Cấu trúc dữ liệu

CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN (BASIC ABSTRACT DATA TYPES)

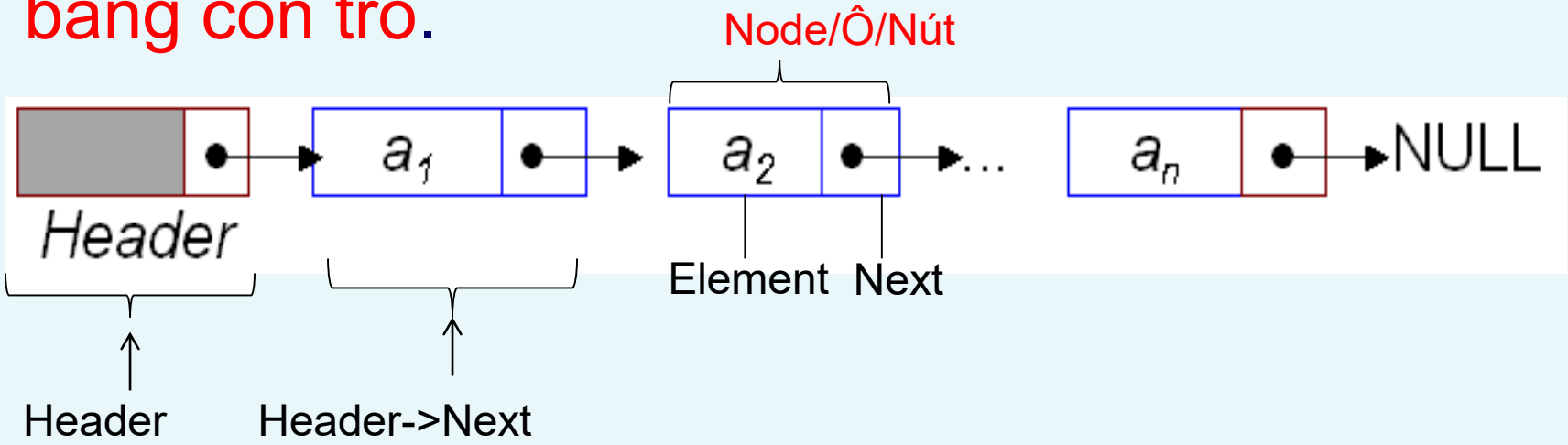
Bộ môn Công Nghệ Phần Mềm

THỰC HÀNH - DANH SÁCH LIÊN KẾT

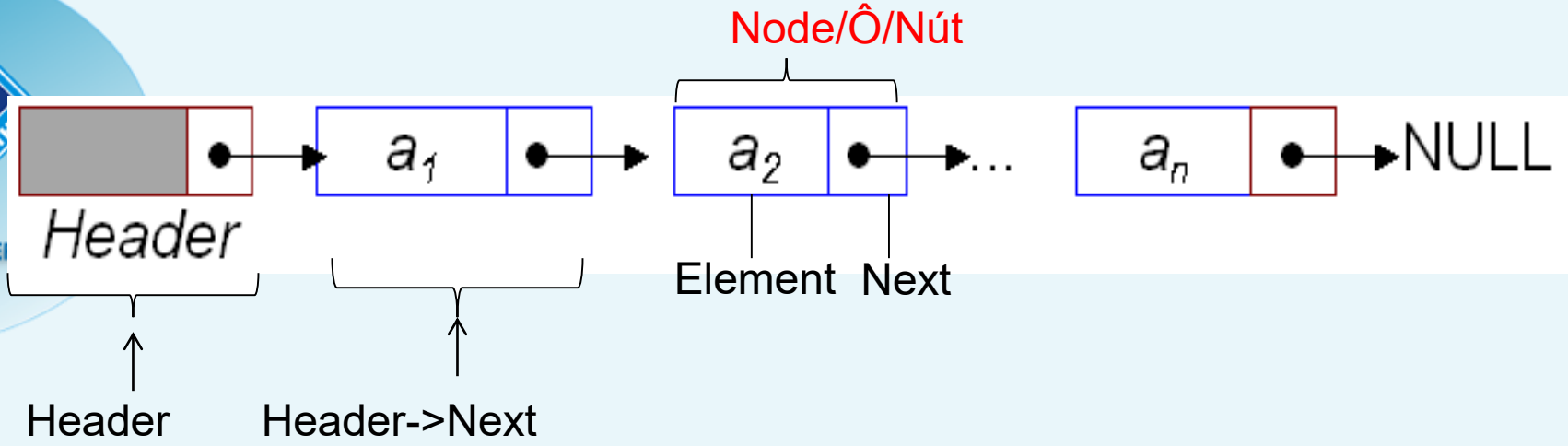
- **Cài đặt** danh sách bằng con trỏ (danh sách liên kết)
  - Tổ chức lưu trữ: **cấu trúc dữ liệu** (*khai báo dữ liệu*).
  - Viết **chương trình con** thực hiện các phép toán (*khai báo phép toán*).
- **Ứng dụng** kiểu danh sách

MỤC TIÊU

- Nắm vững các kiểu dữ liệu trừu tượng danh sách.
- Cài đặt kiểu dữ liệu trừu tượng danh sách bằng con trỏ.



