Cấu trúc dữ liệu

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

Bộ môn Công Nghệ Phần Mềm

MỤC TIÊU

- Nắm vững các kiểu dữ liệu trừu tượng như: danh sách, ngăn xếp, hàng đợi.
- Cài đặt các kiểu dữ liệu trừu tượng bằng ngôn ngữ lập trình cụ thể.
- Ứng dụng được các kiểu dữ liệu trừu tượng trong bài toán thực tế.

NỘI DUNG

- Kiểu dữ liệu trừu tượng danh sách (LIST)
- Kiểu dữ liệu trừu tượng ngăn xếp (STACK)
- Kiểu dữ liệu trừu tượng hàng đợi (QUEUE)
- *Danh sách liên kết kép (Doubly-Linked Lists)*

KHÁI NIỆM VỀ DANH SÁCH

- Là tập hợp *hữu hạn* các phần tử có *cùng kiểu*. Kiểu chung được gọi là kiểu phần tử (element type).
- Ta thường biểu diễn dạng: $a_1, a_2, a_3, ..., a_n$.
- Nếu
  - n=0: danh sách rỗng.
  - n>0: phần tử đầu tiên là a_1, phần tử cuối cùng là a_n.
- *Độ dài của danh sách*: số phần tử của danh sách.
- Các phần tử trong danh sách có *thứ tự tuyến tính theo vị trí* xuất hiện. Ta nói a_i đứng trước a_{i+1} (i=1..n-1).

CANTHO UNIVERSITY

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
fullList(L)	Kiểm tra xem danh sách L có đầy 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

DANH SÁCH

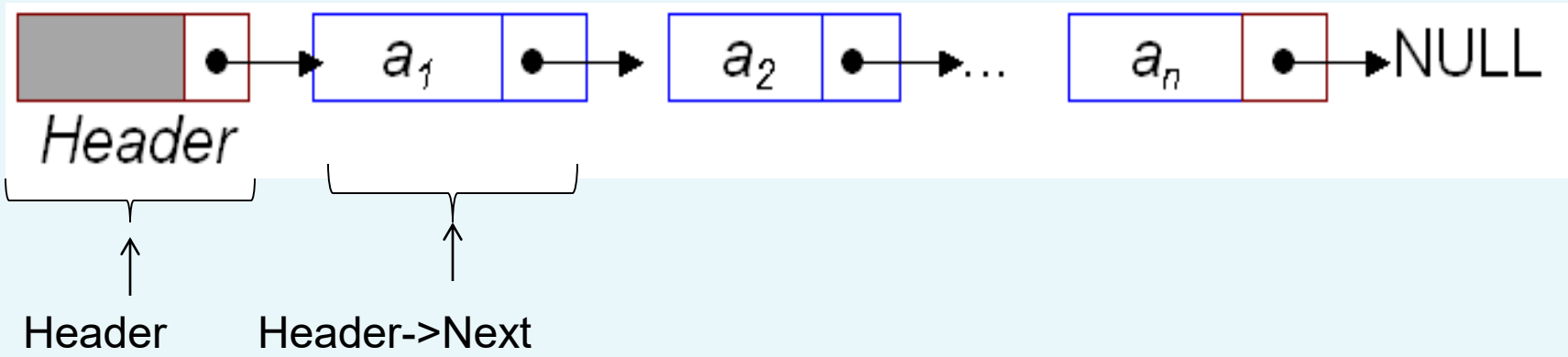
- Khái niệm danh sách
- Các phép toán trên danh sách
- Cài đặt danh sách
  - Dùng mảng (Danh sách ĐẶC)
  - Dùng con trỏ (Danh sách LIÊN KẾT)

Kiểu dữ liệu trừu tượng - Lưu ý

- Cài đặt kiểu dữ liệu trừu tượng:
  - 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*).

