CHƯƠNG 5 (tiếp)

5.4. DANH SÁCH MÓC NỐI

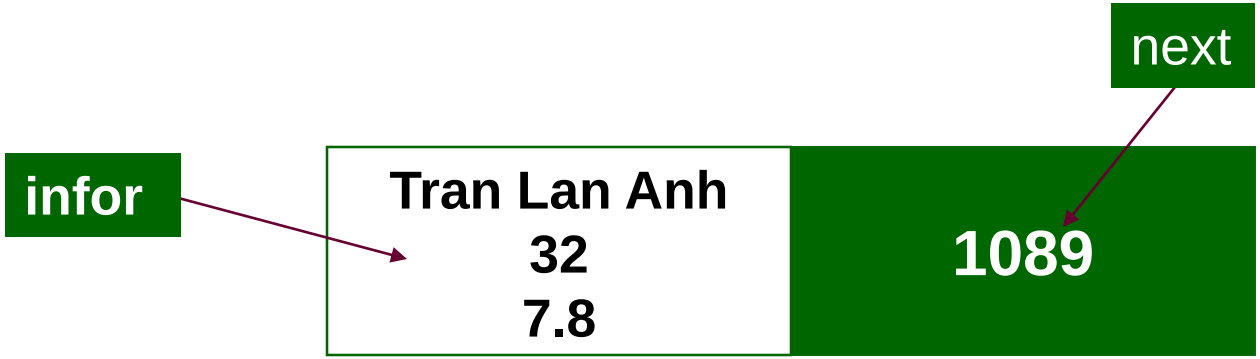
5.4.1. Khái niệm danh sách móc nối đơn

- **Nguyên tắc tạo thành danh sách**
  - Danh sách được tạo thành từ các phần tử gọi là nút *(Node)*
  - Các *node* có thể nằm bất kỳ đâu trong bộ nhớ
  - Mỗi *node* là một cấu trúc gồm 2 thành phần
    - infor chứa thông tin của 1 phần tử của danh sách L
    - next là một con trỏ, nó trỏ vào node đứng sau

Một node trong danh sách

Khái niệm danh sách móc nối đơn (tt)

- **Ví dụ**



Một node trong danh sách sinh viên

1089 là địa chỉ vùng nhớ của node đứng sau

Khái niệm danh sách móc nối đơn (tt)

- Để truy nhập vào các node trong danh sách ta phải đi từ node đầu tiên
- Cần một con trỏ, trỏ vào node đầu trong danh sách
- Phần tử cuối cùng của danh sách có next = NULL

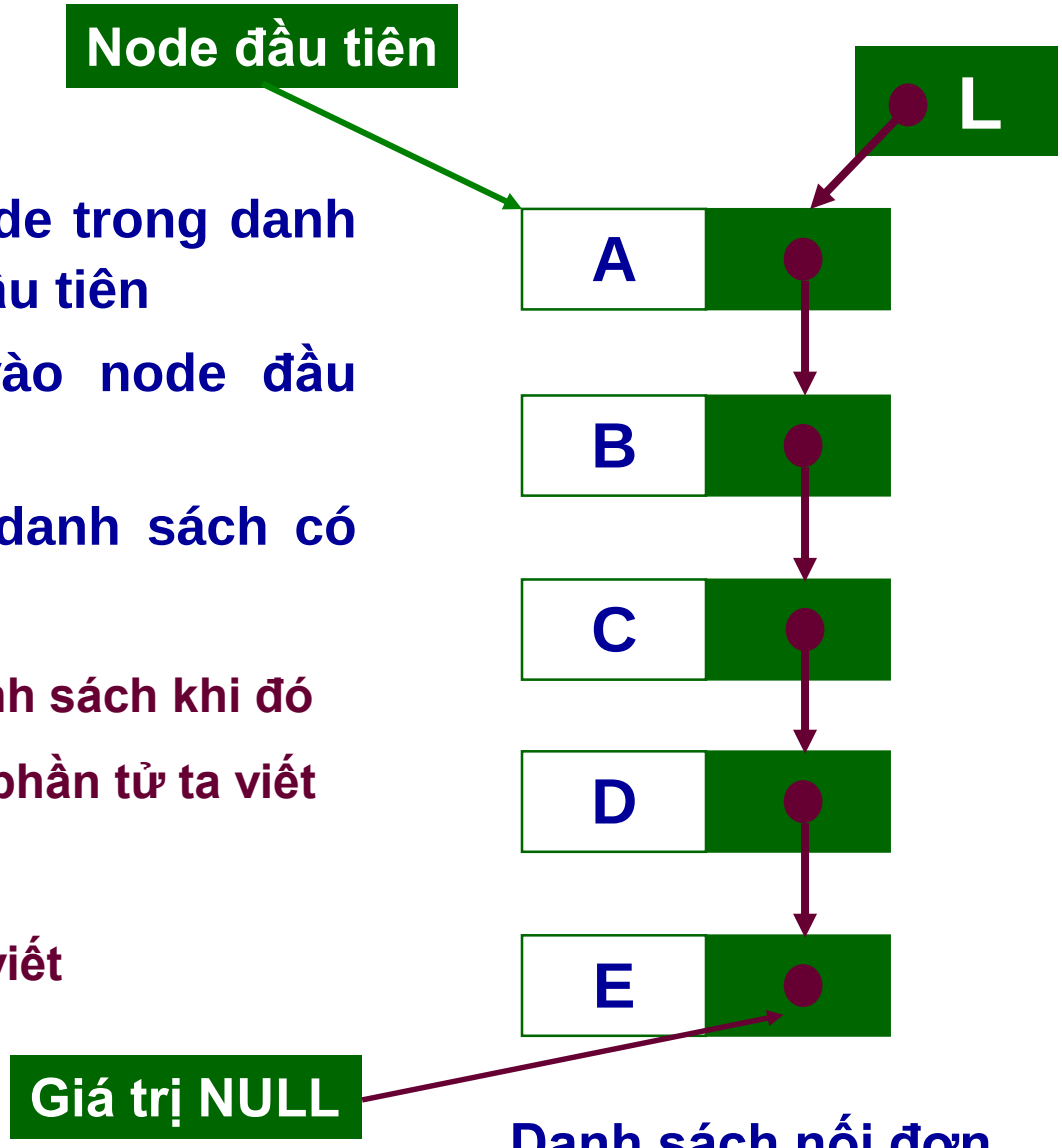
L trỏ vào node đầu tiên của danh sách khi đó