- Ứng dụng được kiểu dữ liệu trừu tượng danh sách trong bài toán thực tế.



- Cài đặt kiểu dữ liệu trừu tượng danh sách bằng con trỏ.

Các phép toán trên danh sách liên kết các số nguyên — Element là số nguyên

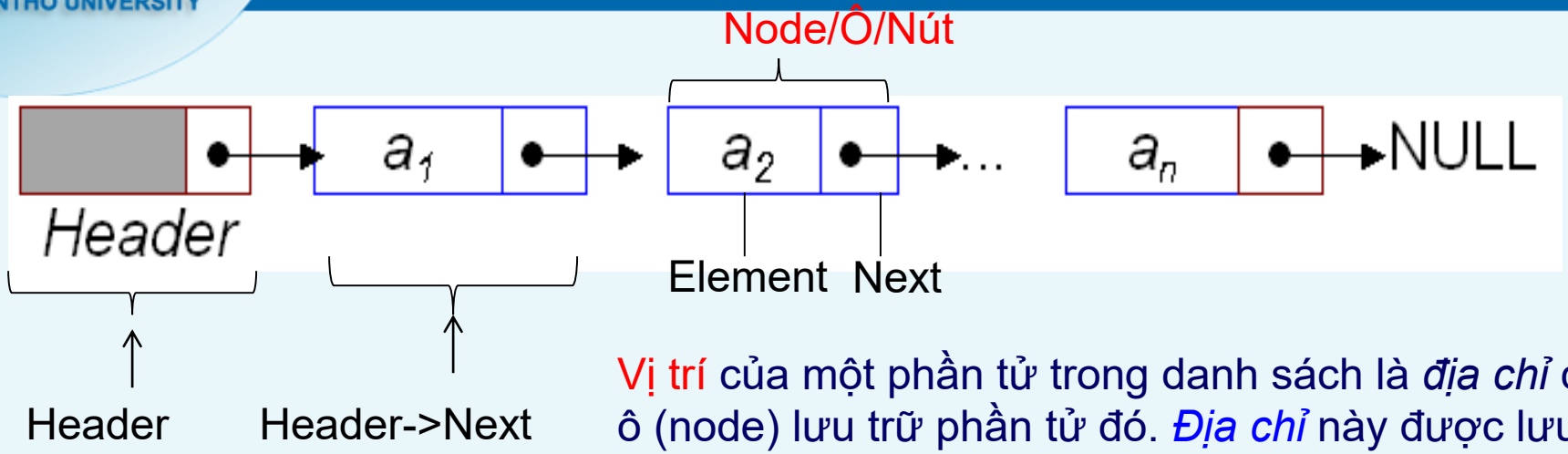
Danh sách sinh viên — Element là cấu trúc SinhVien

Đa thức — Element là cấu trúc DonThuc

- Ứng dụng được kiểu dữ liệu trừu tượng danh sách trong bài toán thực tế.

