KHOA CÔNG NGHỆ THÔNG TIN
Faculty of Computer Science and Engineering-Thuyloi University

BÀI GIẢNG MÔN LẬP TRÌNH NÂNG CAO

Giảng viên: TS.GVC Bùi Thị Thanh Xuân
Bộ môn: Tin học và KTTT
Năm học: 2020-2021

Chương 4: Kiểu cấu trúc và hợp

4.1. Kiểu cấu trúc

4.2. Cấu trúc tự trỏ và danh sách liên kết

4.2.1. Cấu trúc tự trỏ

4.2.2. Khái niệm danh sách liên kết đơn

4.2.3. Các phép toán trên danh sách liên kết đơn

4.3. Con trỏ tới thành phần

4.4. Kiểu hợp

4.5. Kiểu liệt kê

Cấu trúc tự trỏ

- Một cấu trúc có chứa ít nhất một thành phần con trỏ có kiểu của chính cấu trúc đang định nghĩa được gọi là cấu trúc tự trỏ. Có thể khai báo cấu trúc tự trỏ bởi một trong những cách sau:

Cách 1:

typedef struct <tên cấu trúc> <tên kiểu> ;

struct <tên cấu trúc>{

    các thành phần chứa thông tin … ;

    <tên kiểu> *con trỏ ;

} ;

Cách 2:

struct <tên cấu trúc>{

    các thành phần chứa thông tin … ;

    <tên cấu trúc> *con trỏ ;

} ;

typedef <tên cấu trúc> <tên kiểu> ; // định nghĩa tên cấu trúc tự trỏ

Cấu trúc tự trỏ

Cách 3:

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

Cách 4:

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

Cấu trúc tự trỏ

- Một lớp có thể chứa con trỏ trỏ tới đối tượng của chính lớp đó, nhưng không thể chứa đối tượng của lớp đó.

```
class someclass
{
        someclass * ptr;  // Đúng
};
```

```
class someclass
{
        someclass obj;  //Sai
};
```

Ví dụ: Sử dụng cấu trúc tự trỏ

```
class LinkedList
{
    private:
        int data;
        LinkedList *next, *first;
    public:
        LinkedList();
        void  Insert (int d);
        void Show();
};
```

Ví dụ: Sử dụng cấu trúc tự trỏ

```
struct SinhVien{
        char hoten[30];
        float diem;
        SinhVien *next;
};
```

Hoặc

```
struct SV{
        char hoten[30];
        float diem;
        SV *next;
};
typedef  SV SinhVien;
```

Danh sách liên kết

- Biểu diễn Danh sách trên máy tính

  1. Danh sách cài đặt bằng mảng

  2. Danh sách cài đặt bằng con trỏ

Danh sách cài đặt bằng mảng

- Gọi là cấu trúc dữ liệu danh sách **đặc**, hoặc cấu trúc dữ liệu danh sách kế tiếp, gọi tắt là: Danh sách đặc, hoặc danh sách kế tiếp, **thuộc loại cấu trúc dữ liệu tĩnh.**

- N: Là số phần tử tối đa trong danh sách
- Item: Kiểu dữ liệu của các phần tử trong danh sách
- Dùng một mảng kích cỡ N để lưu các phần tử trong DS, giả sử đặt tên là Elems.
- Count: Là biến đếm, đếm số phần tử hiện có trong danh sách

Dạng biểu diễn

Có thể định nghĩa DS như một cấu trúc gồm 2 trường:

- Elems: Chứa các phần tử trong danh sách
- Count: Đếm số phần tử hiện có trong DS (chiều dài danh sách)

```
#define N 100          //so phan tu toi da la 100
struct  List {          //kieu danh sach List
    item Elems[N];     //mang kieu item
    int count;          //so phan tu toi da cua mang
};
List  L;
```

Ví dụ vận dụng

- Nhập danh sách trúng tuyển gồm n sinh viên với các thông tin: Mã SV, Họ và tên, Ngày sinh, Điểm trúng tuyển
- In ra danh sách trúng tuyển
- Sắp xếp danh sách theo họ tên theo abc
- Tìm sinh viên có điểm trúng tuyển cao nhất

Ví dụ

```
struct Date {
        int day;
        int month;
        int year;
        };
struct SinhVien{
        string MaTS;
        string HoTen;
        Date  NgaySinh;
        float Diem;
    };
struct  List {                          //kieu danh sach List
    SinhVien  Elems[MAX];               //mang kieu SinhVien
    int count;                          //so phan tu toi da cua
    mang
     };
     List  DS;
```

```cpp
#include <iostream>
#include <string>
using namespace std;

#define MAX 1000
struct Date {
    int day;
    int month;
    int year;
  };
struct SinhVien{
        string MaTS;
        string HoTen;
        Date NgaySinh;
        float Diem;
    };
struct List{
    SinhVien Elems[MAX];
    int count;
};
```

```cpp
void nhapSV(SinhVien &SV){
    cin.ignore();
    cout<<"Nhap Ma SV: "; getline(cin,SV.MaTS);
    cout<<"Nhap Ten SV: "; getline(cin,SV.HoTen);
    cout<<"Nhap ngay sinh SV: "; cin>>SV.NgaySinh.day;
    cout<<"Nhap thang sinh SV: "; cin>> SV.NgaySinh.month;
    cout<<"Nhap nam sinh SV: "; cin>>SV.NgaySinh.year;
    cout<<"Nhap Diem: "; cin>>SV.Diem;
}
void NhapDS(List &DS){
    cout<<"Cho so sinh vien n = "; cin>>DS.count;
    int n = DS.count;
    for(int i = 0; i<n;i++)
        nhapSV(DS.Elems[i]);
}
```

```cpp
void xuatSV(SinhVien SV){
    cout<<SV.MaTS<<"\t"<<SV.HoTen <<"\t"<<SV.NgaySinh.day<<"/";
    cout<< SV.NgaySinh.month<<"/"<<SV.NgaySinh.year<<"\t\t"<<SV.Diem<<endl;
}
void XuatDS(List DS){
    cout<<"In thong tin SV:"<<endl;
    int n = DS.count;
    for(int i = 0; i<n;i++)
        xuatSV(DS.Elems[i]);
}
```

```cpp
void swap(SinhVien *sv1, SinhVien *sv2){
    SinhVien sv = *sv1;
    *sv1 = *sv2;
    *sv2 = sv;
}
void SapXepDiem(List &DS){
    int n = DS.count;
    for(int i = 0; i<n-1; i++)
        for(int j = i+1; j<n; j++)
        if(DS.Elems[i].Diem>DS.Elems[j].Diem)
            swap(DS.Elems[i],DS.Elems[j]);
}
void SapXepTen(List &DS){
    int n = DS.count;
    for(int i = 0; i<n-1; i++)
        for(int j = i+1; j<n; j++)
        if(DS.Elems[i].HoTen>DS.Elems[j].HoTen)
            swap(DS.Elems[i],DS.Elems[j]);
}
```

```cpp
int main(){
    List DS;
    NhapDS(DS);
    SapXepDiem(DS);
    XuatDS(DS);
    SapXepTen(DS);
    XuatDS(DS);
    return 0;
}
```

Danh sách cài đặt bằng mảng

- Hạn chế:
- Khai thác bộ nhớ không linh hoạt do phải khai báo trước kích thước
- Không thể làm việc với những danh sách quá lớn vì mảng cần được cấp 1 vùng nhớ liên tục
- Các thao tác chèn, xóa đòi hỏi dồn chỉ số cho các phần tử mảng