Để truy xuất vào thông tin của phần tử ta viết

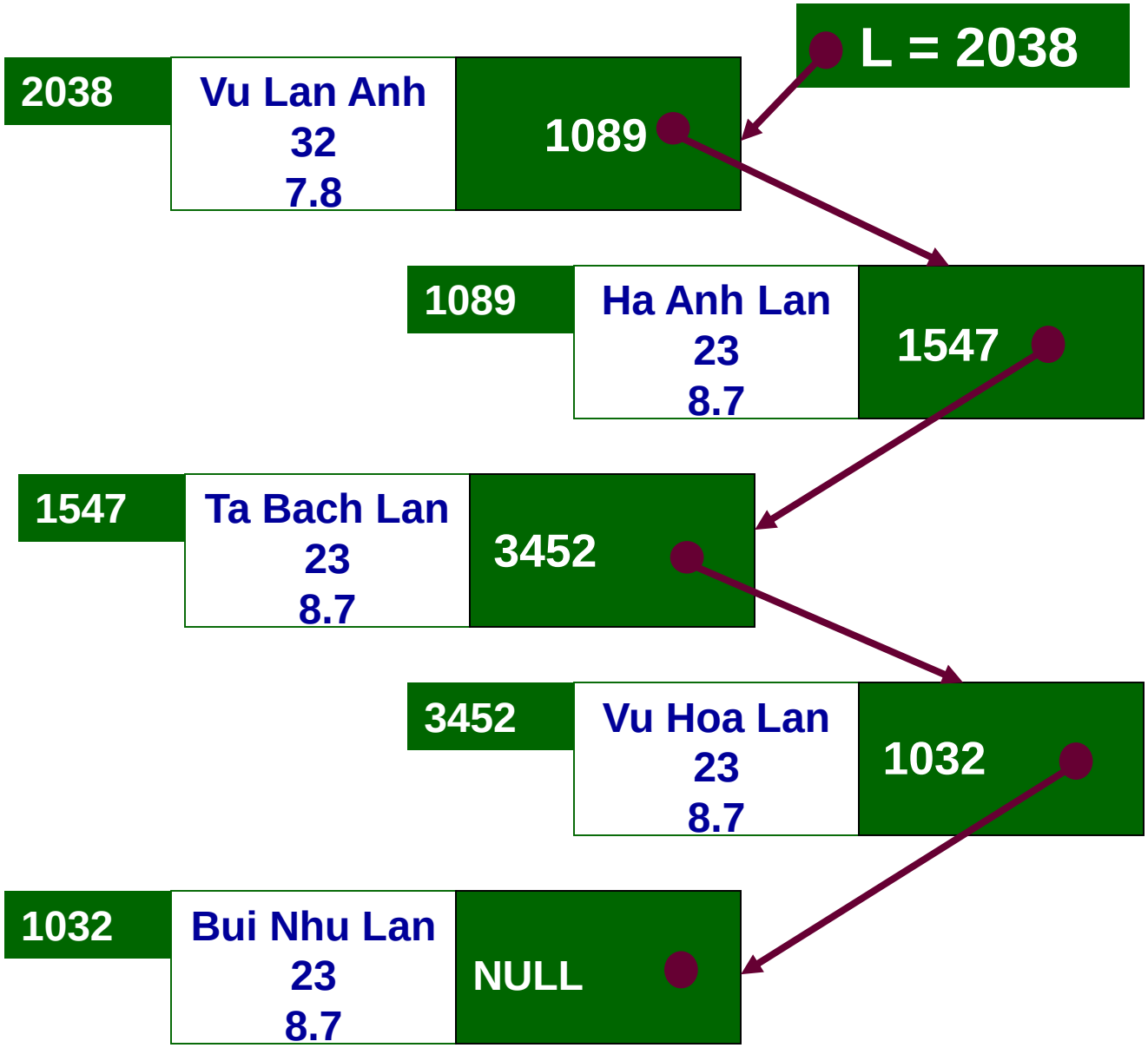
L->infor

Để chỉ ra phần tử đứng sau ta viết

L->next



Khái niệm danh sách móc nối đơn (tt) – Ví dụ



5.4.2. Ưu và nhược điểm của danh sách nối đơn

- **Ưu điểm:**
  - Tiết kiệm bộ nhớ.
  - Các thao tác thêm và xóa thực hiện nhanh vì không phải dịch chuyển các phần tử.
- **Nhược điểm:**
  - Việc truy xuất vào các phần tử chậm vì luôn phải xuất phát từ phần tử đầu tiên.
  - Chỉ duyệt được danh sách theo một chiều nhất định, từ trên xuống.
  - Các thao tác khá phức tạp, khó hiểu với người mới lập trình.

5.4.3. Khai báo cấu trúc dữ liệu

Khai báo kiểu dữ liệu phần tử

```
struct Item {
   //Dữ liệu phần tử;
};
```

Khai báo kiểu dữ liệu Node

```
struct Node {
      Item infor;
      Node *next;
};
```

Khai báo kiểu con trỏ Node

```
typedef Node *TRO;
```

Con trỏ L trỏ vào Node đầu

```
TRO L;
```

Khai báo cấu trúc dữ liệu (tt) – Ví dụ

Khai báo kiểu dữ liệu phần tử

```
struct SinhVien {
    int id;
    char hoTen[30];
    int tuoi;
    float diemTk;
};
```

Khai báo kiểu dữ liệu Node

```
struct Node {
    SinhVien infor;
    Node *next;
};
```

Khai báo kiểu con trỏ Node

```
typedef Node *TRO;
```

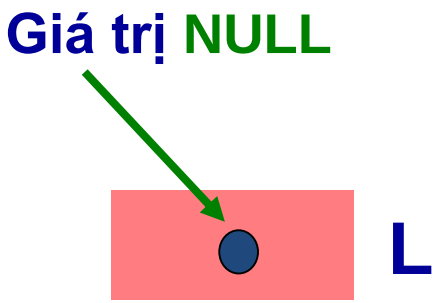
Con trỏ L trỏ vào Node đầu

```
TRO L;
```

5.4.4. Các phép toán trên danh sách

1. Khởi tạo danh sách rỗng
2. Kiểm tra danh sách rỗng
3. Duyệt danh sách
4. Tìm kiếm một node trên danh sách
5. Bổ sung node mới vào đầu danh sách
6. Bổ sung node mới vào sau một node
7. Xóa node đầu danh sách
8. Xóa node đứng sau một node trong danh sách
9. Sắp xếp danh sách

5.4.4.1. Khởi tạo danh sách rỗng



Danh sách nối đơn rỗng

L = NULL -> ds L rỗng

5.4.4.1. Khởi tạo danh sách rỗng

a) Khởi tạo danh sách rỗng

```
void creat(TRO &L){
      L = NULL;//cho L trỏ đến NUll
}
```

5.4.4.2. Kiểm tra danh sách rỗng

b) Kiểm tra danh sách rỗng

```
int empty(TRO L){
      return L == NULL;
}
```

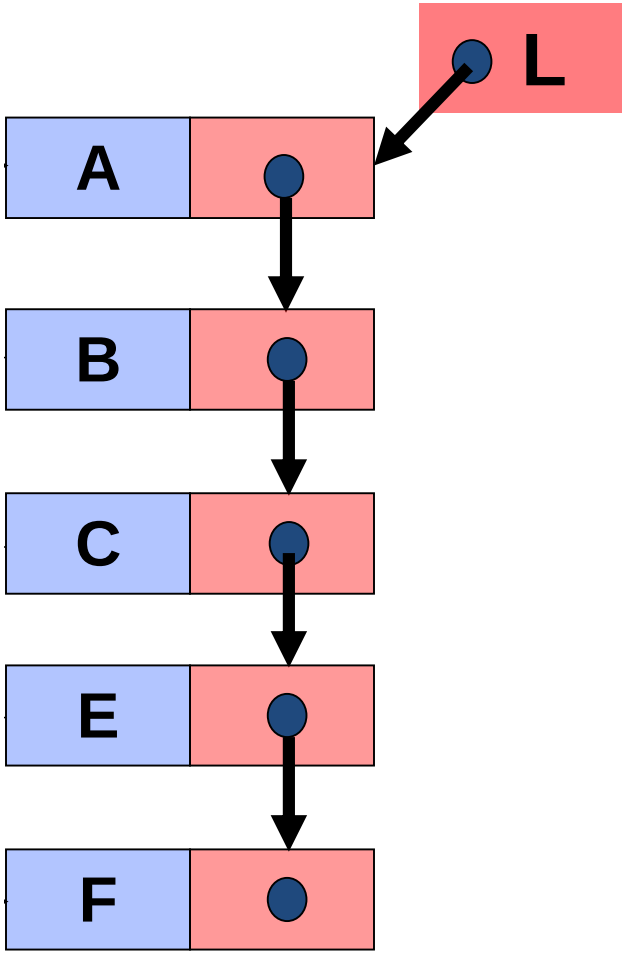
5.4.4.3. Duyệt danh sách

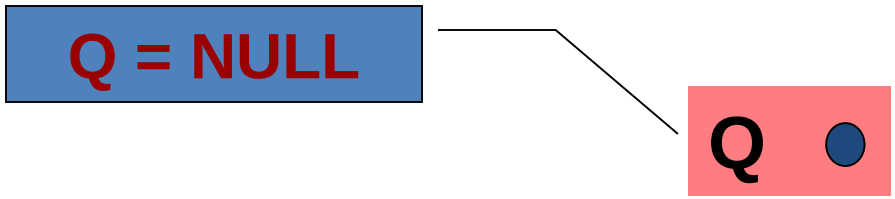
1. Nếu danh sách không rỗng, cho con trỏ Q trỏ vào node đầu tiên: Q = L;

2. Nếu Q != NULL thì (thực hiện yêu cầu) và chuyển Q xuống node ngay sau nó:

   Q=Q->next;

