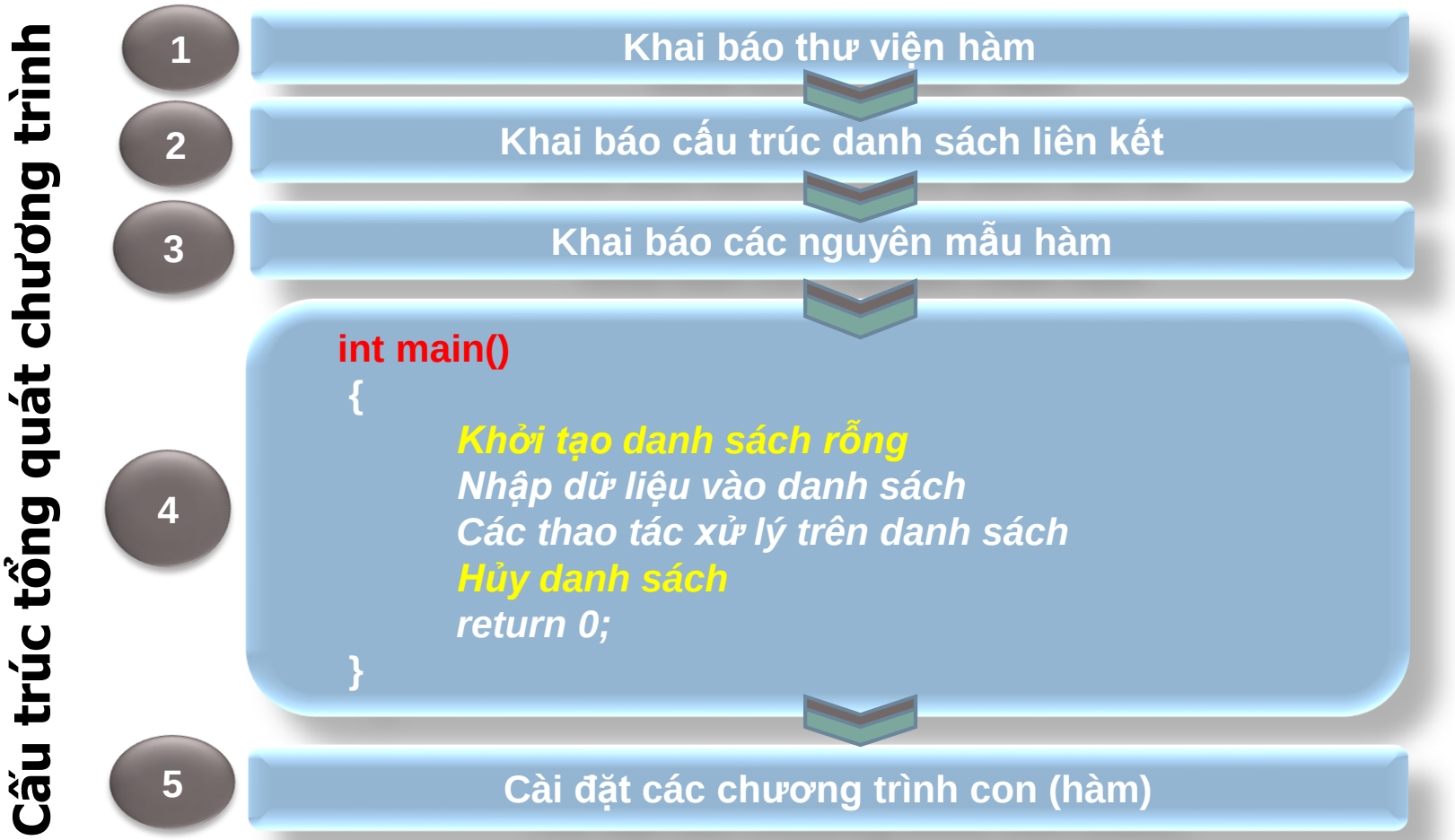
- **Tổ chức, quản lý:**
  - Để quản lý một DSLK đơn chỉ cần biết địa chỉ phần tử đầu danh sách
  - Con trỏ **pHead** sẽ được dùng để lưu trữ địa chỉ phần tử đầu danh sách. Ta có khai báo:

    Node *pHead;
  - Để tiện lợi, có thể sử dụng thêm một con trỏ **pTail** giữ địa chỉ phần tử cuối danh sách. Khai báo **pTail** như sau:

    Node *pTail;



Cài đặt chương trình DSLK đơn



Danh sách liên kết đơn (DSLK đơn)

- Khai báo
- Các thao tác cơ bản trên DSLK đơn
- Sắp xếp trên DSLK đơn

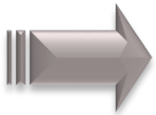
DSLK đơn

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - …

Tạo lập danh sách rỗng

pHead chưa xác định

Trước khi khởi tạo



pHead trỏ vào NULL (rỗng)

Sau khi khởi tạo

```
void Init(pNode &pHead)
{
        pHead = NULL;
}
```

Kiểm tra danh sách rỗng

List 

Danh sách rỗng

```
bool IsEmpty(pNode pHead)
{
        return pHead==NULL;
}
```

Tạo 1 nút mới


```
pNode CreateNode(int value)
{
    pNode p = new Node
      p->next = NULL;
      p->data = value;
       return p;
}
```

DSLK đơn

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - …

DSLK đơn – Các thao tác cơ sở

- **Thêm một phần tử vào danh sách**: Có 3 vị trí thêm
  - Gắn vào đầu danh sách
  - Gắn vào cuối danh sách
  - Chèn vào vị trí bất kỳ /sau (trước) nút q trong danh sách
- Chú ý trường hợp danh sách ban đầu rỗng