Danh sách cài đặt bằng con trỏ

- Danh sách được cài đặt bởi con trỏ ta còn gọi là cấu trúc dữ liệu danh sách liên kết - gọi tắt là danh sách liên kết, đây thuộc loại cấu trúc dữ liệu động.
- Các ô nhớ chứa các phần tử trong danh sách có thể nằm rải rác trong bộ nhớ, và chúng gắn kết với nhau thông qua cơ chế móc nối - lưu địa chỉ của nhau, các ô nhớ này được cấp phát động qua **con trỏ.**

Một số kiến thức về Con trỏ

- Khái niệm: biến trỏ dùng để lưu trữ địa chỉ của đối tượng khác
- Khai báo:

<kiểu dữ liệu> *<tên biến con trỏ>;

Ví dụ:

int *p; //p là biến con trỏ, trỏ tới vùng nhớ kiểu int (4 bytes)

- **Cách truy xuất**

Với con trỏ p bên trên ta có 2 phép truy xuất là:

- p : Lấy địa chỉ mà nó lưu giữ (trỏ tới)
- *p : Lấy giá trị trong vùng nhớ mà nó trỏ tới.

Một số kiến thức về Con trỏ

- Định nghĩa kiểu con trỏ:

  typedef <kiểu dữ liệu> *<tên kiểu con trỏ>;
- Khai báo biến:

  <tên kiểu con trỏ> <tên biến con trỏ>;
- Ví dụ:

  typedef int *item;

  item p;

Ví dụ về con trỏ

```
#include <iostream>
using namespace std;

int main()
{ typedef int *pInt;
   pInt p;
   int a[10] = {1,2,3,4,5,6,7,8,9,0};
   cout<<"Hien thi day so:";
   for(p = a; p< a+10; p++)
     cout<<" "<<*p;
   return 0;
}
```

Cấp phát và thu hồi vùng nhớ

```
<Kiểu> *p = new <Kiểu>;
//su dung vung nho tai p
//sau do giai phong no
delete p;
```

- hoặc

```
<Kiểu> *p = new <Kiểu>[Số phần tử] ;
//su dung vung nho tai p
//sau do giai phong no
delete[] p;
```

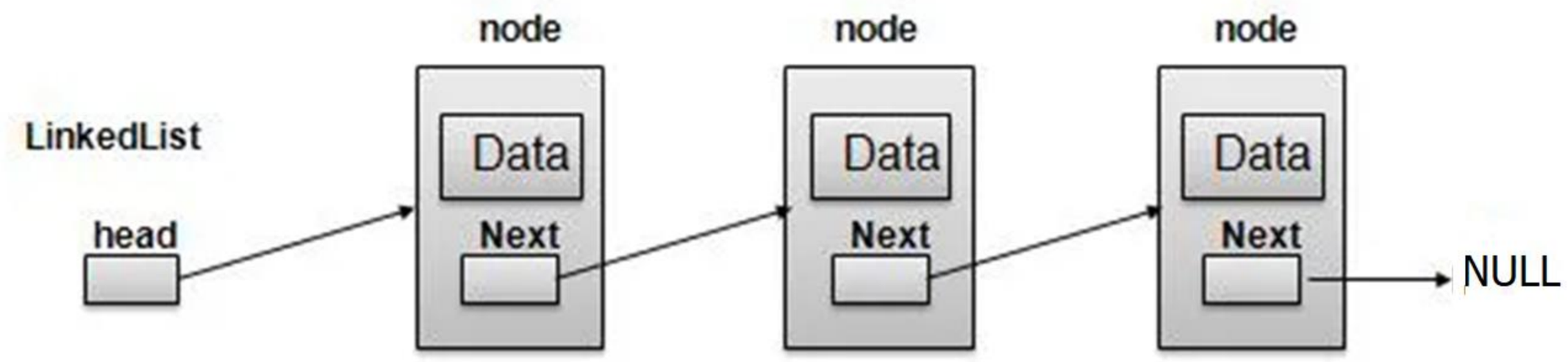
Các hình thức tổ chức liên kết các phần tử trong danh sách:

- Danh sách liên kết là một dãy các phần tử có cùng kiểu dữ liệu, trong đó cần chỉ rõ mối liên kết trước-sau của các phần tử trong danh sách.
  - Liên kết đơn: tương ứng ta có cấu trúc dữ liệu danh sách liên kết đơn – gọi tắt là danh sách liên kết đơn
  - Liên kết vòng: Tương ứng ta có cấu trúc dữ liệu danh sách liên kết vòng – gọi tắt là danh sách liên kết vòng
  - Liên kết đôi: Tương ứng ta có cấu trúc dữ liệu danh sách liên kết đôi – gọi tắt là danh sách liên kết đôi/kép
  - Đa liên kết: Tương ứng ta có danh sách đa liên kết/đa móc nối

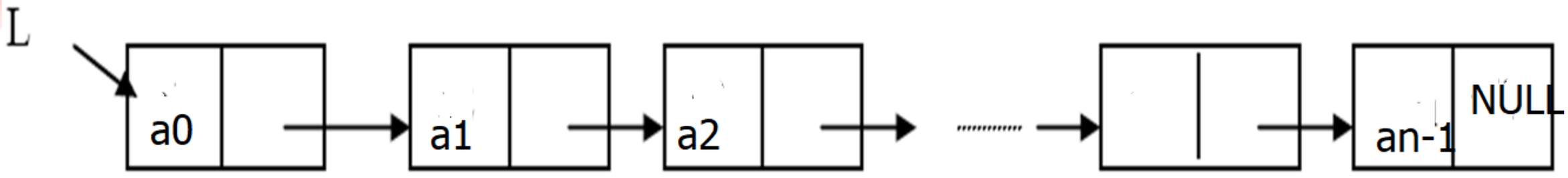
→ Việc cấp phát động cho các phần tử của danh sách cho phép khai thác bộ tốt hơn
→Không phụ thuộc vào kích thước danh sách vì không phải khai báo trước.
→Các phần tử của danh sách động được cấp phát riêng rẽ nên DS không bị hạn chế
→Các thao tác chèn xóa không đòi hỏi phải dồn phần tử

Danh sách liên kết đơn (Single Link List)

- Liên kết đơn: Mỗi phần tử trong danh sách chứa địa chỉ của phần tử đứng ngay sau nó.
  - 1 phần tử trong danh sách là 1 ô nhớ (1 node),
  - 1 ô nhớ là một cấu trúc ít nhất là hai ngăn,
    + 1 ngăn chứa dữ liệu của phần tử đó,
    + 1 ngăn là con trỏ chứa địa chỉ của ô nhớ đứng kế sau nó trong danh sách



Hình ảnh danh sách có dạng như sau:



- Nút cuối cùng trong danh sách không có nút đứng sau, nên Trường ***next*** của phần tử cuối trong danh sách, trỏ đến một giá trị đặc biệt là **NULL (**không trỏ tới đâu**).**
- Để truy nhập vào d/s ta phải truy nhập tuần tự đến vị trí mong muốn, xuất phát từ phần tử đầu tiên,
- Do đó để quản lý danh sách ta chỉ cần quản lý địa chỉ ô nhớ

chứa phần tử đầu tiên của danh sách, tức là cần một con trỏ trỏ đến phần tử đầu tiên này - giả sử con trỏ L.