3. Lặp lại bước 2





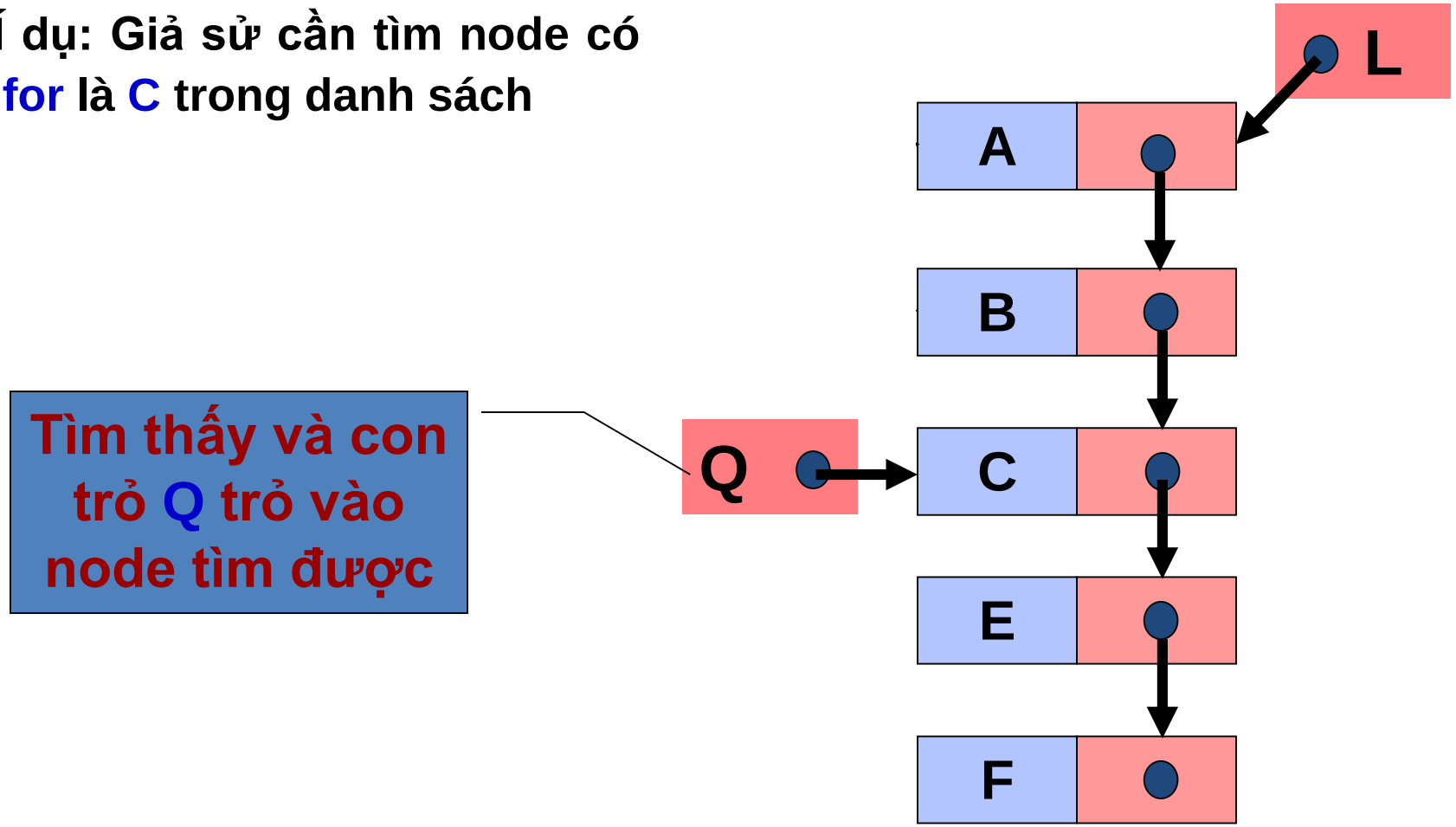
Duyệt danh sách (tt)

- **Hàm duyệt danh sách như sau**

```
void travel(TRO L)
{
        TRO Q; //tạo 1 Node Q để duyệt
        if (!empty(L))
        {
                Q = L; //cho Q trỏ vào Node đầu tiên
                int i=0; //biến đếm
                while (Q != NULL)
                {
                   i++;//Các lênh (nếu muốn tính độ dài của danh sách
                   Q = Q->next; //cho Q trỏ dến Node tiếp theo
                }
                return i; //trả lại số Node của danh sách;
        }
}
```

5.4.4.4. Tìm kiếm một nút trên danh sách

Ví dụ: Giả sử cần tìm node có infor là C trong danh sách



Tìm kiếm một nút trên danh sách (tt)

Thuật toán tìm kiếm một nút trên danh sách

1. Nếu danh sách không rỗng, cho con trỏ Q trỏ vào node đầu tiên: Q = L;
2. Nếu (Q != NULL) và (chưa trỏ vào node cần tìm) thì (có thể thực hiện yêu cầu) và chuyển Q xuống node ngay sau nó:

   Q=Q->next;

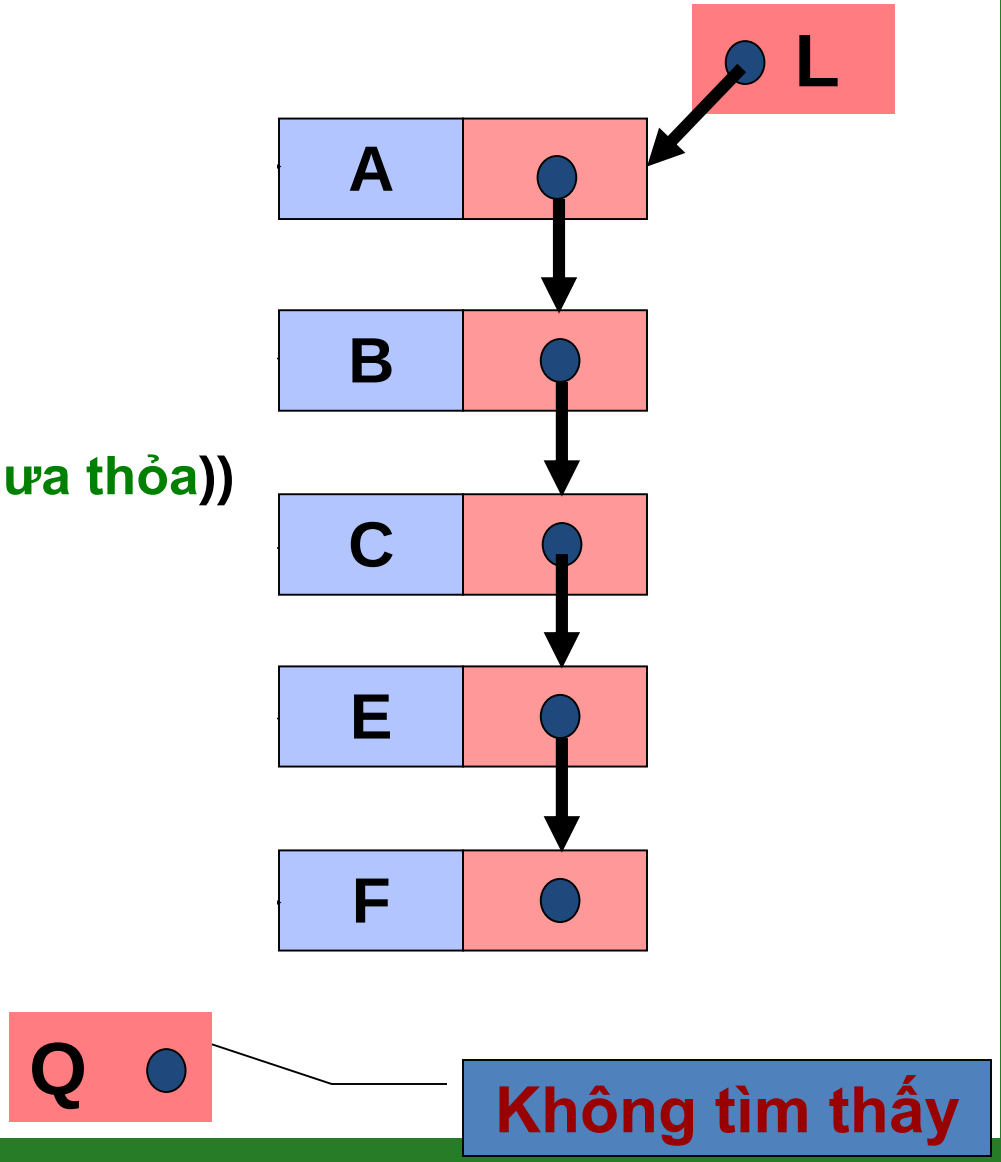
   Ngược lại thì sang bước 4
3. Lặp lại bước 2
4. Trả về con trỏ Q: return Q;