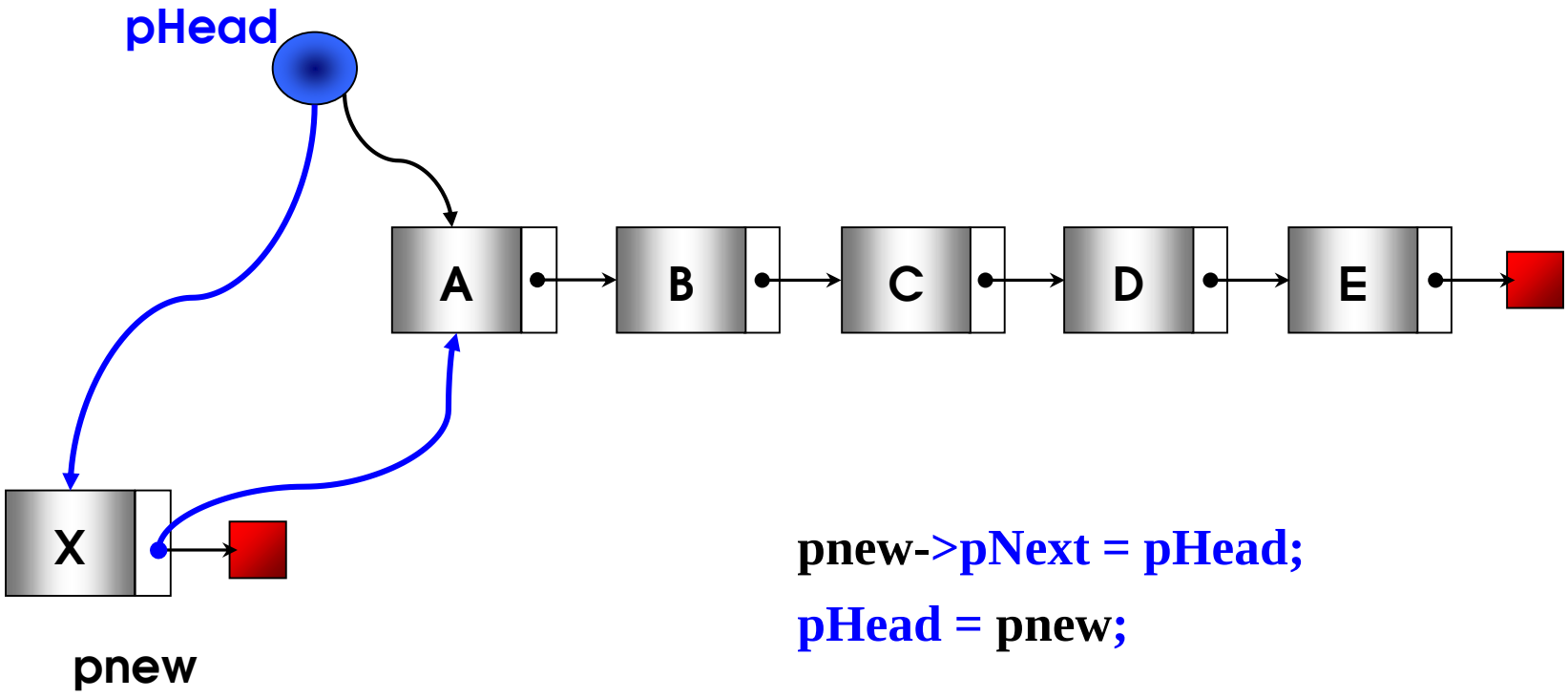
Thêm một nút vào danh sách

TH danh sách rỗng


```
if(IsEmpty(pHead))
{
        pHead = pNew;
}
```

DSLK đơn – Các thao tác cơ sở

- Thêm một phần tử
  - Nếu danh sách ban đầu không rỗng:
    - Gắn node vào đầu danh sách



pnew->pNext = pHead;

pHead = pnew;

DSLK đơn – Các thao tác cơ sở

Thuật toán: Gắn nút vào đầu DS

// input: danh sách, phần tử mới new_node

// output: danh sách với new_node ở đầu DS

- Nếu DS rỗng thì
  - pHead = pnew;
- Ngược lại