- L còn gọi là con trỏ quản lý danh sách. Danh sách L rỗng khi: L = NULL

Dạng biểu diễn danh sách:

Trong cài đặt, mỗi phần tử trong danh sách được cài đặt như một **Node** có hai trường:

- Trường ***data*** chứa giá trị của các phần tử trong danh sách;
- Trường **next** là một con trỏ giữ địa chỉ của ô kế tiếp nó trong danh sách.

```
typedef <Kiểu dữ liệu> item; // Kieu cac phan tu dinh nghia la item
struct Node{      // Xay dung mot Node trong danh sach
    item data;                  // Du lieu co kieu item
    Node *next;             //next la con tro, tro den 1 Node tiep theo
};
typedef Node  *List;          //List la mot danh sach cac Node
List L; //L là con trỏ  trỏ đến phần tử đầu tiên của danh sách. Từ giá trị của L ta có thể truy cập đến node đầu tiên, sau đó trường next truy cập đến node thứ hai.
```

Cài đặt các phép toán cơ bản của danh sách liên kết đơn

1- Tạo danh sách rỗng

```
void Init(List &L){
      L=NULL;
   }
```

2- Kiểm tra một danh sách rỗng

```
void IsEmpty(List L){
      if(L == NULL)
         cout<<"DS rong!\n";
      else
         cout<<"DS khong rong!\n";
   }
```

3- Tính độ dài danh sách

- Ta dùng 1 node để duyệt từ đầu đến cuối, vừa duyệt vừa đếm

```
int Length(List L)
{
 int count = 0;
 Node *node = L;
 while (node != NULL)
 {
     count++;
     node = node->next;
 }
 return count;
}
```

4- Tạo 1 node trong danh sách

- Trước tiên ta sẽ phải cấp phát vùng nhớ cho node và sau đó gán data vào

```
Node *CreateNode(item init_data)
{
   Node *node = new Node;
   node->data = init_data;
   node->next = NULL;
   return node;
}
```

Thêm một phần tử vào đầu DS

```
List InsertToHead(List L, item x){
    Node *p = new Node;
    p->data = x;
    p->next = L;
    L = p;
    return L;
}
```

Hoặc sử dụng hàm CreateNode(x)

```
List InsertToHead(List L, item x){
    Node *p=CreateNode(x);
    p->next = L;
    L = p;
    return L;
}
```

Thêm 1 phần tử vào cuối DS

```
List InsertToLast(List L, item x){
    Node *temp, *p;
    temp = CreateNode(x);
    if(L == NULL){
        L = temp;
    }
    else{
        p  = L;
        while(p->next != NULL){
            p = p->next;
        }
        p->next = temp;
    }
    return L;
}
```

5- Nhập dữ liệu cho danh sách: có n phần tử

```
void Input(List &L, int n){
    item x;
    for(int i =0; i<n; i++){
        cout<<"Nhap phan tu: ";cin>>x;
        if(L== NULL)
            L = InsertToHead(L,x);
        else
            L= InsertToLast(L,x);
    }
}
```

5- Nhập dữ liệu cho danh sách: Nhập cho tới khi nhập số 0

```
void Input(List &L){
    int n=0;
    item x;
    do{
        n++;
        cout<<"Nhap phan tu thu "<<n<<": ";cin>>x;
        if(L== NULL)
            L = InsertToHead(L,x);
        else
            if(x!=0)   L = InsertToLast(L, x);
    }while(x!=0);
}
```

6- In dữ liệu trong DS

```
void Output(List L){
    Node *p=L;
    cout<<"Hien thi danh sach:";
    while(p!= NULL)
        {
            cout<<" "<<p->data;
            p=p->next;
        }
}
```

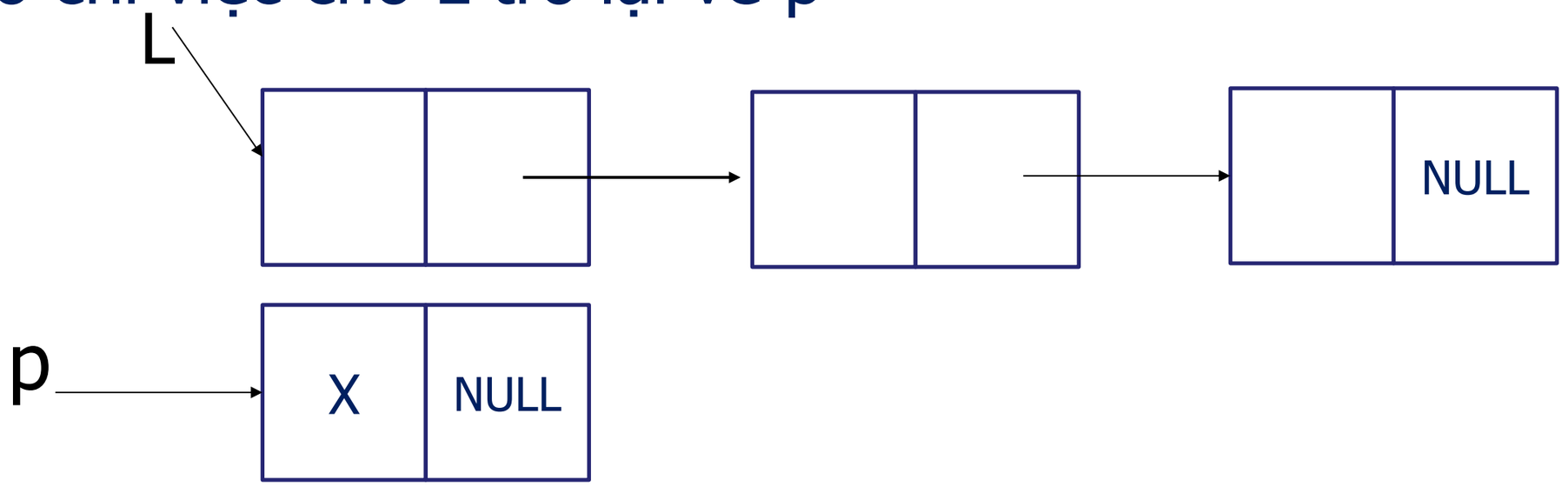
7- Chèn 1 phần tử vào danh sách

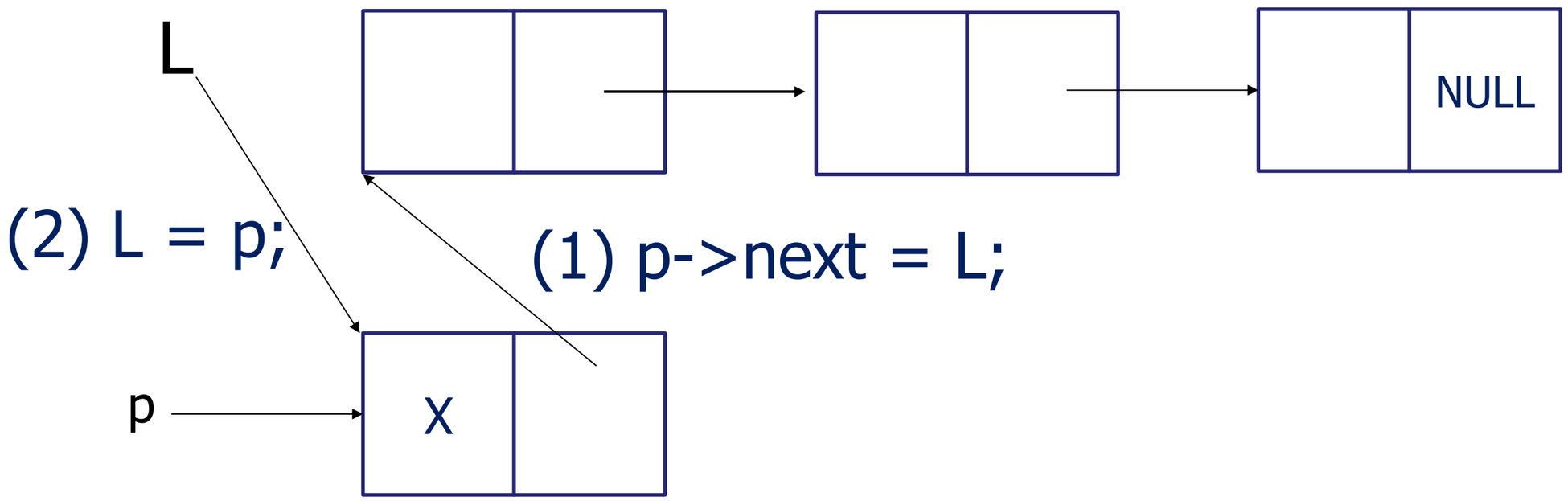
Chèn một phần tử có giá trị x vào danh sách L tại vị trí k ta cần:

- **Cấp phát** một ô nhớ để lưu trữ phần tử mới này: Giả sử con trỏ **p** trỏ tới ô nhớ này
- Đổ dữ liệu cần chèn vào ô nhớ vừa cấp phát: **p->data= x**
- **Nối kết lại các con trỏ** để đưa ô nhớ mới này vào vị trí k. Gồm:
    + Nếu ds rỗng: if (L==NULL) L=p; p>next=NULL
    + Nếu ds khác rỗng: ta ktra vị trí k
        -) Nếu k=0: ta thêm vào đầu ds p->next=L; L=p;
        -) Nếu 0<k<length(L): ta thêm vào giữa DS
        -) Nếu k>= len(L): thêm vào cuối DS

Chèn node có giá trị x vào đầu danh sách

- Xin cấp phát một nút mới cho con trỏ p lưu giá trị x.
- Để chèn p vào đầu danh sách trước tiên ta cho p trỏ đến L, sau đó chỉ việc cho L trỏ lại về p





Chèn Node p vào vị trí đầu tiên

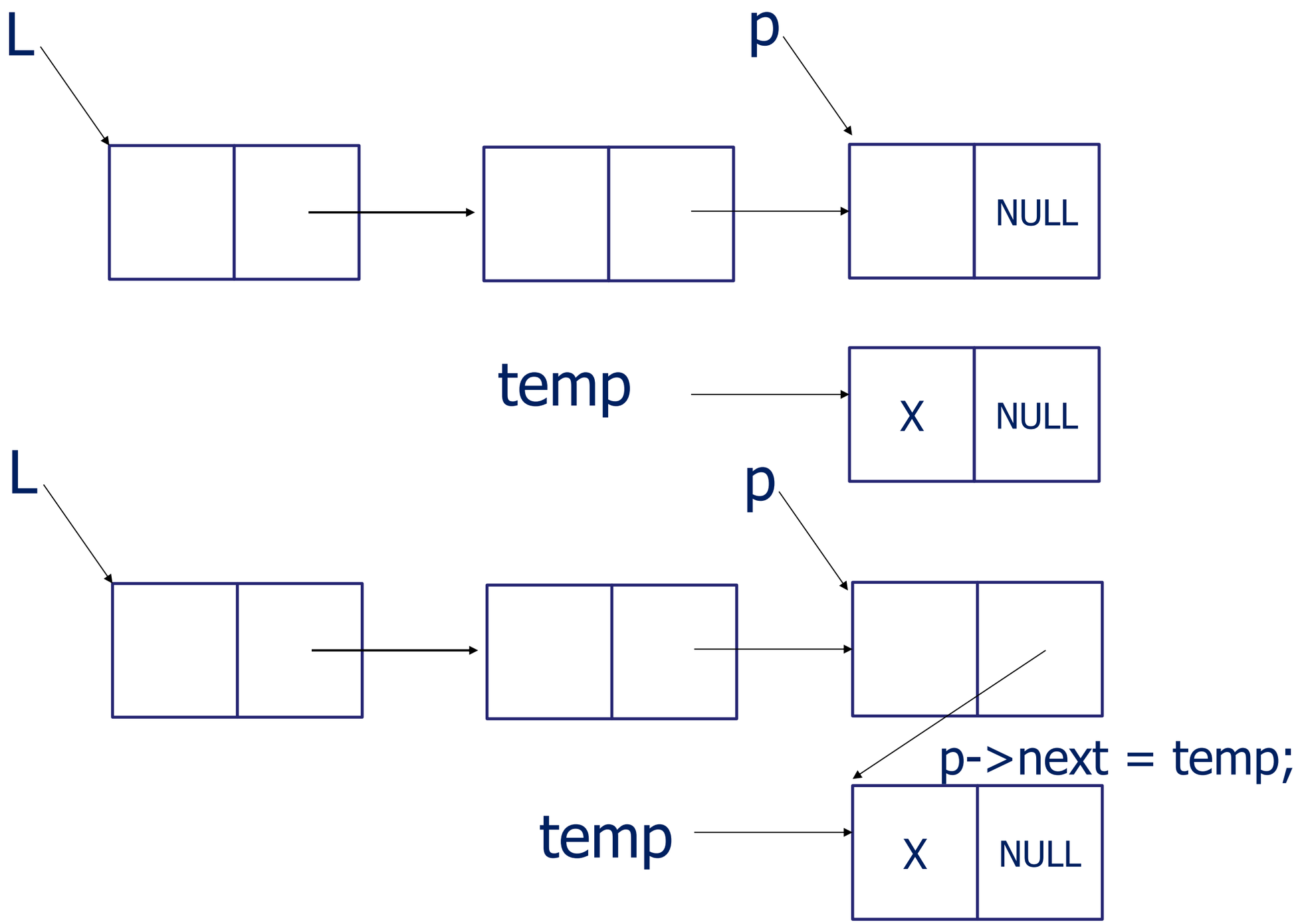
- Để chèn p vào đầu danh sách trước tiên ta cho p trỏ đến L, sau đó chỉ việc cho L trỏ lại về p

```
List InsertToHead(List L, item x){
    Node *p=CreateNode(x);
    p->next = L;
    L = p;
    return L;
}
```

- Hoặc

```
void InsertToHead(List &L, item x){
    Node *p=CreateNode(x);
    p->next = L;
    L = p;
}
```

