
Danh sách liên kết



Giảng viên: Văn Thị Thiên Trang

Đại học Kỹ Thuật Công Nghệ TP. HCM

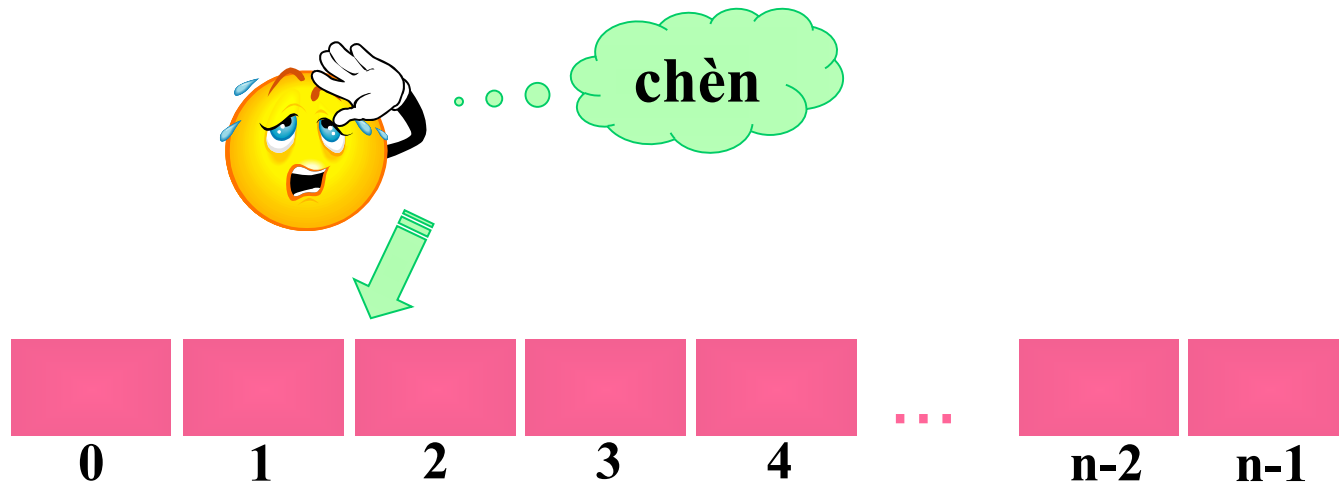
Nội dung

- Danh sách liên kết đơn
 - Giới thiệu
 - Cài đặt
 - Thao tác
 - Ứng dụng
- Danh sách vòng
- Danh sách liên kết kép

Singly Linked List - Giới thiệu

■ Mảng 1 chiều

- Kích thước cố định (fixed size)
- Các phần tử tuần tự theo chỉ số $0 \Rightarrow n-1$
- Truy cập ngẫu nhiên (random access)
- Chèn 1 phần tử vào mảng rất khó

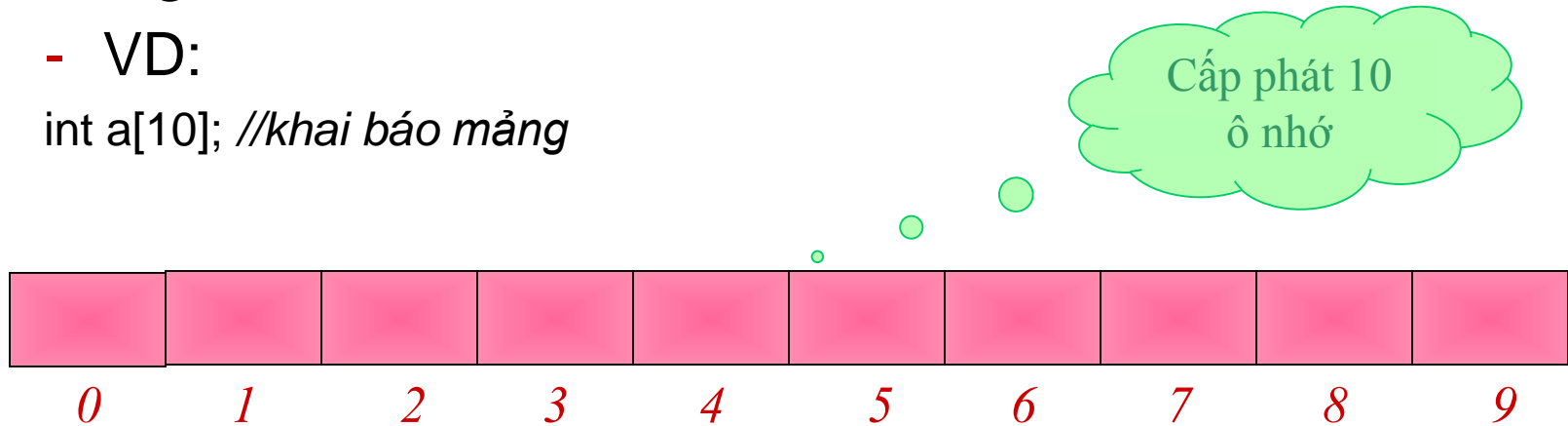


Singly Linked List - Giới thiệu

■ Mảng 1 chiều

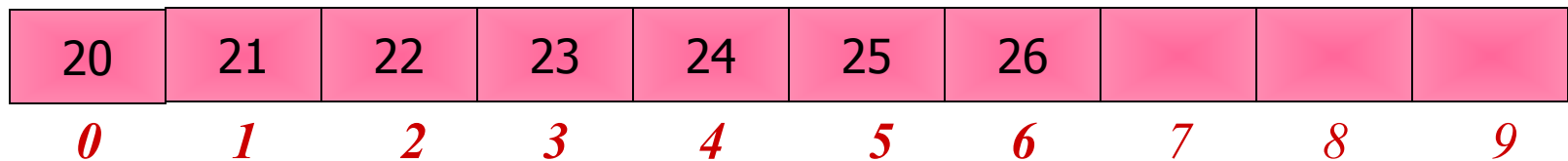
- VD:

```
int a[10]; //khai báo mảng
```



```
Int n=7; //khai báo số phần tử của mảng
```

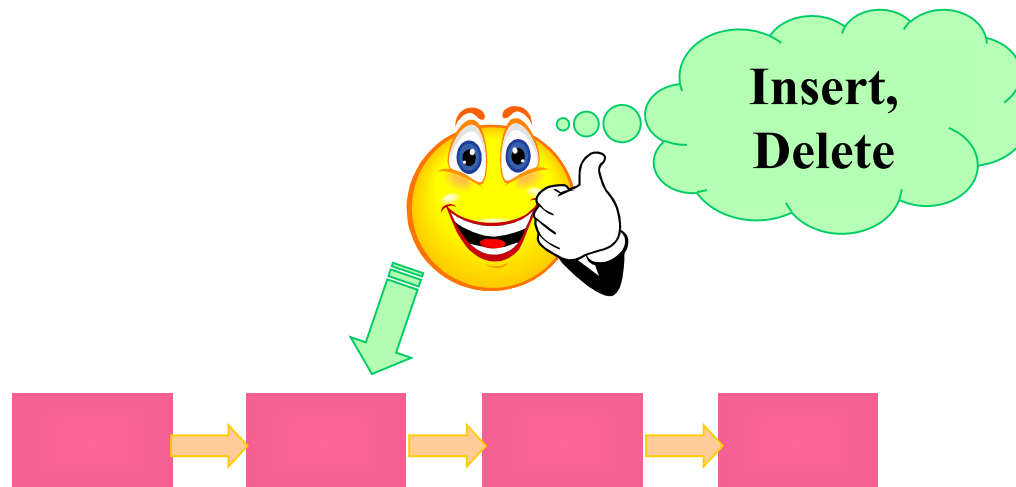
```
a={20, 21, 22, 23, 24, 25, 26}
```



Chèn phần tử 19 vào mảng???

SLL - Giới thiệu

- Danh sách liên kết
 - 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 xóa đơn giản



SLL – Ôn pointer

■ Nhắc lại pointer

`int i, *pi;`

<code>i</code>	<code>1FF0</code> ?	<code>pi</code>	<code>1FF4</code> ?
----------------	------------------------	-----------------	------------------------

`pi = &i;`

<code>i</code>	<code>1FF0</code>	<code>pi</code>	<code>1FF4</code> <code>1FF4</code> <code>1FF0</code>
<code>*pi</code>			

`i = 10 or *pi = 10`

<code>i</code>	<code>1FF0</code>	<code>pi</code>	<code>1FF4</code> <code>1FF0</code>
<code>*pi</code>	? <code>10</code>		

SLL – Ôn pointer

```
typedef struct SINHVIEN{  
    long MaSV;  
    char* HoTen;  
    float DiemTB;  
}SV;
```

```
SV s;
```

```
s.MaSV=12345;
```

```
s.HoTen="Ly  
An";
```

```
s.DTB=2.4;
```

```
SV *p;
```

```
(*p).MaSV=1234  
5;
```

```
(*p).HoTen="Ly  
An";
```

```
(*p).DTB=2.4;
```

```
SV *p;
```

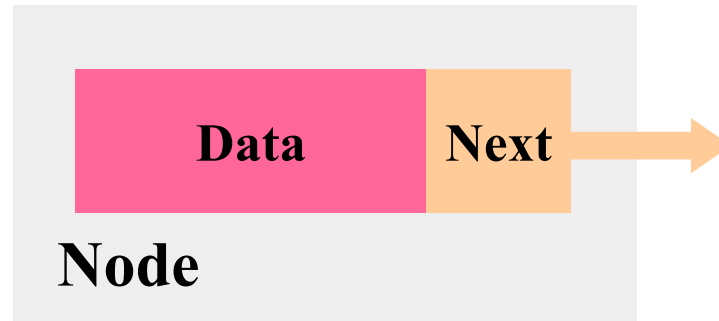
```
p→MaSV=12345;
```

```
p→.HoTen="Ly  
An";
```

```
p→.DTB=2.4;
```

SLL - định nghĩa

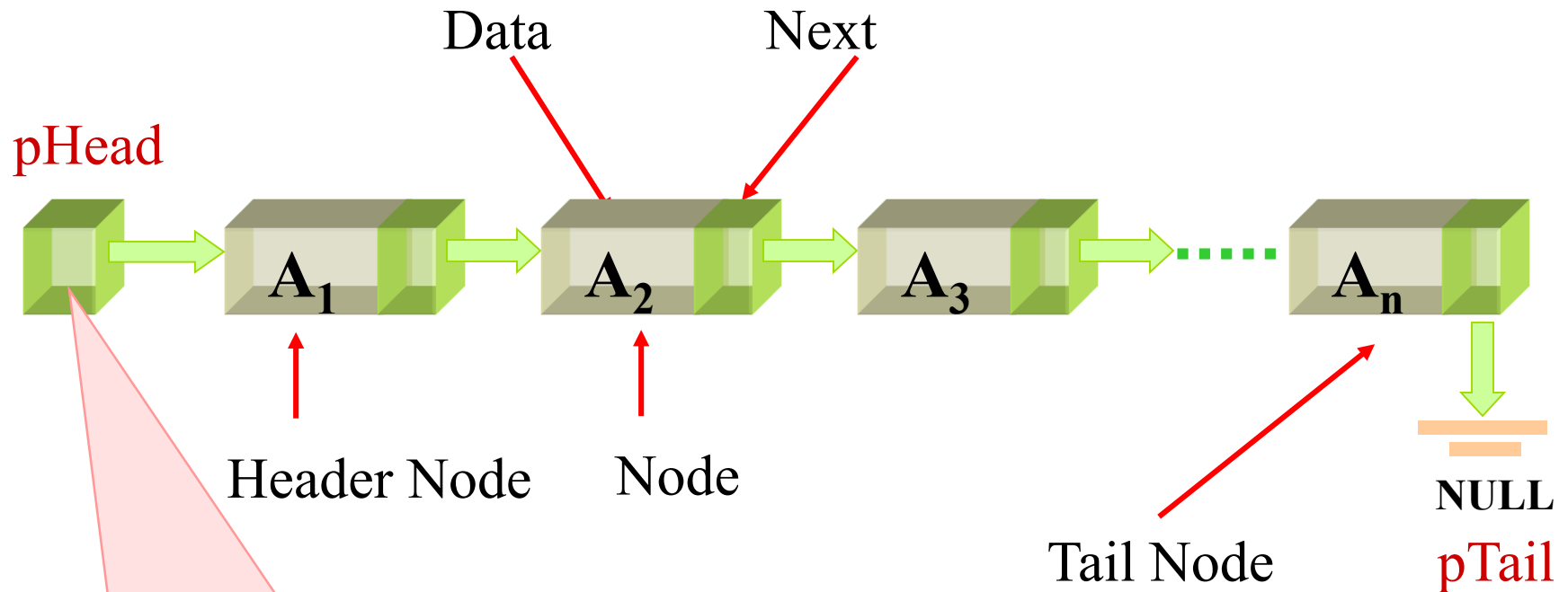
- DSLK đơn là một danh sách, mỗi phần tử gồm 2 thành phần:
 - Phần chứa dữ liệu - Data
 - Phần chỉ vị trí của phần tử tiếp theo trong danh sách – Next



- Mỗi phần tử được gọi là một Node
- Phần next là con trỏ, trỏ đến node kế tiếp

SLL – Minh hoạ

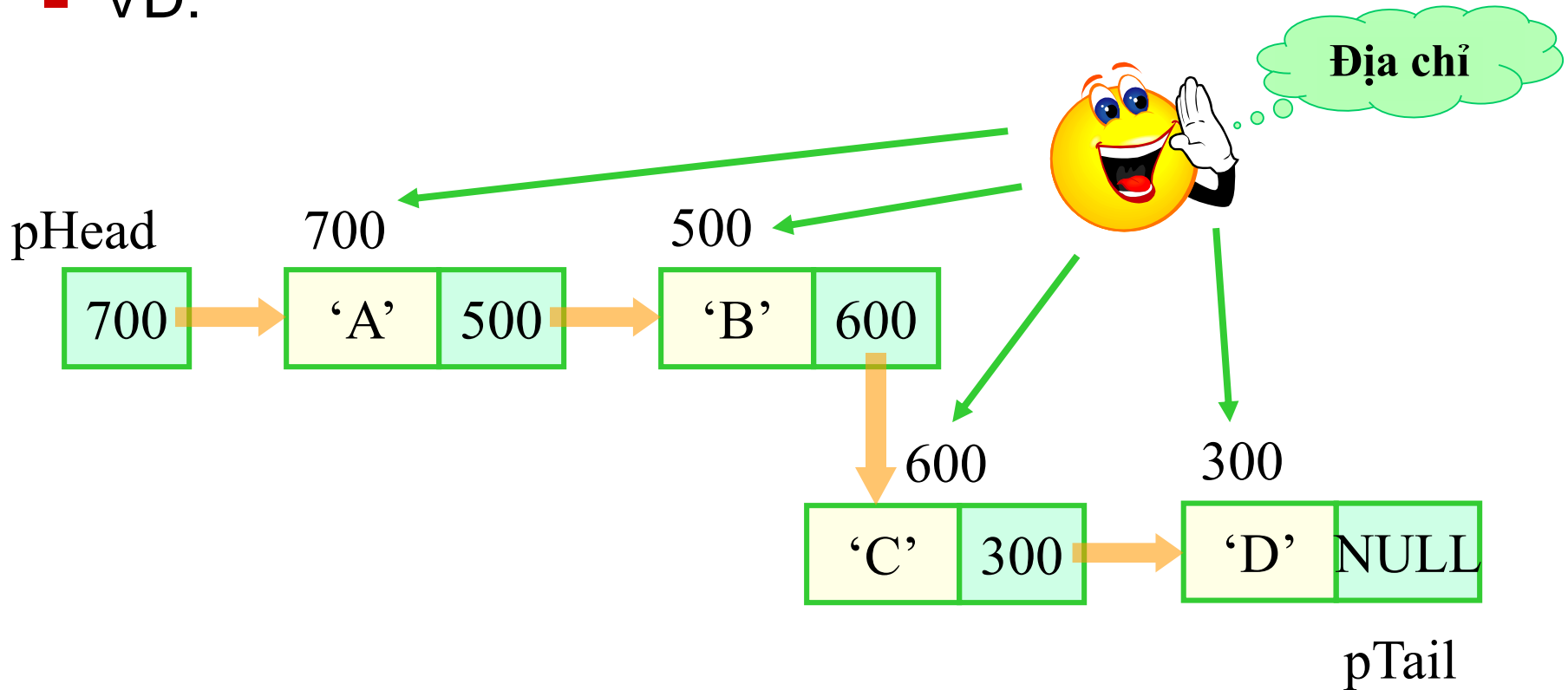
■ Mô tả DSLK



*Con trỏ đến node đầu tiên
(giữ địa chỉ của node đầu tiên)*

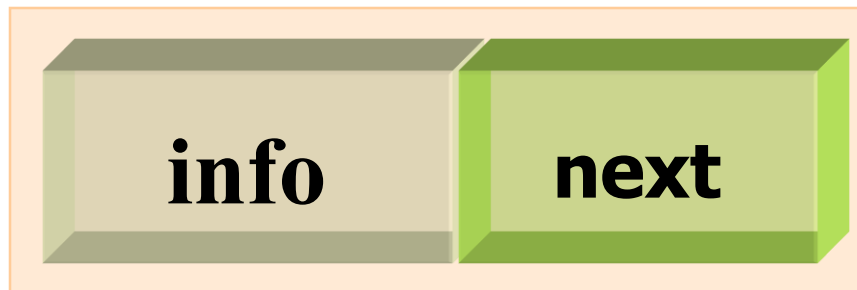
SLL – Minh họa

■ VD:



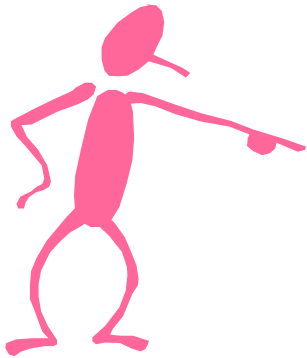
SLL- Khai báo một NODE

```
typedef struct node
{
    DataType    info;
    struct node * next;
}NODE;
```



SLL – Khai báo phần data

- Khai báo DSLK - DataType
 - Kiểu dữ liệu định nghĩa trước
 - Chứa dữ liệu, thông tin của từng node



```
typedef int DataType;  
typedef char DataType;  
typedef struct  
{  
    char Ten[30];  
    char MaSo[10];  
    DateTime NgaySinh;  
    char Khoa[10];  
} SinhVien;  
typedef SinhVien DataType;
```

SLL – Khai báo phần data

```
typedef struct node
{
    DataType    info;
    struct node * next;
}Node;
```

Cấu trúc
node

```
typedef struct node
{
    int    info;
    struct node * next;
}Node;
```

```
typedef struct node
{
    SinhVien    info;
    struct node * next;
}Node;
```

SLL – Khai báo DSLK

■ Khai báo và khởi tạo

```
typedef struct node
{
    DataType    info;
    struct node * next;
}Node;
```

Node* pHead;



pHead quản lý ds

pHead = NULL;



Khởi tạo dslk

SLL – Thao tác cơ bản

■ Các thao tác cơ bản

1. Init
2. IsEmpty
3. InsertFirst
4. InsertAfter
5. DeleteFirst
6. DeleteAfter
7. DeleteAll
8. ShowList
9. Search
10. Sort



Phần minh
hoạ sẽ dùng
DataType là
int

```
typedef struct node
{
    int    info;
    struct node * next;
}Node;

Node*    pHead;

pHead = NULL;
```

SLL – 1. Khởi tạo

- **Init:** khởi động danh sách, ban đầu chưa có phần tử

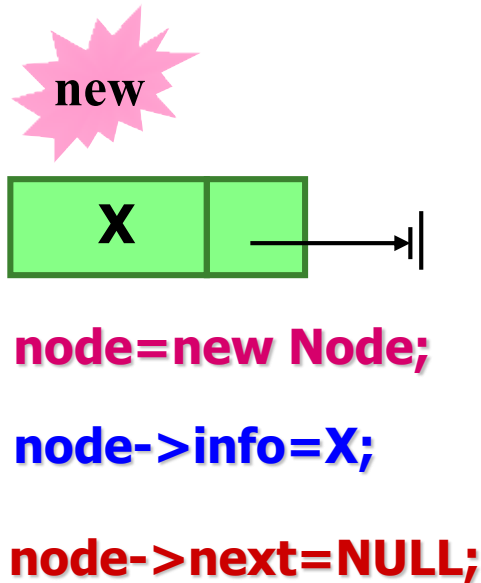
```
void Init(Node * &pHead)
{
    pHead = NULL;
}
```