  - pnew->pNext = pHead;
  - pHead = pnew;

DSLK đơn – Các thao tác cơ sở

Cài đặt: Gắn nút vào đầu DS

```
pNode AddHead(pNode pHead, int value){
    pNode temp = CreateNode(value);
    if(pHead == NULL){
        pHead = temp;
    }
    else{
        temp->next = pHead;
        pHead = temp;
    }
    return pHead;
}
```

DSLK đơn – Các thao tác cơ sở

Thuật toán: Thêm một thành phần dữ liệu vào đầu DS

// input: danh sách l

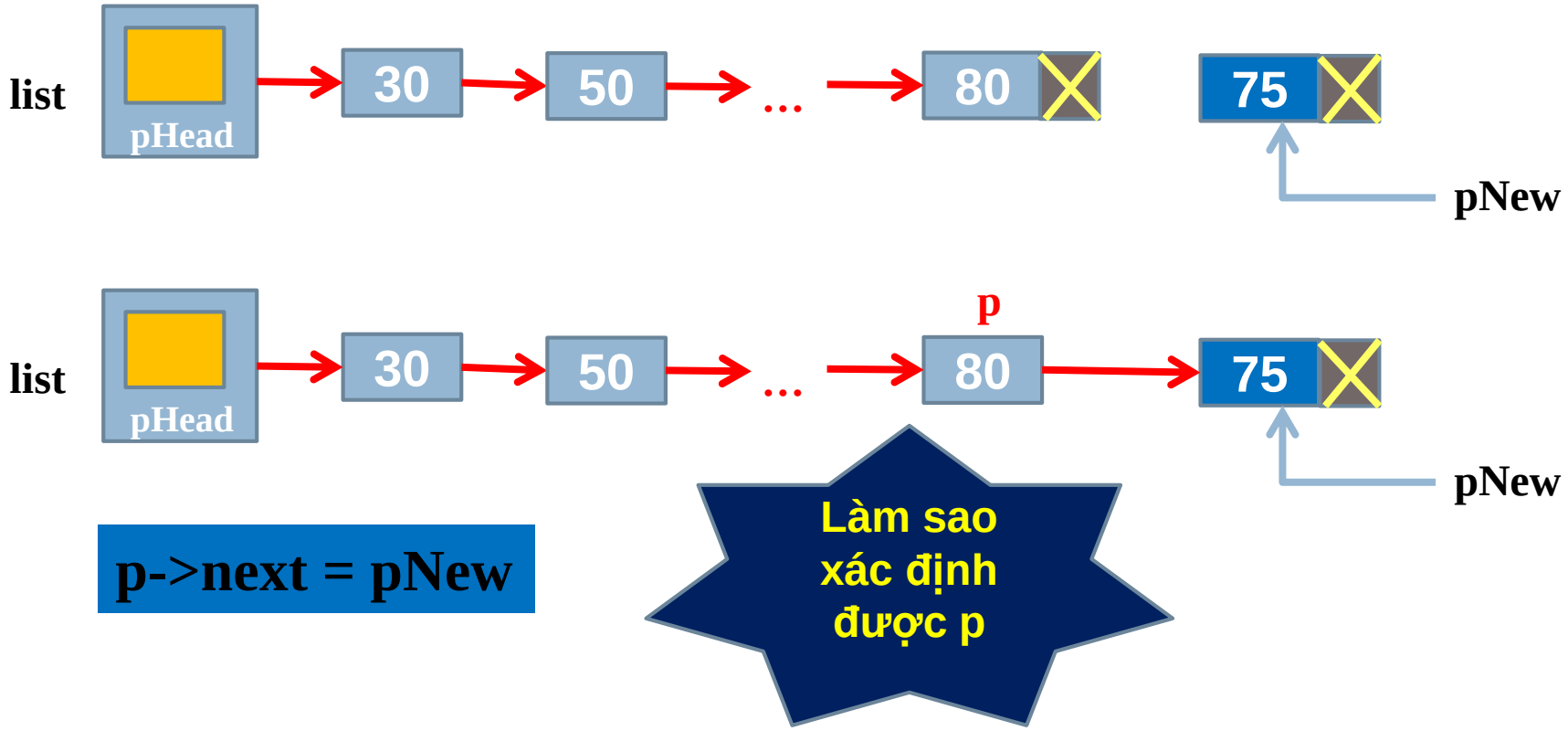
// output: danh sách l với phần tử chứa X ở đầu DS