Tìm kiếm một nút trên danh sách (tt)

Ví dụ: Giả sử cần tìm node có infor là K trong danh sách

```
TRO Search(TRO L) {
    TRO Q = L;
    while (Q != NULL && (ĐKTK chưa thỏa))
        Q = Q->next;
    return Q;
}
```

Hàm Search trả về NULL nếu không tìm thấy, ngược lại trả về con trỏ trỏ vào node tìm được



5.4.4.5. Chèn một nút vào đầu danh sách

Danh sách có phần tử đầu tiên được trỏ bởi con trỏ L

Giả sử dữ liệu của phần tử lưu trong biến X

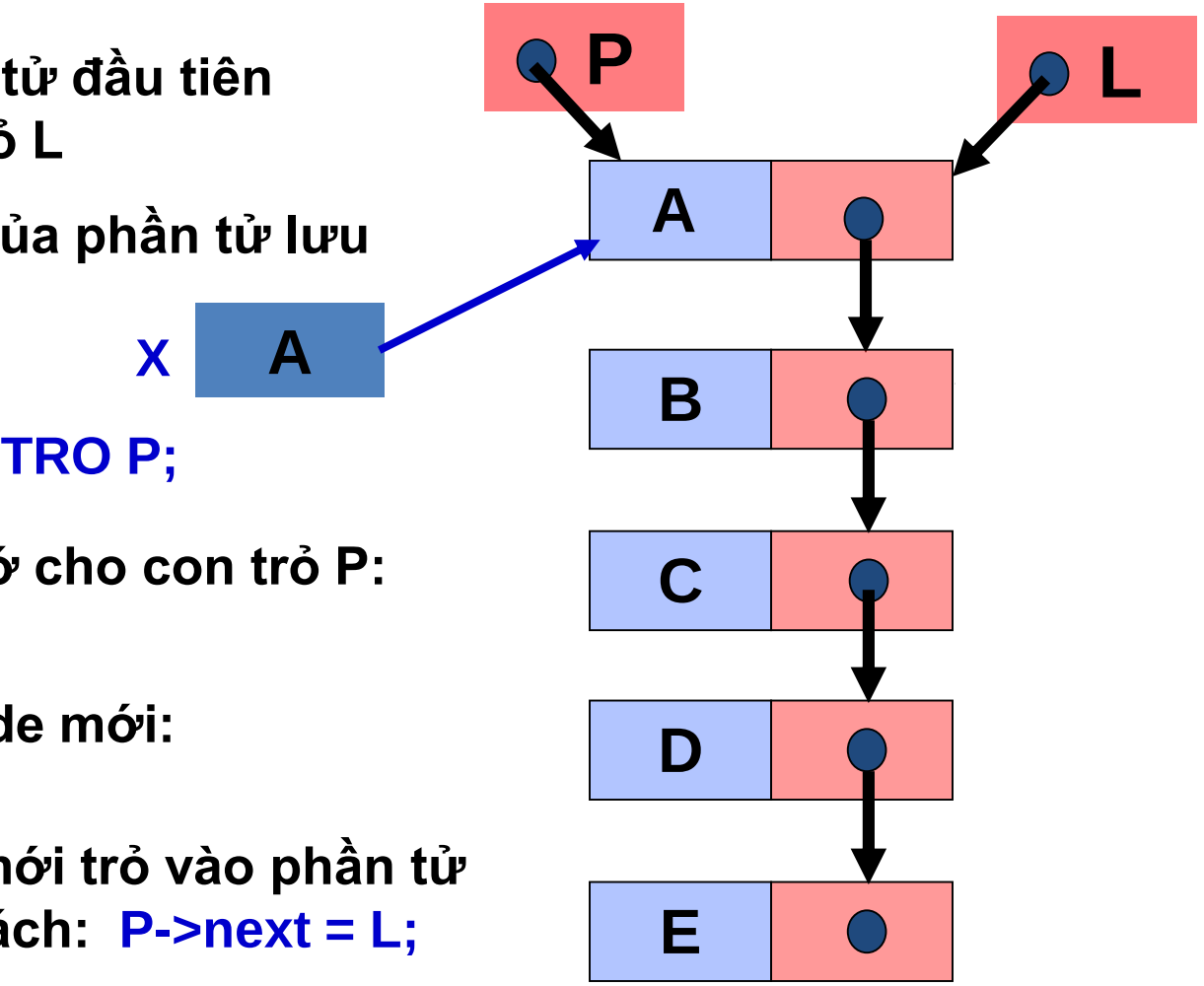
Khai báo con trỏ P: TRO P;

Cấp phát bộ nhớ cho con trỏ P:
P = new Node;

Đưa dữ liệu vào node mới:
P ->infor = X;

next của node mới trỏ vào phần tử đầu của danh sách: P->next = L;

L trỏ vào node mới: L = P;

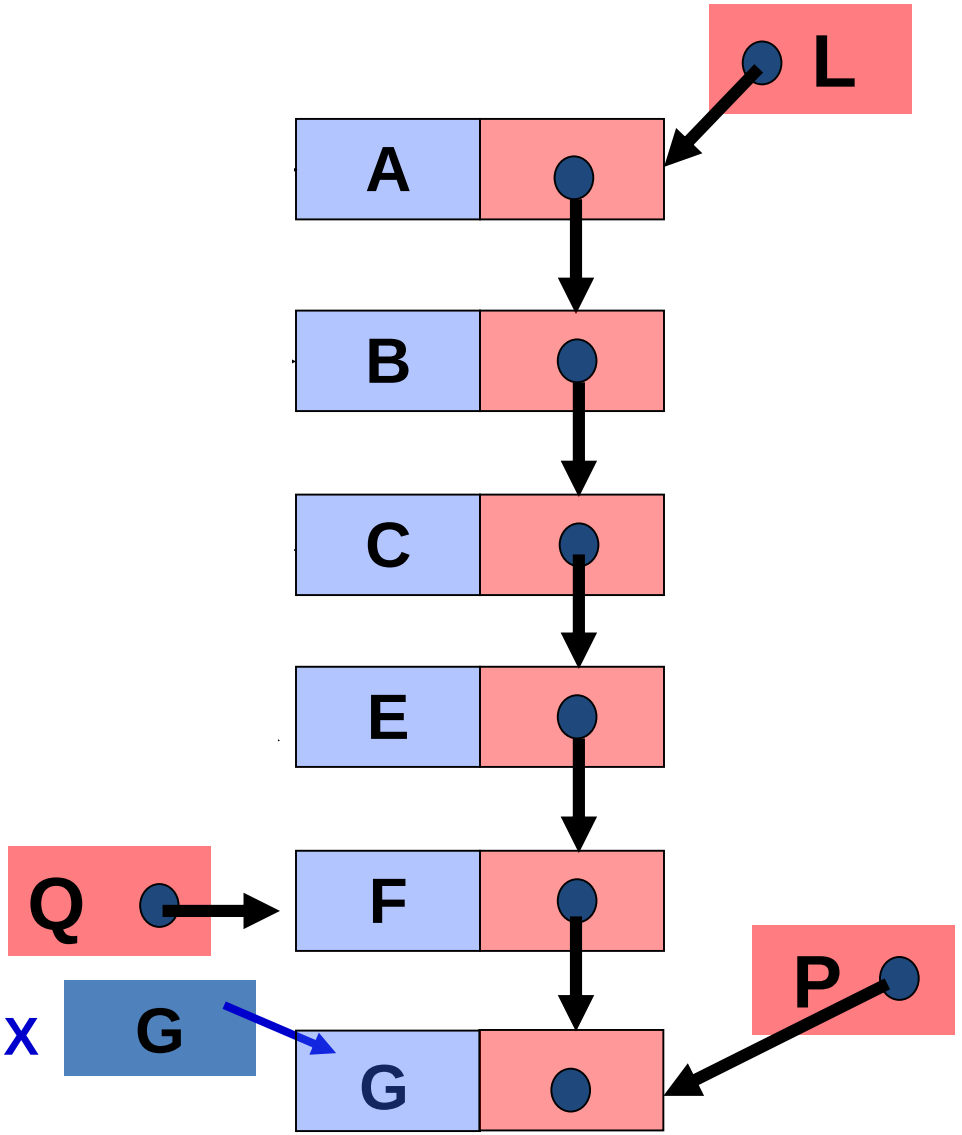


5.4.4.5. Chèn một nút vào đầu danh sách

```
void First_Add(TRO &L, Item X)
{
        TRO P;
        P = new Node;
        P->infor = X;
        P->next = L;
        L = P;
}
```