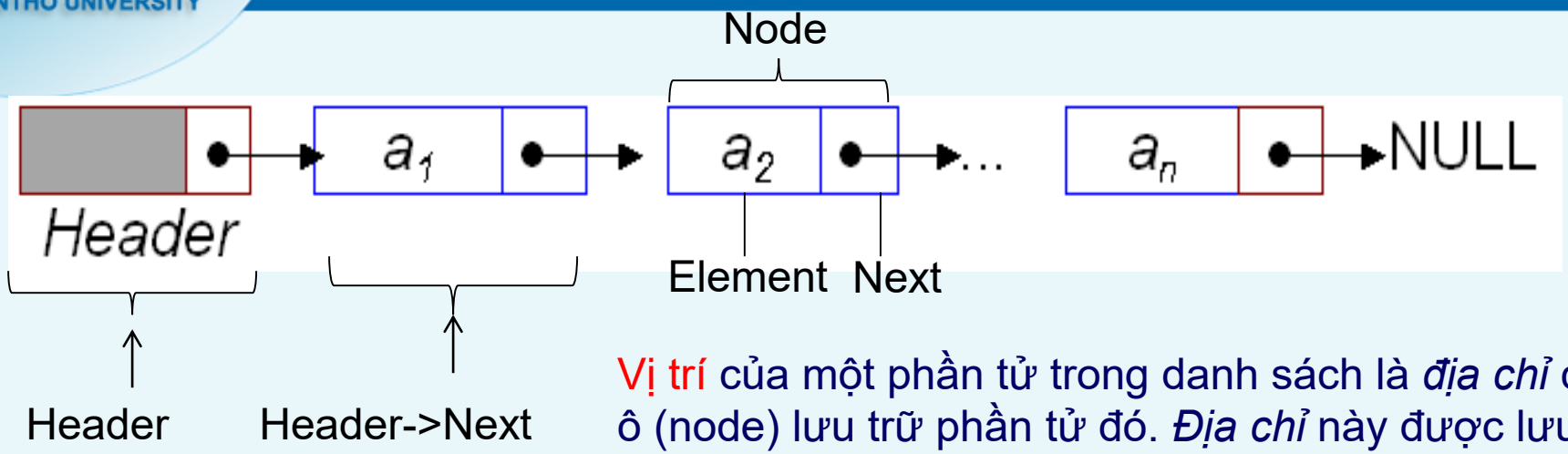
CÀI ĐẶT DANH SÁCH BẰNG CON TRỎ

- Mô hình



- Nối kết các phần tử liên tiếp nhau bằng con trỏ
  - Phần tử a_i trỏ tới phần tử a_{i+1}.
  - Phần tử a_n trỏ tới phần tử đặc biệt NULL.
  - Phần tử Header trỏ tới phần tử đầu tiên a_1.

CÀI ĐẶT DANH SÁCH BẰNG CON TRỎ

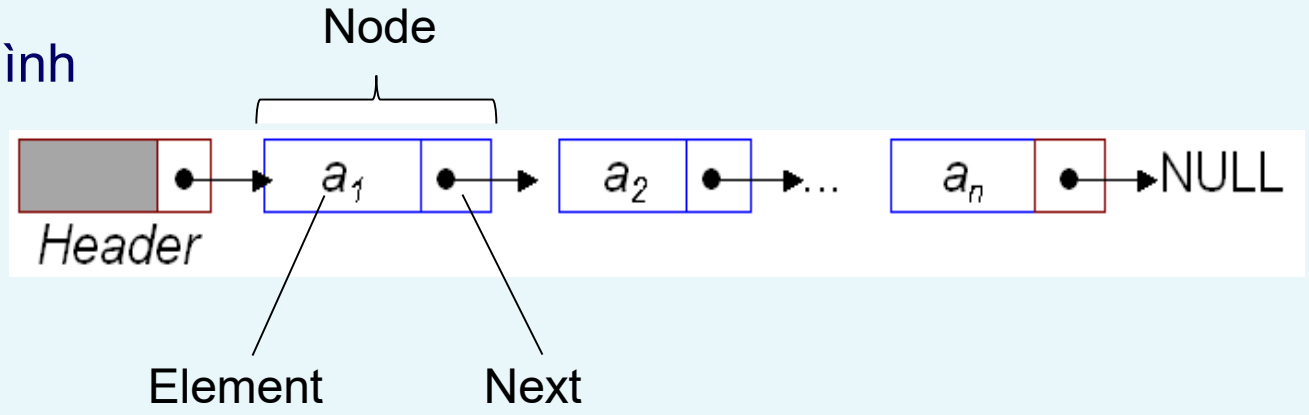


Vị trí của một phần tử trong danh sách là *địa chỉ* của ô (node) lưu trữ phần tử đó. *Địa chỉ* này được lưu trong trường Next của ô đứng trước.

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

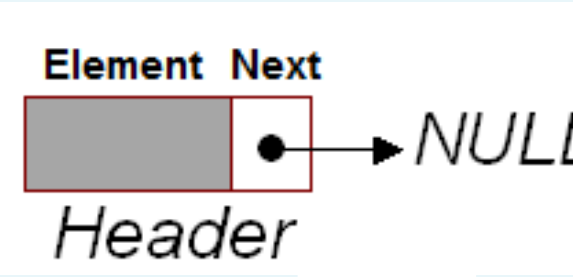
- Mô hình



- 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
```

KHỞI TẠO DANH SÁCH RỖNG

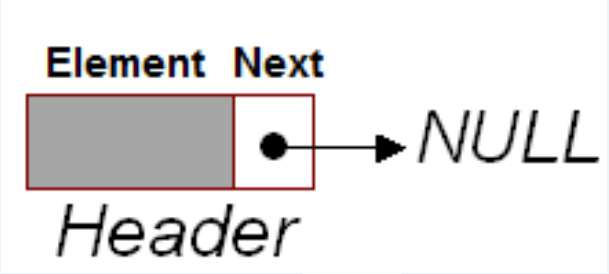



```
<kiểu kết quả>  Tên hàm (          Tham số hình thức
                    [<kiểu t số> <tham số>]
                    [,<kiểu t số><tham số>][…])
{
      [Khai báo biến cục bộ]
      [Các câu lệnh thực hiện hàm]
      [return [<Biểu thức>];]
}
```

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

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

KHỞI TẠO DANH SÁCH RỖNG



- Hàm khởi tạo danh sách rỗng:

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

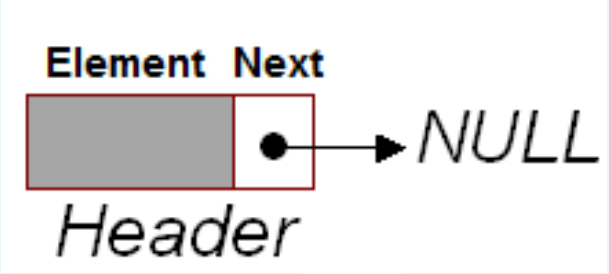
- Giả sử ta có khai báo biến:

```
List   L;
```

```
makenullList(&L);
```

Hãy viết lời gọi hàm makenullList() khởi tạo L rỗng?

KHỞI TẠO DANH SÁCH RỖNG



- Hàm khởi tạo danh sách rỗng:

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

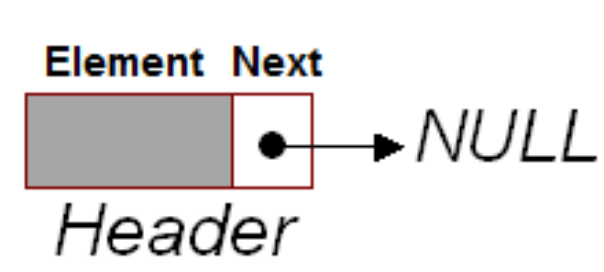
- Giả sử ta có khai báo biến:

```
List    *pL; …
```

```
makenullList(pL);
```

Hãy viết lời gọi hàm makenullList() khởi tạo danh sách được trỏ bởi pL rỗng?

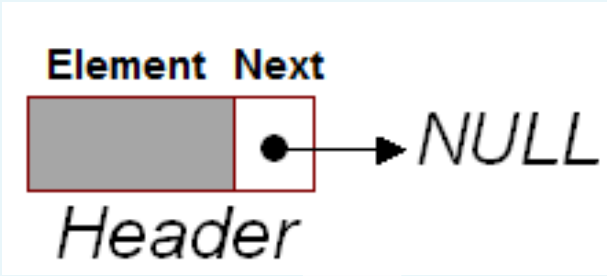
KHỞI TẠO DANH SÁCH RỖNG



- Hãy viết hàm makenullList không có tham số vào và trả về một danh sách rỗng?

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

KHỞI TẠO DANH SÁCH RỖNG



- Hàm khởi tạo danh sách rỗng:

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

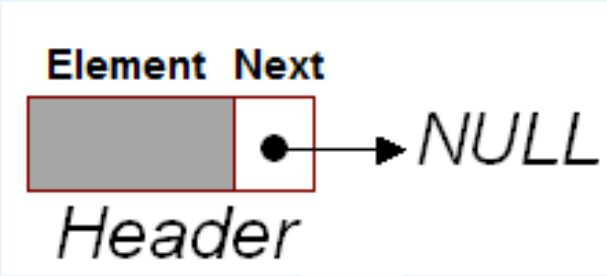
- Giả sử ta có khai báo biến:

```
List   L;
```

```
L=makenullList();
```

Hãy viết lời gọi hàm makenullList() khởi tạo L rỗng?

KHỞI TẠO DANH SÁCH RỖNG



- Hàm khởi tạo danh sách rỗng:

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

- Giả sử ta có khai báo biến:

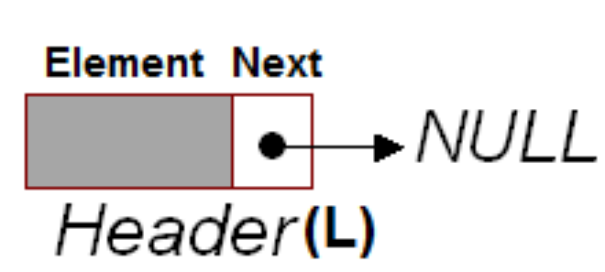
```
List    *pL;…
```

```
(*pL)=makenullList();
```

Hãy viết lời gọi hàm makenullList() khởi tạo danh sách được trỏ bởi pL rỗng?

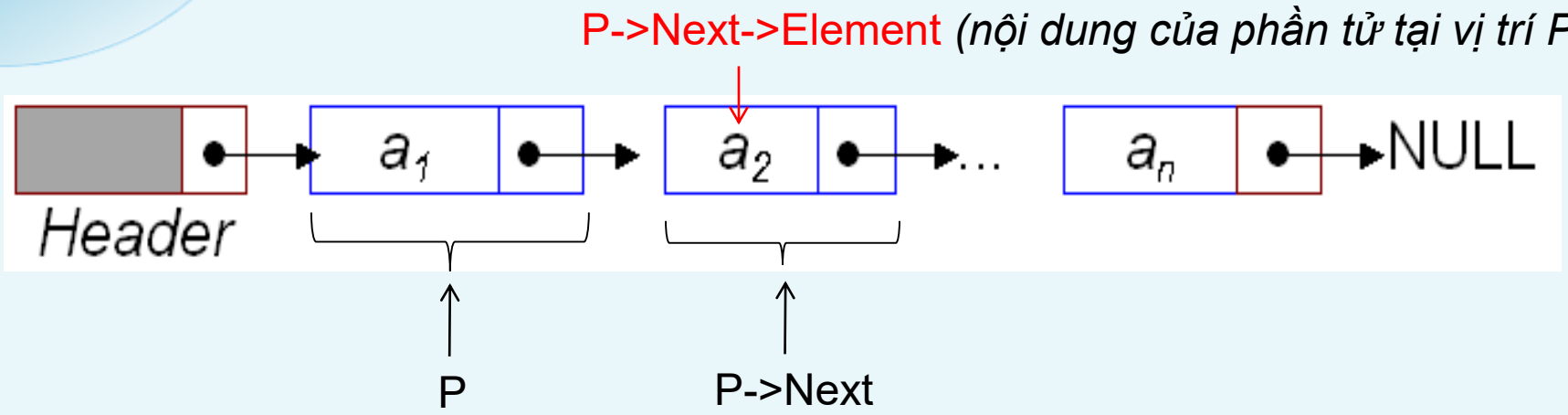
KIỂM TRA DANH SÁCH RỖNG

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




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

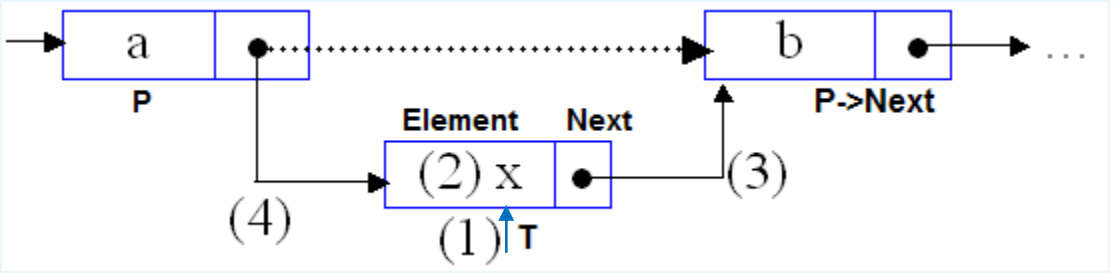
XÁC ĐỊNH NỘI DUNG PHẦN TỬ TẠI VỊ TRÍ P