Áp dụng danh sách liên kết các số nguyên

CÀI ĐẶT DANH SÁCH BẰNG CON TRỎ

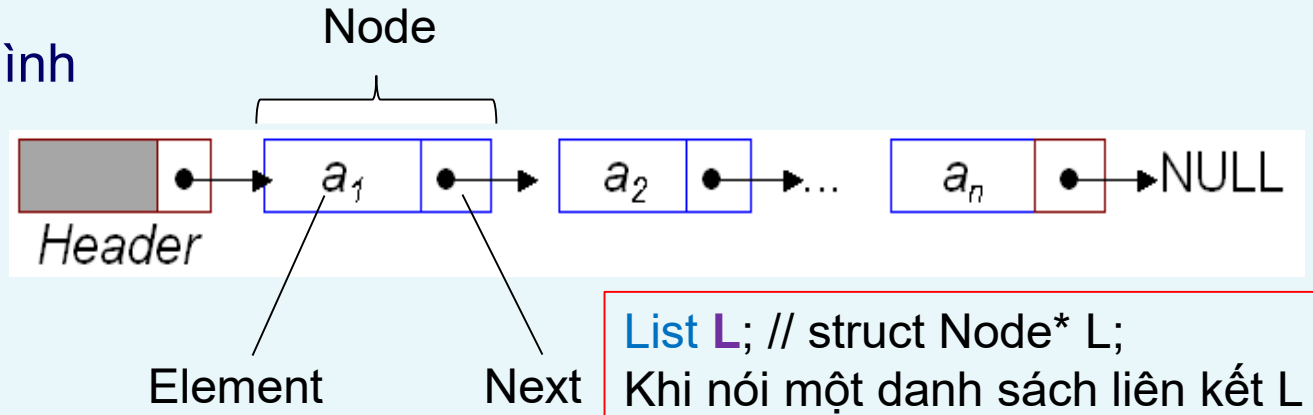


Vị trí của một phần tử trong danh sách là *địa chỉ* của ô (node) lưu trữ phần tử đó. *Địa chỉ* này được lưu trong trường Next của ô đứng trước.

Phần tử	Giá trị	Địa chỉ
1	a_1	Chứa trong trường Next của ô Header
2	a_2	Chứa trong trường Next của phần tử 1 (ô lưu a_1)
…	…	…
n	a_n	Chứa trong trường Next của phần tử n-1 (ô lưu a_{n-1})
Sau phần tử cuối cùng	Không xác định	Chứa trong trường Next của phần tử n (ô lưu a_n) và có giá trị NULL

CÀI ĐẶT DANH SÁCH BẰNG CON TRỎ

- Mô hình



List L; // struct Node* L;
Khi nói một danh sách liên kết L thì **L chính là địa chỉ của nút (ô) đầu tiên (ô Header)**.

- Khai báo

```
typedef <DataType> ElementType; //kiểu của phần tử trong danh sách
struct Node{
    ElementType Element; //Chứa nội dung của phần tử
    struct Node *Next;   //Con trỏ chỉ đến phần tử kế tiếp
};
typedef struct Node* Position;  //Kiểu vị trí
typedef Position List;          //Kiểu danh sách
```

CÁC PHÉP TOÁN TRÊN DANH SÁCH

Tên phép toán	Công dụng
makenullList(L)	Khởi tạo một danh sách L rỗng
emptyList(L)	Kiểm tra xem danh sách L có rỗng hay không
first(L)	Trả về kết quả là vị trí của phần tử đầu danh sách, endList(L) nếu danh sách rỗng
endList(L)	Trả về vị trí *sau phần tử cuối* trong ds L
insertList(x,P,L)	Xen phần tử có nội dung x vào danh sách L tại vị trí P, phép toán không được xác định (thông báo lỗi) nếu vị trí P không tồn tại trong danh sách
deleteList(P,L)	Xóa phần tử tại vị trí P trong danh sách L, phép toán không được xác định (thông báo lỗi) nếu vị trí P không tồn tại trong danh sách

CANTHO UNIVERSITY

CÁC PHÉP TOÁN TRÊN DANH SÁCH

Tên phép toán	Công dụng
retrieve(P,L)	Trả về nội dung phần tử tại vị trí P trong danh sách L, kết quả không xác định (có thể thông báo lỗi) nếu vị trí P không có trong danh sách
locate(x,L)	Trả về kết quả là vị trí của phần tử có nội dung x đầu tiên trong danh sách L, endList(L) nếu không tìm thấy
next(P,L)	Trả về kết quả là vị trí của phần tử đi sau phần tử tại vị trí P trong danh sách L, endList(L) nếu phần tử tại vị trí P là phần tử cuối cùng, kết quả không xác định nếu vị trí P không có trong danh sách
previous(P,L)	Trả về kết quả là vị trí của phần tử đứng trước phần tử tại vị trí P trong danh sách L, kết quả không xác định nếu ví trí P là vị trí đầu tiên hoặc không có trong danh sách L
printList(L)	Hiển thị các phần tử trong danh sách L theo thứ tự xuất hiện

KHỞI TẠO DANH SÁCH RỖNG

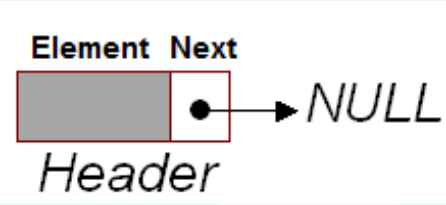
- Khai báo

```
<kiểu kết quả>  Tên hàm (          Tham số hình thức
                    [<kiểu t số> <tham số>]
                    [,<kiểu t số><tham số>][…])
{
      [Khai báo biến cục bộ]
      [Các câu lệnh thực hiện hàm]
      [return [<Biểu thức>];]
}
```

```
typedef <DataType> ElementType;
struct Node{
     ElementType Element;
     struct Node *Next;
};
typedef struct Node* Position;
typedef Position List;
```

- Cấp phát vùng nhớ cho Header
- Cho trường Next của Header trỏ đến NULL

```
struct Node **pL
```




```
void makenullList(List *pL) {
  (*pL)pL =(struct Node*)malloc(sizeof(struct Node));
  (*pL)-> ->Next = NULL;
}
```