SLL – 2. Kiểm tra DS rỗng

- **IsEmpty**: kiểm tra danh sách rỗng

```
//trả về 1: danh sách rỗng  
//trả về 0: danh sách không rỗng  
  
int IsEmpty(Node* pHead)  
{  
    return (pHead==NULL)? 1: 0;  
}
```

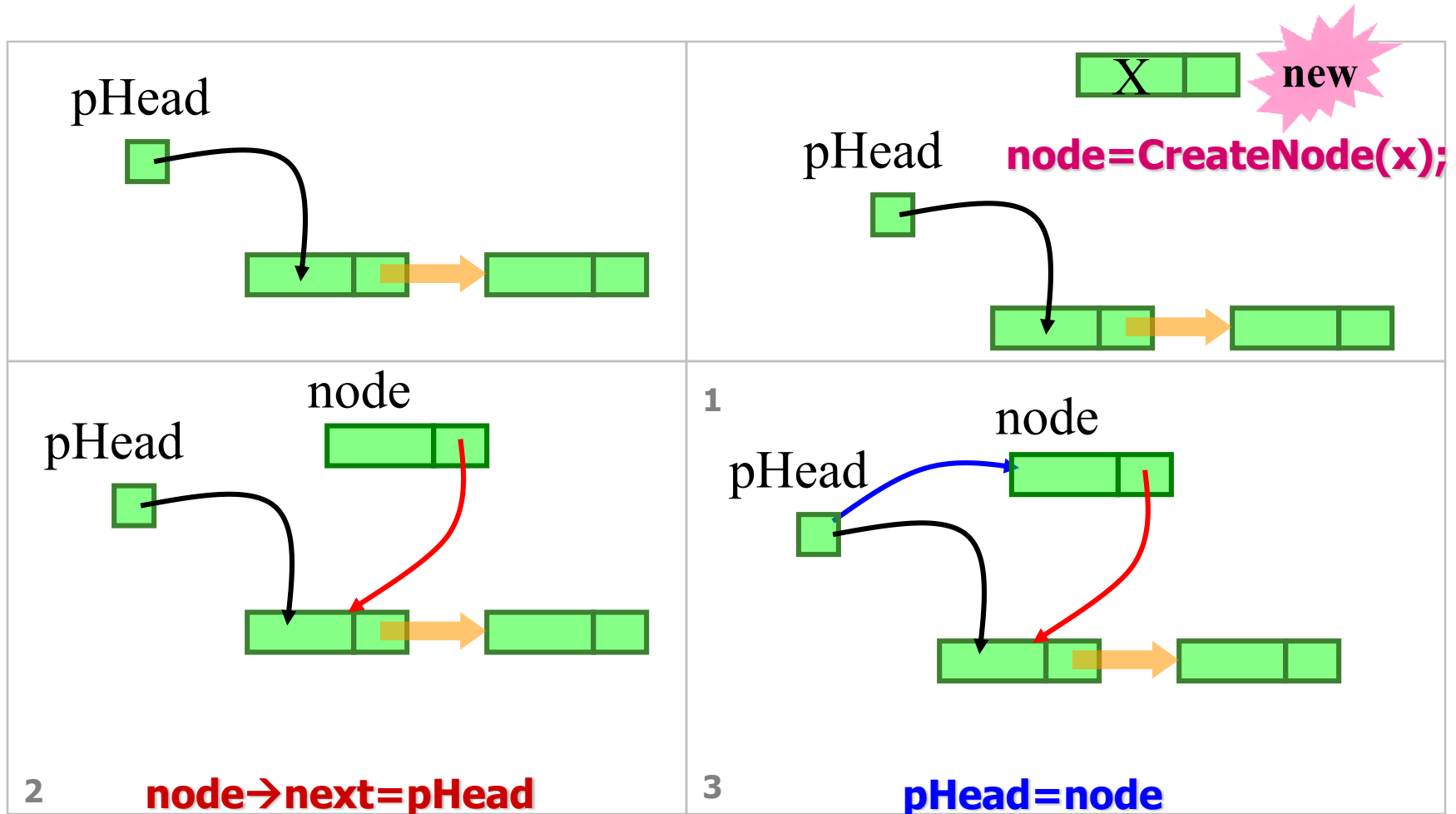
SLL – Tạo nút có dữ liệu X



```
Node* CreateNode(int X)
{
    Node* node=new Node;
    node->info=X;
    node->next=NULL;
    return node;
}
```

SLL – 3. Thêm vào đầu

■ Thêm vào đầu danh sách



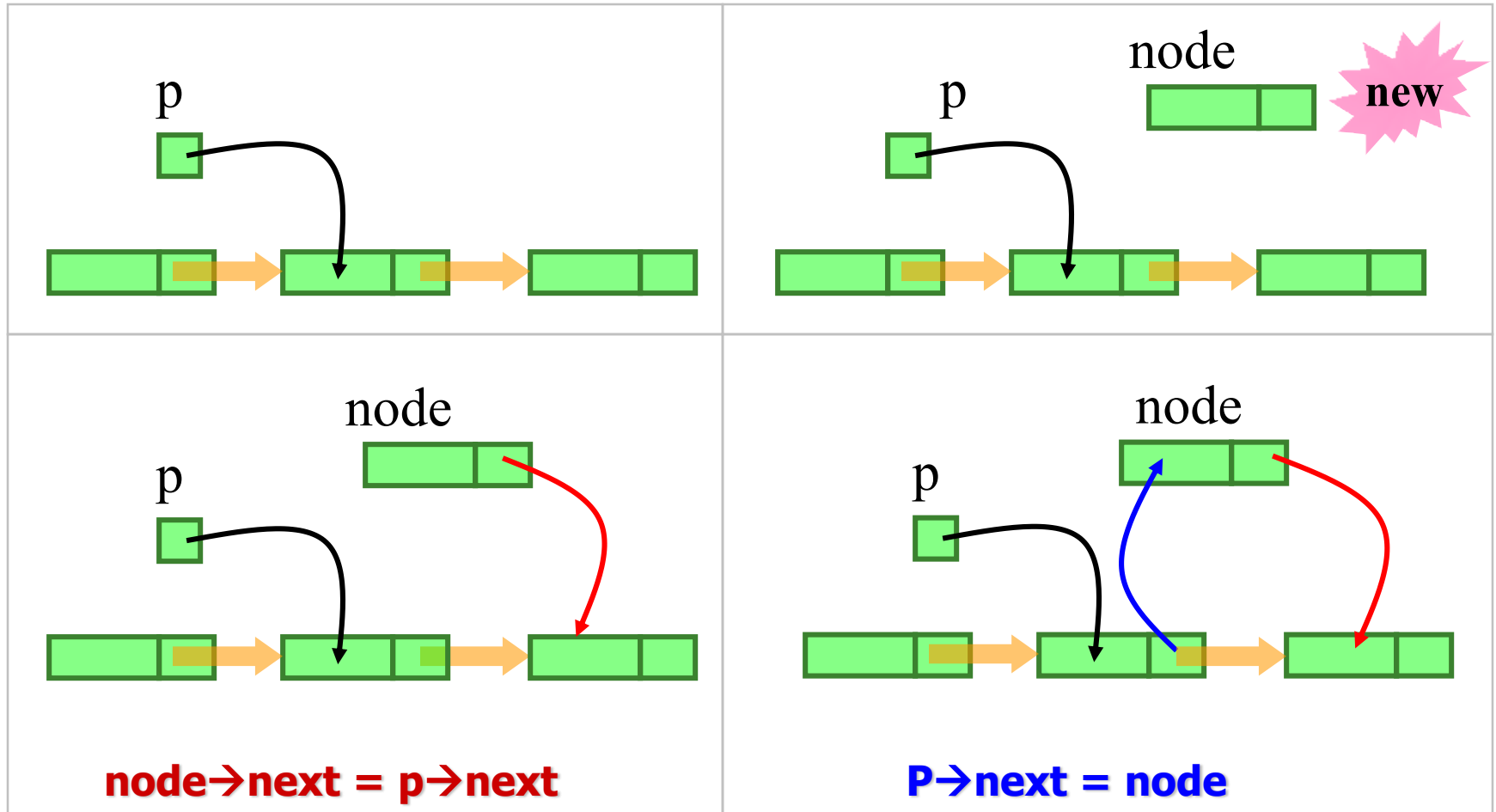
SLL – 3. Thêm vào đầu

- InsertFirst: thêm nút có nội dung x vào đầu ds

```
void InsertFirst(Node* &pHead, int x)
{
    Node* node;
    node = CreateNode(x); //Tạo nút có dữ liệu X
    node->next = pHead;
    pHead = node;
}
```

SLL – 4. Thêm vào sau 1 phần tử

- Thêm vào sau node p trong danh sách



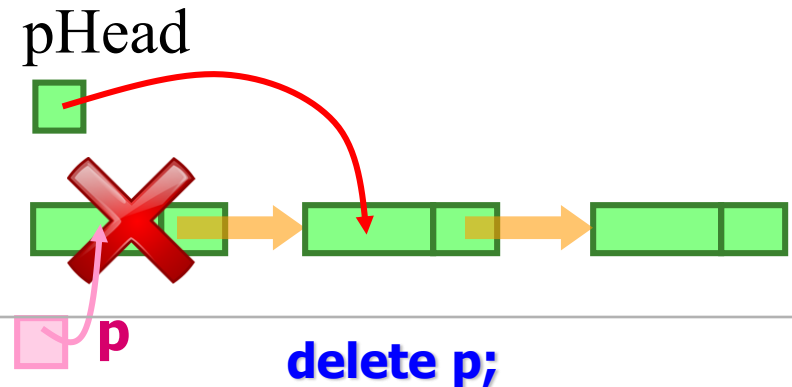
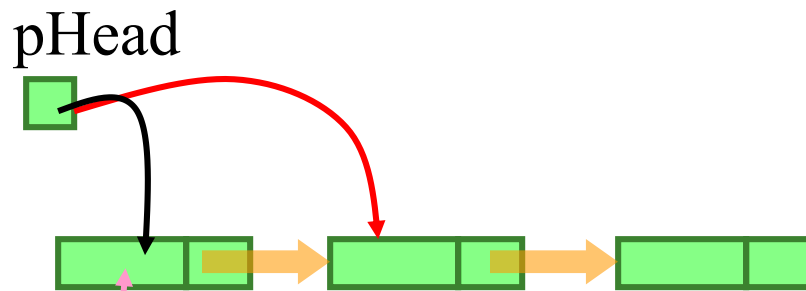
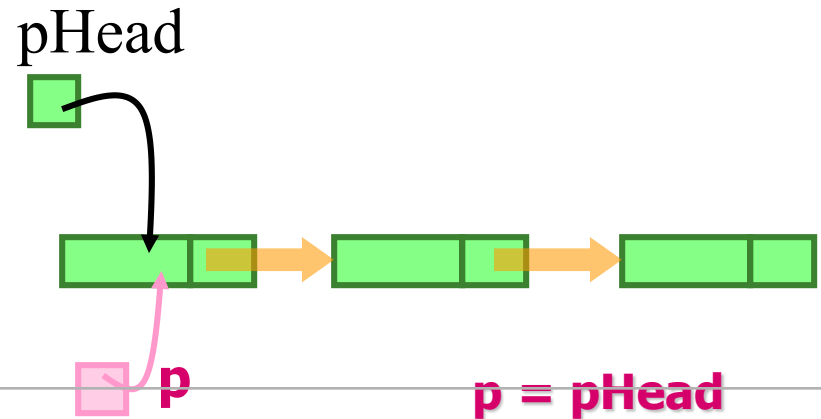
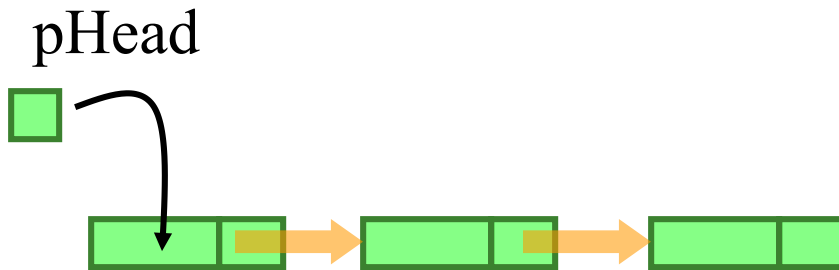
SLL – 4. Thêm vào sau 1 phần tử

- InsertAfter: thêm node có nội dung x sau node p

```
void InsertAfter(Node* &p, int x){
    Node* node;
    if (p == NULL)
        printf("Cannot insert new node!");
    else
    {
        node = CreateNode(x);
        node->next = p->next;
        p->next = node;
    }
}
```

SLL – 5. Xóa phần tử đầu danh sách

■ Xóa phần tử đầu danh sách



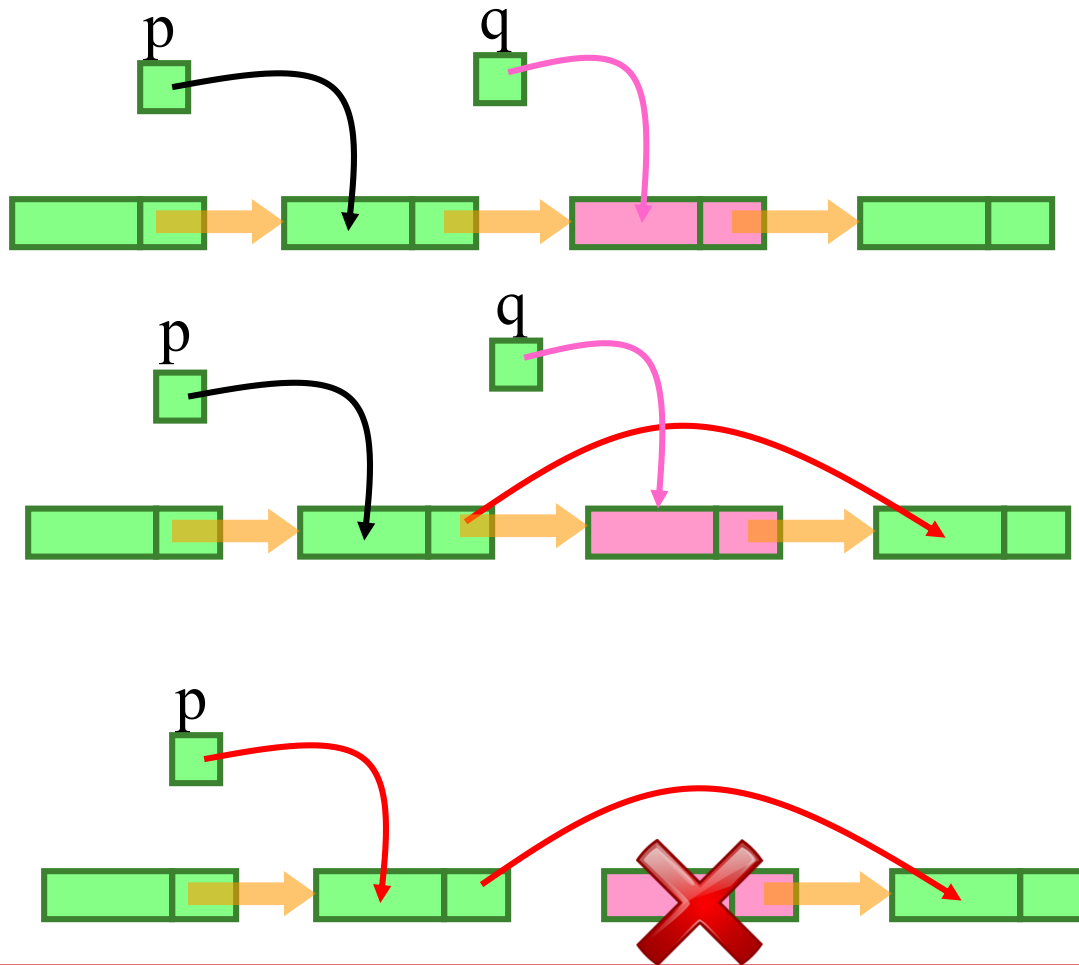
SLL – 5. Xóa phần tử đầu

- DeleteFirst: xóa node đầu tiên của danh sách

```
void DeleteFirst(Node* &pHead){  
    Node* p;  
    if (IsEmpty(pHead))  
        printf("List is empty!");  
    else {  
        p = pHead;  
        pHead = pHead->next;  
        delete p;  
    }  
}
```