```
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

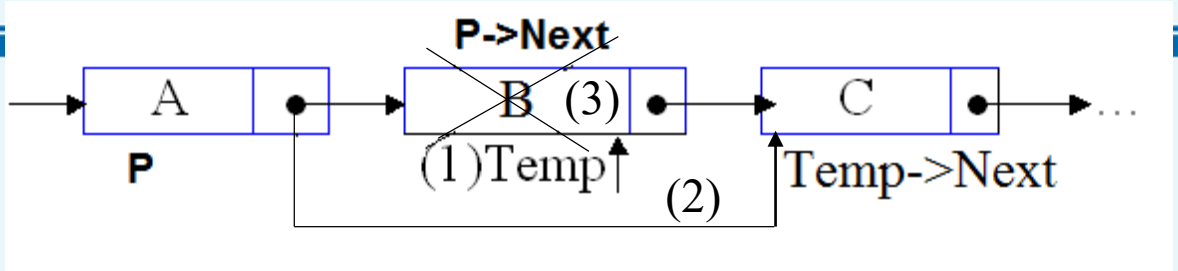


- Để xen phần tử x vào vị trí P của L, ta làm như sau:
  - Cấp phát 1 ô mới để lưu trữ phần tử x.
  - Nối kết lại các con trỏ để đưa ô mới này vào vị trí P.

```
void insertList(ElementType x,Position P, List *pL){
        Position T;
        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



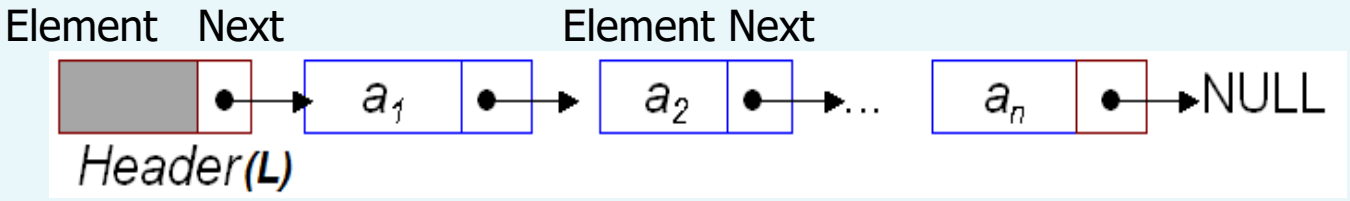
=>Muốn xóa phần tử ở vị trí P trong danh sách ta cần nối kết lại các con trỏ bằng cách cho P trỏ tới phần tử đứng sau phần tử thứ P.

```
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

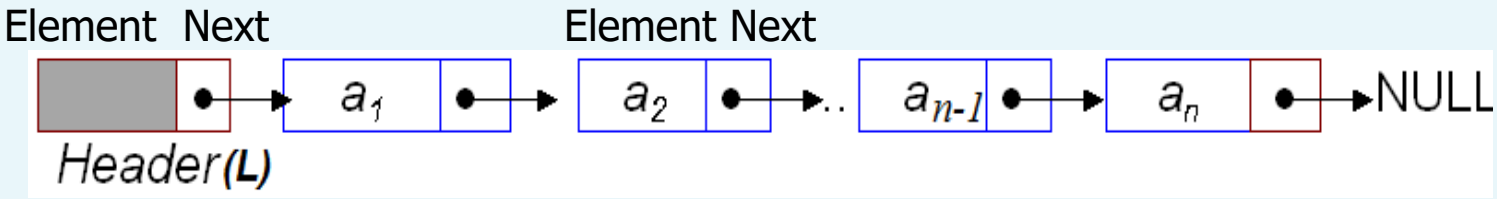


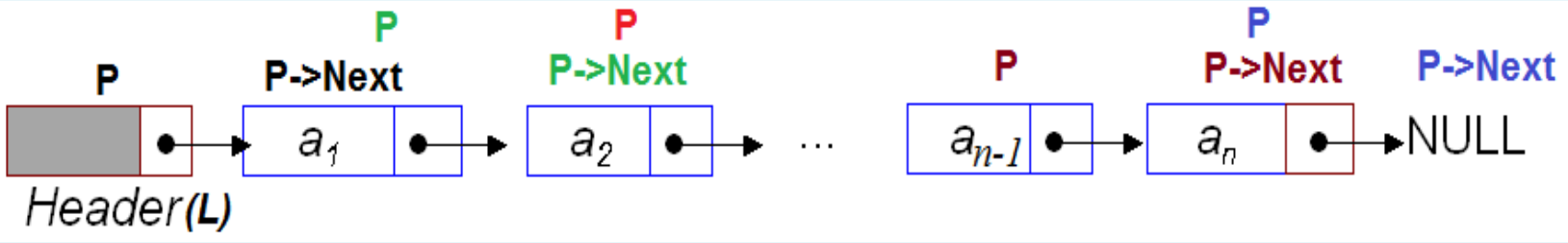

```
Position first(List L) {
        return L;
}
```