- Nhập dữ liệu cho X (???)
- Tạo nút mới chứa dữ liệu X (???)
- Nếu tạo được:
  - Gắn nút mới vào đầu danh sách (???)

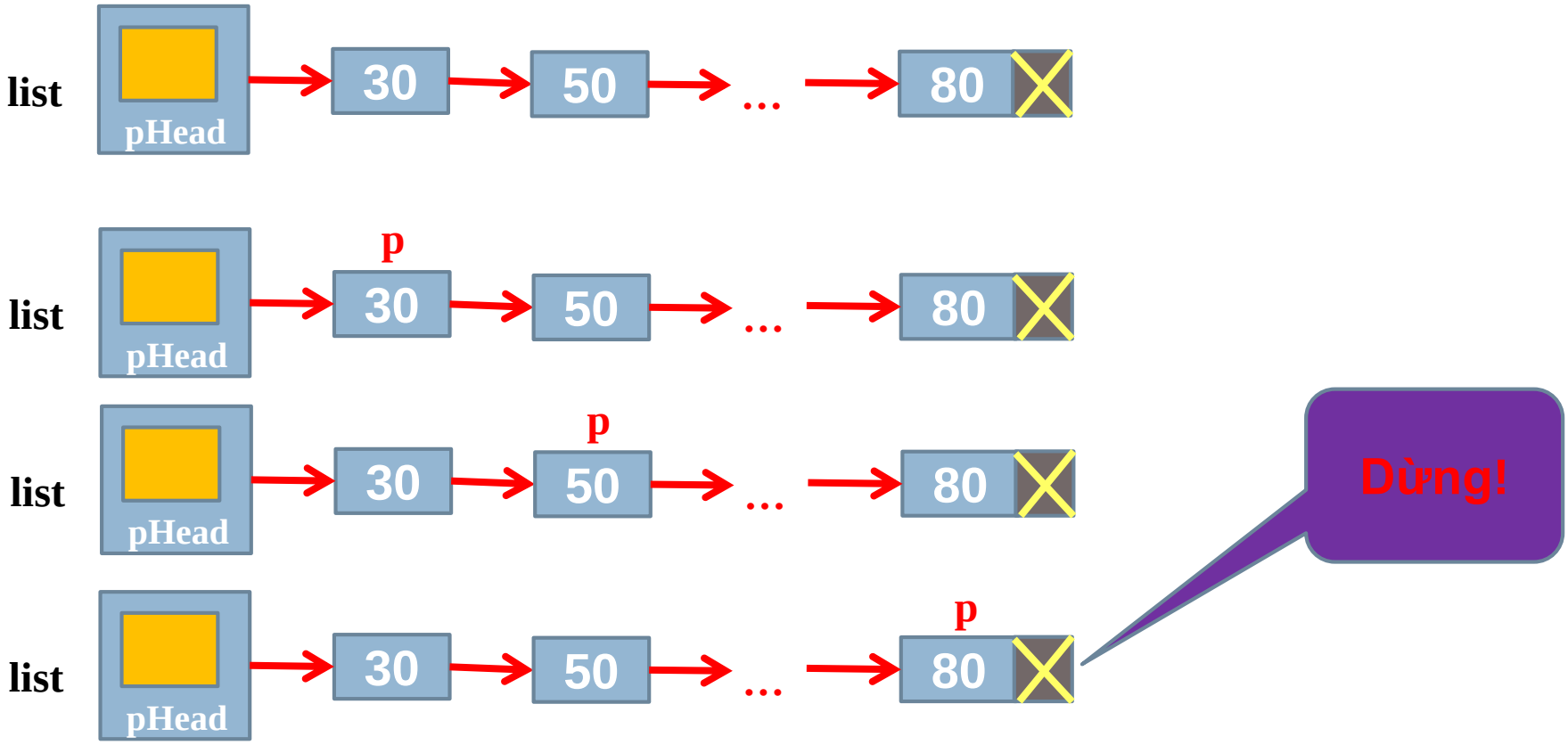
DSLK đơn – Các thao tác cơ sở

- **Thêm một phần tử vào danh sách**: Có 3 vị trí thêm
  - Gắn vào đầu danh sách
  - Gắn vào cuối danh sách
  - Chèn vào vị trí bất kỳ /sau (trước) nút q trong danh sách
- Chú ý trường hợp danh sách ban đầu rỗng

Trường hợp thêm một nút vào cuối danh sách



Trường hợp thêm một nút vào cuối danh sách



Thuật toán


```
pNode AddTail(pNode pHead, int value){
    pNode temp, p; // Khai báo 2 node tam temp và p
    temp = CreateNode(value);//Goi ham createNode de khoi tao node
temp có next tro toi NULL và giá tri là value
    if(pHead == NULL){
        pHead = temp;
    }
    else{
        p  = pHead;
        while(p->next != NULL){
            p = p->next;
        }
          p->next = temp;
    }
    return pHead;
}
```

DSLK đơn – Các thao tác cơ sở

Thuật toán: Thêm một thành phần dữ liệu vào cuối ds

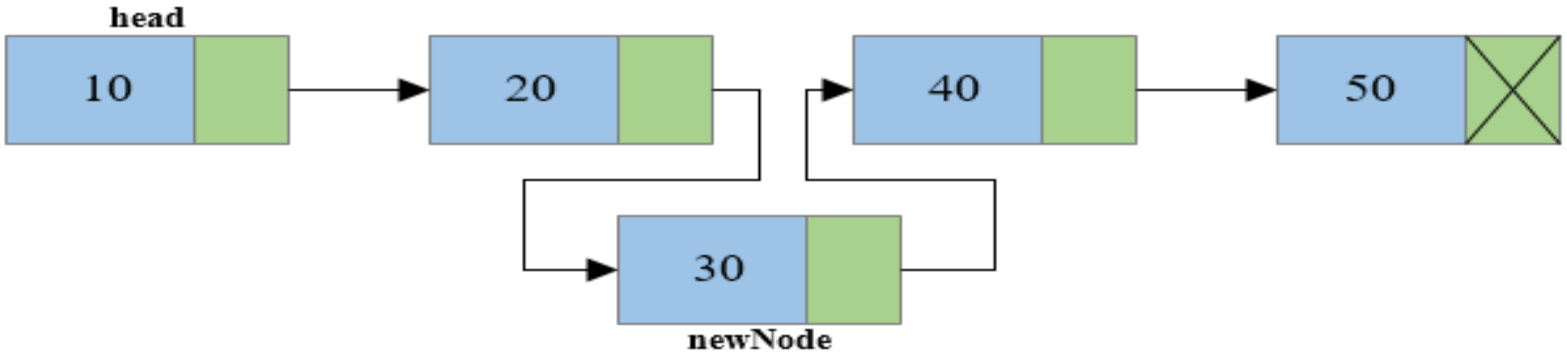
// input: danh sách thành phần dữ liệu X

// output: danh sách với phần tử chứa X ở cuối DS

- Nhập dữ liệu cho X (???)
- Tạo nút mới chứa dữ liệu X (???)
- Nếu tạo được:
  - Gắn nút mới vào cuối danh sách (???)