SLL – 6. Xoá phần tử sau 1 node

- Xoá phần tử sau node p trong danh sách



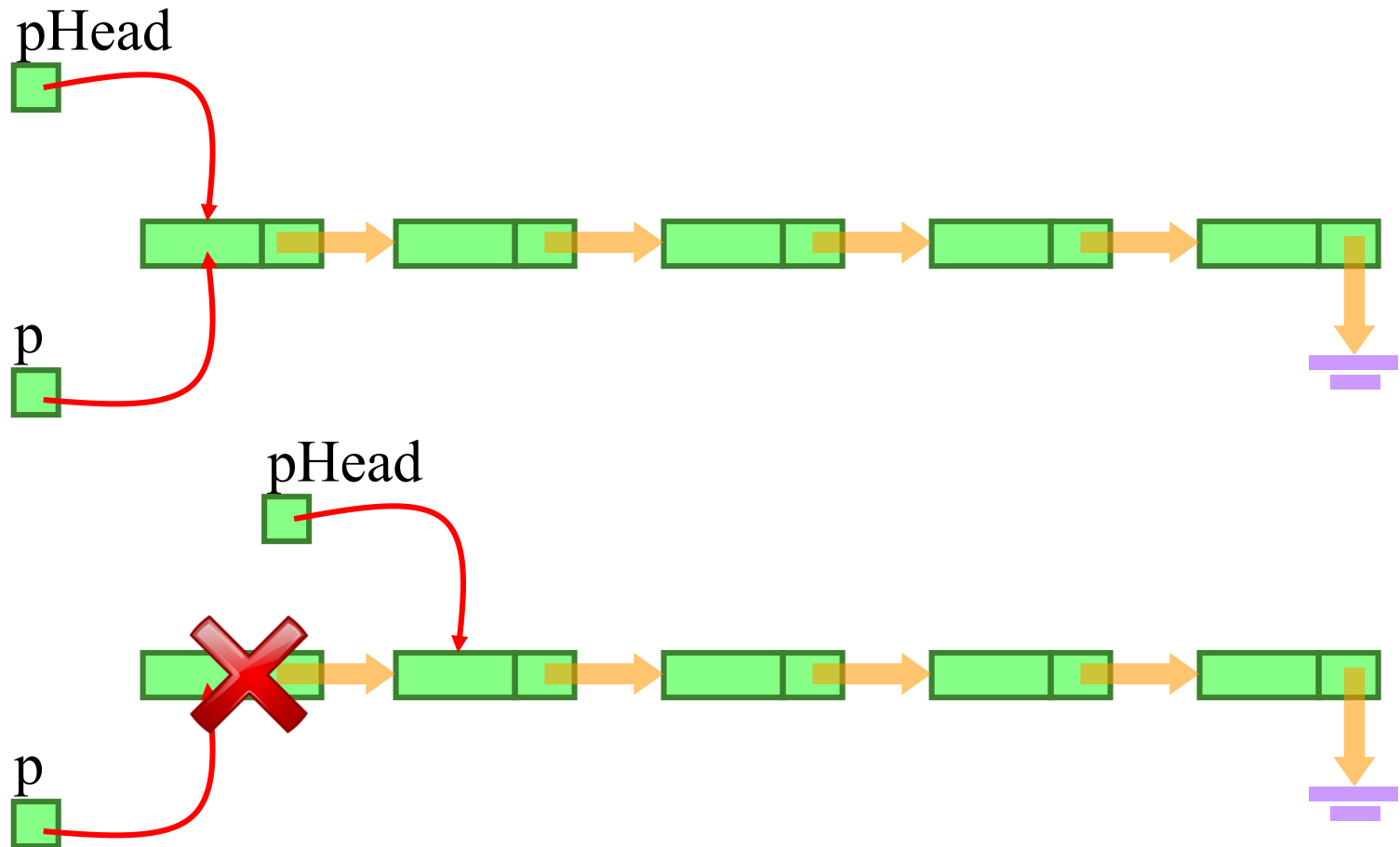
SLL – 6. Xoá phần tử sau 1 node

- DeleteAfter: xoá node sau node p trong danh sách

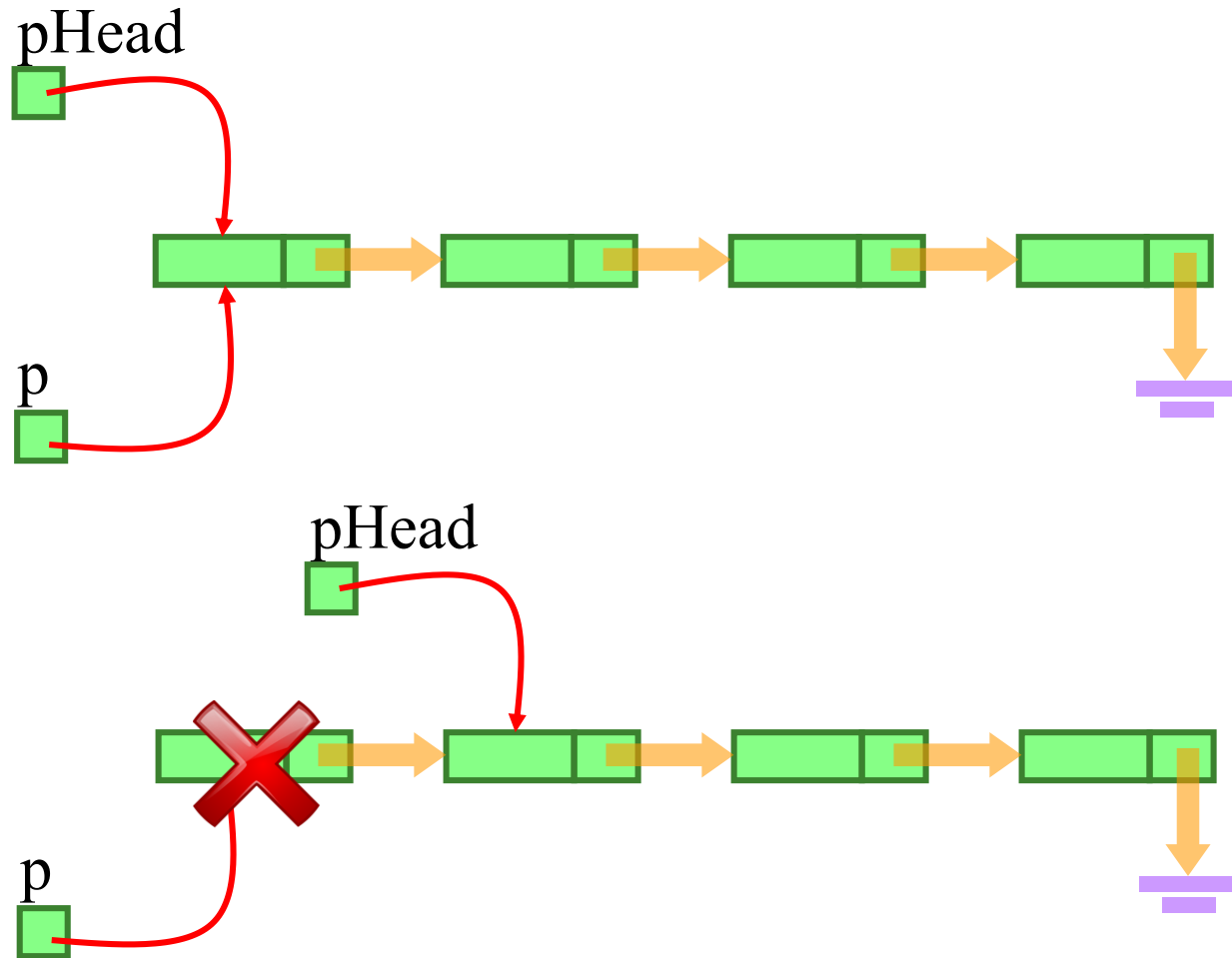
```
void DeleteAfter(Node* &p){  
    Node* q;  
    if (p->next == NULL)  
        printf("Cannot delete node!");  
    else  
    {  
        q = p->next;  
        p->next = q->next;  
        delete q;  
    }  
}
```

SLL – 7. Xoá toàn bộ danh sách

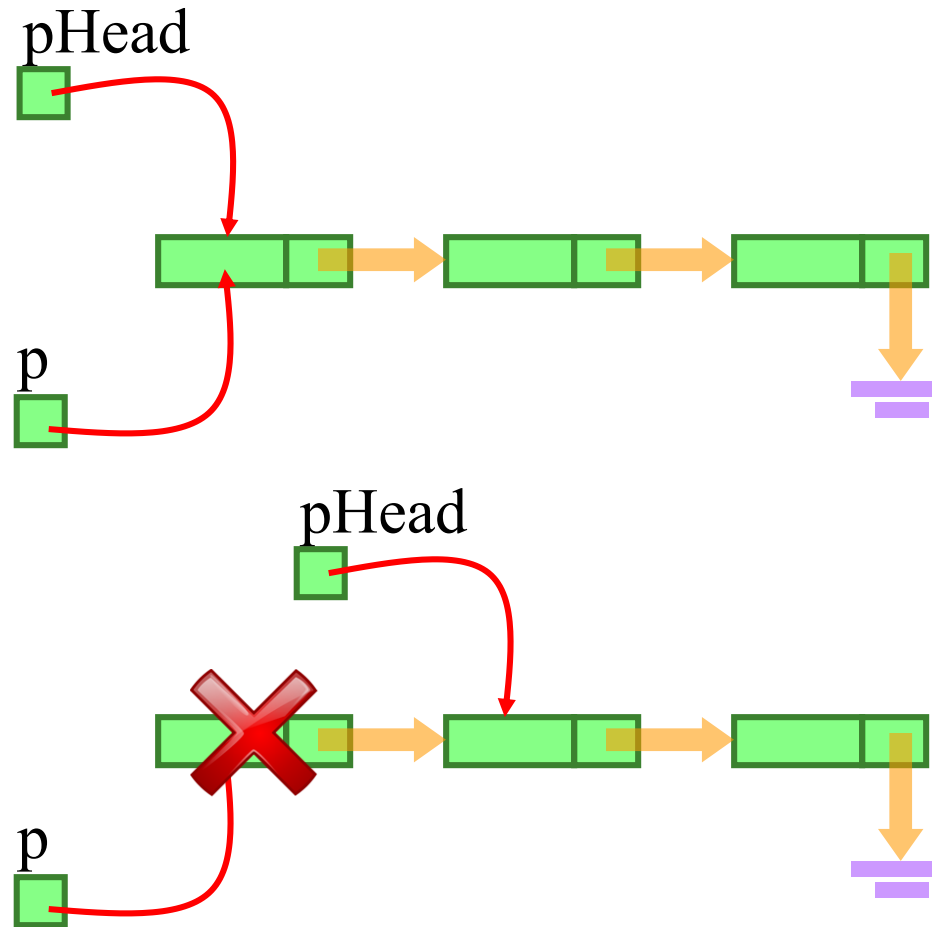
- Xoá toàn bộ danh sách



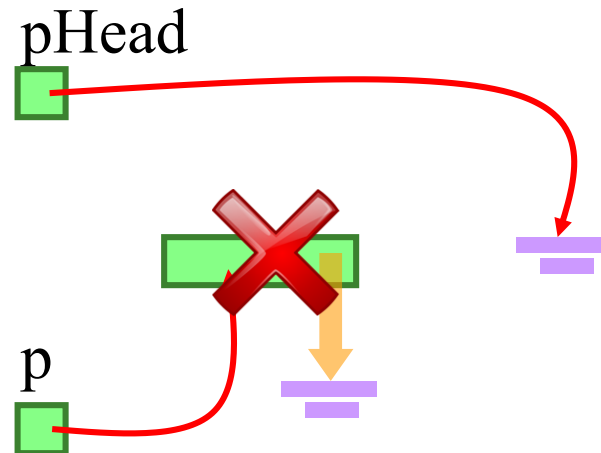
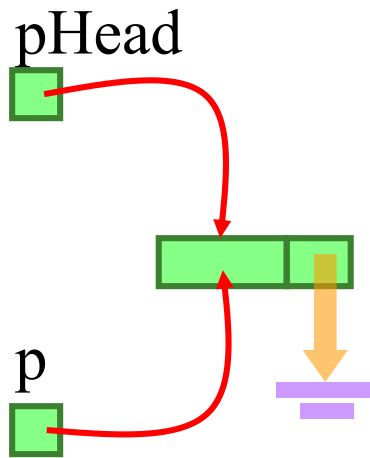
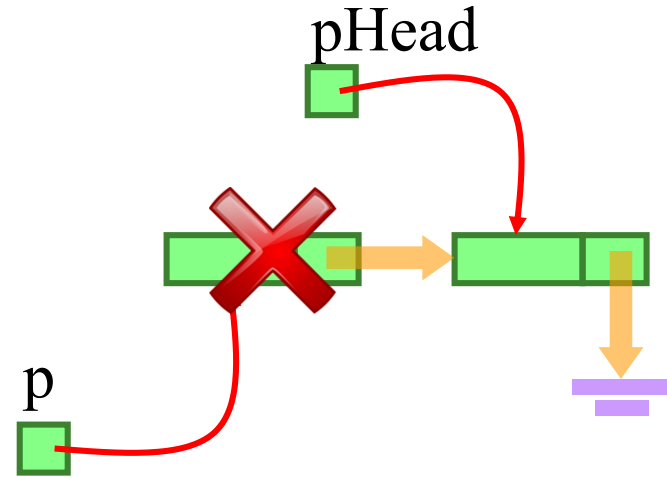
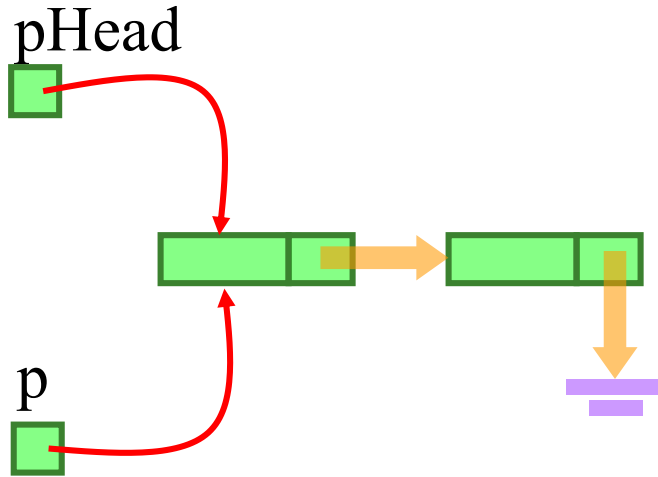
SLL – 7. Xoá toàn bộ danh sách



SLL – 7. Xoá toàn bộ danh sách



SLL- 7. Xoá toàn bộ danh sách



SLL- 7. Xoá toàn bộ danh sách

- DeleteAll: xoá toàn bộ danh sách

```
void DeleteAll(Node* &pHead)
{
    Node* p;
    while (pHead!=NULL)
    {
        p = pHead;
        pHead = p->next;
        delete p;
    }
}
```

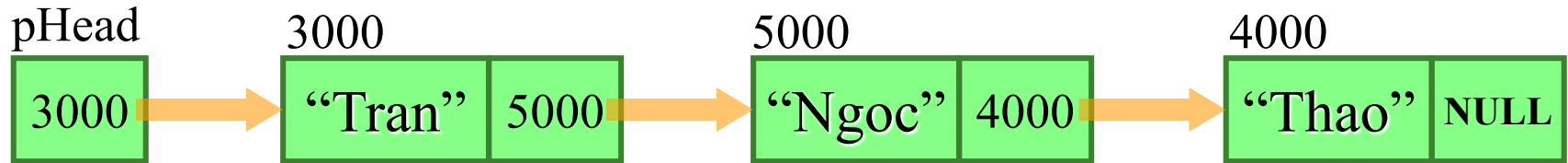
SLL – 8. In danh sách

- **ShowList:** in toàn bộ danh sách

```
void ShowList(Node* pHead)
{
    Node* p;
    p = pHead;
    while (p!=NULL)
    {
        ShowNode(p);
        p = p->next;
    }
}
```

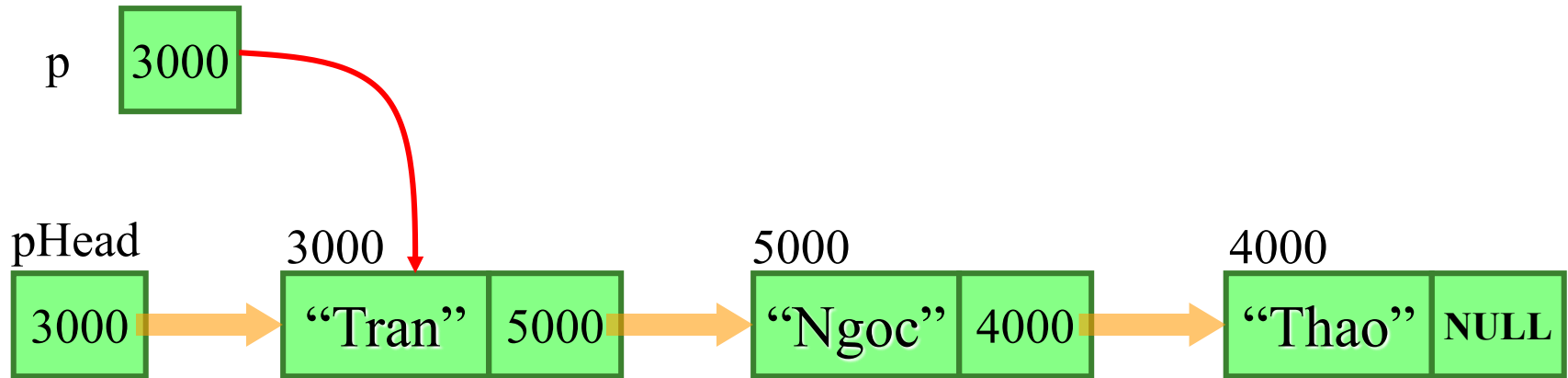

SLL – Minh họa in danh sách

p



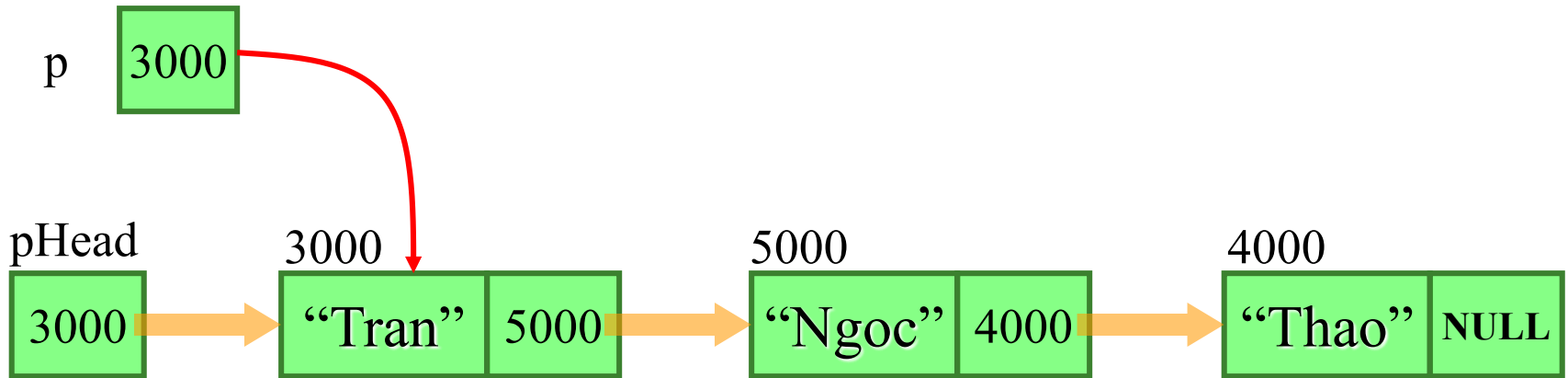
```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

SLL – Minh họa in danh sách



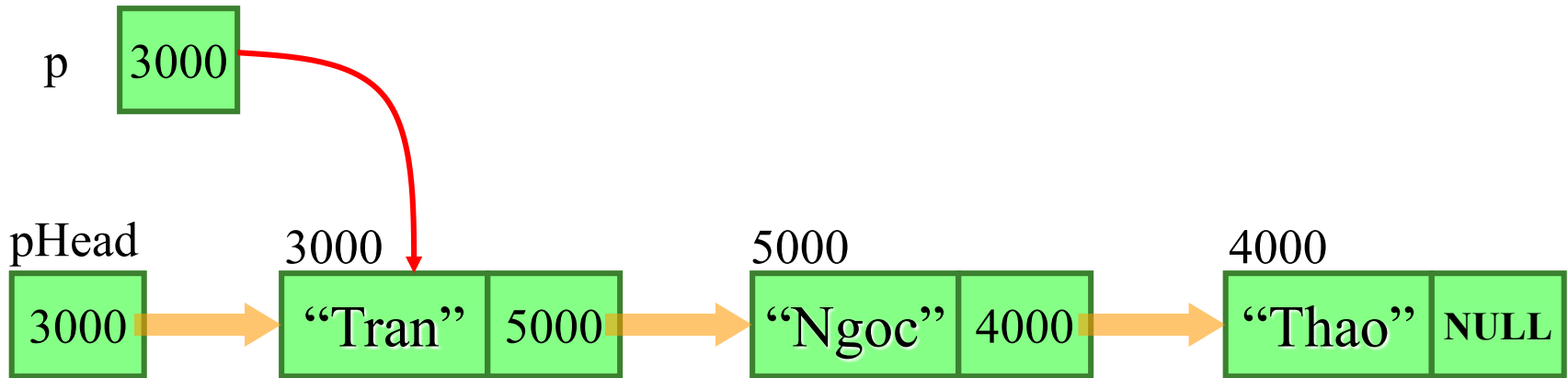
```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