Chèn giá trị x vào cuối DS

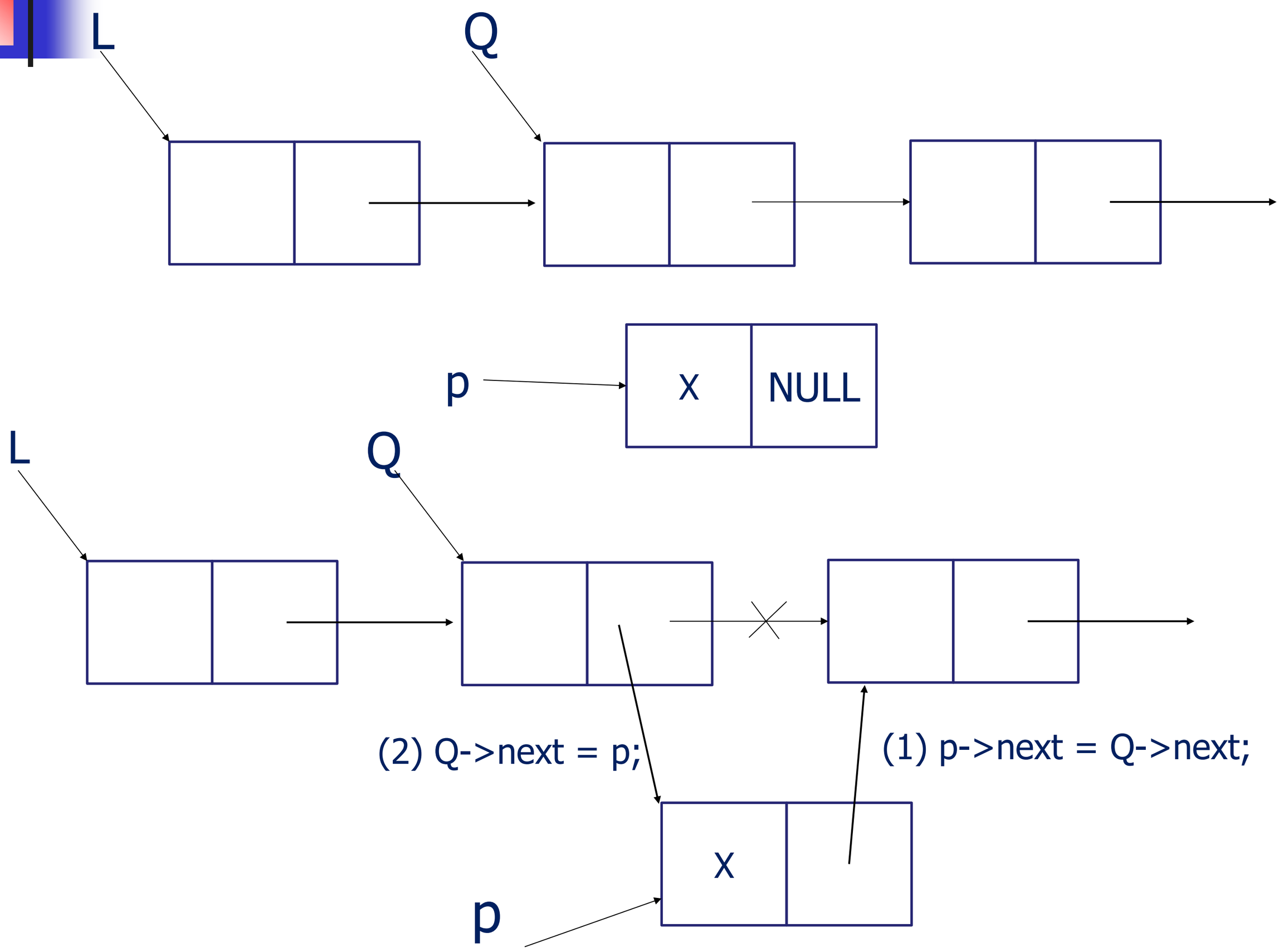


Chèn giá trị x vào cuối DS

```
List InsertToLast(List L, item x){
    List temp, p;
    temp = CreateNode(x);
    if(L == NULL){
        L = temp;
    }
    else{
        p = L;
        while( p->next != NULL){
            p = p->next;
        }
        p->next = temp;
    }
    return L;
}
```

```
void InsertToLast(List &L, item x){
    List temp, p;
    temp = CreateNode(x);
    if(L == NULL){
        L = temp;
    }
    else{
        p = L;
        while( p->next != NULL){
            p = p->next;
        }
        p->next = temp;
    }
}
```

Chèn giá trị x vào sau vị trí Q



Chèn giá trị x vào sau vị trí Q

```
void InsertToMiddle(List &L, Node *Q, item x){
    Node *p = new Node;
    p->data = x;
    p->next = Q->next;
    Q->next = p;
}
```

8- Tìm phần tử có giá trị x trong danh sách

- Ta duyệt danh sách cho đến khi tìm thấy hoặc kết thúc và trả về vị trí nếu tìm thấy, ngược lại trả về -1

```
int Search(List L, item x){
    int position = 0;
     Node *p;
     for(p = L; p != NULL; p = p->next){
        if(p->data == x){
            return position;
        }
        ++position;
    }
    return -1;
}
```

8- Tìm phần tử có giá trị x trong danh sách

- Ta duyệt danh sách cho đến khi tìm thấy hoặc kết thúc và trả về vị trí nếu tìm thấy, ngược lại trả về NULL

```
Node *Search(List L, item x){
   Node *node = L;
   while (node != NULL && node->data != x)
      node = node->next;
   if (node != NULL)
      return node;
   return NULL;
}
```

9- Xóa phần tử ra khỏi danh sách

- Tương tự như khi thêm một phần tử vào danh sách liên kết, muốn xóa một phần tử khỏi danh sách ta cần:
- Kiểm tra ds:
  - Nếu Ds rỗng (L=NULL): Báo lỗi ds rỗng
  - Nếu ds khác rỗng thì ta Xác định vị trí của phần tử muốn xóa trong danh sách L, giả sử ví trí thứ K:

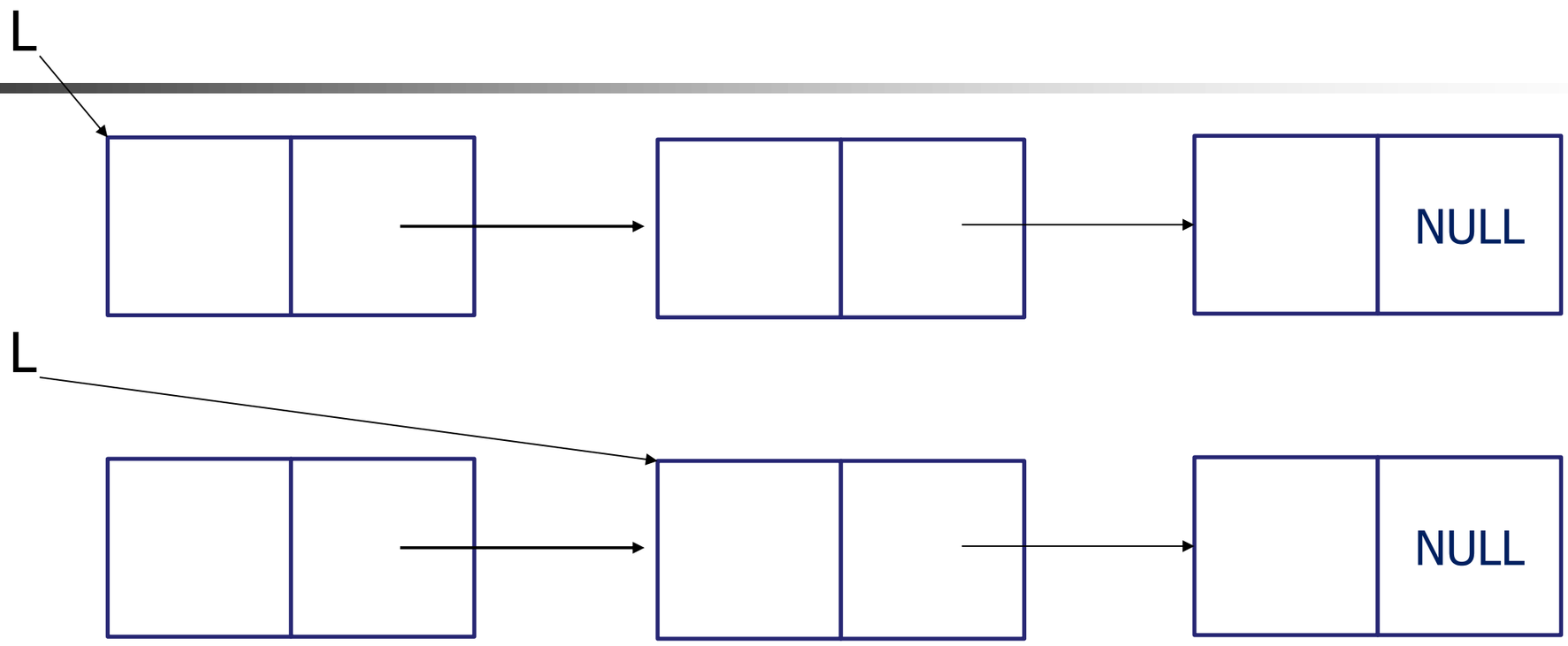
+ Nếu K<0 or K >= len(L): vị trí này ko tồn tại trong danh sách