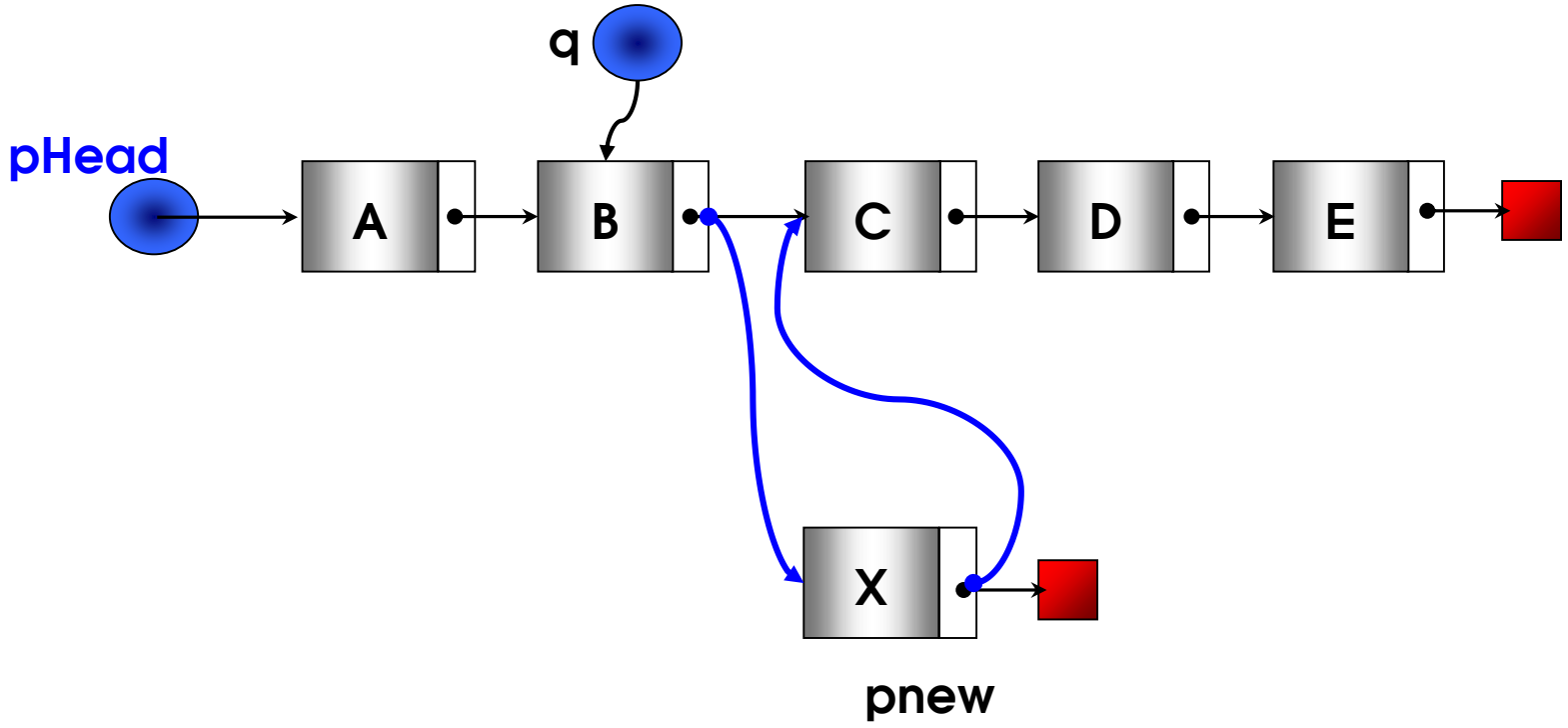
DSLK đơn – Các thao tác cơ sở

- **Thêm một phần tử vào danh sách**: Có 3 vị trí thêm
  - Gắn vào đầu danh sách
  - Gắn vào cuối danh sách
  - Chèn vào vị trí bất kỳ /sau (trước) nút q trong danh sách
- Chú ý trường hợp danh sách ban đầu rỗng