SLL – Minh họa in danh sách



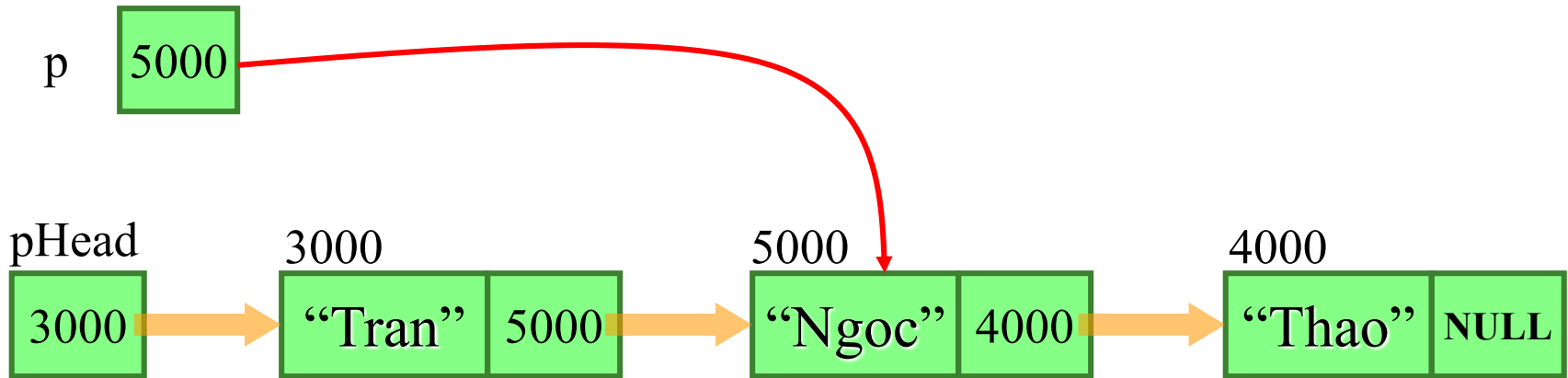
```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

SLL – Minh họa in danh sách



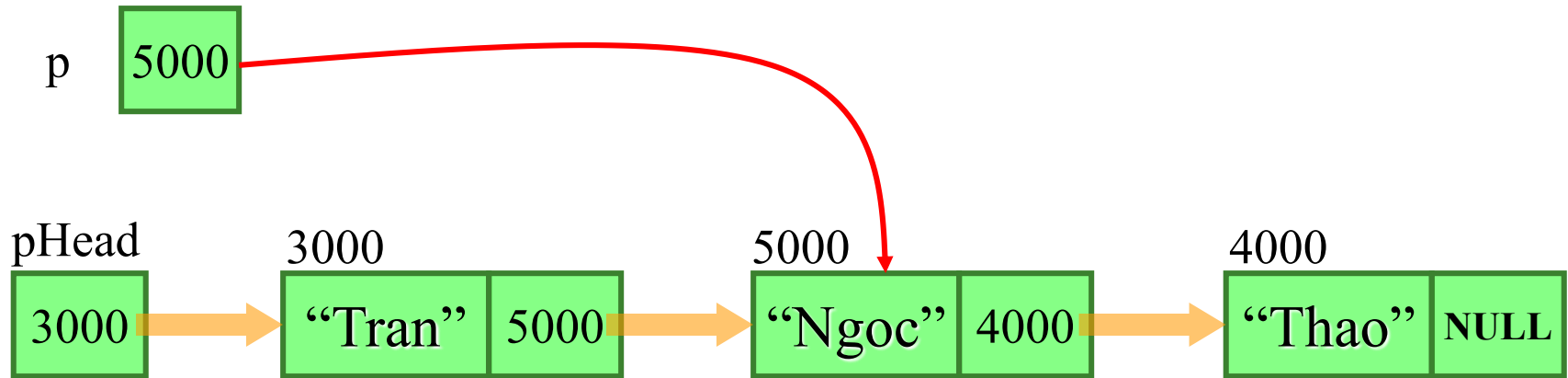
```
p = pHead;
while (p!=NULL)
{
    ShowNode(p);
    p = p->next;
}
```

SLL – Minh họa in danh sách



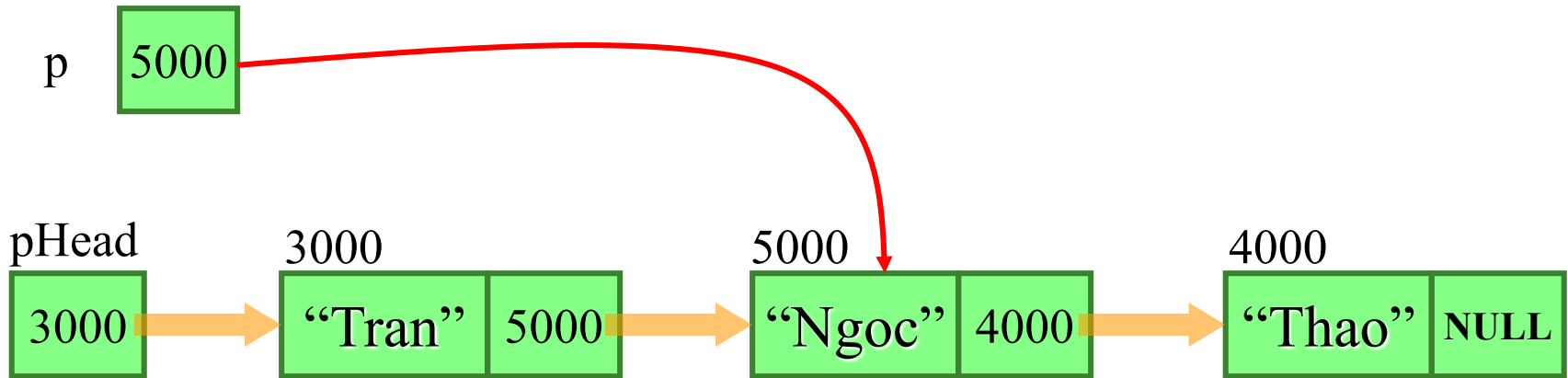
```
p = pHead;
while (p!=NULL)
{
    ShowNode(p);
    p = p->next;
}
```

SLL – Minh họa in danh sách



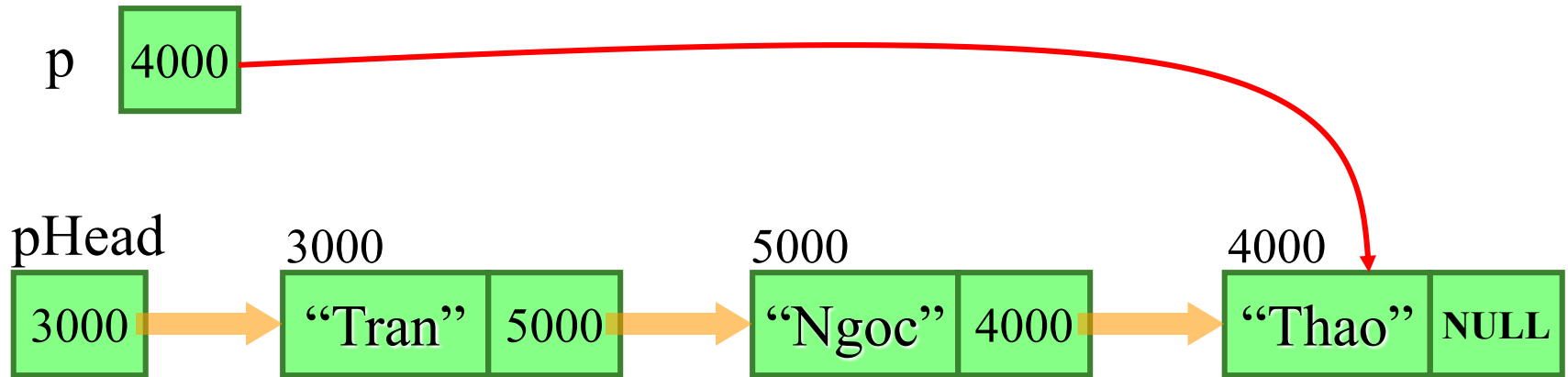
```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

SLL – Minh họa in danh sách



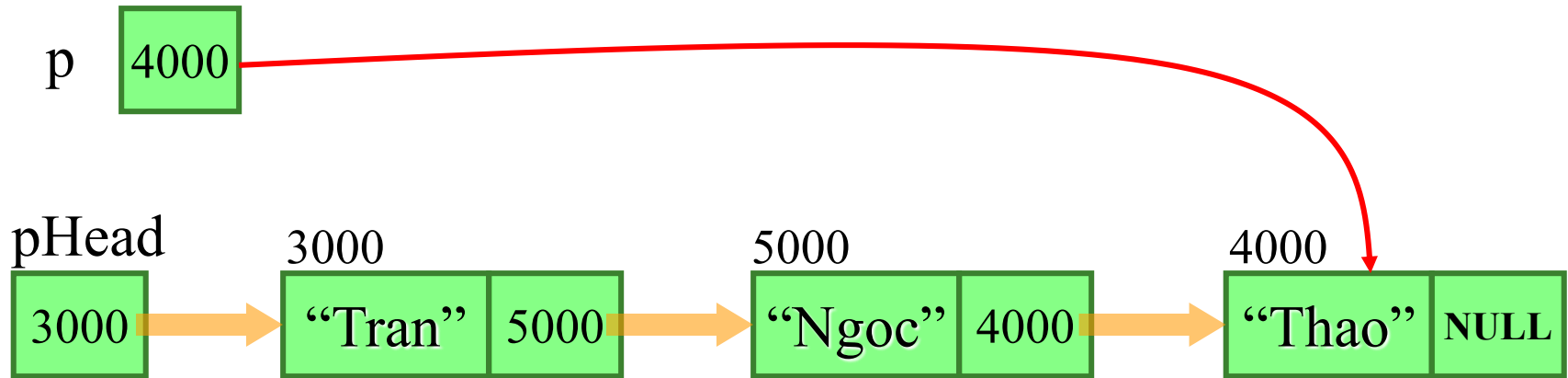
```
p = pHead;
while (p!=NULL)
{
    ShowNode(p);
    p = p->next;
}
```

SLL – Minh họa in danh sách



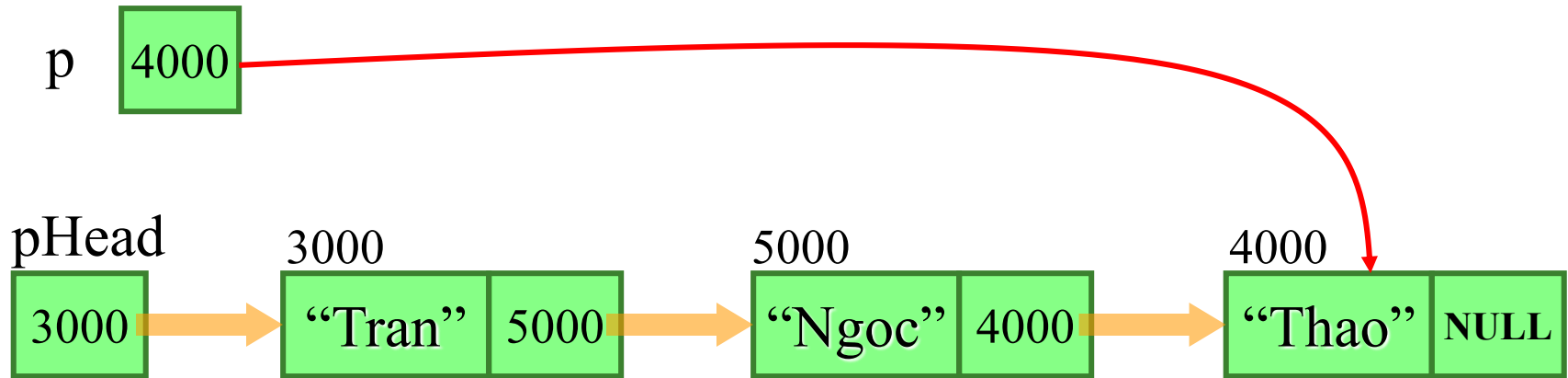
```
p = pHead;
while (p!=NULL)
{
    ShowNode(p);
    p = p->next;
}
```


SLL – Minh họa in danh sách



```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

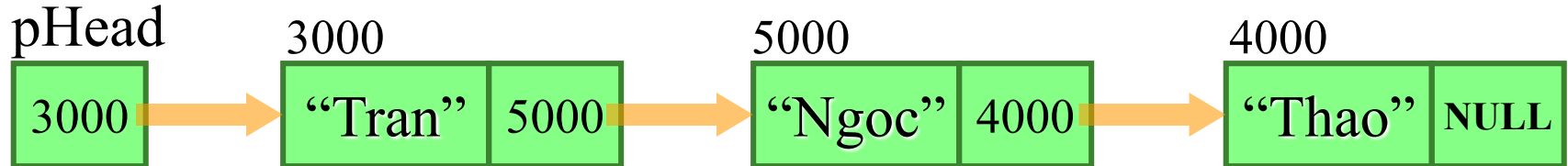
SLL – Minh họa in danh sách



```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

SLL – Minh họa in danh sách

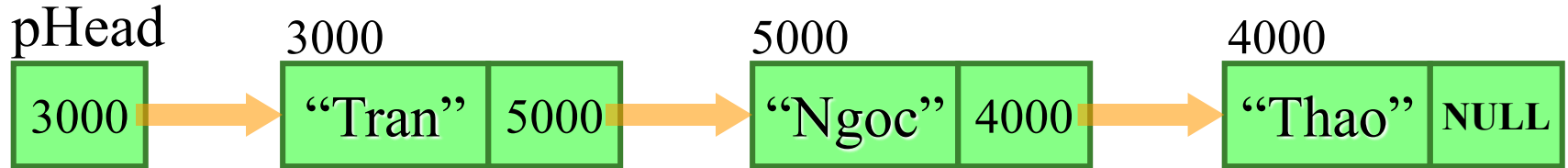
p NULL



```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

SLL – Minh họa in danh sách

p NULL



```
p = pHead;  
while (p!=NULL)  
{  
    ShowNode(p);  
    p = p->next;  
}
```

Kết thúc



SLL– 9. Tìm kiếm

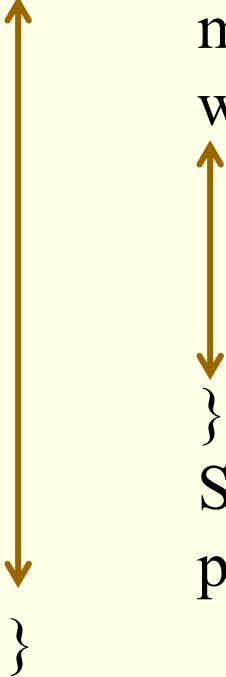
- Search: Tìm kiếm phần tử x trong danh sách

```
Node* Search(Node* pHead, int x)
{
    Node* p;
    p = pHead;
    while ( (p != NULL) && (p->info != x))
        p = p->next;
    return p;
}
```

SLL- 10. Sắp xếp

- Sắp xếp ds theo thứ tự tăng dần, dùng Selection Sort

```
void Sort(Node* &pHead) {  
    Node* q, *min, *p = pHead;  
    while (p!=NULL) {  
        min = p; q = p;  
        while (q!=NULL) {  
            if (q->info < min->info)  
                min = q;  
            q = q->next;  
        }  
        Swap(p->info, min->info);  
        p = p->next;  
    }  
}
```



SLL– Bài tập bổ sung

- Các thao tác bổ sung (SV cài đặt)
 1. Thêm vào cuối danh sách
 2. Sắp xếp danh sách theo phương pháp Interchange Sort
 3. Xoá 1 phần tử có khoá là x
 4. Thêm phần tử x vào ds đã có thứ tự (tăng) sao cho sau khi thêm vẫn có thứ tự (tăng).
 5. Xác định vị trí của node x trong danh sách
 6. Xác định kích thước của danh sách (số phần tử)
 7. Chèn một phần tử có khoá x vào vị trí pos trong ds
 8. Xoá các phần tử trùng nhau trong danh sách, chỉ giữ lại duy nhất một phần tử (*)
 9. Trộn hai danh sách có thứ tự tăng thành một danh sách cũng có thứ tự tăng. (*)

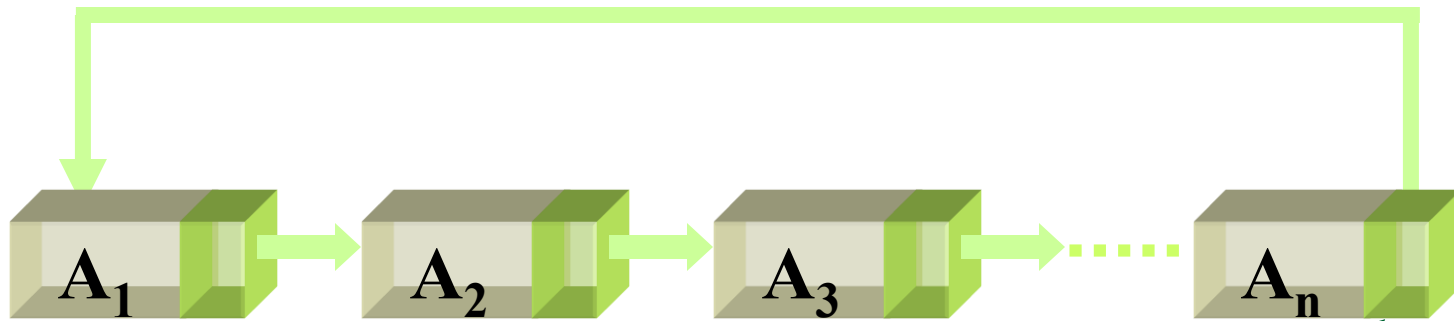
SLL – Bài tập

- Hãy định nghĩa DSLK đơn để quản lý thông tin của các SV, thông tin của SV gồm: MSSV, họ tên, điểm TB
 - Cài đặt các thao tác trên DS sinh viên

Circular Linked List

Circular Linked List

- Tương tự như danh sách liên kết đơn.
- Trường next của nút cuối chỉ đến đầu danh sách

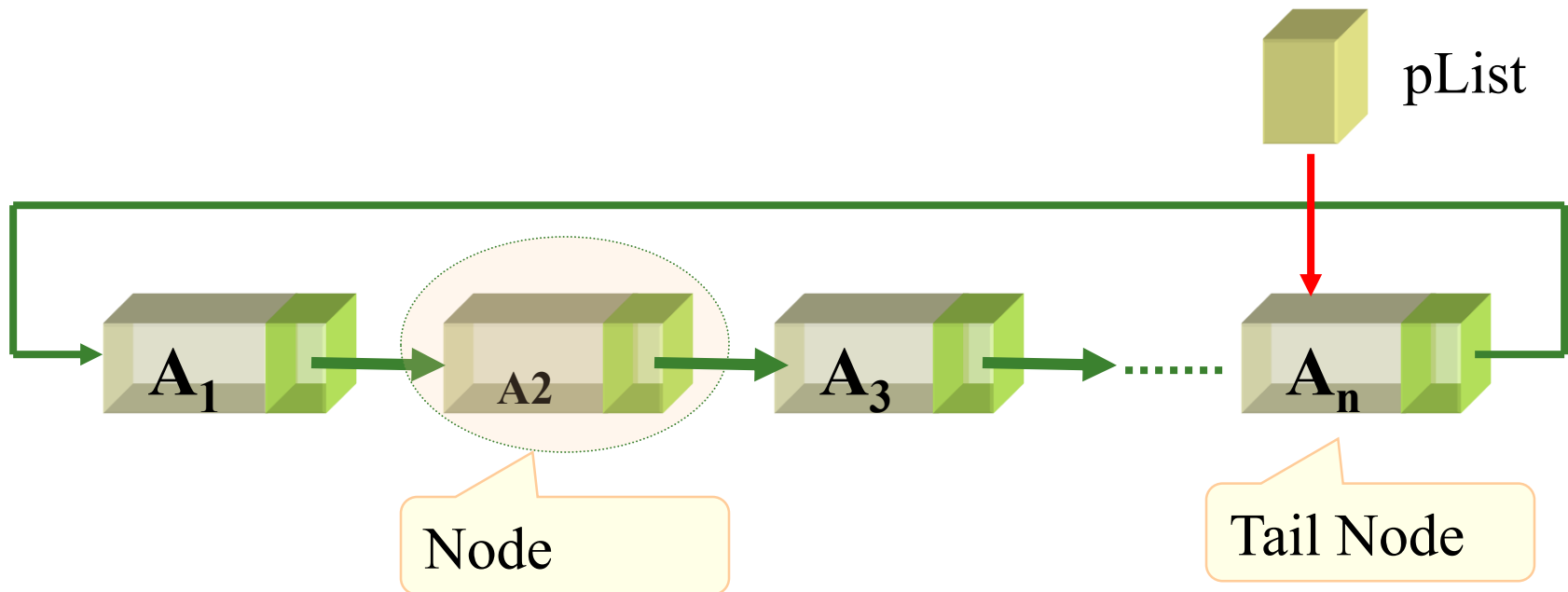


Trường Next của nút cuối ko còn trỏ đến NULL, mà trỏ đến nút đầu

Circular Linked List

■ Mô tả CLL

- Sử dụng pList trỏ đến phần tử cuối của danh sách



Circular Linked List

■ Khai báo & khởi tạo

```
typedef struct node
{
    DataType    info;
    struct node * next;
}Node;
```