+ Nếu K=0: ta thực hiện phép toán xóa phần tử ở vị trí đầu danh sách

+ Nếu 0< K < len(L)-1: ta thực hiện phép toán xóa phần tử ở giữa danh sách (ở vị trí thứ k bất kỳ)

+ Nếu K= len(L)-1: Xóa phần tử ở cuối danh sách

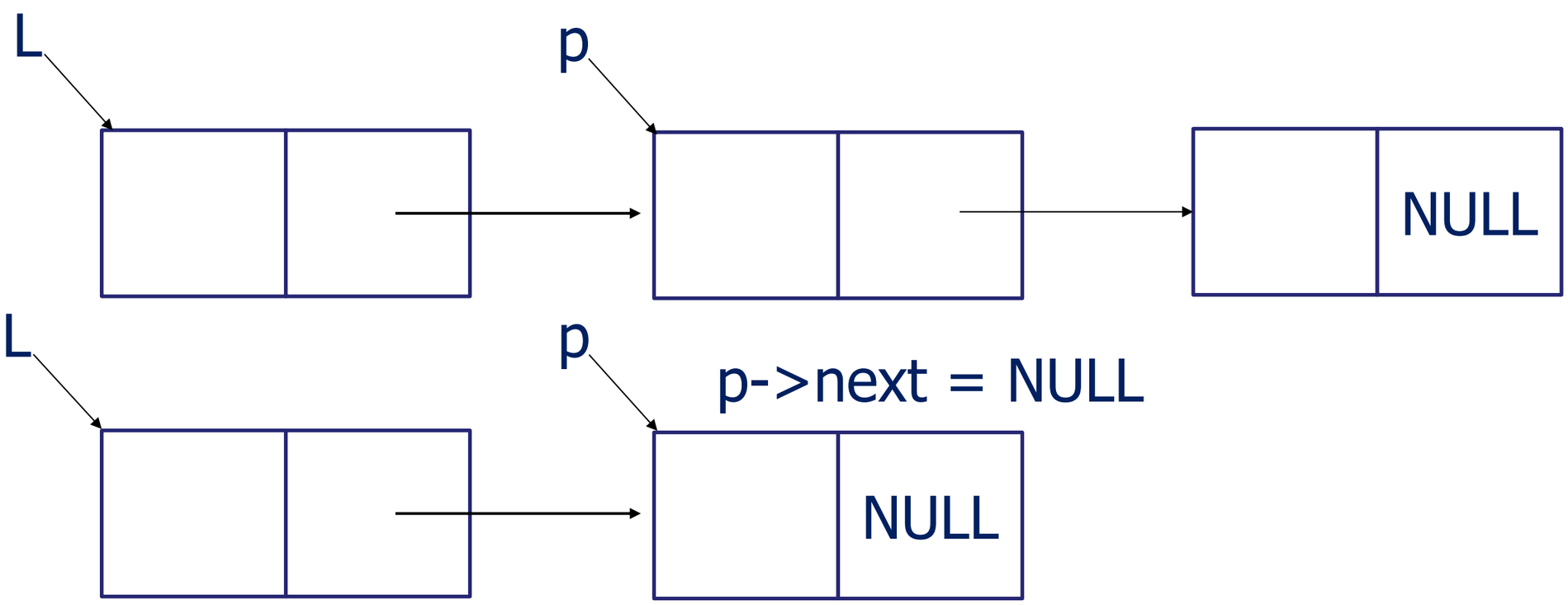
Xóa phần tử ở vị trí đầu tiên




```
List DeleteHead(List L){
    if(L == NULL){
        cout<<"\nDS rong!";
    }else{
        L = L->next;
    }
    return L;
}
```

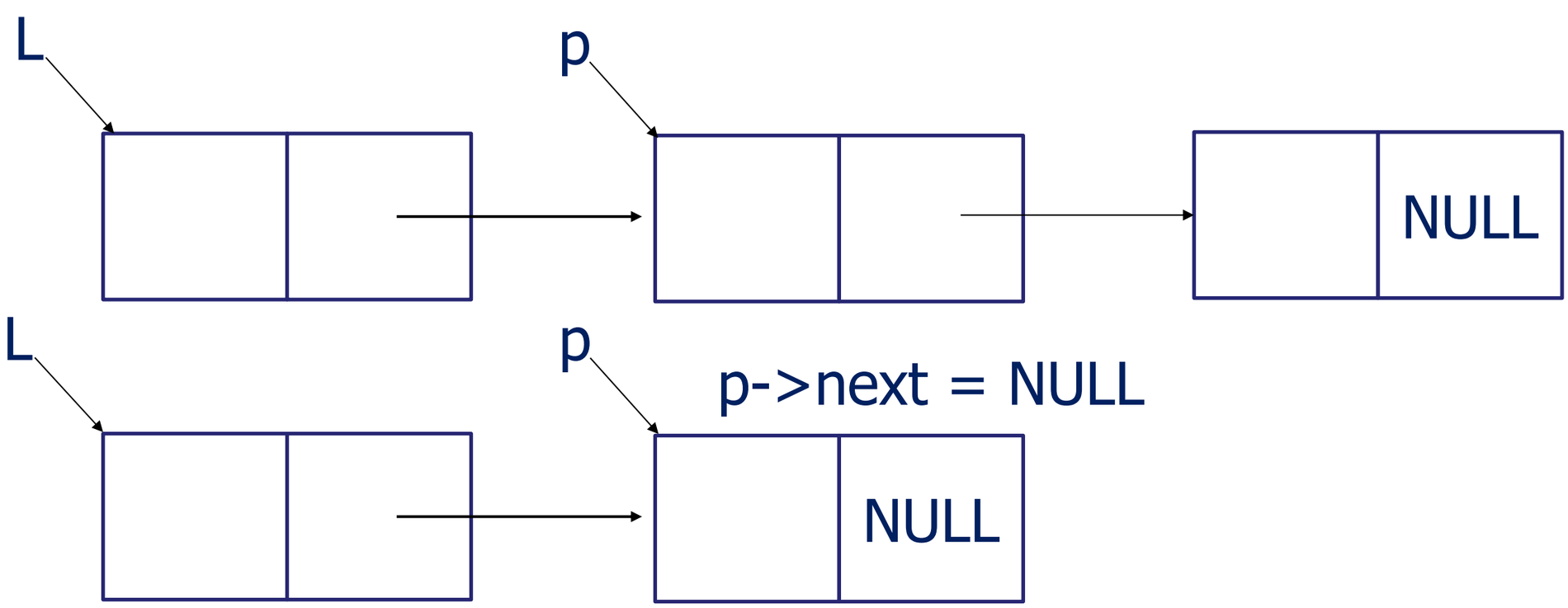
```
void DeleteHead(List &L){
    if(L == NULL){
        cout<<"\nDS rong!";
    }else{
        L = L->next;
    }
}
```