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



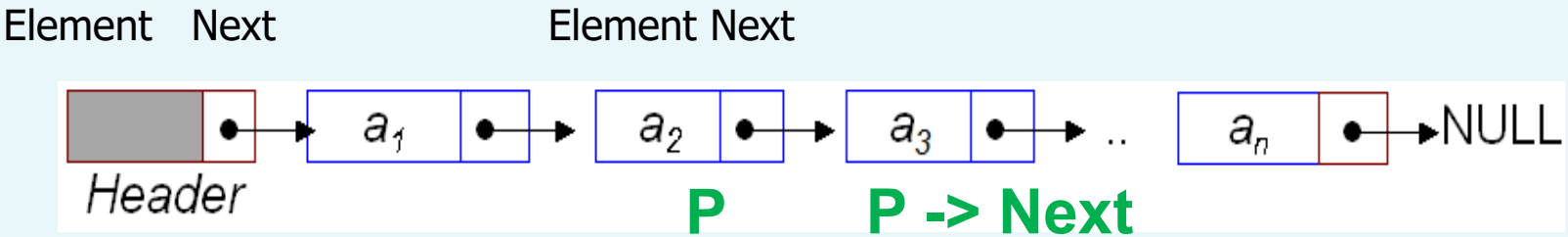



```
Position endList(List L) {
    Position P;
    P=L; // P=first(L);
    while (P->Next!=NULL)
        P=P->Next;
    return P;
}
```

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

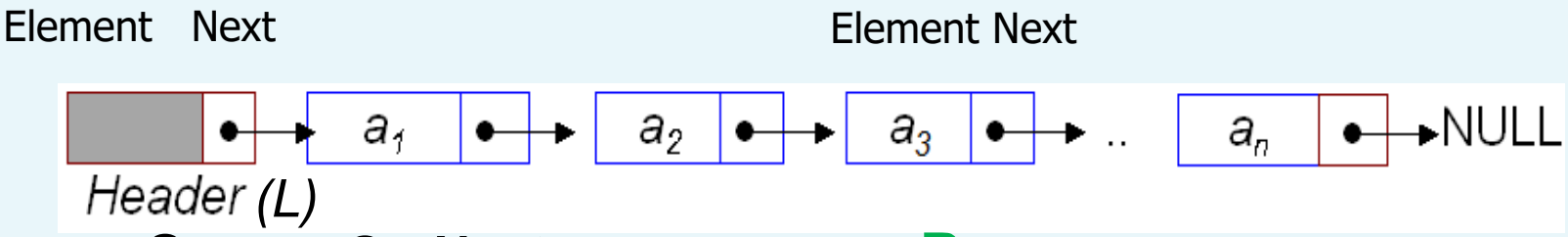



```
Position next(Position P, List L){
        return P->Next;
}
```

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

XÁC ĐỊNH VỊ TRÍ PHẦN TỬ

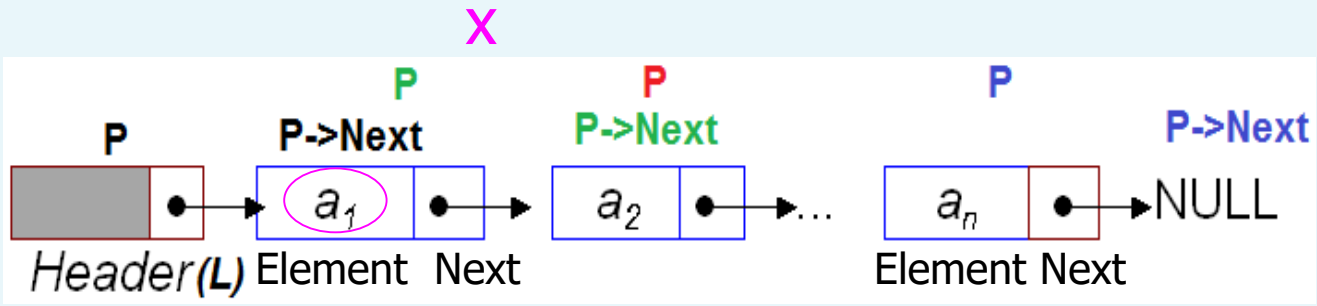
- Vị trí phần tử trước đó




```
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



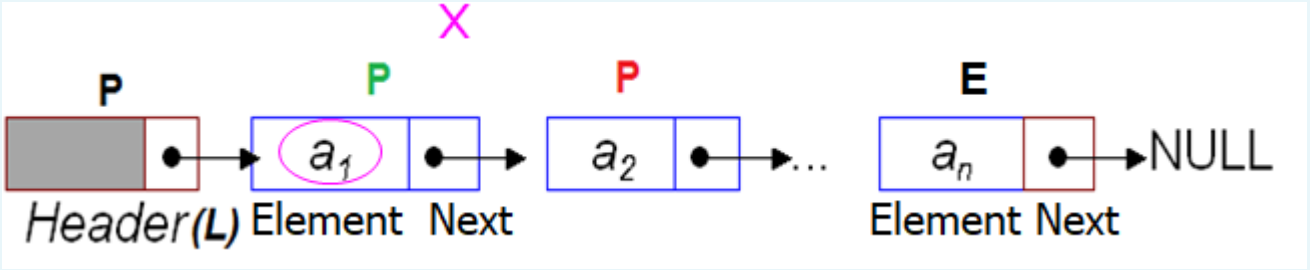
- Bắt đầu từ phần tử đầu tiên trong danh sách, ta tiến hành tìm từ *đầu danh sách* cho đến khi tìm *thấy hoặc cuối danh sách*
  - Nếu giá trị tại vị trí P bằng x

    retrieve(P,L) == x hay P->Next->Element ==x

    thì dừng tìm kiếm
  - Ngược lại (giá trị tại vị trí P khác x) thì đến vị trí kế tiếp

    P = next(P,L) hay P=P->Next

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH

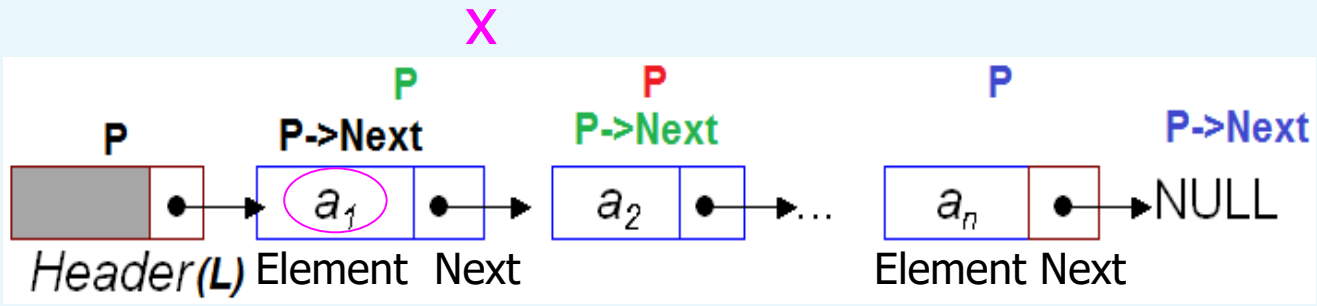