```
Node* pList;
```

```
pList = NULL;
```



pList trỏ nút cuối ds



Khởi tạo dslk

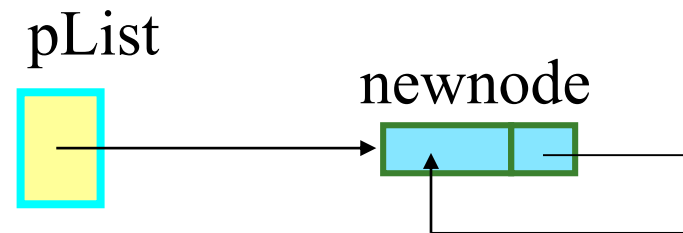
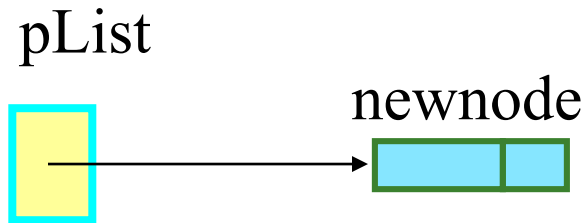
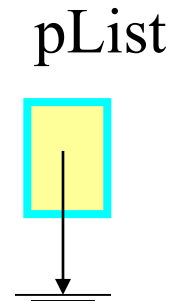
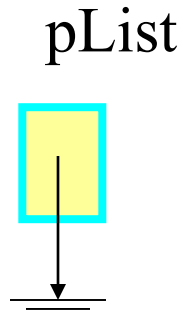
Circular Linked List

- Các thao tác

1. InsertFirst
2. InsertLast
3. DeleteFirst
4. DeleteLast
5. ShowList
6. Search
7. AddList

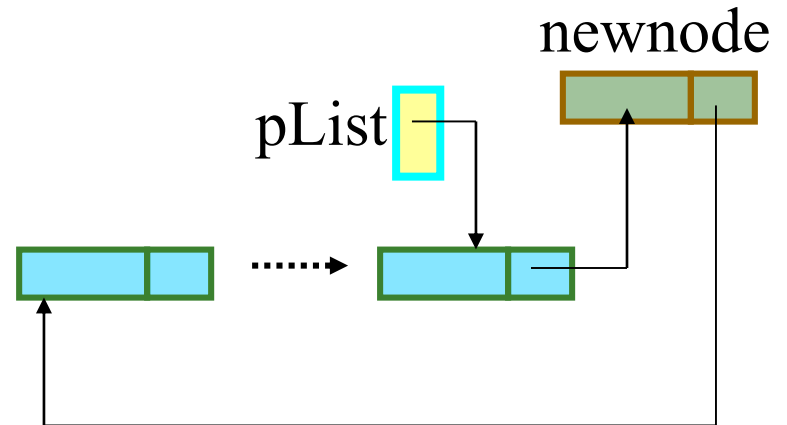
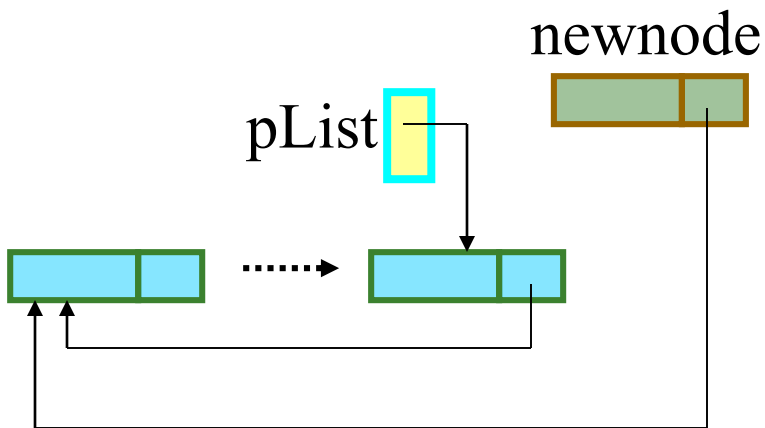
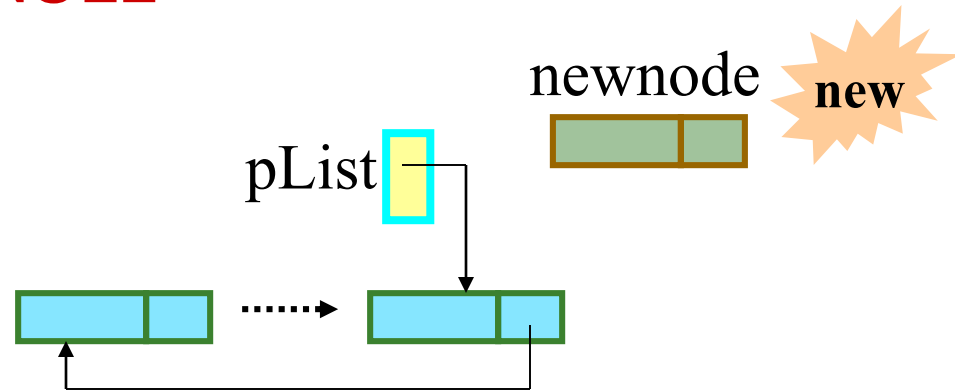
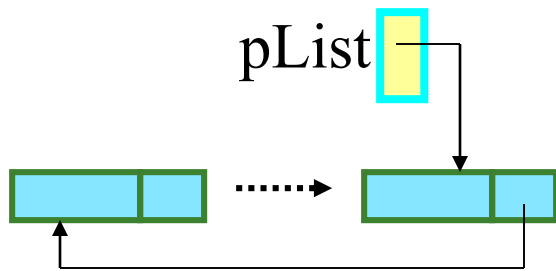
CLL- 1. InsertFirst

- Chèn vào đầu – $pList = \text{NULL}$



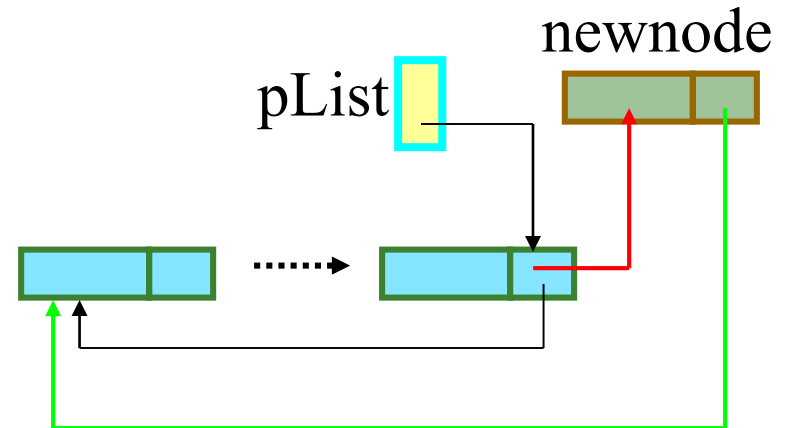
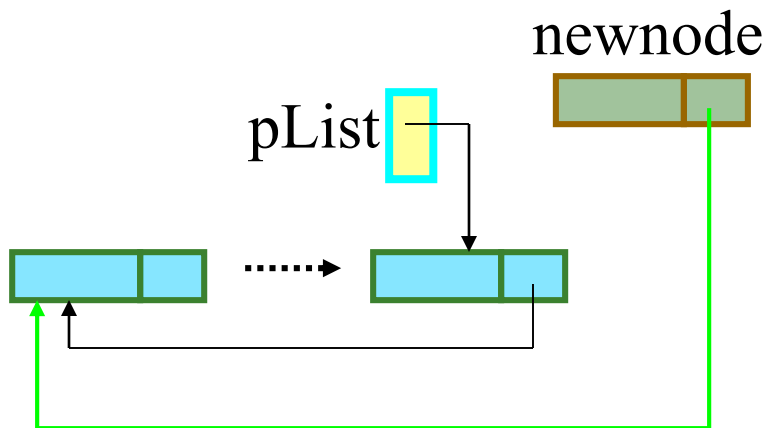
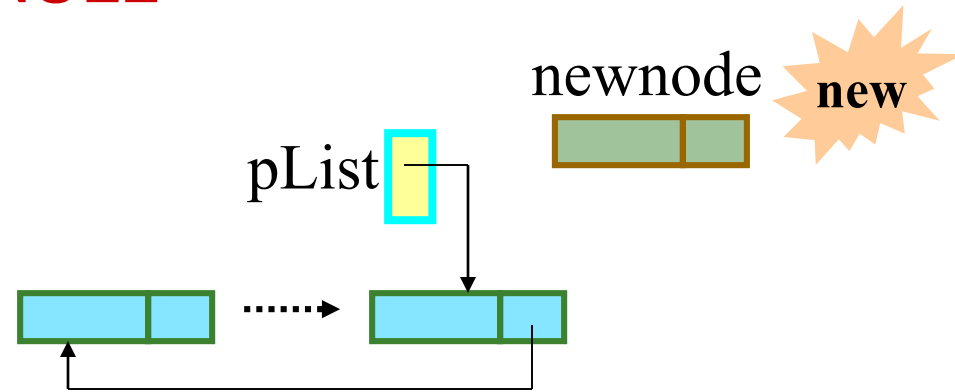
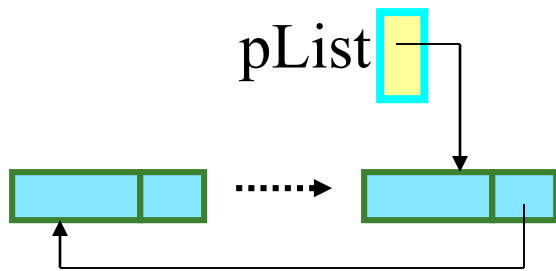
CLL- 1. InsertFirst

- Chèn vào đầu – $pList \neq NULL$



CLL- 1. InsertFirst

- Chèn vào đầu – $pList \neq NULL$

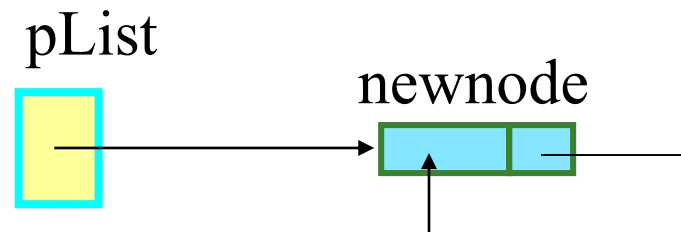
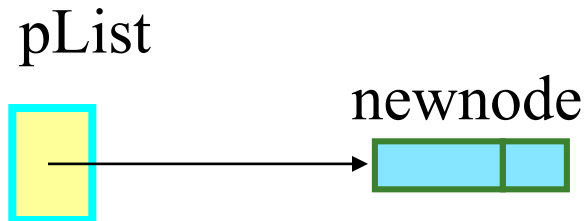
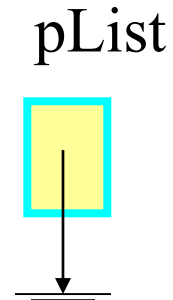
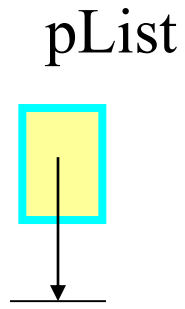


CLL- 1. InsertFirst

```
void InsertFirst(Node* &pList, int x) {  
    Node* newNode;  
    //tạo nút chứa dữ liệu x  
    newNode = CreateNode(x);  
    //nếu danh sách rỗng  
    if (pList == NULL){  
        pList = newNode;  
        pList->next = pList  
    }  
    else {  
        newNode->next = pList->next;  
        pList->next = newNode;  
    }  
}
```

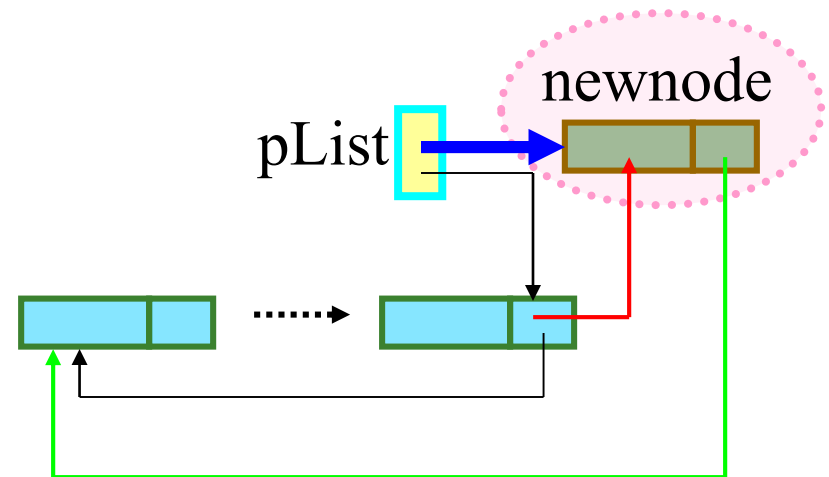
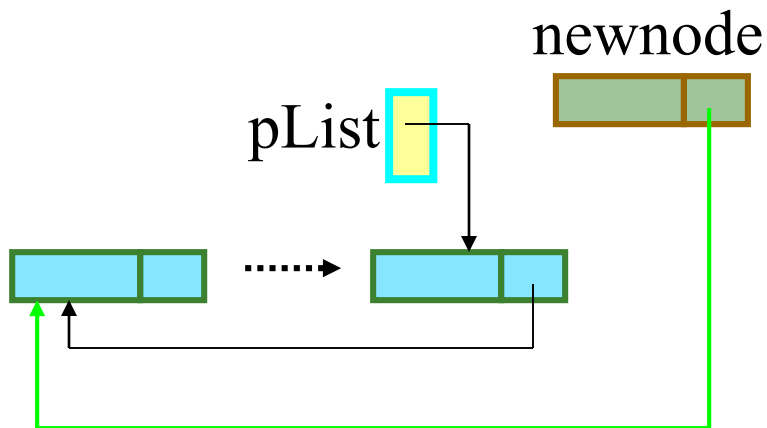
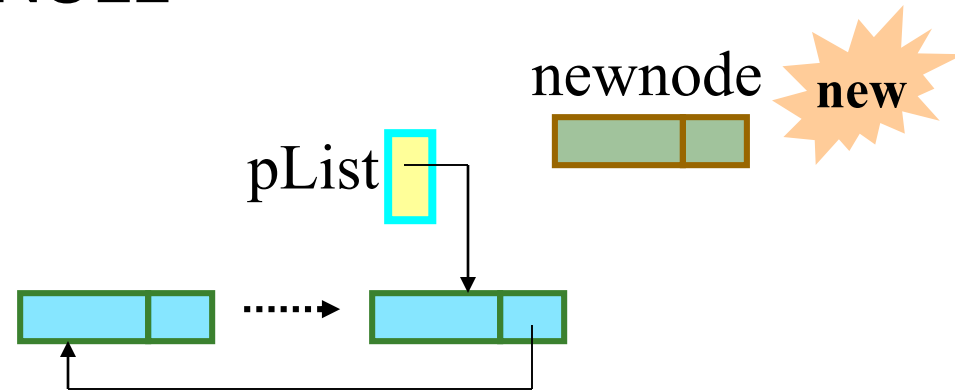
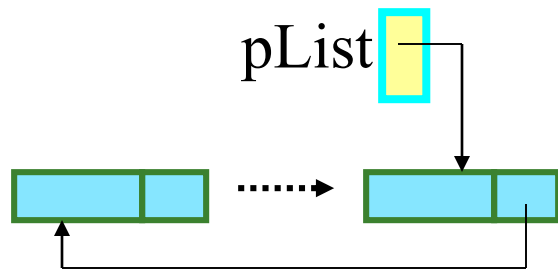
CLL- 2. InsertLast

- Chèn vào cuối – $pList = \text{NULL}$



CLL- 2. InsertLast

- Chèn vào cuối – $pList \neq NULL$

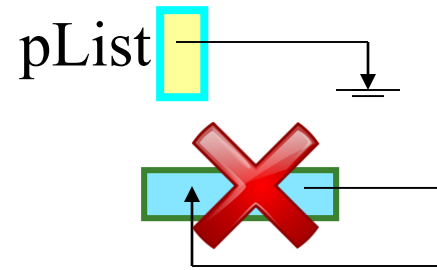
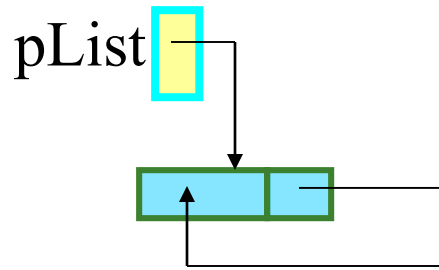


CLL- 2. InsertLast

```
void InsertLast(Node* &pList, int x) {  
    Node* newNode;  
    newNode = CreateNode(x);  
    if (pList == NULL){  
        pList = newNode;  
        pList->next = pList;  
    }  
    else {  
        newNode->next = pList->next;  
        pList->next = newNode;  
        pList = newNode;  
    }  
}
```

CLL- 3. DeleteFirst

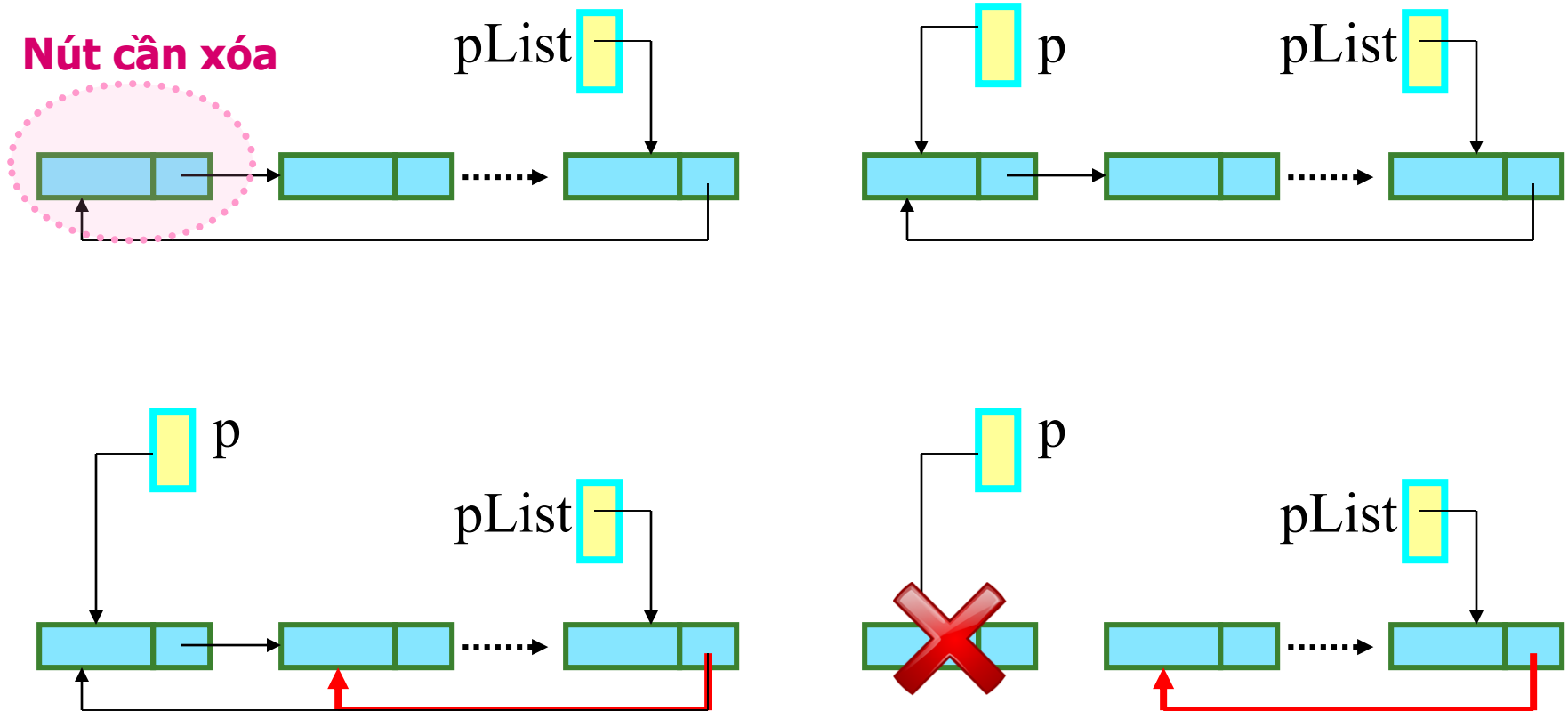
- Xóa nút đầu:
 - Danh sách chỉ có 1 nút ($pList \rightarrow next == pList$)



CLL- 3. DeleteFirst

■ Xóa nút đầu

- Danh sách có nhiều nút ($pList \rightarrow next \neq pList$)

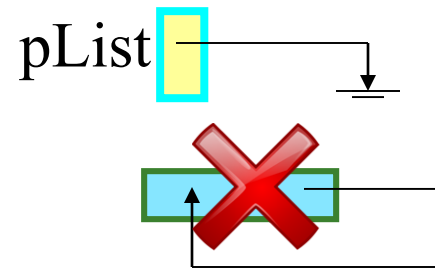
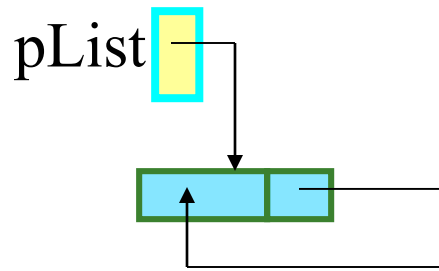


CLL- 3. DeleteFirst