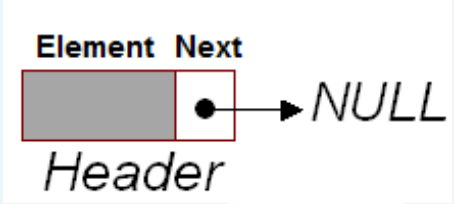
KHỞI TẠO DANH SÁCH RỖNG

- Khai báo

```
<kiểu kết quả>  Tên hàm (          Tham số hình thức
                    [<kiểu t số> <tham số>]
                    [,<kiểu t số><tham số>][…])
{
      [Khai báo biến cục bộ]
      [Các câu lệnh thực hiện hàm]
      [return [<Biểu thức>];]
}
```

```
typedef <DataType> ElementType;
struct Node{
      ElementType Element;
      struct Node *Next;
};
typedef struct Node* Position;
typedef Position List;
```

- Cấp phát vùng nhớ cho Header
- Cho trường Next của Header trỏ đến NULL




```
List makenullList() {
   List L;  // struct Node* L;
   L=(struct Node*)malloc(sizeof(struct Node));
   L->Next= NULL; // L.Next=NULL
   return L;
}
```

LƯU Ý (Định nghĩa hàm, gọi hàm, phương pháp truyền tham số)

```
void makenullList(List *pL){
(*pL)=(struct Node*)
          malloc(sizeof(struct Node));
(*pL)->Next= NULL;
}
```

- Giả sử ta có khai báo biến:

```
List   L1;
List  *L2;
```

Hãy viết lời gọi hàm makenullList() khởi tạo danh sách L1 và danh sách được trỏ bởi con trỏ *L2 rỗng?

```
makenullList(&L1);
 makenullList(L2);
```

```
List makenullList(){
  List L;
  L=(struct Node*)
     malloc(sizeof(struct Node));
  L->Next= NULL;
   return L;
}
```

- Giả sử ta có khai báo biến:

```
List   L1;
List  *L2;
```

Hãy viết lời gọi hàm emptyList() kiểm tra xem danh sách L1 và danh sách được trỏ bởi con trỏ *L2 có rỗng hay không?

```
L1=makenullList();
(*L2)=makenullList();
```

KIỂM TRA DANH SÁCH RỖNG

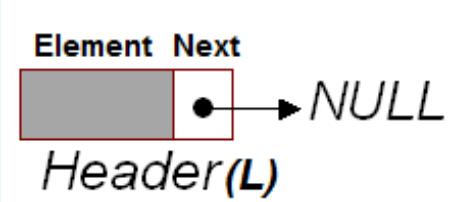
- Khai báo

```
<kiểu kết quả>  Tên hàm (          Tham số hình thức
                  [<kiểu t số> <tham số>]
                  [,<kiểu t số><tham số>][…])
{
      [Khai báo biến cục bộ]
      [Các câu lệnh thực hiện hàm]
      [return [<Biểu thức>];]
}
```

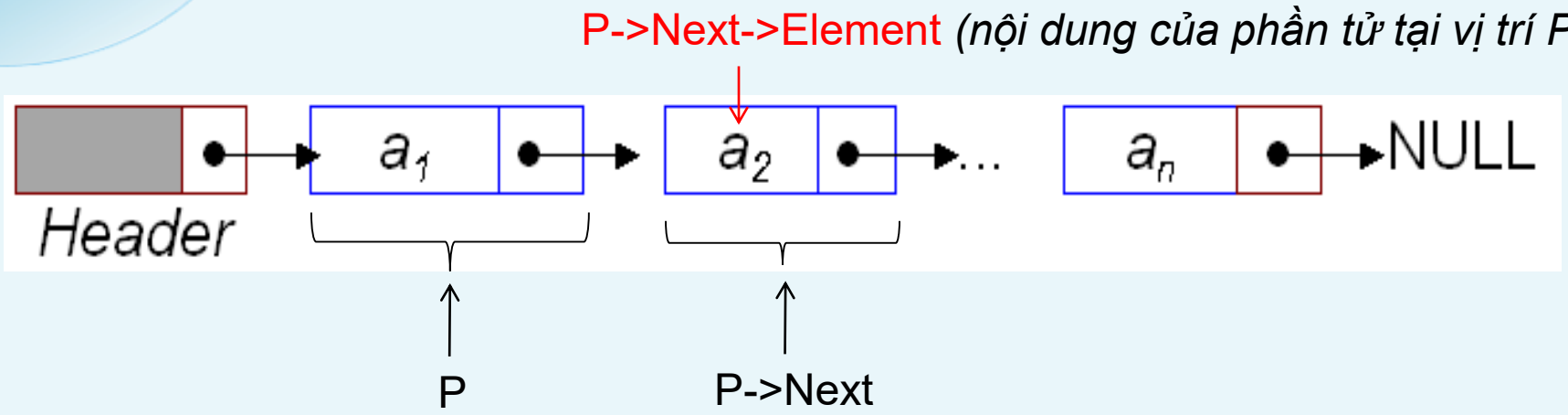
```
typedef <DataType> ElementType;
struct Node{
    ElementType Element;
    struct Node *Next;
};
typedef struct Node* Position;
typedef Position List;
```