Xóa phần tử ở vị trí cuối DS



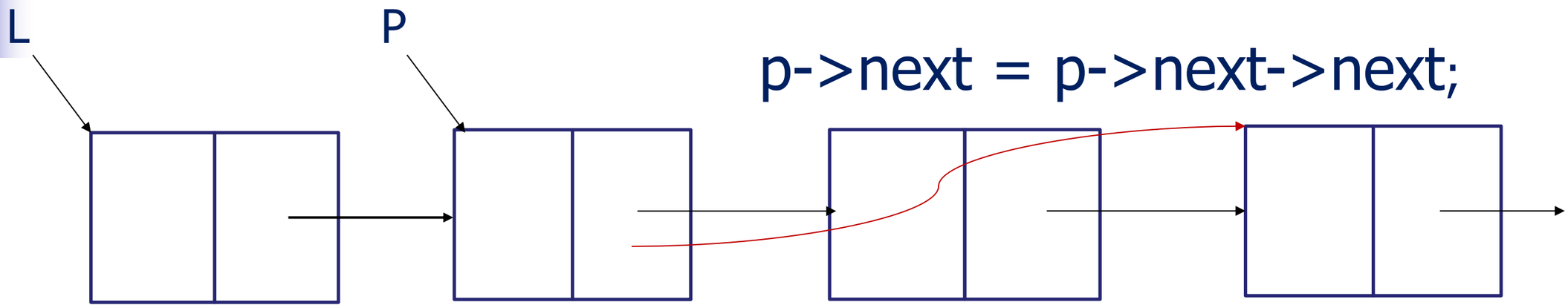

```
List DeleteTail(List L){
    if (L == NULL || L->next == NULL){
        return DeleteHead(L);
    }
    Node *p = L;
    while(p->next->next != NULL){
        p = p->next;
    }
     p->next = NULL ;
     return L;
}
```

Xóa phần tử ở vị trí cuối DS




```
void DeleteTail(List &L){
    if (L == NULL || L->next == NULL){
        return DeleteHead(L);
    }
    Node *p = L;
    while(p->next->next != NULL){
        p = p->next;
    }
    p->next = p->next->next; // Cho next bang NULL
    // Hoac viet p->next = NULL cung duoc
}
```

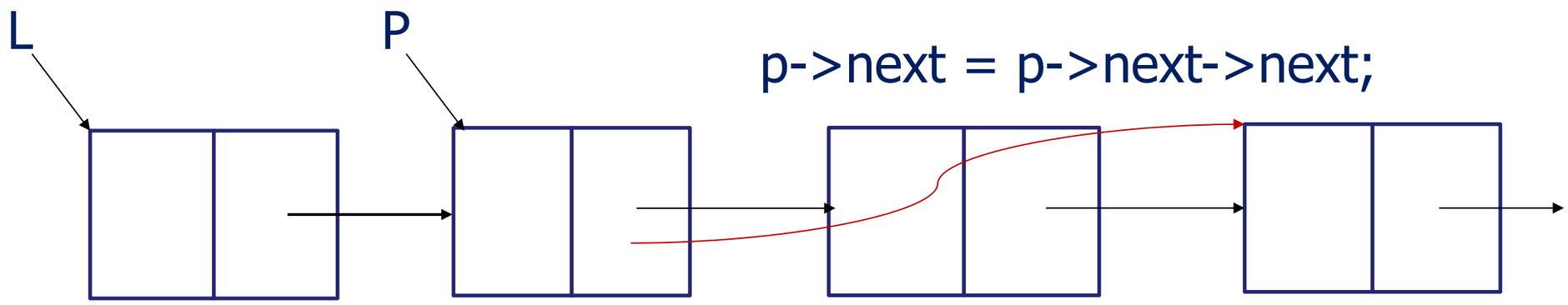
Xóa phần tử ở vị trí position




```
List DeleteAt(List L, int position){
    if(position == 0 || L == NULL || L->next == NULL){
        L = DeleteHead(L);
      }else{
      int k = 1;
        List p = L;
        while(p->next->next != NULL && k != position){
            p = p->next;
            ++k;
        }

        if(k != position){
            L = DeleteTail(L);
        }
      else{
            p->next = p->next->next;
        }
    }
    return L;
}
```

Xóa phần tử ở vị trí position




```
void DeleteAt(List &L, int position){
    if(position == 0 || L == NULL || L->next == NULL){
        L = DeleteHead(L);
    }else{
        int k = 1;
        List p = L;
        while(p->next->next != NULL && k != position){
            p = p->next;
            ++k;
        }

        if(k != position){
            L = DeleteTail(L);
        }
      else{
            p->next = p->next->next;
        }
    }
}
```

Ví dụ 1

- Viết chương trình nhập vào một danh sách gồm n số nguyên
- In ra danh sách
- Tính độ dài của danh sách
- Nhập số nguyên x. Thêm x vào cuối danh sách.

```
cho so phan tu n = 4
Nhap phan tu: 1
Nhap phan tu: 2
Nhap phan tu: 3
Nhap phan tu: 4
Hien thi danh sach: 1 2 3 4
Danh sach co do dai = 4
Cho phan tu can them vao cuoi DS x = 5
Hien thi danh sach: 1 2 3 4 5
```

```cpp
#include <iostream>
using namespace std;

typedef int item;
struct Node{
    item data;
    Node *next;
};
typedef Node *List;

void Init(List &L){
    L= NULL;
}
void IsEmpty(List L){
    if(L == NULL)
        cout<<"DS rong!\n";
    else
        cout<<"DS khong rong!\n";
}
```

```cpp
int Length(List L)
{
    int count = 0;
    Node *node = L;
    while (node != NULL)
    {
        count++;
        node = node->next;
    }
    return count;
}
List CreateNode(item x)
{
    Node *node = new Node;
    node->data = x;
    node->next = NULL;
    return node;
}
```

```cpp
List InsertToHead(List L, item x){
    Node *p = new Node;
    p->data = x;
    p->next = L;
    L = p;
    return L;
}
List InsertToLast(List L, int x){
    List temp, p;
    temp = CreateNode(x);
    if(L == NULL)  L = temp;
    else{
        p = L;
        while(p->next != NULL) p = p->next;
        p->next = temp;
        }
    return L;
}
```

```cpp
void Input(List &L, int n){
    item x;
    for(int i =0; i<n;i++){
        cout<<"Nhap phan tu: ";cin>>x;
        if(L== NULL)
            L = InsertToHead(L,x);
        else
            L= InsertToLast(L,x);
    }
}
void Output(List L){
    Node *p=L;
    cout<<"Hien thi danh sach:";
    while(p!= NULL){
            cout<<" "<<p->data;
            p=p->next;
        }
}
```

```cpp
int main(){
    List L;
    cin.clear();
    Init(L);
    IsEmpty(L);
    int n;
    cout<<"cho so phan tu n = ";cin>>n;
    Input(L,n);
    Output(L);
    cout<<"\nDanh sach co do dai = "<<Length(L)<<endl;
    item x;
    cout<<"Cho phan tu can them vao cuoi DS x = ";cin>>x;
    Output(InsertToLast(L,x));
    return 0;
}
```

Ví dụ 2

- Viết chương trình nhập vào một danh sách n số nguyên
- In ra danh sách. Nhập số nguyên x. Tìm xem x có xuất hiện trong DS hay không? Nếu x không có trong DS thì thêm x vào cuối danh sách.