```
void DeleteFirst(Node* &pList) {  
    Node* p;  
    if (pList == NULL)                //ds rỗng  
        return;  
    else if (pList == pList->next {    //ds chỉ có 1 phần tử  
        delete pList;  
        pList = NULL;  
    }  
    else {                             //ds có 2 phần tử trở lên  
        p = pList->next;  
        pList->next = p->next;  
        delete p;  
    }  
}
```

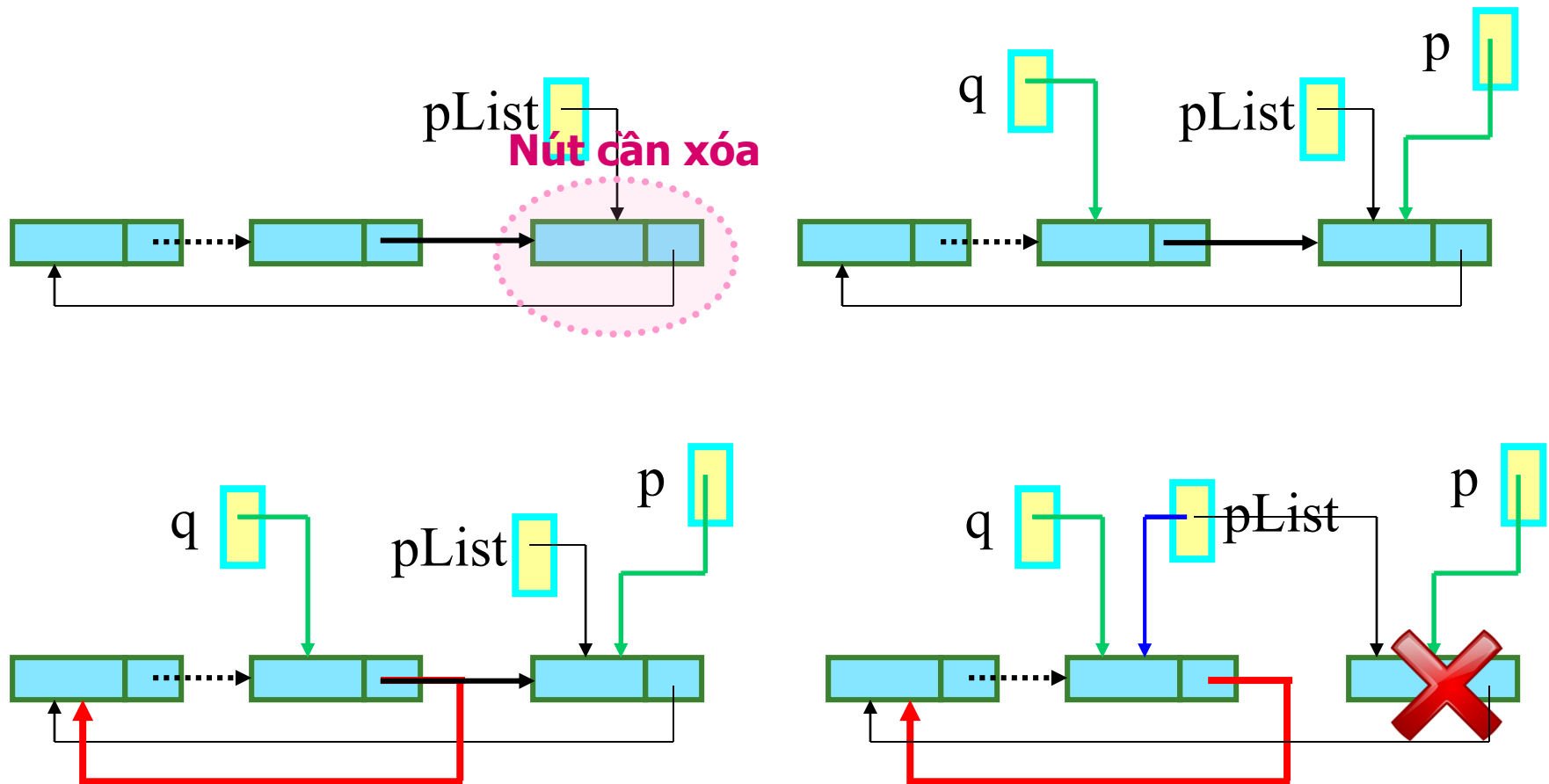
CLL- 4. DeleteLast

- Xóa nút cuối:
 - Danh sách chỉ có 1 nút ($pList \rightarrow next == pList$)



CLL- 4. DeleteLast

- Xóa nút cuối – $pList \rightarrow next \neq pList$



CLL- 4. DeleteLast

```
void DeleteLast(Node* &pList) {
```

```
}
```

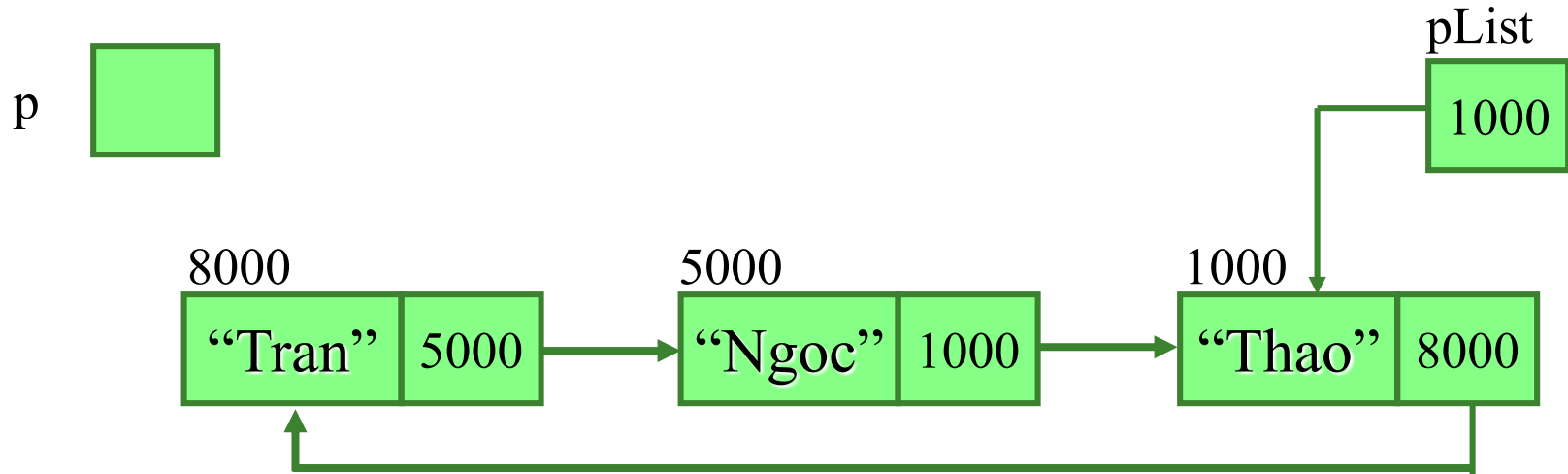
CLL- 5. ShowList

■ ShowList:

- Duyệt từ đầu danh sách
- Đến khi nào quay lại phần tử đầu thì dừng

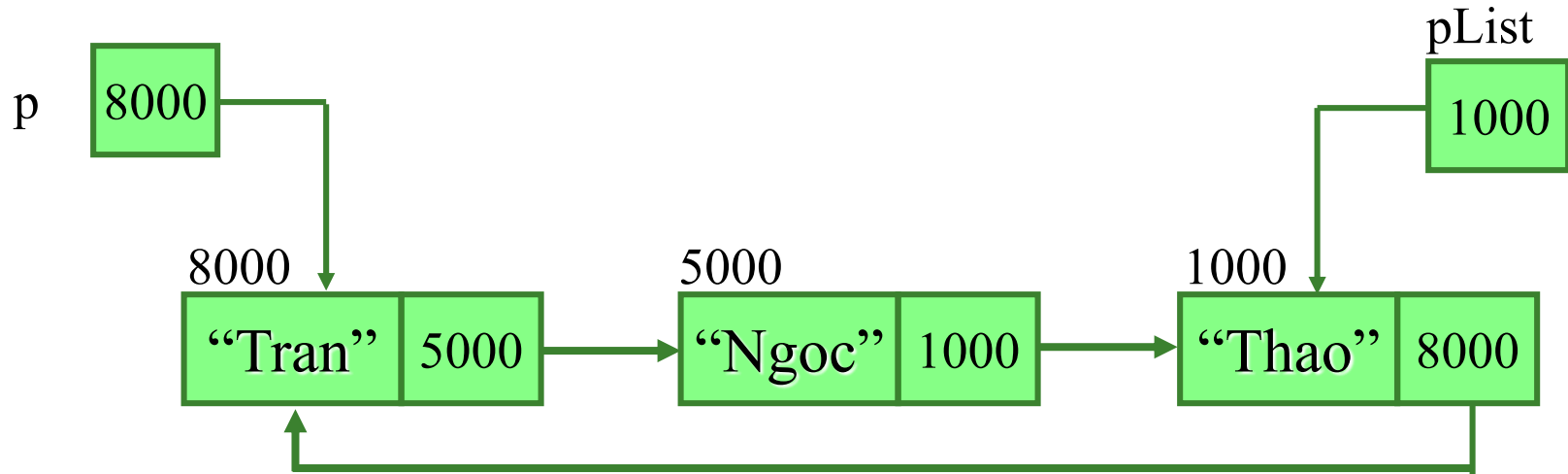
```
void ShowList(Node* pList) {  
    Node* p;  
    if (pList == NULL ) return;  
    p = pList->next;  
    do {  
        ShowNode(p);  
        p = p->next;  
    } while (p!=pList->next);  
}
```

CLL- 5. ShowList



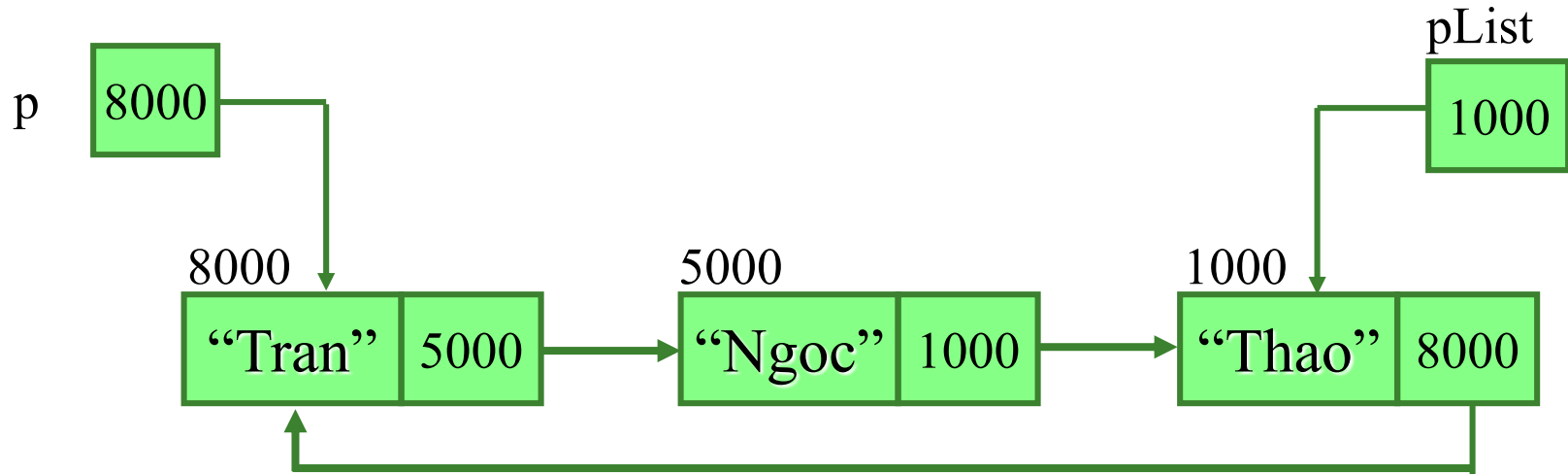
```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

CLL- 5. ShowList



```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

CLL- 5. ShowList



```
p = pList->next;
```

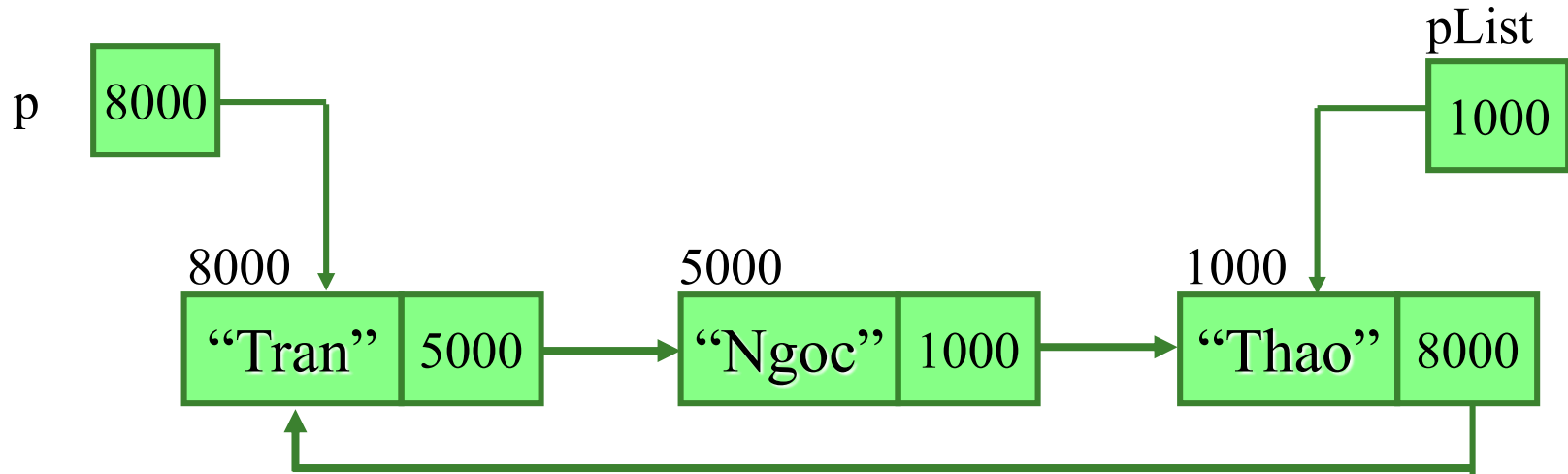
```
do {
```

```
    ShowNode(p);
```

```
    p = p->next;
```

```
} while (p != pList->next);
```

CLL- 5. ShowList



```
p = pList->next;
```

```
do {
```

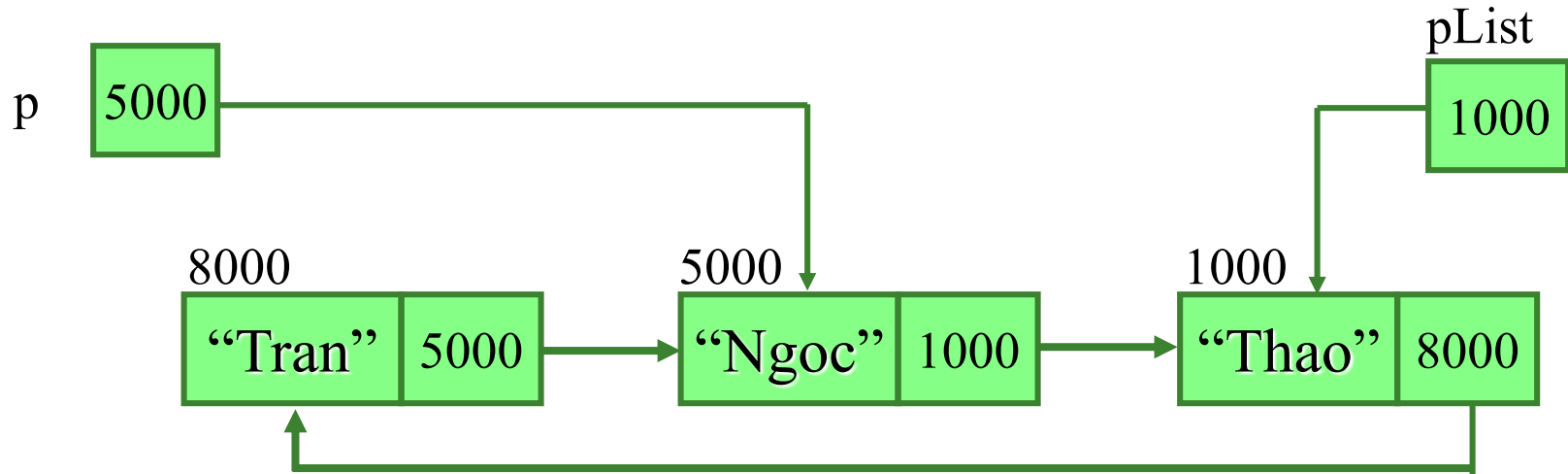
```
    ShowNode(p);
```

```
    p = p->next;
```

```
} while (p != pList->next);
```

"Tran"

CLL- 5. ShowList



```
p = pList->next;
```

```
do {
```

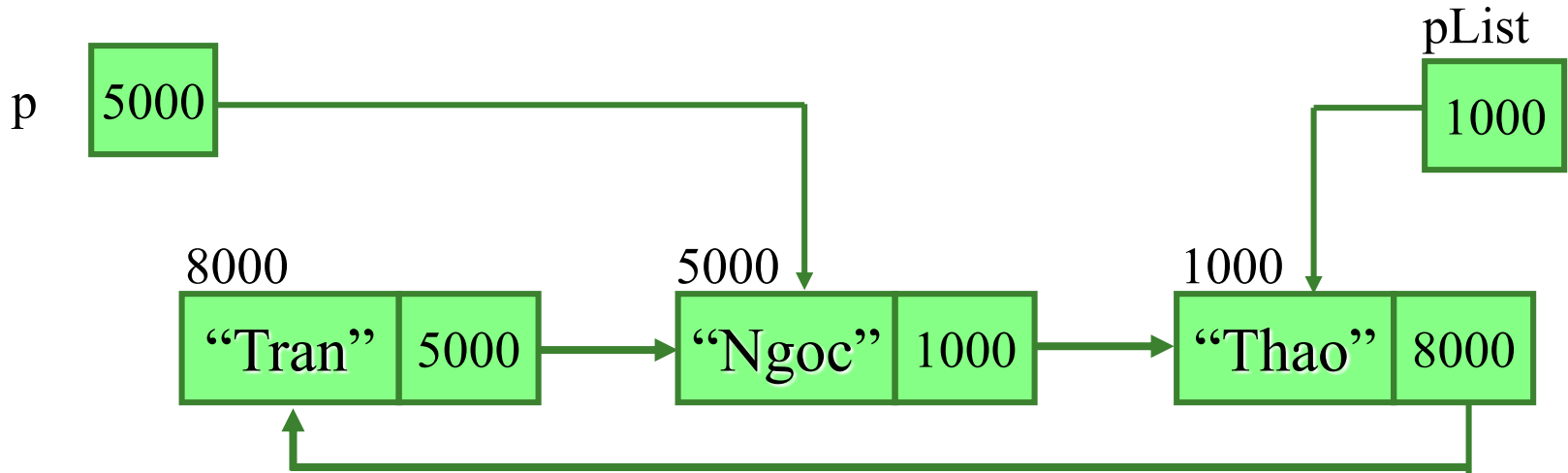
```
    ShowNode(p);
```

```
    p = p->next;
```

```
} while (p != pList->next);
```

"Tran"

CLL- 5. ShowList

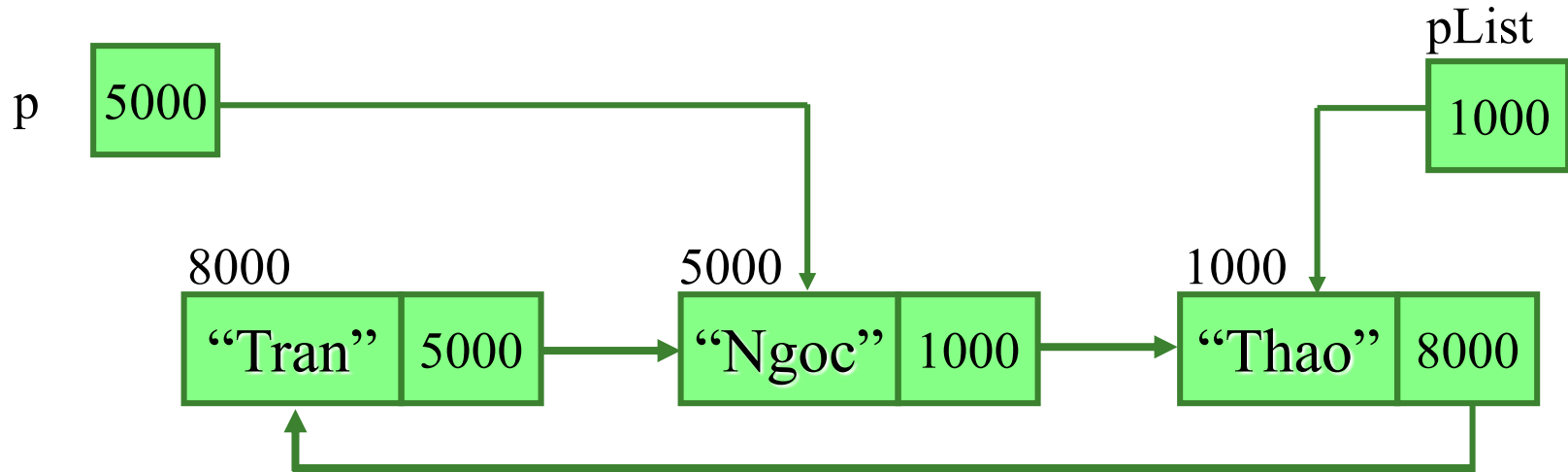