- Xem trường Next của ô Header có trỏ đến NULL hay không?

```
int emptyList(List L) { //struct Node* L
    return (L->Next==NULL);
}
```



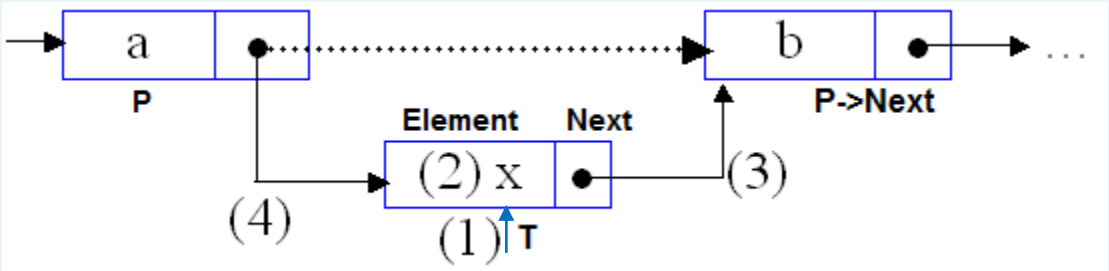
XÁC ĐỊNH NỘI DUNG PHẦN TỬ TẠI VỊ TRÍ P




```
ElementType retrieve(Position P, List L){
  if (P->Next!=NULL)
     return P->Next->Element;
}
```

So sánh với cách cài đặt bằng mảng

XEN MỘT PHẦN TỬ VÀO DANH SÁCH

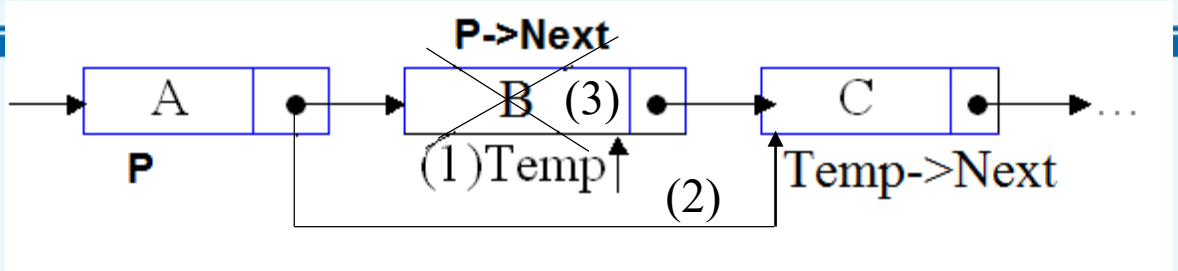


- Để xen phần tử x vào vị trí P của L, ta làm như sau:
  - Cấp phát 1 ô mới để lưu trữ phần tử x.
  - Nối kết lại các con trỏ để đưa ô mới này vào vị trí P.

```
void insertList(ElementType x,Position P, List *pL){
      Position T; //struct Node*  T
      T=(struct Node*)malloc(sizeof(struct Node));
      T->Element=x;
      T->Next=P->Next;
      P->Next=T;
}
```

Cho nhận xét đánh giá độ phức tạp so với cách dùng mảng

XÓA MỘT PHẦN TỬ KHỎI DANH SÁCH



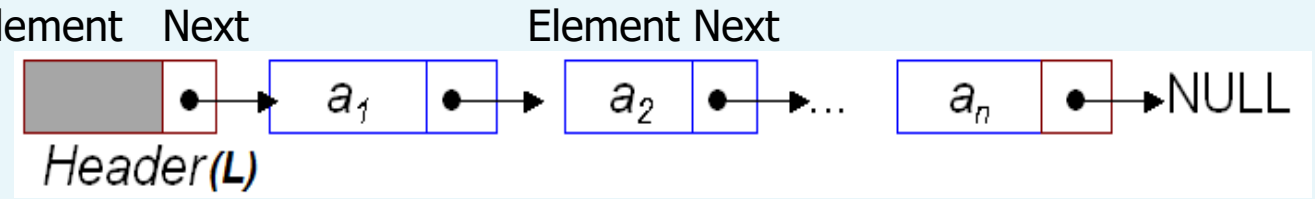
=>Muốn xóa phần tử ở vị trí P trong danh sách ta cần nối kết lại các con trỏ bằng cách cho P trỏ tới phần tử đứng sau phần tử thứ P.

```
void deleteList(Position P, List *pL){
   Position Temp;
    if (P->Next!=NULL){
       //Giữ ô chứa phần tử bị xoá để thu hồi vùng nhớ
       Temp=P->Next;
       //Nối kết con trỏ trỏ tới phần tử kế tiếp
       P->Next=Temp->Next;
       //Thu hồi vùng nhớ
       free(Temp);
     }
}
```