5.4.4.6. Chèn một nút vào cuối danh sách

1. Khởi tạo một node mới và đưa dữ liệu vào node mới.
2. Đưa con trỏ Q tìm đến cuối danh sách.
3. Nối node cuối với node mới



5.4.4.6. Chèn một nút vào cuối danh sách

```
void Add(TRO &L, Item X)
{
        TRO P,Q;
        P = new Node;  P->infor = X;
        P->next = NULL;
        if (L==NULL)  L = P;
        else {
                Q = L;
                while (Q->next  != NULL)
                        Q = Q->next;
                Q->next = P;
        }
}
```

5.4.4.7. Chèn một nút vào sau nút trỏ bởi Q

Danh sách có phần tử đầu tiên được trỏ bởi con trỏ L

Q trỏ vào node mà node mới được bổ sung vào sau nó

Dữ liệu lưu trong biến X

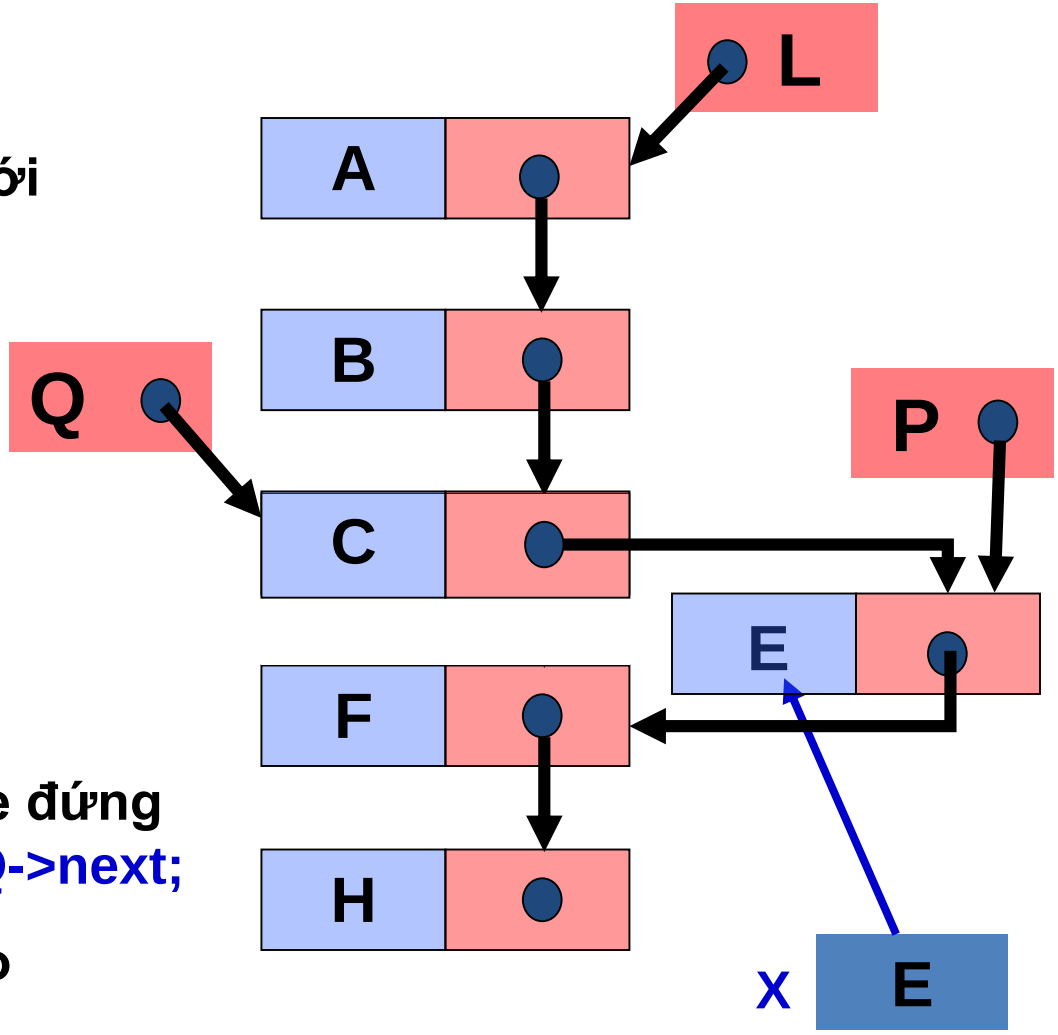
Khai báo con trỏ P: TRO P;

Cấp phát bộ nhớ cho con trỏ P:
P = new Node;

Đưa dữ liệu vào node mới:
P ->infor = X;

next của node mới trỏ vào node đứng sau node trỏ bởi Q: P->next = Q->next;

next của node trỏ bởi Q trỏ vào node mới: Q->next = P;



5.4.4.7. Chèn một nút vào sau nút trỏ bởi Q

```
void Insert(TRO &L, TRO Q, Item X)
{       TRO P;
        P = new Node;
        P->infor = X;
        P->next = Q->next;
        Q->next = P;
}
```

Hàm Insert cũng thỏa mãn nếu bổ sung phần tử vào cuối danh sách, khi đó Q trỏ vào node cuối danh sách

chèn x vào vị trí k trong danh sách

```
void Insert_k (List *L, item x, int k) //chen x vao vi tri k trong danh sach
{
    Node *P, *Q = L;
    int i=1;
    if (k<1 || k> len(L)+1) //kiem tra dieu kien
    else
    {
        P = Make_Node(P,x); //tao 1 Node P
        if (k == 1) Insert_first(L,x); //chen vao vi tri dau tien
        else //chen vao k != 1
        {
            while (Q != NULL && i != k-1) //duyet den vi tri k-1
            {
                i++;
                Q = Q->next;
            }
            P->next = Q->next;
            Q->next = P;
        }
    }
}
```

5.4.4.8. Xóa nút đầu tiên trong danh sách

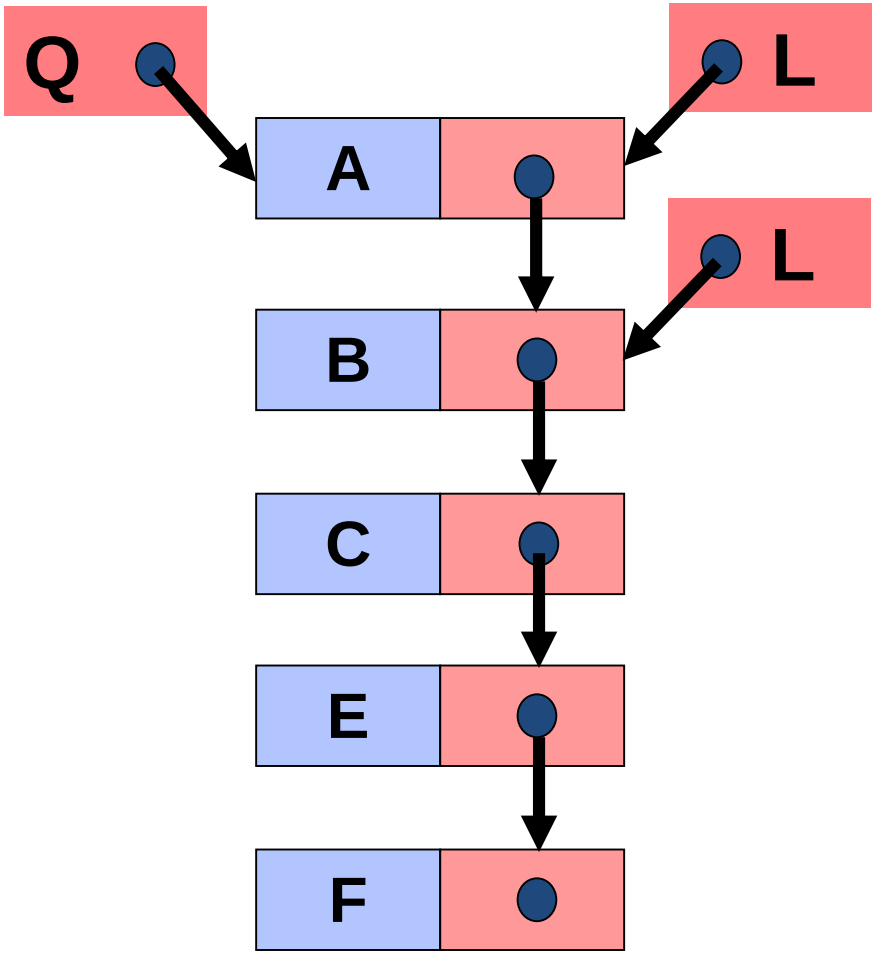
Buổi sau tiếp tục ở đây

1. Khai báo con trỏ Q: TRO Q;

2. Cho Q trỏ vào node đầu tiên:
   Q = L;

3. Chuyển L xuống node thứ 2:
   L = L->next;

4. Xóa node trỏ bởi con trỏ Q:
   delete Q;



5.4.4.8. Xóa nút đầu tiên trong danh sách

```
void First_Delete(TRO &L)
{
        TRO Q;
        Q = L;
        L = L->next;
        delete Q;
}
```