```
Position locate(ElementType x, List L){
  Position P,E;  int Found=0;
  P = first(L);
  E = endList(L);
  while ((P!=E) && (Found == 0))
     if (retrieve(P,L) == x)
           Found = 1;
     else P = next(P,L);
  return P;  // endList(L)
}
```

!Found

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

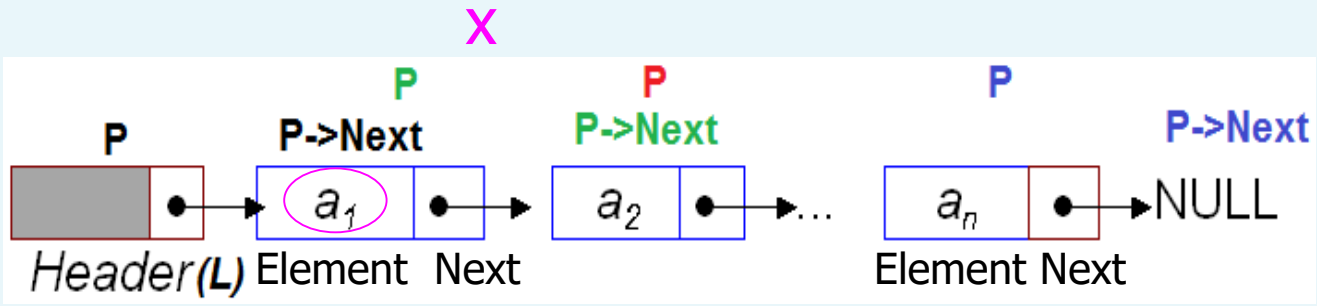
TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH




```
Position locate(ElementType x, List L){
  Position P;
  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

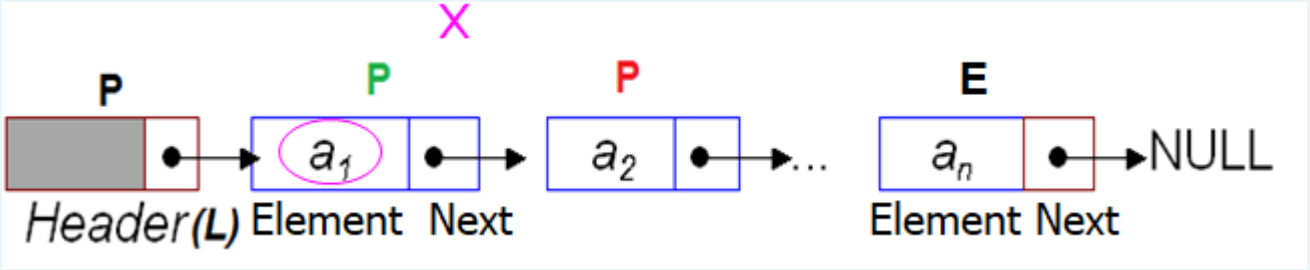
TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH




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

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

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH




```
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)
}
```

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

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;
}
```

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH

Bài tập: cài đặt hàm

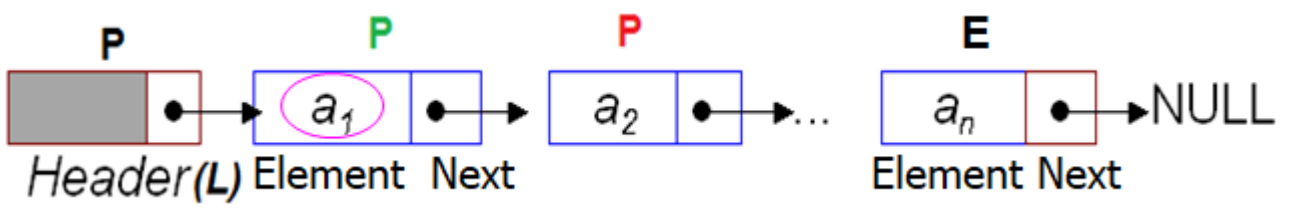
```
Position myLocate(ElementType X, int i,
                  List L)
```

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.

TÌM KIẾM MỘT PHẦN TỬ TRONG 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;
}
```

