CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GV : ThS. Trần Văn Thọ
E-mail : thotv@hufi.edu.vn

Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHƯƠNG III:

DANH SÁCH LIÊN KẾT

MỤC TIÊU

- Tìm hiểu các loại CTDL danh sách liên kết.
- Tìm hiểu những thao tác trên danh sách liên kết đơn. Từ đó ứng dụng vào một số bài toán cụ thể.

Nội dung

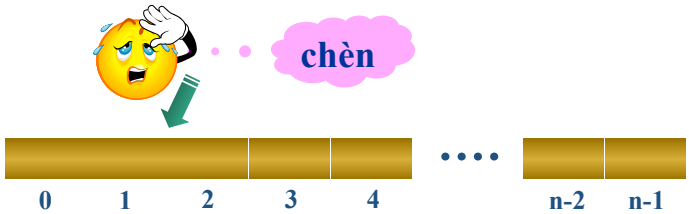
- Danh sách liên kết đơn (**Single Linked List**):
  - Giới thiệu
  - Minh họa
  - Cài đặt các thao tác
  - Ứng dụng.

Single Linked List

(Danh sách liên kết đơn)

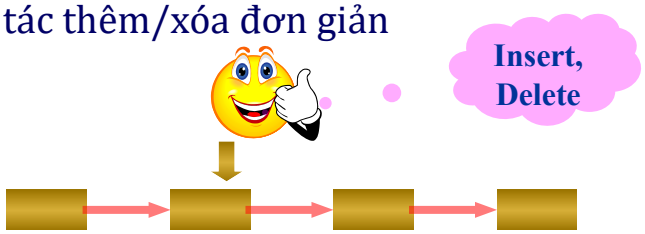
Giới thiệu

- **Mảng 1 chiều**
  - Có kích thước cố định (*cấp phát tĩnh*)
  - Thêm/ xóa có độ phức tạp cao
  - Các phần tử tuần tự theo chỉ số 0 ⇒ n-1
  - Truy cập ngẫu nhiên

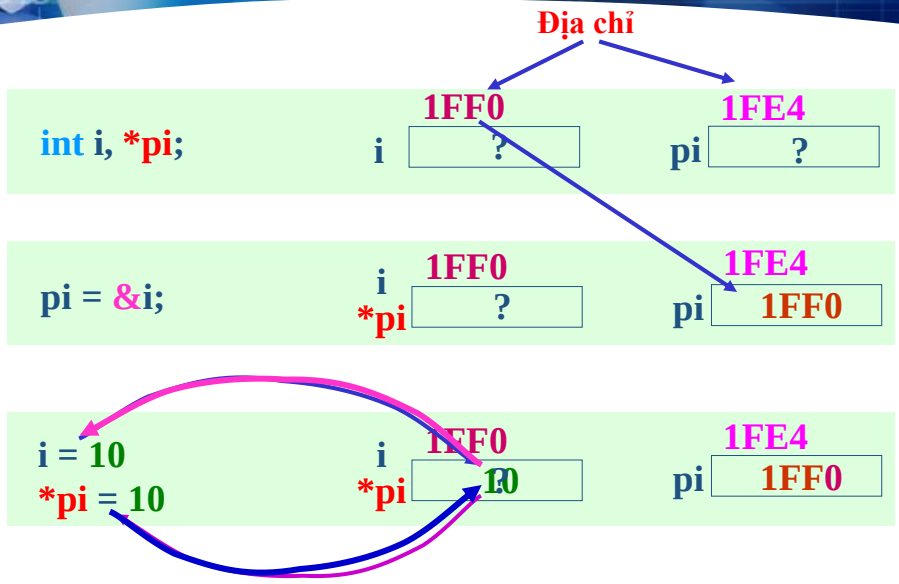


Giới thiệu

- **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

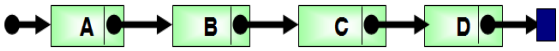


Con trỏ: Pointer



Định nghĩa

- ***Danh sách liên kết đơn*** là chuỗi các phần tử (**Node**), được tổ chức theo thứ tự tuyến tính. Mỗi phần tử liên kết với 1 phần tử đứng liền kế sau trong danh sách.