5.4.4.9. Xóa nút đứng sau nút trỏ bởi con trỏ P

1. Khai báo con trỏ Q: TRO Q;

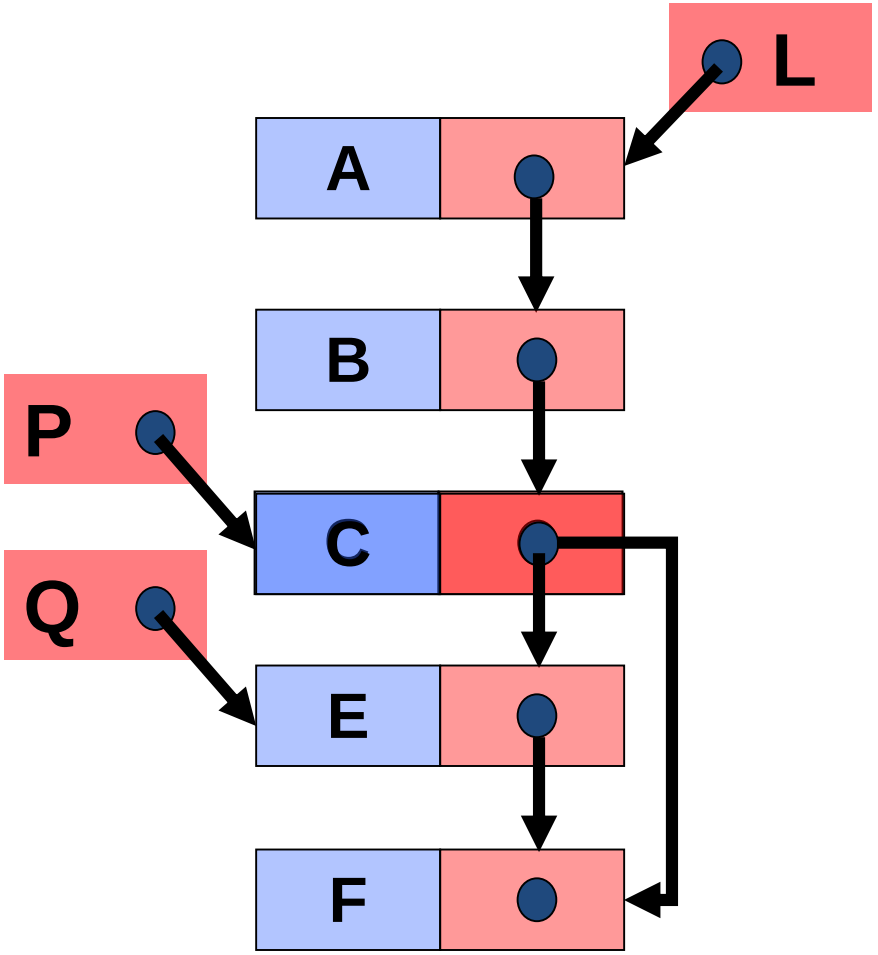
2. Cho Q trỏ vào node ở sau node trỏ bởi P: Q = P->next;

3. next của P trỏ vào node sau node trỏ bởi Q:

P->next = Q->next;

4. Xóa node trỏ bởi con trỏ Q:

delete Q;



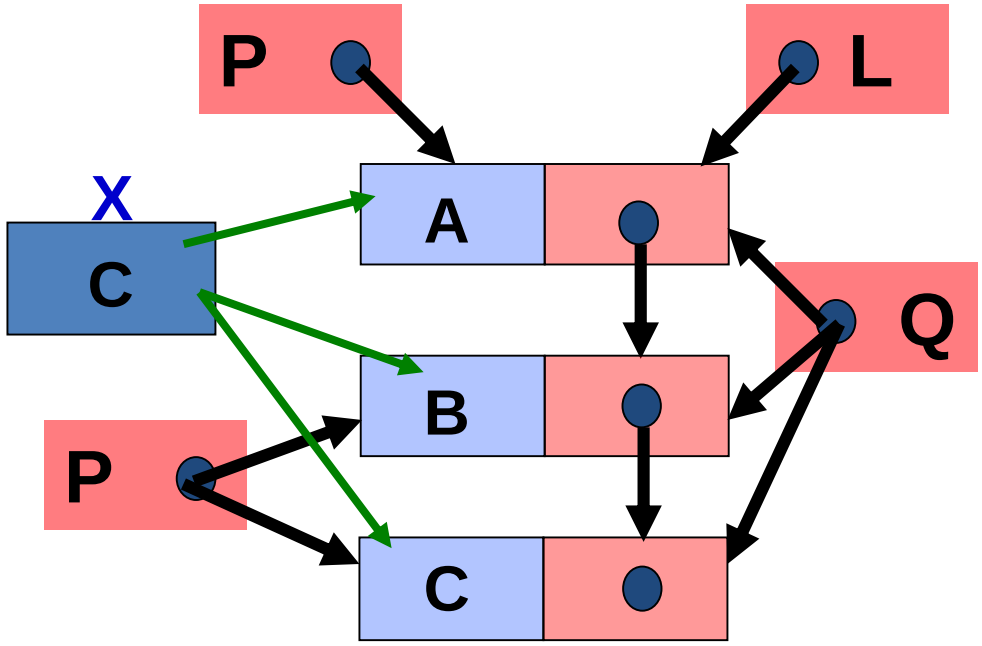
5.4.4.9. Xóa nút đứng sau nút trỏ bởi con trỏ P

```
void after_Delete(TRO &L, TRO P)
{
        TRO Q;
        Q = P->next;
        P->next = Q->next;
        delete Q;
}
```

5.4.5. Tạo một danh sách mới

Xuất phát từ một danh sách rỗng:
creat(L);

1. Khai báo 2 con trỏ P, Q và biến X: TRO P, Q; Item X;
2. Nhập dữ liệu cho biến X;
3. Cấp phát bộ nhớ cho con trỏ P và đưa dữ liệu vào chỗ nhớ đó, đồng thời P->next=NULL;
4. Nếu L=NULL thì L trỏ vào P Ngược lại next của node trỏ bởi Q trỏ vào node mới
5. Cho Q trỏ vào node mới
6. Nếu thỏa mãn điều kiện nhập tiếp thì lặp lại bước 2, ngược lại kết thúc



5.4.5. Tạo một danh sách mới

- **Hàm tạo mới danh sách**

```
void input_List(TRO &L)
{
        TRO P, Q; Item X; char tieptuc;      //Bước 1
        creat(L);
        do{
                input(X);                    //Bước 2
                P = new Node;
                P->infor = X; P->next = NULL;      //Bước 3
                if (L==NULL) L = P;                //Bước 4
                else  Q->next = P;
                Q = P;                       //Bước 5
                cout<<"Co nhap nua khong(C/K)?:"; cin>>tieptuc;
        }while (toupper(tieptuc) == 'C');          //Bước 6
}
```

Bài tập 1

1. **Chương trình quản lý sinh viên (mã SV, họ tên, năm sinh, điểm tổng kết) bằng danh sách nối đơn với các chức năng**
   - **Tạo mới danh sách**
   - **Hiển thị danh sách**
   - **Xác định chiều dài danh sách**
   - **Tìm kiếm sinh viên theo mã và hiển thị thông tin của sinh viên nếu tìm thấy**