```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

"Tran"

true

Circular Linked List



```
p = pList->next;
```

```
do {
```

```
    ShowNode(p);
```

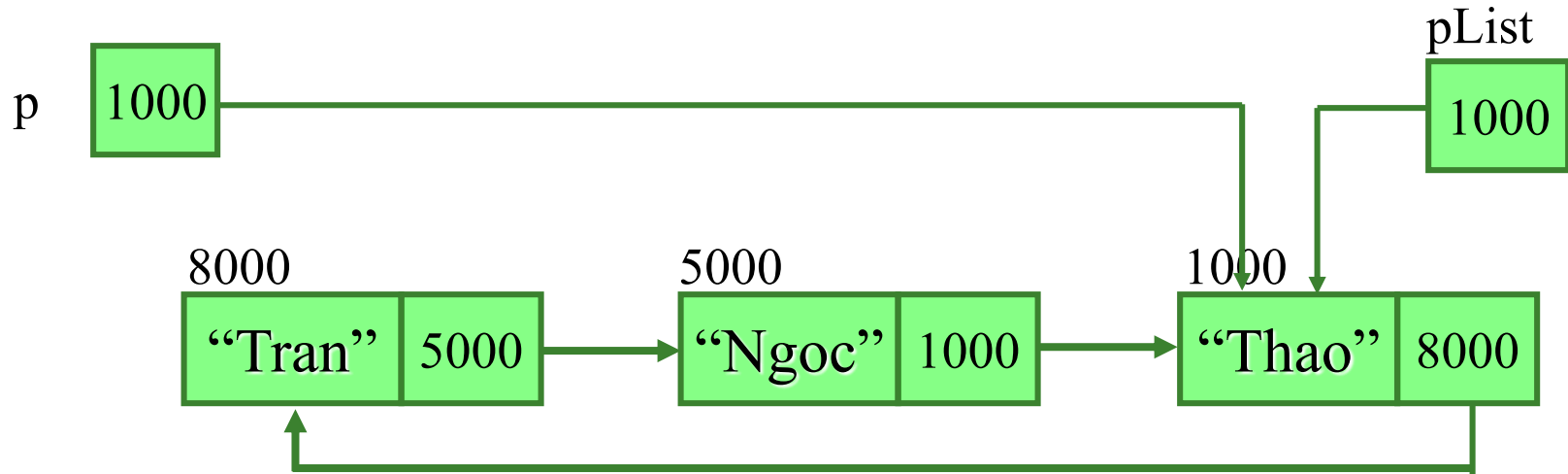
```
    p = p->next;
```

```
} while (p != pList->next);
```

Tran

Ngoc

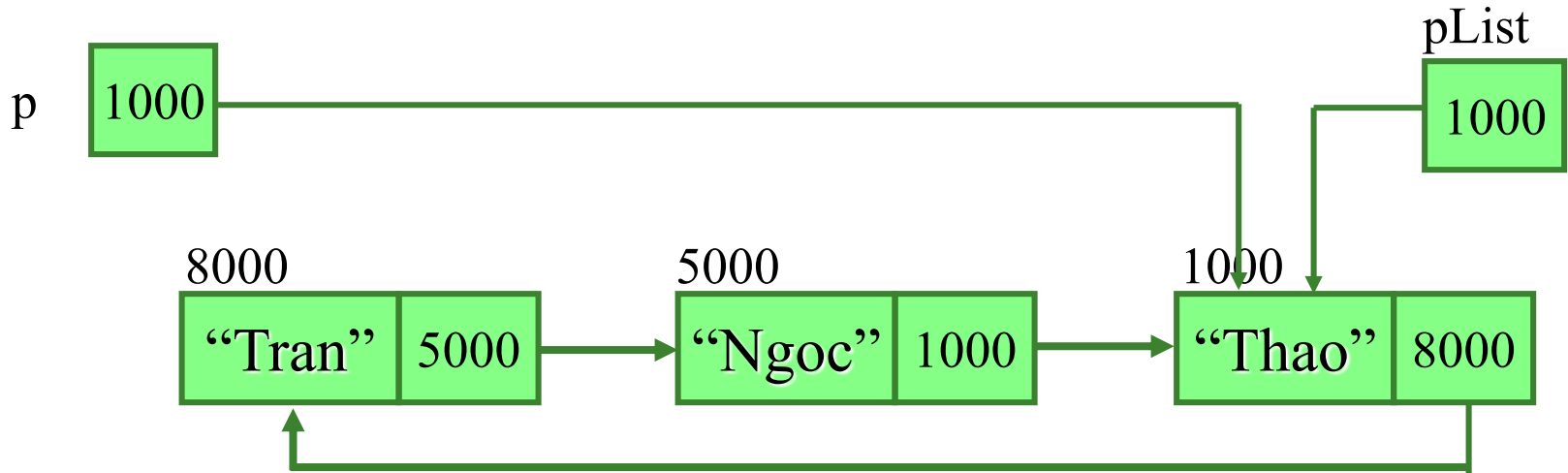
CLL- 5. ShowList



```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

Tran
Ngoc

CLL- 5. ShowList

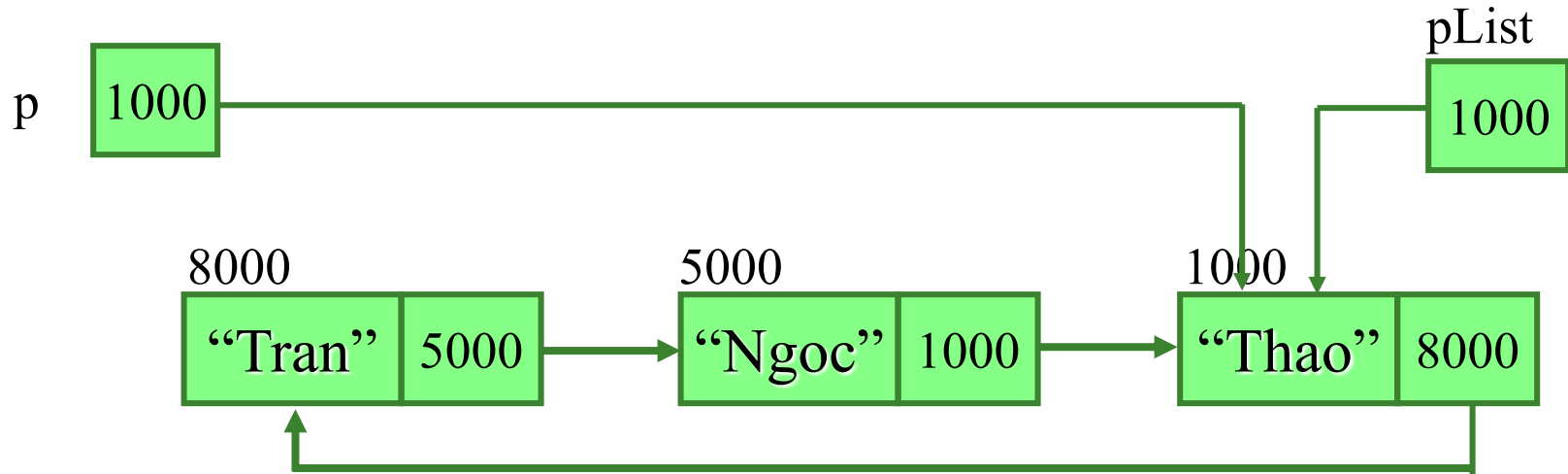


```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

Tran
Ngoc

true

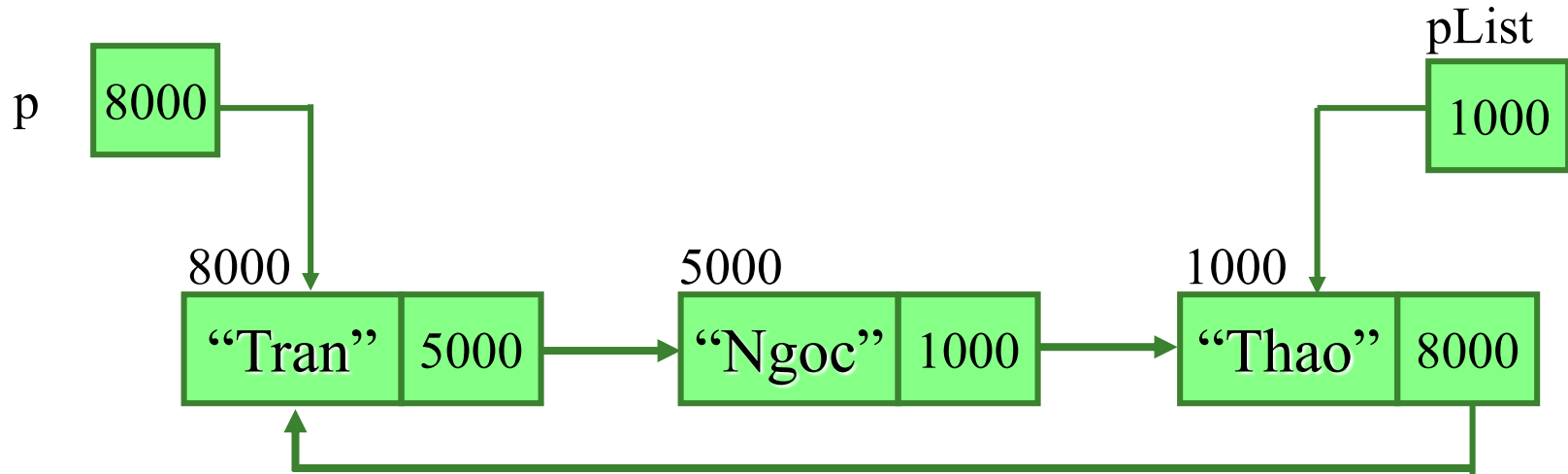
CLL- 5. ShowList



```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

Tran
Ngoc
Thao

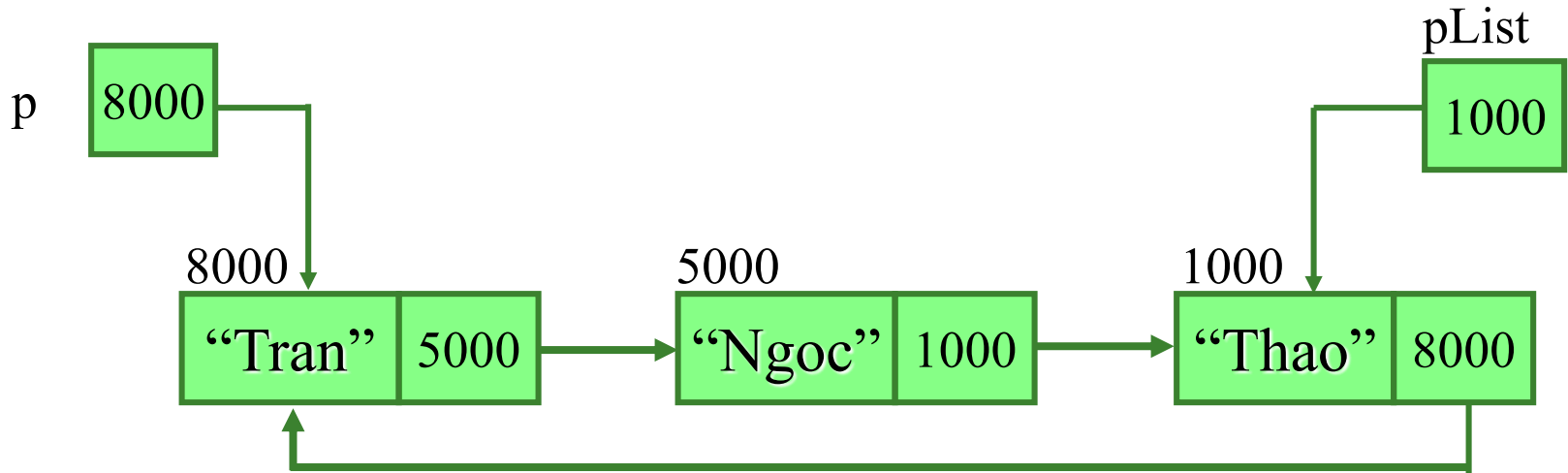
CLL- 5. ShowList



```
p = pList->next;  
do {  
    ShowNode(p);  
    p = p->next;  
} while (p != pList->next);
```

Tran
Ngoc
Thao

CLL- 5. ShowList



```
p = pList->next;
do {
    ShowNode(p);
    p = p->next;
} while (p != pList->next);
```

Tran
Ngoc
Thao

false

CLL- 6. Search

■ Search:

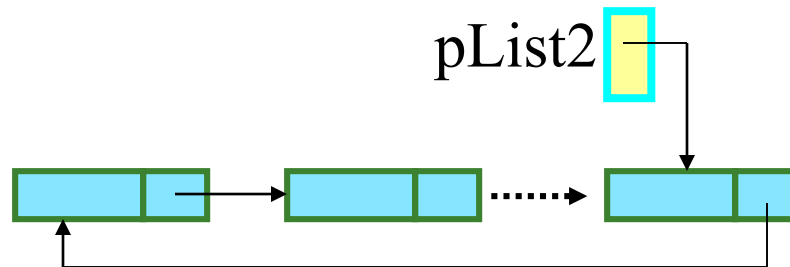
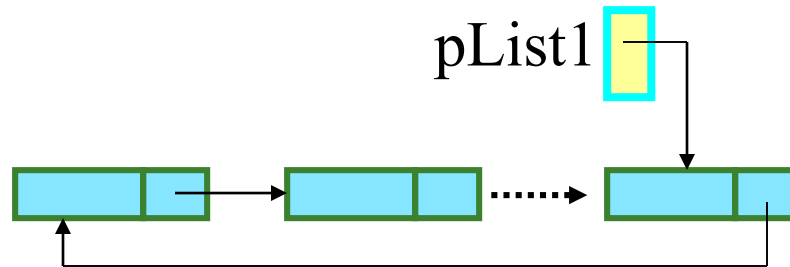
- Xuất phát từ đầu danh sách
- Nếu tìm thấy trả về địa chỉ nút đó
- Ngược lại qua phần tử tiếp theo
- Điều kiện dừng khi quay lại phần tử đầu tiên
- Không tìm thấy trả về NULL

CLL- 6. Search

```
Node* Search(Node* pList, int x) {  
    Node* p;  
    if (pList == NULL) return NULL;  
  