Cho nhận xét đánh giá độ phức tạp so với cách dùng mảng

XÁC ĐỊNH VỊ TRÍ PHẦN TỬ

- Vị trí phần tử đầu tiên

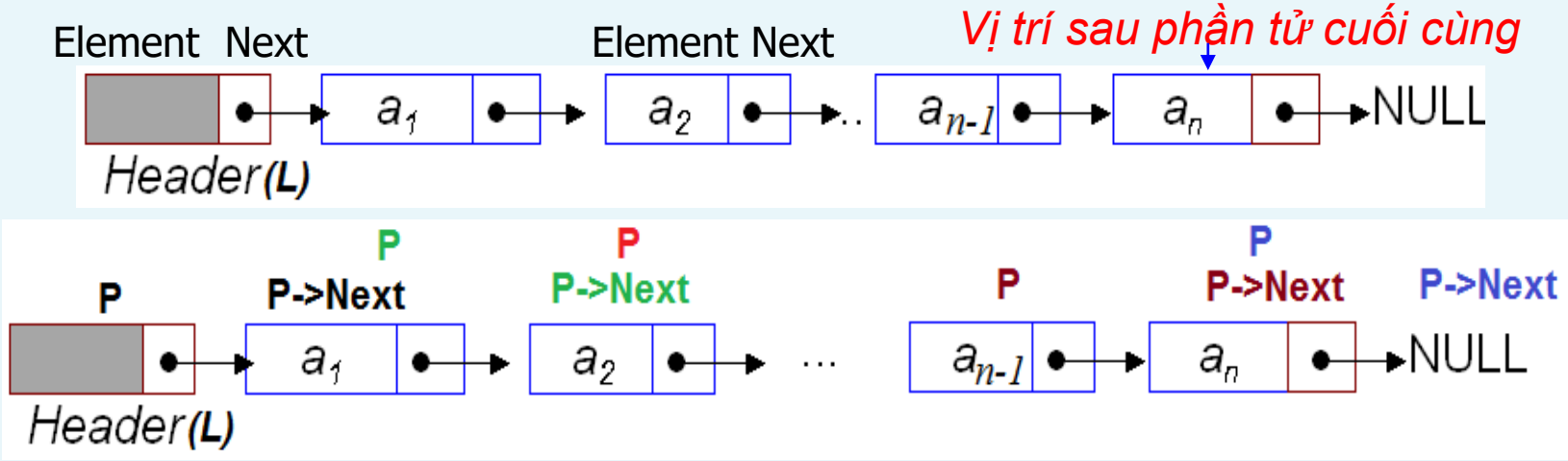



```
Position first(List L) {
        return L;
}
```

Cho nhận xét đánh giá độ phức tạp so với cách dùng mảng

XÁC ĐỊNH VỊ TRÍ PHẦN TỬ

- Vị trí sau phần tử cuối cùng

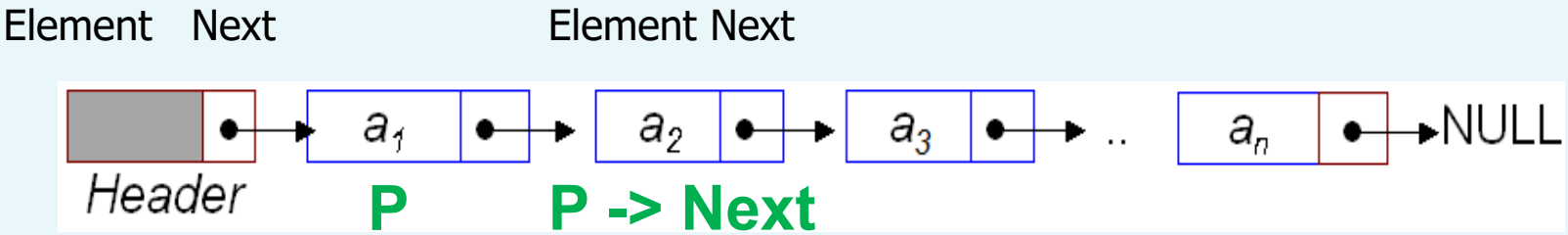



```
Position endList(List L) {
    Position P;
    P=L; // P=first(L);
    while (P->Next!=NULL)
        P=P->Next;
    return P;
}
```