Bài tập 2

1. **Chương trình quản lý sinh viên (mã SV, họ tên, năm sinh, điểm tổng kết) bằng danh sách nối đơn với các chức năng**
   - **Tạo mới danh sách**
   - **Hiển thị danh sách**
   - **Hiển thị danh sách sinh viên có điểm tổng kết từ 6.5 trở lên**
   - **Chèn một sinh viên mới vào danh sách theo vị trí k (k nhập từ bàn phím)**

Bài tập 3

1. **Chương trình quản lý sinh viên (mã SV, họ tên, năm sinh, điểm tổng kết) bằng danh sách nối đơn với các chức năng**
   - **Tạo mới danh sách 7 phần tử**
   - **Hiển thị danh sách sinh viên sinh năm 1998**
   - **Xóa phần tử đầu tiên trong danh sách, hiển thị lại danh sách**
   - **Xóa phần tử thứ 5 trong danh sách hiển thị lại danh sách.**
   - **Xóa sinh viên khi biết mã (nhập từ bàn phím)**

Bài tập 4

1. **Chương trình quản lý sinh viên (mã SV, họ tên, năm sinh, điểm tổng kết) bằng danh sách nối đơn với các chức năng**
   - **Tạo mới danh sách 6 phần tử**
   - **Hiển thị danh sách**
   - **Thêm một phần tử vào đầu danh sách, hiển thị lại danh sách**
   - **Thêm một phần tử vào cuối danh sách, hiển thị lại danh sách**
   - **Thêm một phần tử vào vị trí thứ 5 trong danh sách, hiển thị lại danh sách.**
   - **Sắp xếp danh sách theo chiều tăng dần của điểm tổng kết.**

5.5. Danh sách móc nối vòng

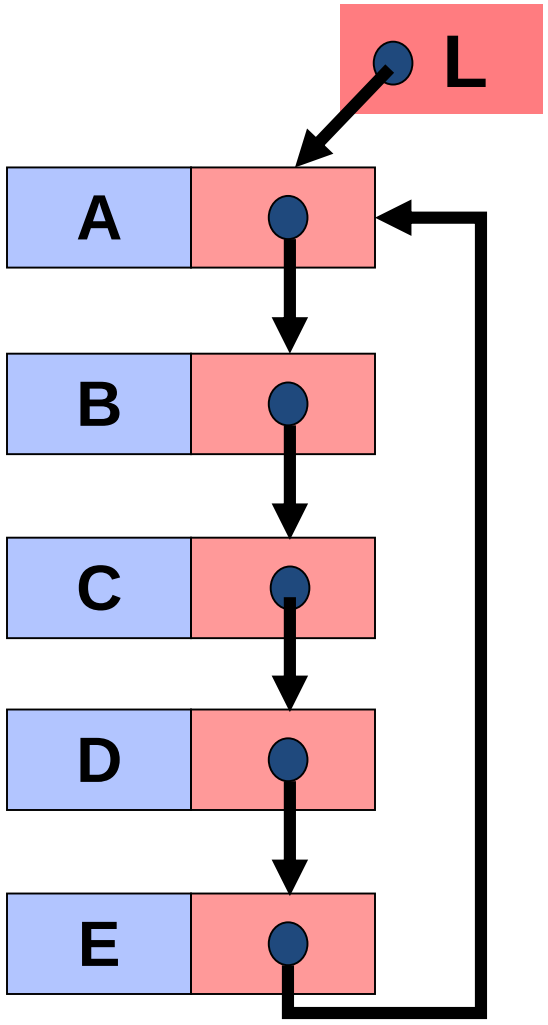
Là một danh sách nối đơn

Có next của node cuối cùng trỏ vào node đầu tiên

DS nối vòng có ưu điểm là xuất phát từ một vị trí bất kỳ có thể duyệt hết danh sách

Về cấu trúc dữ liệu và các phép toán tương tự như DS nối đơn

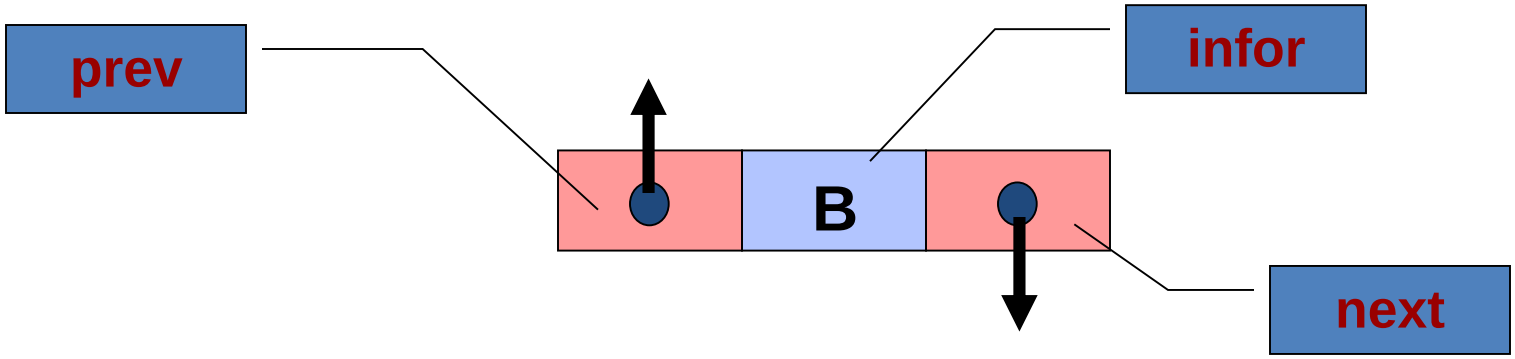
Sinh viên tự nghiên cứu trong tài liệu



5.6. Danh sách móc nối 2 chiều

Còn gọi là danh sách nối đôi.

Là một danh sách móc nối mà mỗi node có ba thành phần



Thành phần infor chứa dữ liệu

Con trỏ next trỏ vào node đứng sau

Con trỏ prev trỏ vào node đứng trước

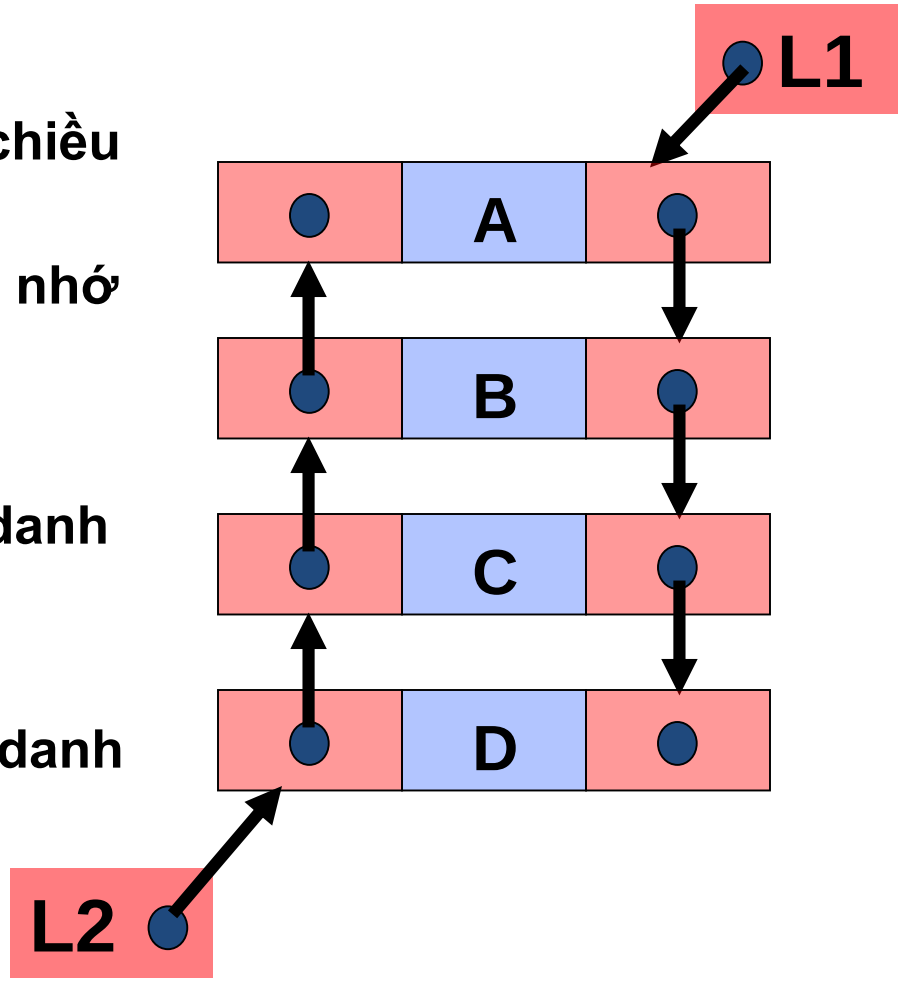
5.6. Danh sách móc nối 2 chiều

Hình ảnh danh sách móc nối hai chiều

Để quản lý danh sách trong bộ nhớ người ta dùng hai con trỏ

Con trỏ L1 trỏ vào node đầu của danh sách

Con trỏ L2 trỏ vào node cuối của danh sách



5.6.1. Khai báo cấu trúc dữ liệu

➢ Khai báo Cấu trúc dữ liệu MẪU

Khai báo kiểu dữ liệu phần tử

```
struct      Item {
            Các thành phần dữ liệu;
};
```