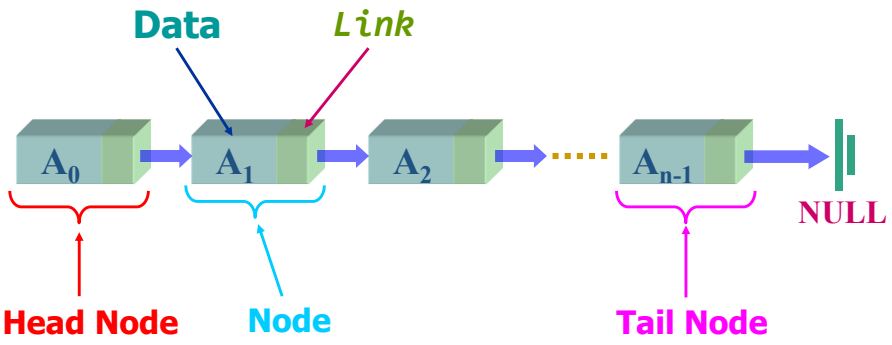


- **Mỗi phần tử (*Node*) gồm 2 thành phần:**
  - ***Thành phần dữ liệu* (*Data*)**: Lưu trữ thông tin về bản thân phần tử (***Information***).

  - ***Thành phần liên kết (Link)***: Lưu địa chỉ phần tử đứng kế sau (***Next***) trong danh sách, hoặc lưu trữ giá trị **NULL** nếu là phần tử cuối danh sách.

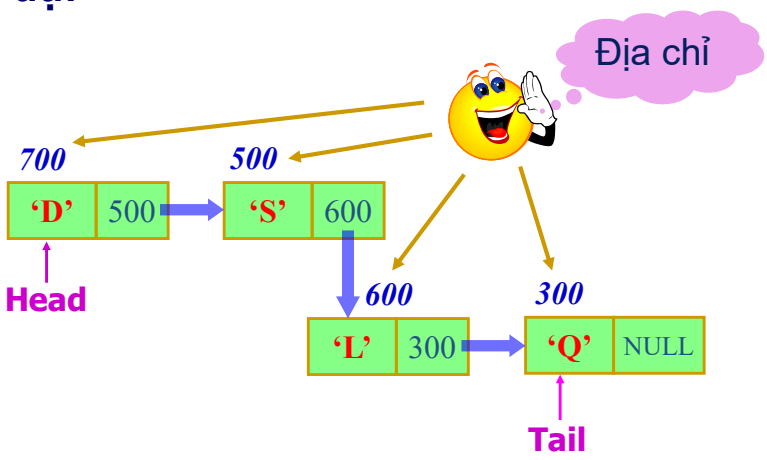
Minh Họa

- **Mô tả danh sách**



Minh Họa

❖Ví dụ:

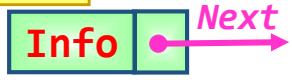


Khai báo phần tử của danh sách

❖Khai báo cấu trúc một nút (Node):

```
struct  SNode
{
    ItemType Info;
    SNode* Next;
};
```

❖Trong đó ItemType:



- Là kiểu dữ liệu được định nghĩa trước.
- Chứa thông tin (*hay dữ liệu*) của từng Node.

ItemType có thể là: int, long, float, SV, NV,….

Khai báo phần tử của danh sách

```
struct SNode
{
    ItemType Info;
    SNode* Next;
};
```

Cấu trúc SNode

```
struct SNode
{
    int Info;
    SNode* Next;
};
```

```
struct SNode
{
    SinhVien Info;
    SNode* Next;
};
```


Khai báo phần tử của danh sách

```
typedef int ItemType;
struct SNode
{
    ItemType Info;
    SNode* Next;
};
```

Kiểu Dữ liệu Node

Dữ liệu của Node là một số nguyên

trỏ đến nút kế sau

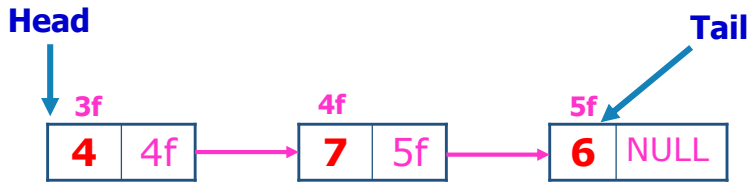
Khai báo kiểu con trỏ SNode

```
typedef SNode* SNodePtr;
```

Khai báo cấu trúc dữ liệu DSLK đơn

```
struct SList
{// Tạo mới kiểu danh sách liên kết đơn tên SList
  SNode* Head; //Lưu địa chỉ SNode đầu tiên trong List
  SNode* Tail; //Lưu địa chỉ của SNode cuối cùng trong List
};
```


Ví dụ tổ chức DSLK đơn trong bộ nhớ



Trong ví dụ trên thành phần dữ liệu là 1 số nguyên

Thao tác cơ bản trên DSLK đơn

- Khởi tạo một danh sách rỗng.
- Kiểm tra danh sách rỗng.
- Tạo một nút mới chứa khóa x.
- Thêm phần tử có khóa x vào danh sách.
- Xóa phần tử có khóa x khỏi danh sách.
- Tìm kiếm phần tử có khóa bằng với x trong danh sách.
- Sắp xếp danh sách.
- Duyệt danh sách.