Ví dụ 2

- Viết chương trình nhập vào một danh sách n số nguyên
- In ra danh sách. Nhập số nguyên x. Tìm xem x có xuất hiện trong DS hay không? Nếu x không có trong DS thì thêm x vào cuối danh sách.

```
Node *Search(List L, int x)
{
    Node *node = L;
    while (node != NULL && node->data !=
x)
        node = node->next;
    if (node != NULL)
        return node;
    return NULL;
}
```

```
int main(){
    List L;
    int n;
    Init(L);
    Input(L,n);
    Output(L);
    cout<<"\nDanh sach co do dai =
"<<Length(L)<<endl;
    int x;
    cout<<"Cho x = "; cin>>x;
    if(Search(L,x)!= NULL)
        cout<<"x xuat hien trong DS!";
    else
    {
        cout<<"x khong xuat hien trong DS!
Them x vao cuoi DS!\n";
        L = InsertToLast(L,x);
        Output(L);
    }

    return 0;
}
```

Ví dụ 2b

- Viết chương trình nhập vào một danh sách các số nguyên gồm n phần tử
- In ra danh sách
- Nhập số nguyên x và vị trí k. Chèn x vào vị trí k (nếu k>= độ dài DS thì chèn vào cuối, nếu k<=0 thì chèn vào đầu)
- Nhập vị trí k. Xóa phần tử ở vị trí k (nếu k>= độ dài DS thì xóa ptử cuối, nếu k<=0 thì xóa đầu
- Cho số nguyên y. Tìm y xuất hiện ở vị trí nào trong DS.

```
----Tao 1 danh sach lien ket-----
Nhap so phan tu n = 4
Nhap gia tri: 1
Nhap gia tri: 2
Nhap gia tri: 3
Nhap gia tri: 4

  1  2  3  4
----Them 1 phan tu vao DSLK---
Cho phan tu muon them: 5
Cho vi tri muon chen: 4

  1  2  3  4  5
----Xoa 1 phan tu khoi DSLK----
Cho vi tri muon xoa: 3

  1  2  3  5
----Tim kiem 1 phan tu trong DSLK----
Cho phan tu muon tim: 3

Tim thay tai chi so: 2
```

Ví dụ 3

- Viết chương trình nhập vào một danh sách các số nguyên gồm n phần tử
- In ra danh sách
- Nhập số nguyên x. Thêm x vào đầu, cuối danh sách
- Nhập vị trí k. Xóa phần tử ở vị trí k.

Ví dụ 4

- Nhập danh sách trúng tuyển gồm n sinh viên với các thông tin: Mã SV, Họ và tên, Ngày sinh, Điểm trúng tuyển
- In ra danh sách trúng tuyển
- Sắp xếp danh sách theo họ tên theo abc
- Tìm sinh viên có điểm trúng tuyển cao nhất

Cấu trúc với thành phần kiểu bit

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 với số bít n theo sau (0<=n<=15)

Cấu trúc với thành phần kiểu bit

Trường bit

- Ví dụ: Khai báo cấu trúc ngày tháng năm

```
struct Date {
    int day: 5;
    int month: 4;
    int year;
};
```

Đặc điểm: Cần chú ý các đặc điểm sau của một cấu trúc có chứa trường bit:

- Các bit được bố trí liên tục trên dãy các byte.
- Kiểu trường bit phải là int (signed hoặc unsigned).
- Độ dài mỗi trường bit không quá 16 bit.

Cấu trúc với thành phần kiểu bit

- Có thể bỏ qua một số bit nếu bỏ trống tên trường, ví dụ:

```
struct tu{
        int: 8;
        int x:8;
}
```

mỗi một biến cấu trúc theo khai báo trên gồm 2 byte, bỏ qua không sử dụng byte thấp và trường x chiếm byte (8 bit) cao.

- Không cho phép lấy địa chỉ của thành phần kiểu bit.
- Không thể xây dựng được mảng kiểu bit.
- Không được trả về từ hàm một thành phần kiểu bit.

Ví dụ nếu b là một thành phần của biến cấu trúc x có kiểu bit thì câu lệnh sau là sai:

```
return x.b ; // sai
```

tuy nhiên có thể thông qua biến phụ như sau:

```
int tam = x.b ;
return tam ;
```

- Tiết kiệm bộ nhớ
- Dùng trong kiểu union để lấy các bit của một từ (xem ví dụ trong phần kiểu
hợp).

Kiểu hợp

- 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
- Khai báo:

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

- 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).

Kiểu hợp

```
#include <iostream>
using namespace std;
union phanso{
    int x, y;
    }a;
struct PhanSo{
    int x, y;
    }b;
int main(){
    cout << "Nhap phan so a = " ; cin >> a.x>>a.y;
    cout << "Nhap phan so b = " ; cin >> b.x>>b.y;
    cout << "Kich thuoc cua a = " << sizeof(a) << endl;
    cout << "Kich thuoc cua b = " << sizeof(b) << endl;
    return 0;
}
```

```
Nhap phan so a = 1 2
Nhap phan so b = 1 2
Kich thuoc cua a = 4
Kich thuoc cua b = 8
```

Ví dụ

```
#include <iostream>
using namespace std;
union SV{
    char ten[30];
    int tuoi;
}a;
struct SV1{
    char ten[30];
    int tuoi;
}b;
```

```
Nhap SV a:
Ho ten: Hoang Linh
Tuoi: 20
Nhap SV b:
Ho ten: Hoang Linh
Tuoi: 20
Kich thuoc bo nho cua union a = 32
Dia chi bo nho union: 0x4a7040 0x4a7040
Kich thuoc bo nho cua struct b = 36
Dia chi bo nho struct: 0x4a7060 0x4a7080
```

```
int main(){
   cout << "Nhap SV a: \n" ;
   cout<<"Ho ten: "; cin.getline(a.ten,30);
   cout<<"Tuoi: "; cin>>a.tuoi;
   cin.ignore();
   cout << "Nhap SV b: \n" ;
   cout<<"Ho ten: "; cin.getline(b.ten,30);
   cout<<"Tuoi: "; cin>>b.tuoi;
   cout <<"Kich thuoc bo nho cua union a = " << sizeof(a) << endl;
   cout<<"Dia chi bo nho union: "<<&a.ten<<" "<<&a.tuoi<<endl;
   cout <<"Kich thuoc bo nho cua struct b = " << sizeof(b) << endl;
   cout<<"Dia chi bo nho struct: "<<&b.ten<<" "<<&b.tuoi<<endl;
   return 0;
}
```

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
```

Kiểu liệt kê

Ví dụ: enum Bool {false, true};

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
```

Kiểu liệt kê

```
enum dayOfWeek{Mon = 2, Tue = 3, Wed = 4, Thur = 5,
Fri = 6, Sat = 7, Sun = 8};
int main(){
    enum dayOfWeek day;
     day=Mon;
     cout<<"Ngày hom nay: "<<day<<endl;
     cout<<"Các ngày trong tuan: "<<Mon<<"; "<<Tue<<";
"<<Wed<<"; "
    <<Thur<<"; "<<Fri<<"; "<<Sat<<"; "<<Sun;
    return 0;
}
```