Khai báo kiểu dữ liệu Node

```
struct Node {
            Item infor;
            Node *next;
            Node *prev;
};
```

Khai báo kiểu con trỏ trỏ vào Node

```
typedef Node * TRO;
```

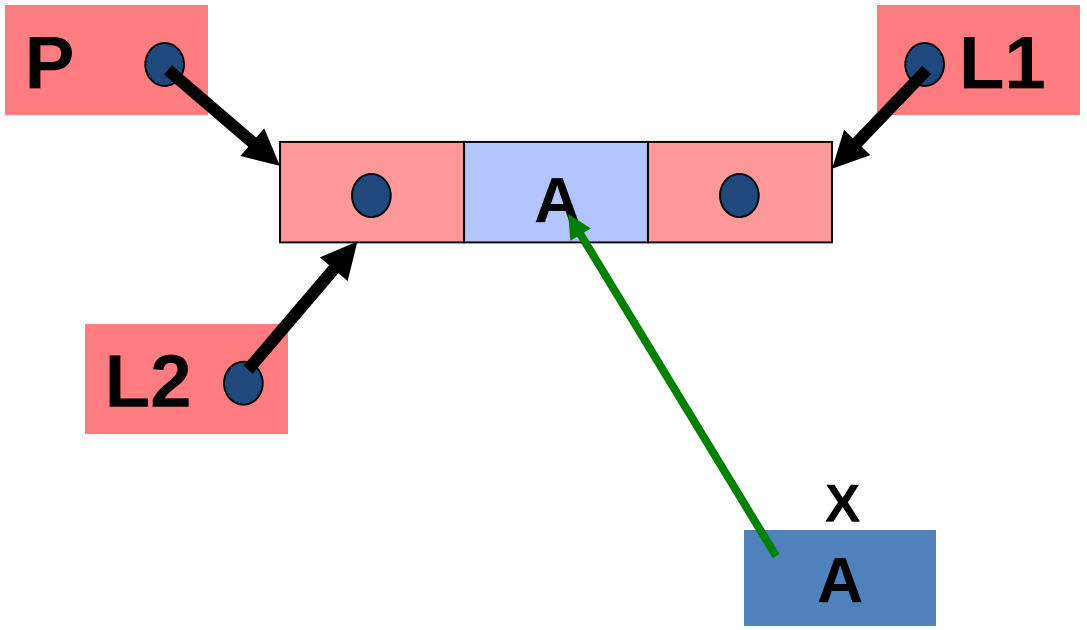
KB con trỏ trỏ vào Node đầu tiên và node cuối

```
TRO L1, L2;
```

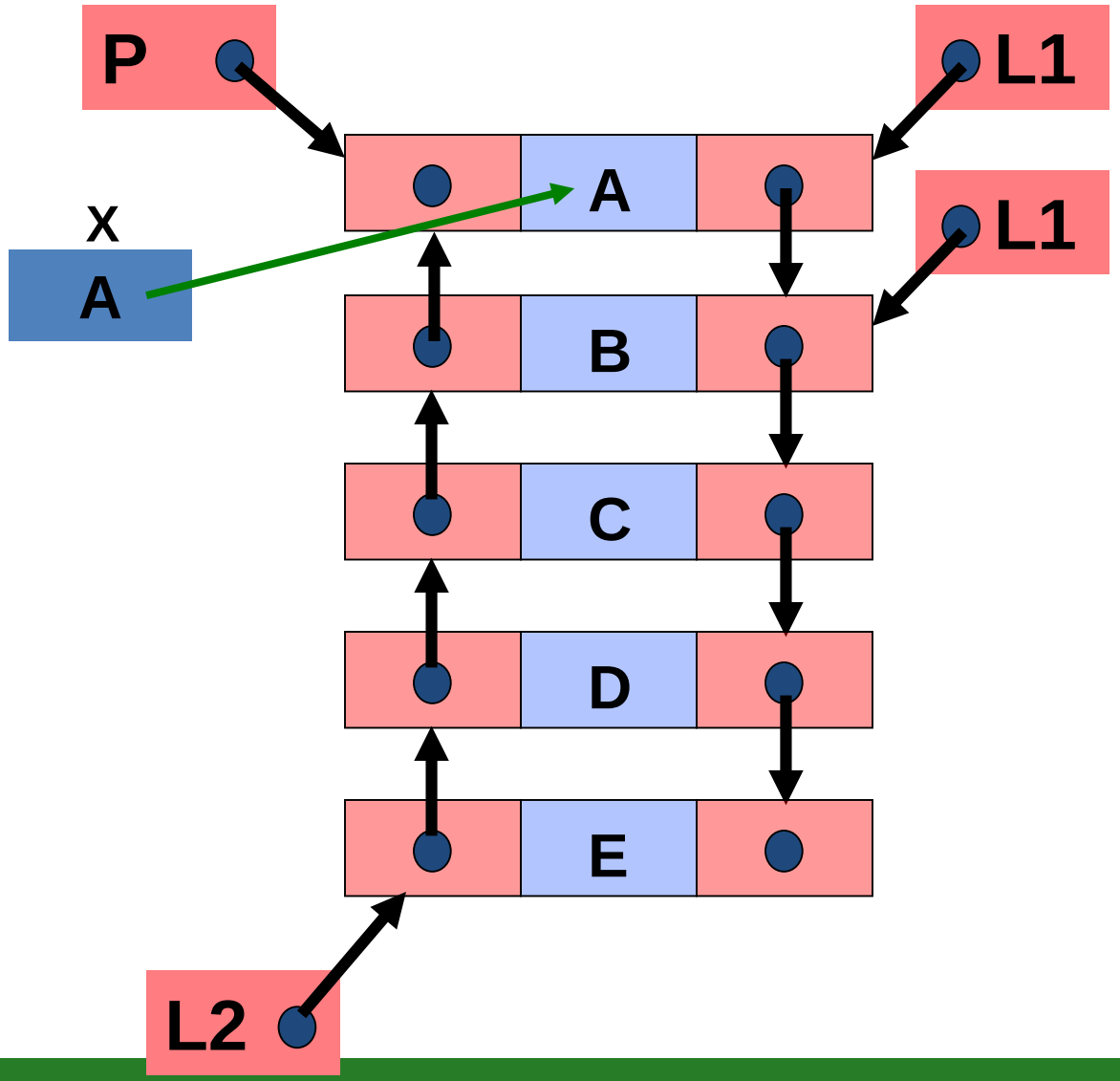
L1/L2=NULL -> ds L rỗng

5.6.2. Chèn nút mới vào đầu danh sách

Trường hợp danh sách rỗng



5.6.2. Chèn nút mới vào đầu danh sách



5.6.2. Chèn nút mới vào đầu danh sách

- **Hàm bổ sung node mới vào đầu danh sách**

```
void first_Add(TRO &L1, TRO &L2, Item X)
{
        TRO P;              //Khai báo con trỏ P
        P = new Node;
        P->infor = X;
        P->prev = NULL;
        P->next = L1;
        if (L2==NULL)               //Trường hợp danh sách rỗng
                L2 = P;
        else
                L1->prev = P;
         L1 = P;
}
```

Bổ sung node mới vào đầu danh sách

Các hàm chèn vào cuối, chèn vào vị trí nào đó,… xoá một node, đếm số lượng node,….

Tự ngâm kứu! ^^