Cho nhận xét đánh giá độ phức tạp so với cách dùng mảng

XÁC ĐỊNH VỊ TRÍ PHẦN TỬ

- Vị trí phần tử kế tiếp

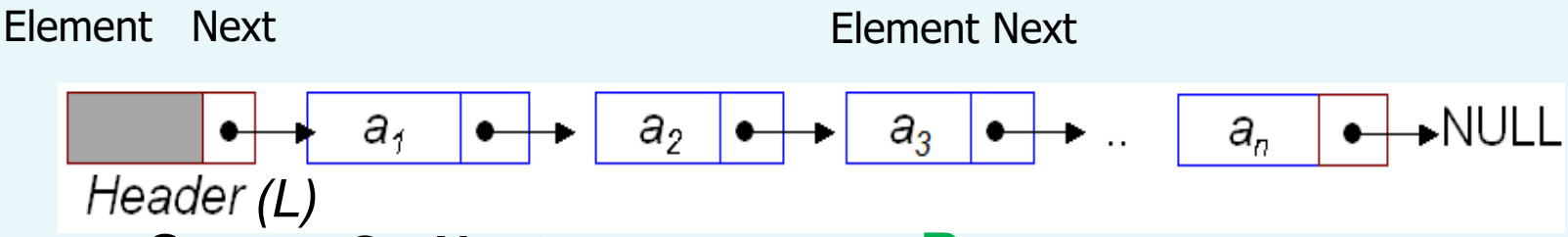



```
Position next(Position P, List L){
        return P->Next; // P=P+1
}
```

Cho nhận xét đánh giá độ phức tạp so với cách dùng mảng

XÁC ĐỊNH VỊ TRÍ PHẦN TỬ

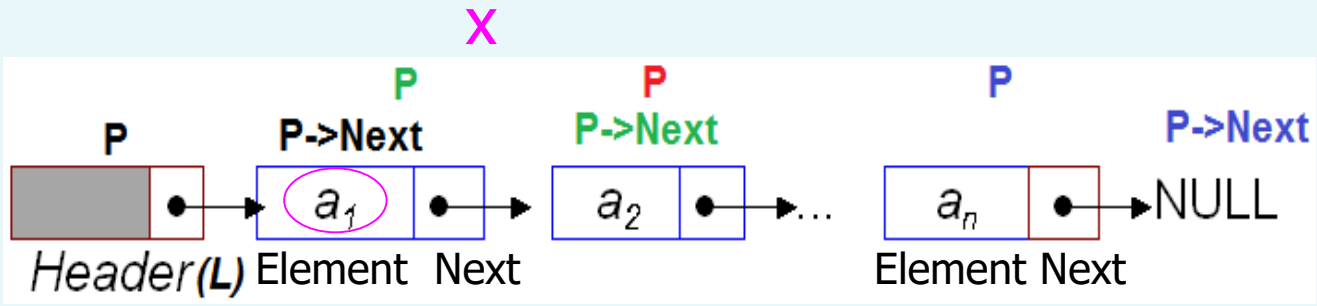
- Vị trí phần tử trước đó




```
Position previous(Position P, List L){
    Position Q=L;
    while(Q->Next!=P)
        Q=Q->Next;
    return Q;
}
```

Cho nhận xét đánh giá độ phức tạp so với cách dùng mảng

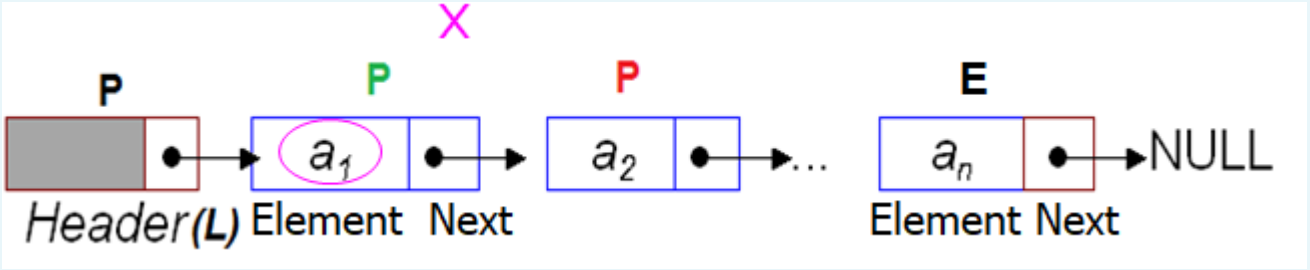
TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH




```
Position locate(ElementType x, List L){
  Position P;
  int Found = 0;
  P = L;
  while ((P->Next != NULL) && (Found == 0))
       if (P->Next->Element == x) Found = 1;
       else P = P->Next;
  return P;
}
```