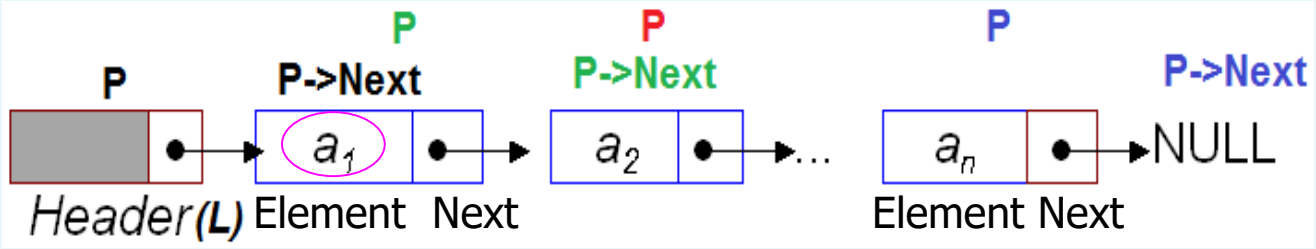
IN DANH SÁCH RA MÀN HÌNH




```
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**

IN DANH SÁCH RA MÀN HÌNH




```
void printList(List L){
   Position P;
   P = L;
   while (P->Next != NULL){
      printf("%d ",P->Next->Element);
      P = P->Next;
   }
   printf("\n");
}
```

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

SO SÁNH 2 PHƯƠNG PHÁP CÀI ĐẶT DS

- Bạn hãy phân tích ưu và khuyết điểm của
  - Danh sách đặc
  - Danh sách liên kết
- Bạn nên chọn phương pháp cài đặt nào cho ứng dụng của mình?

BÀI TẬP

Vận dụng các phép toán trên danh sách liên kết để viết chương trình:

- Nhập vào một danh sách các số nguyên
- Hiển thị danh sách vừa nhập ra màn hình
- Thêm phần tử có nội dung x vào danh sách tại vị trí P (trong đó x và P được nhập từ bàn phím)
- Xóa phần tử đầu tiên có nội dung x (nhập từ bàn phím) ra khỏi danh sách
- Viết hàm

```
Position myLocate(ElementType x, int i, List L)
```

NHẬP DANH SÁCH TỪ BÀN PHÍM

```
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);
  }
}
```

HIỂN THỊ DANH SÁCH RA MÀN HÌNH

```
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");
}
```

CHƯƠNG TRÌNH CHÍNH

```
int main(){
    List L;
    ElementType x;
    Position P;
    readList(&L); // Nhap danh sach
    printList(L); //In danh sach len man hinh

    // Nhap noi dung phan tu can them
    scanf("%d",&x);
    // Xen phan tu x vao cuoi danh sach
    insertList(x,endList(L),&L);
    // In danh sach sau khi them phan tu
    printList(L);

    // Nhap noi dung phan tu can xoa
    scanf("%d",&x);
    P=locate(x,L);
    deleteList(P,&L);
    // In danh sach sau khi xoa
    printList(L);
    return 0;                    }
```

Q&A?

ĐỌC THÊM

- Bài tập
- Cài đặt danh sách bằng con nháy
- Danh sách liên kết kép (Doubly-Linked Lists)

BÀI TẬP

Vận dụng các phép toán trên danh sách liên kết để viết chương trình:

- Nhập vào một danh sách các số nguyên
- Hiển thị danh sách vừa nhập ra màn hình
- Thêm phần tử có nội dung x vào danh sách tại vị trí P (trong đó x và P được nhập từ bàn phím)
- Xóa phần tử đầu tiên có nội dung x (nhập từ bàn phím) ra khỏi danh sách
- Viết hàm

```
Position myLocate(ElementType x, int i, List L)
```

NHẬP DANH SÁCH TỪ BÀN PHÍM

```
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;
}
```

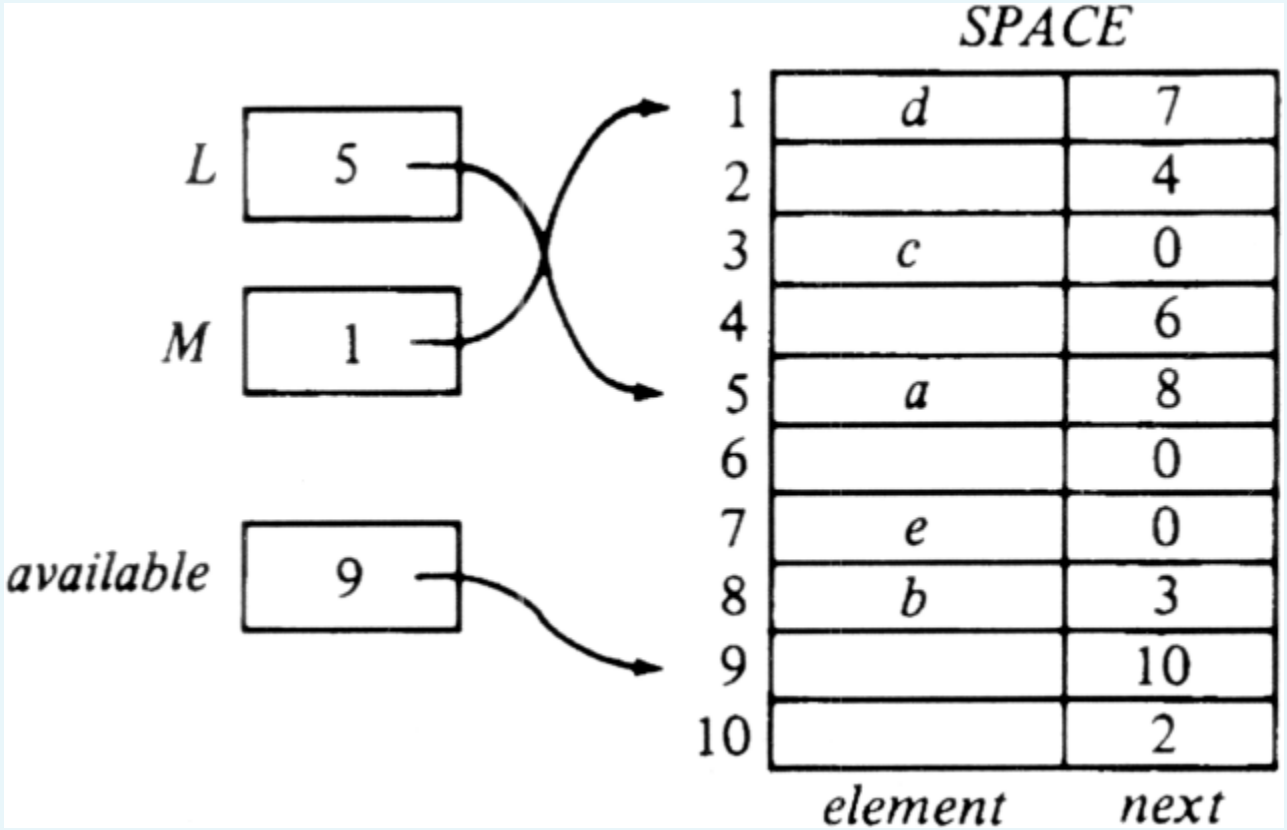
CHƯƠNG TRÌNH CHÍNH