    p = pList->next;           //Lấy nút đầu DS  
    while (p->info != x && p != pList->next)  
        p = p->next;  
    if ( p->info == x)        return p;  
    return NULL;  
}
```

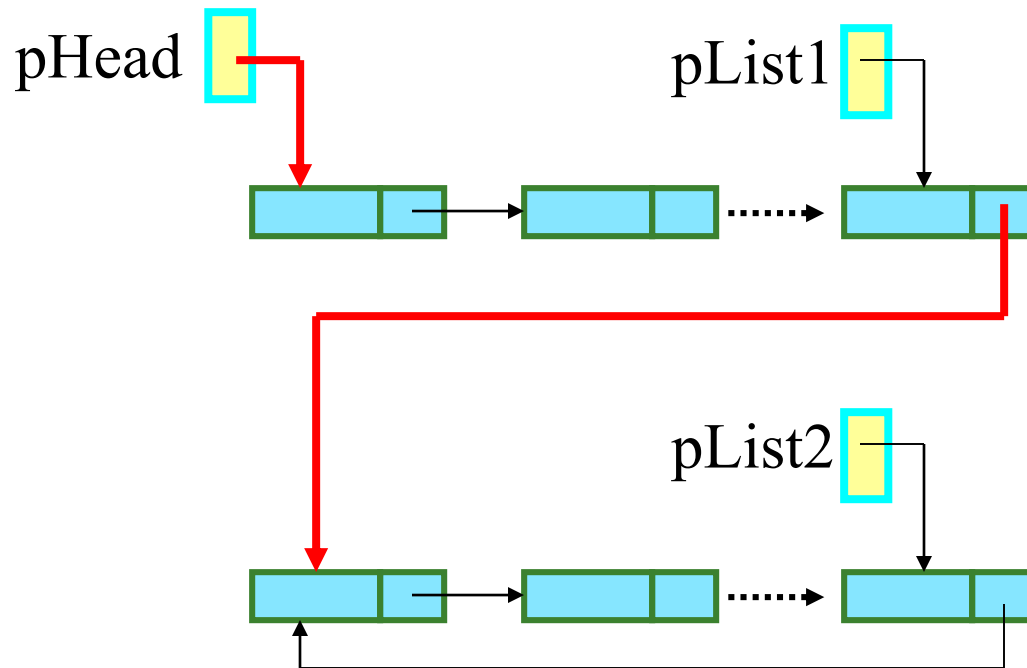
CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



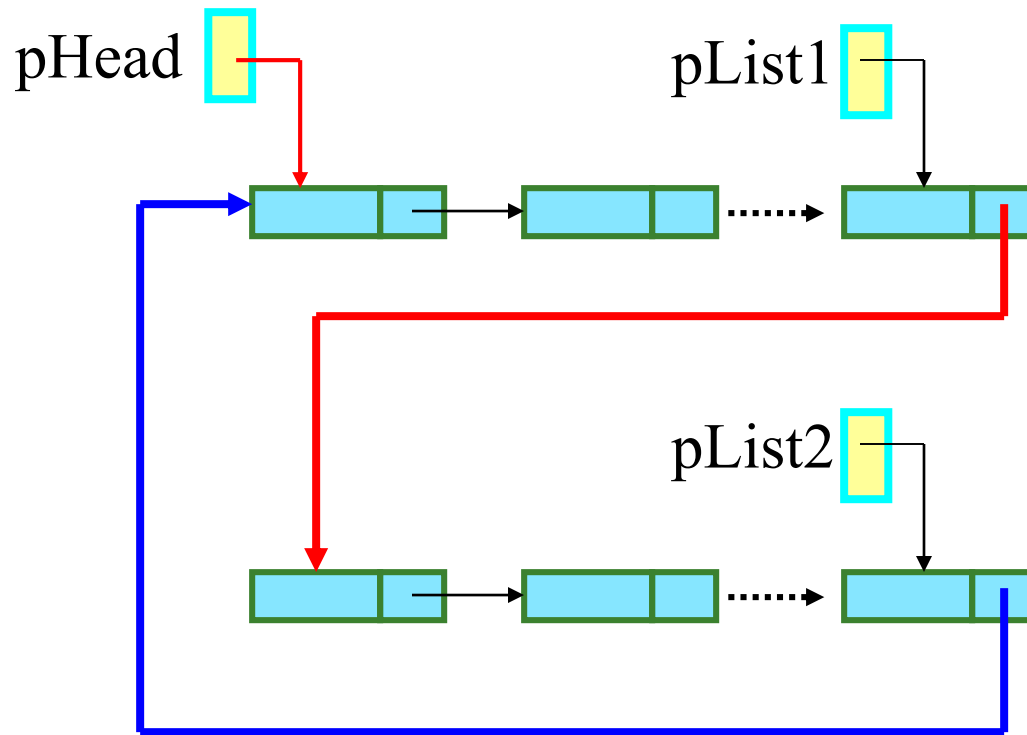
CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



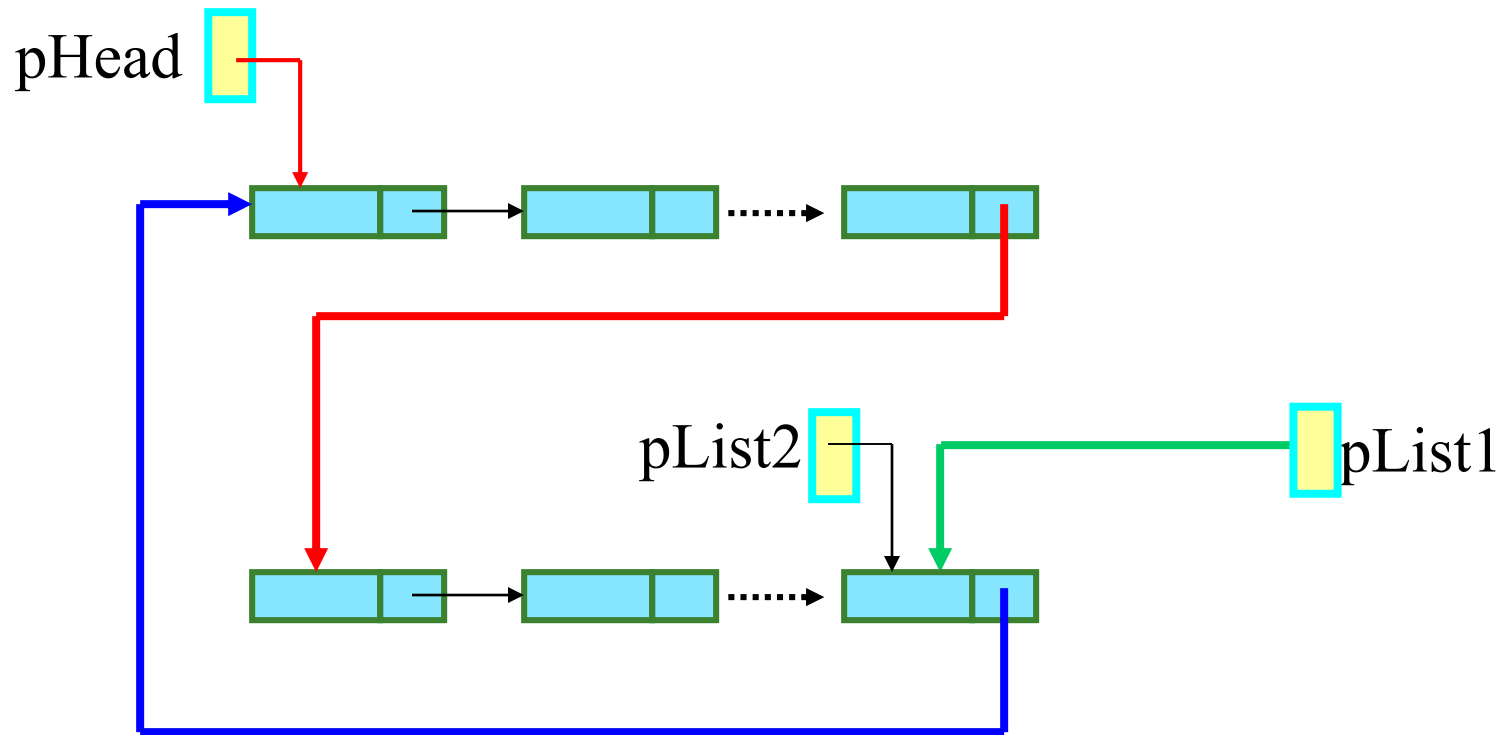
CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



CLL- 7. AddList

- Nối danh sách vòng pList2 vào pList 1



CLL- 7. AddList

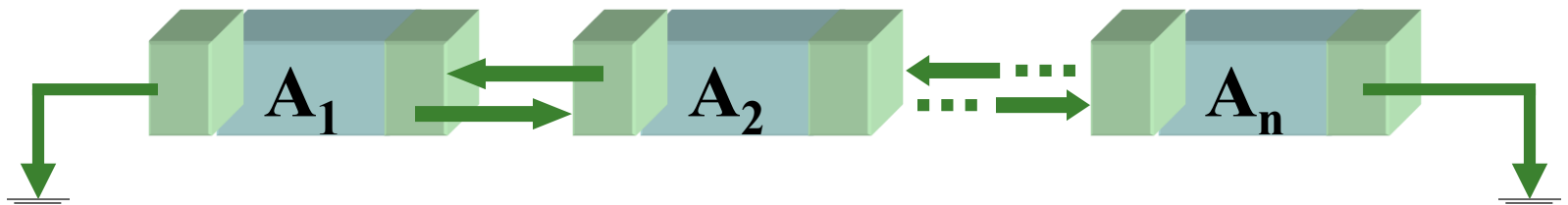
```
void AddList(NodePtr &pList1, NodePtr &pList2)
{

}
}
```

Doubly Linked List

Doubly Linked List

- Cho phép di chuyển 2 chiều đến nút trước và sau.
 - Liên kết nút trước là: prev
 - Liên kết nút sau là: next
- Nút đầu có prev là NULL
- Nút cuối có next là NULL



Doubly Linked List

■ Khai báo

```
typedef struct node
```

```
{
```

```
    DataType    info;
```

```
    struct node * prev;
```

```
    struct node * next;
```

```
}Node;
```

```
Node* pHead;
```

```
pHead = NULL;
```

trở đến nút trước

trở đến nút sau

pHead quản lý ds kép

Khởi tạo dslk

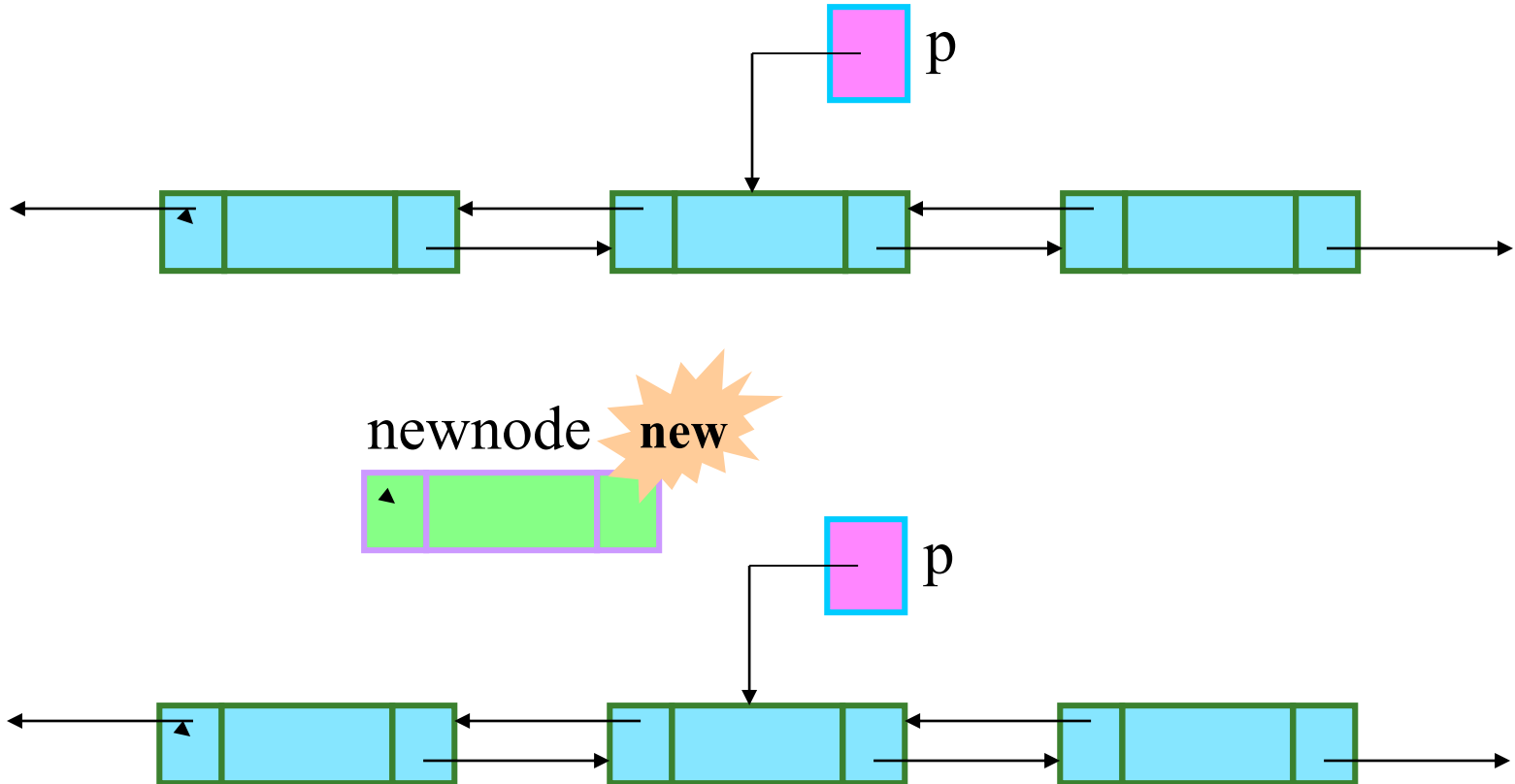
Doubly Linked List

■ Các thao tác cơ bản

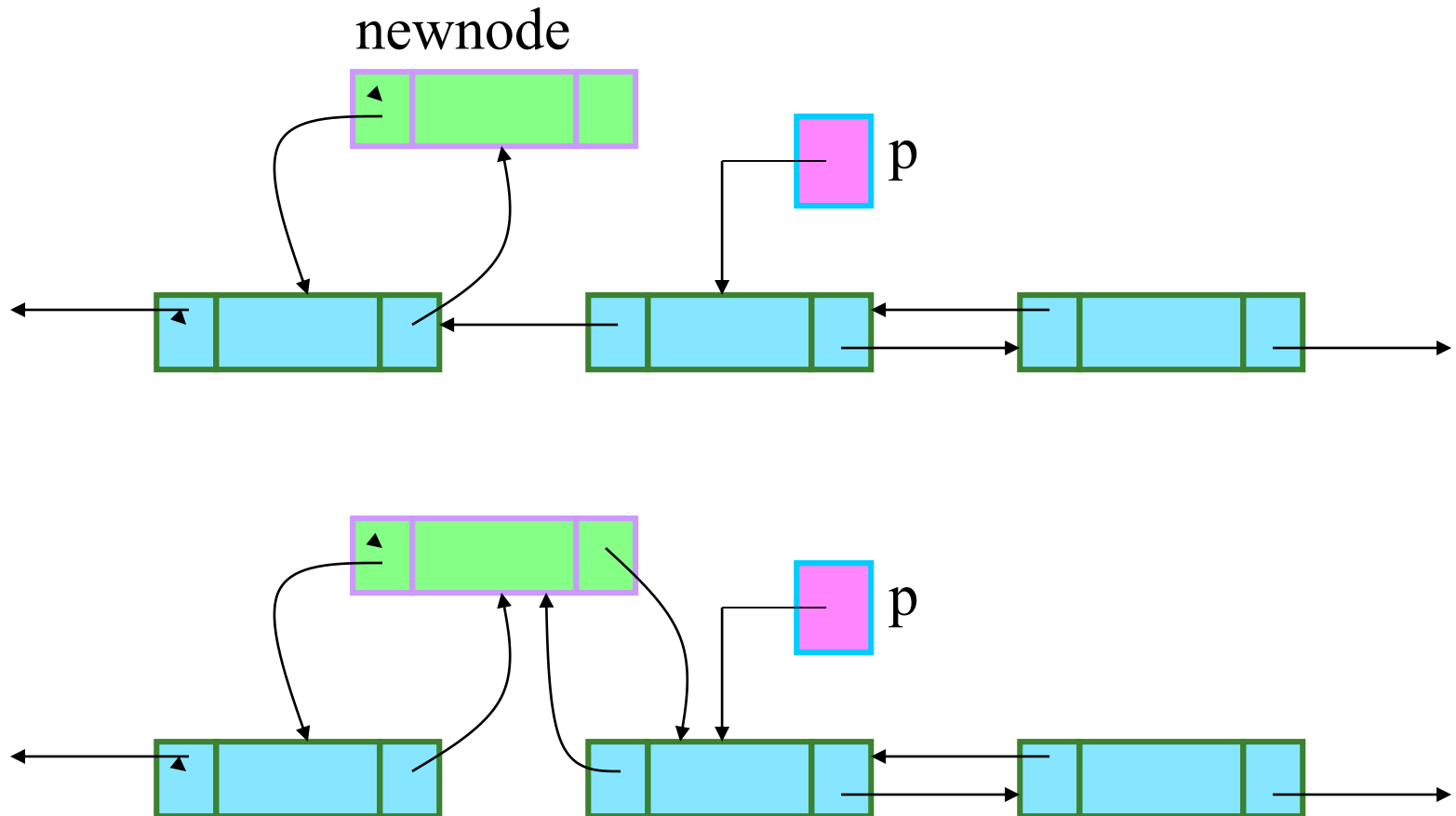
- CreateNode, Init, IsEmpty...
- InsertFirst: chèn vào đầu
- InsertPrev: chèn trước nút p
- InsertNext: chèn sau nút p
- DeleteFirst: xoá nút đầu
- DeleteNode: xoá nút p
- ShowList: duyệt ds
- ShowReverse: duyệt từ cuối danh sách
- ClearList: xoá toàn bộ ds

Doubly Linked List

- InsertPrev: chèn vào trước nút p



Doubly Linked List



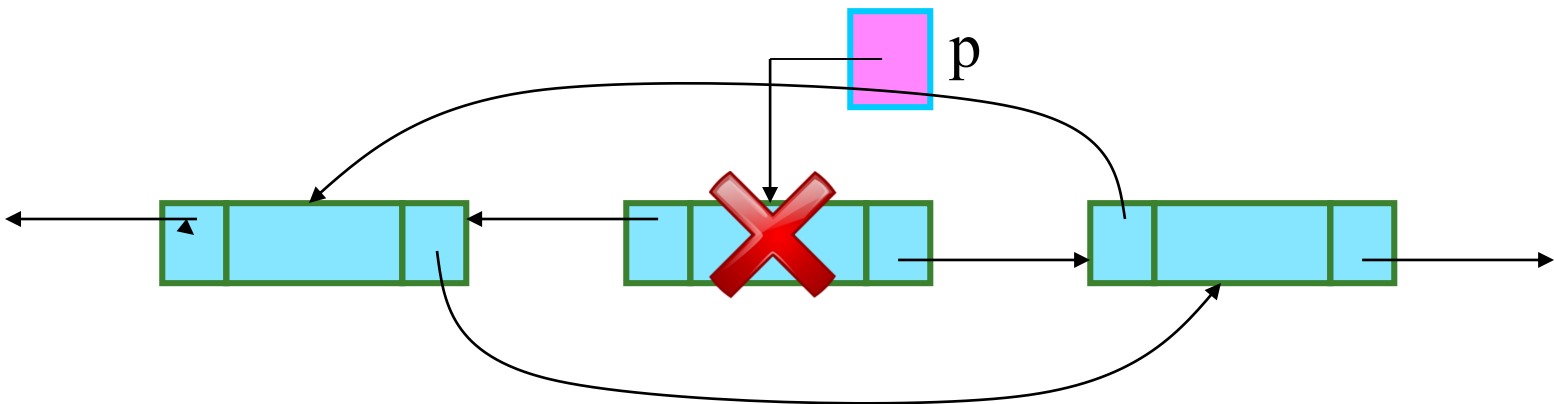
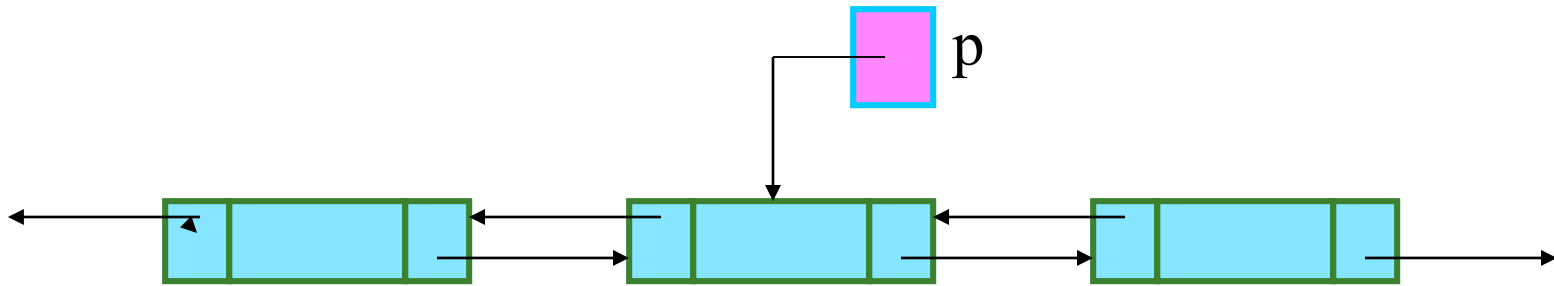
Doubly Linked List

```
void InsertPrev(Node*&pHead, Node* &p, int x){
    Node* newnode, *left;
    if (p == NULL)        return;
    if (p == pHead)       InsertFirst(pHead,x);
    else {
        newnode = CreateNode(x); //tạo nút mới chứa dl x
        left = p->prev;         //lấy nút trước nút p
        newnode->prev = left;   //gắn nút mới vào nút left
        left->next = newnode;

        newnode->next = p;      //gắn nút mới vào nút p
        p->prev = newnode;
    }
}
```

Doubly Linked List

- DeleteNode: xoá nút p



Doubly Linked List

```
void DeleteNode(Node* &pHead, Node* &p) {  
    Node* left, *right;  
    if (p == NULL) return;  
    if (p==pHead) DeleteFirst(pHead);  
    else {  
        left = p ->prev;  
        right = p->next;  
        left->next = right;  
        if (right != NULL)  
            right->prev = left;  
        delete p;  
    }  
}
```

Doubly Linked List

- Các thao tác còn lại SV tự làm!

Bài tập nâng cao

- Xây dựng cấu trúc danh sách liên kết đôi vòng
 - Mỗi nút trên danh sách có hai trường liên kết
 - Prev: trở đến nút trước
 - Next: trở đến nút sau
 - Nút cuối cùng trong danh sách có trường next là nút đầu tiên
 - Nút đầu tiên có trường prev là nút cuối cùng.
 - Các thao tác trên danh sách:
 - Init, IsEmpty, NewNode, FreeNode
 - InsertFrist, InsertLast, InsertPrev, InsertNext, InsertPos
 - DeleteFirst, DeleteLast, DeleteNext, DeletePrev, DeletePos
 - ShowList, ShowInvert
 - Search, Sort.
 - ClearList