head
10
20
40
50
30
newNode

DSLK đơn – Các thao tác cơ sở

- Thêm một phần tử
  - Nếu danh sách ban đầu không rỗng
    - **Chèn một phần tử sau q**



DSLK đơn – Các thao tác cơ sở

Thuật toán: Chèn một phần tử sau q

// input: danh sách l, q, phần tử mới new_node

// output: danh sách với new_node ở sau q

- Nếu (q != NULL) thì:

  pnew -> next = q -> next;

  q -> next = pnew;

  Nếu (q ->next==NULL) thì chèn cuối ds

- Ngược lại

  Thêm new_node vào đầu danh sách

DSLK đơn – Các thao tác cơ sở

Thuật toán: Thêm một thành phần dữ liệu vào sau q

// input: danh sách thành phần dữ liệu X

// output: danh sách với phần tử chứa X ở cuối DS

- Nhập dữ liệu cho nút q (???)
- Tìm nút q (???)
- Nếu tồn tại q trong ds thì:
  - Nhập dữ liệu cho X (???)
  - Tạo nút mới chứa dữ liệu X (???)
  - Nếu tạo được:
    - Gắn nút mới vào sau nút q (???)
- Ngược lại thì báo lỗi

Bài tập

- Viết các hàm liên quan cho phép chèn 1 node có giá trị x vào phía sau node có giá trị lớn nhất trong danh sách list (***giả sử danh sách không có giá trị trùng nhau***)
- Viết đoạn code chèn sau vị trí pos cho trước
- Sử dụng những hàm có sẵn, viết bổ sung hàm chèn node có giá trị y vào trước node có giá trị x (*giả sử chỉ có 1 node có giá trị x nếu có*)

DSLK đơn

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - …

DSLK đơn – Các thao tác cơ sở

- Duyệt danh sách
  - Là thao tác thường được thực hiện khi có nhu cầu xử lý các phần tử của danh sách theo cùng một cách thức hoặc khi cần lấy thông tin tổng hợp từ các phần tử của danh sách như:
    - Đếm các phần tử của danh sách
    - Tìm tất cả các phần tử thoả điều kiện
    - Hủy toàn bộ danh sách (và giải phóng bộ nhớ)
    - …

DSLK đơn – Các thao tác cơ sở

- Duyệt danh sách
  - Bước 1: p = pHead; *//Cho p trỏ đến phần tử đầu danh sách*
  - Bước 2: Trong khi (Danh sách chưa hết) thực hiện:
    - B2.1 : Xử lý phần tử p
    - B2.2 : p=p->next; *// Cho p trỏ tới phần tử kế*

```
void processList (pNode pHead)
{
        pNode p = pHead;
        while (p!= NULL)
        {
                // xử lý cụ thể p tùy ứng dụng
                p = p->next;
        }
}
```

DSLK đơn – Các thao tác cơ sở

Ví dụ: In các phần tử trong danh sách

```
void Output (pNode pHead)
{
        pNode p=pHead;
        while (p!=NULL)
        {
                printf("%2d",p->data);
                p=p ->next;
        }
}
```

DSLK đơn

- **Các thao tác cơ bản**
  - Tạo danh sách rỗng
  - Thêm một phần tử vào danh sách
  - Duyệt danh sách
  - Tìm kiếm một giá trị trên danh sách
  - Xóa một phần tử ra khỏi danh sách
  - Hủy toàn bộ danh sách
  - …

DSLK đơn – Các thao tác cơ sở

- Tìm kiếm một phần tử có khóa x

```
Node* Search (pNode  pHead, int  x)
{
        pNode  p = pHead;
        while (p!=NULL) {
                if (p->data==x)
                        return p;
                p=p->pNext;
        }
        return NULL;
}
```