```
int main(){
    List L;
    ElementType x;
    Position P;
    L=readList(); // Nhap danh sach
    printList(L); //In danh sach len man hinh
    // Nhap noi dung phan tu can them
    scanf("%d",&x);
    // Xen phan tu x vao cuoi danh sach
    insertList(x,endList(L),&L);
    // In danh sach sau khi them phan tu
    printList(L);
    // Nhap noi dung phan tu can xoa
    scanf("%d",&x);
    P=locate(x,L);
    deleteList(P,&L);
    // In danh sach sau khi xoa
    printList(L);
    return 0;
}
```

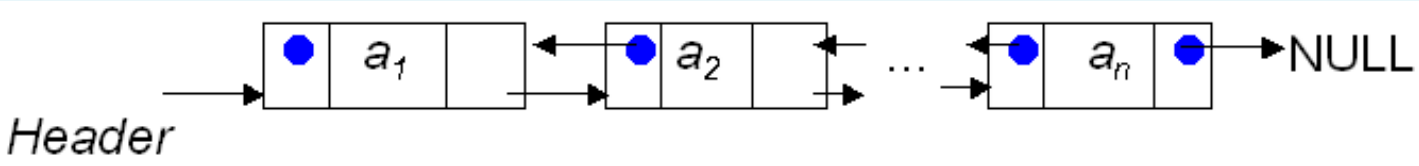
CÀI ĐẶT DANH SÁCH BẰNG CON NHÁY

- Một số ngôn ngữ lập trình không có cung cấp kiểu con trỏ.
- “Giả" con trỏ để cài đặt danh sách liên kết
  - Dùng mảng để chứa các phần tử của danh sách.
  - Các "con nháy" (cursor) sẽ là các biến số nguyên **(int)** để giữ chỉ số của phần tử kế tiếp trong mảng.
  - Như vậy để cài đặt danh sách bằng con nháy ta cần một mảng mà mỗi phần tử xem như là một ô gồm có hai trường:
    - **Element**: lưu trữ giá trị của phần tử trong danh sách;
    - **Next**: là con nháy để chỉ tới vị trí trong mảng của phần tử kế tiếp.

CÀI ĐẶT DANH SÁCH BẰNG CON NHÁY

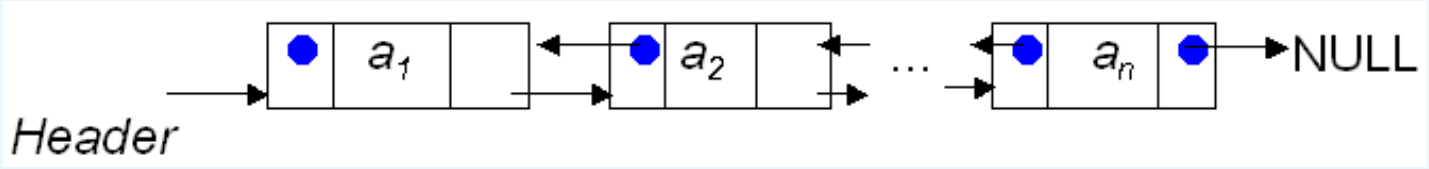


DANH SÁCH LIÊN KẾT KÉP



- Một phần tử của danh sách gồm 3 phần:
  - Element để lưu giữ nội dung phần tử
  - Next để chỉ đến phần tử đứng sau phần tử đang xét
  - Previous để chỉ đến phần tử đứng trước phần tử đang xét
- Để quản lý danh sách liên kết kép ta dùng một con trỏ được gọi là Header
  - Hearder có cùng kiểu với kiểu phần tử trong danh sách
  - Header có thể được cấp phát bộ nhớ hay không cấp phát bộ nhớ

DANH SÁCH LIÊN KẾT KÉP



- Khai báo

```
typedef ... ElementType;//kiểu nội dung của phần tử
struct Node{
    ElementType Element; //lưu trữ nội dung phần tử
    struct Node* Prev; //Con trỏ trỏ tới phần tử trước
    struct Node* Next; //Con trỏ trỏ tới phần tử sau
};
typedef struct Node* Position;
typedef Position DoubleList;
```

ƯU ĐIỂM CỦA DSLK KÉP

- Theo bạn, thuận lợi và bất lợi của việc sử dụng danh sách liên kết kép là gì?