Gắn với cách tổ chức lưu trữ danh sách bằng con trỏ, phụ thuộc chi tiết bên trong

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH




```
Position locate(ElementType x, List L){
  Position P,E;  int Found=0;
  P = first(L);
  E = endList(L);
  while ((P!=E) && (Found == 0))
     if (retrieve(P,L) == x)
           Found = 1;
     else P = next(P,L);
  return P;  // endList(L)
}
```

!Found

Tái sử dụng, tổng quát, **không phụ thuộc vào chi tiết cài đặt bên trong (mảng/con trỏ)**

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH

```
Position locate(ElementType x, List L){
  Position P;
  P = L;
  while (P->Next != NULL)
       if (P->Next->Element == X)
            return P; // break;
       else P = P->Next;
  return P;
}
```

Cài đặt lại hàm Locate bằng cách loại bỏ biết Found.

```
Position locate(ElementType x, List L){
   Position P,E;
   P = first(L);
   E = endList(L);
   while (P!=E)
       if (retrieve(P,L) == x)
            return P;
       else P = next(P,L);
   return P;  // endList(L)
}
```

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH

```
Position locate1(ElementType X, List L){
   Position P, E;
   P = first(L);      E=endList(L);
   while (P != E)
        if (retrieve(P,L) == X)
                return P;
        else P = next(P,L);
   return P;   // endList(L)
}
```

?

```
Position locate2(ElementType X, List L){
   Position P;
   P = first(L);
   while (P!= endList(L))
    if (retrieve(P,L) == X) return P;
    else P = next(P,L);
   return P;
}
```

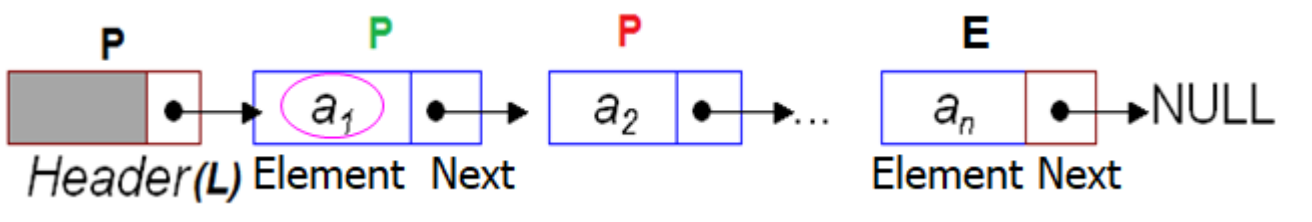
ỨNG DỤNG KIỂU DANH SÁCH

- Cài đặt hàm `myLocate()` trả về vị trí của lần xuất hiện thứ i của x trong L. Nếu không tìm thấy thì trả về vị trí sau phần tử cuối cùng.
- Cài đặt hàm `printList()` in một danh sách L ra màn hình.
- Cài đặt hàm `readList()` nhập danh sách từ bàn phím theo 2 cách: có tham số và không có tham số.

ỨNG DỤNG KIỂU DANH SÁCH

```
Position myLocate(ElementType x, int i, List L){
  Position P, E;
  P=first(L);
  E=endList(L);
  int count =0;
  while (P != E && count < i){
     if (retrieve(P,L)== x)
          count++;
     if (count<i)
          P=next(P,L);
  }
  return P;
}
```

ỨNG DỤNG KIỂU DANH SÁCH




```
void printList(List L){
   Position P, E;
   P = first(L);        E=endList(L);
   while (P != E) {
       printf("%d ",retrieve(P,L));
       P = next(P,L);
   }
   printf("\n");
}
```

Tái sử dụng, tổng quát, **không phụ thuộc vào chi tiết cài đặt bên trong**

ỨNG DỤNG KIỂU DANH SÁCH

```
void readList(List *pL) {
   int i,n;
   ElementType x;
   makenullList(pL);
   printf("So phan tu danh sach n= ");
   scanf("%d",&n);
   for(i=1;i<=n;i++) {
      printf("Phan tu thu %d: ",i);
      scanf("%d",&x);
      //insertList(x,endList(*pL),pL);
      insertList(x,first(*pL),pL);
   }
}
```

ỨNG DỤNG KIỂU DANH SÁCH

```
List readList() {
   List L;
   int i,n;
   ElementType x;
   makenullList(&L);
   printf("So phan tu danh sach n= ");
   scanf("%d",&n);
   for(i=1;i<=n;i++) {
       printf("Phan tu thu %d: ",i);
       scanf("%d",&x);
       //insertList(x,endList(L),&L);
       insertList(x,first(L),&L);
   }
   return L;
}
```

Q&A?