Phần minh họa sẽ dùng **ItemType** là **int**

Khởi tạo danh sách rỗng

- **Gán địa chỉ của 2 con trỏ Head và Tail đồng thời bằng NULL *(trỏ NULL).***

```
void initSList(SList &sl)
{
   sl.Head = NULL;
   sl.Tail = NULL;
}
```

Kiểm tra danh sách rỗng

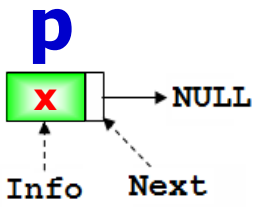
- Danh sách rỗng nếu con trỏ **Head** là **NULL**.

```
int isEmpty(SList sl)
{
    if(sl.Head == NULL)
        return 1;
    else
        return 0;
}
```


Tạo nút mới chứa giá trị x bất kỳ

- **Hàm trả về địa chỉ phần tử mới tạo:**

```
SNode* createSNode(ItemType x)
{
    SNode* p = new SNode;
    if(p == NULL) {
        printf("Không đủ bộ nhớ!");
        getch();
        return NULL;
    }
    p→Info = x;
    p→Next = NULL;
    return p;
}
```



Thao tác thêm phần tử vào danh sách

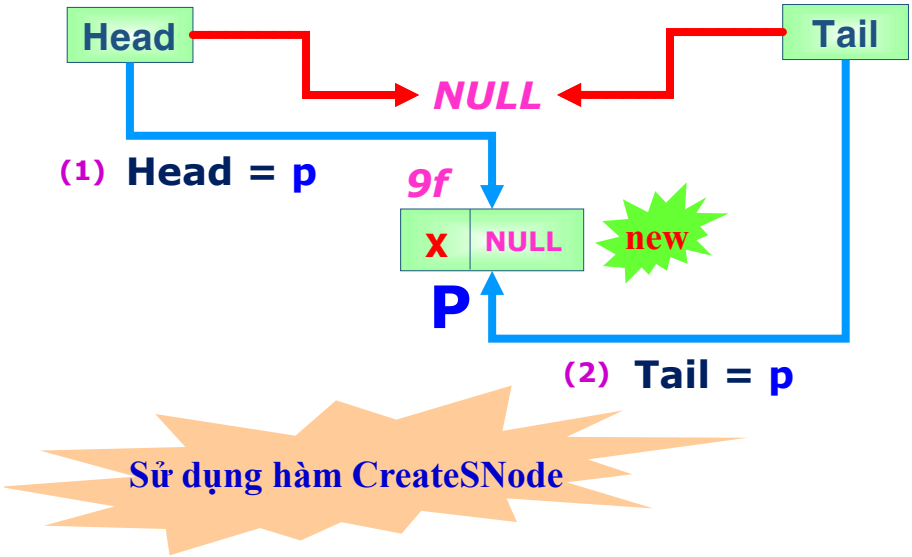
- **Nguyên tắc thêm:** Khi thêm 1 phần tử vào SList thì có làm cho các con trỏ Head, Tail thay đổi?
- **Các vị trí cần thêm 1 phần tử vào SList:**
  - Thêm phần tử vào đầu SList
  - Thêm phần tử vào cuối SList
  - Thêm phần tử p vào sau 1 phần tử q trong SList

Thêm phần tử vào đầu danh sách

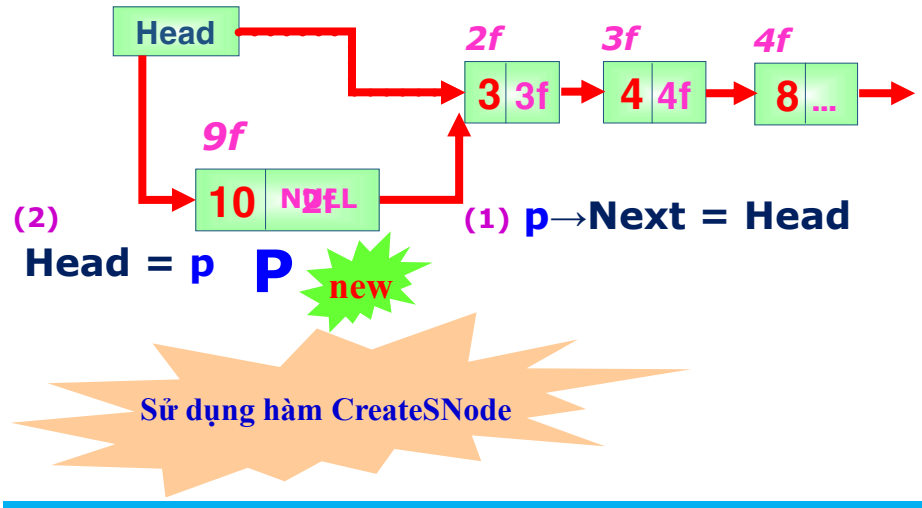
Giải thuật:

- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.
- **Bước 2:** Nếu SList rỗng thì
  + Gán Head = p;
  + Gán Tail = p; *//hoặc Tail = Head;*
- **Bước 3:** Ngược lại
  + Gán p→Next = Head;
  + Gán Head = p;

Ví dụ minh họa Thêm phần tử vào đầu DS



Ví dụ minh họa Thêm phần tử vào đầu DS



Thêm phần tử vào đầu danh sách

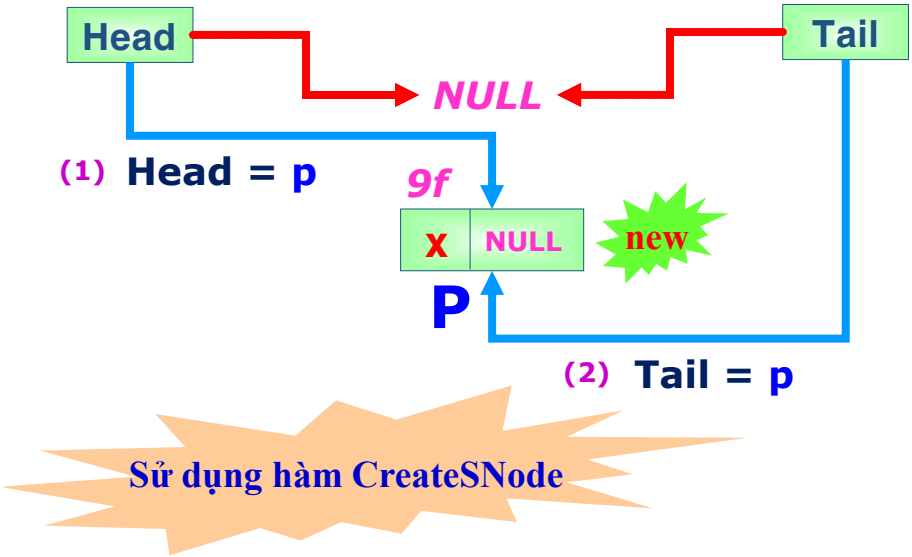
```
int insertHead(SList &sl, SNode* p) {
    if(p == NULL)  return 0;
    if(isEmpty(sl) == 1) {
        sl.Head = p;
        sl.Tail = p;
    }//p làm nút đầu đồng thời là nút cuối
    else {
        p→Next = sl.Head;     (1)
        sl.Head = p;          (2)
    }
    return 1;
}
```


Thêm phần tử vào cuối danh sách

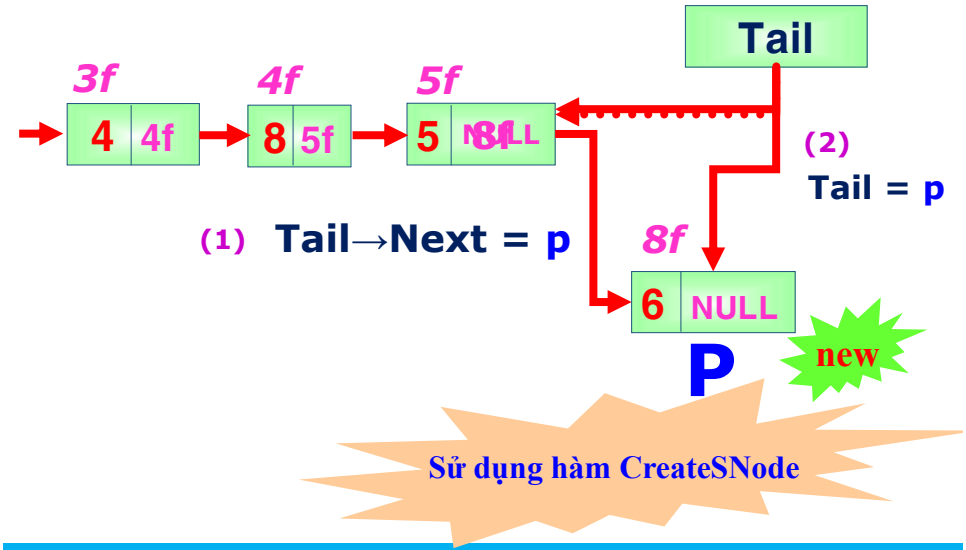
Giải thuật:

- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.
- **Bước 2:** Nếu SList rỗng thì:
  + Gán Head = p;
  + Gán Tail = p;
- **Bước 3:** Ngược lại
  + Gán Tail→Next = p;
  + Gán Tail = p;

Ví dụ minh họa Thêm phần tử vào cuối DS



Ví dụ minh họa Thêm phần tử vào cuối DS



Thêm phần tử vào cuối danh sách

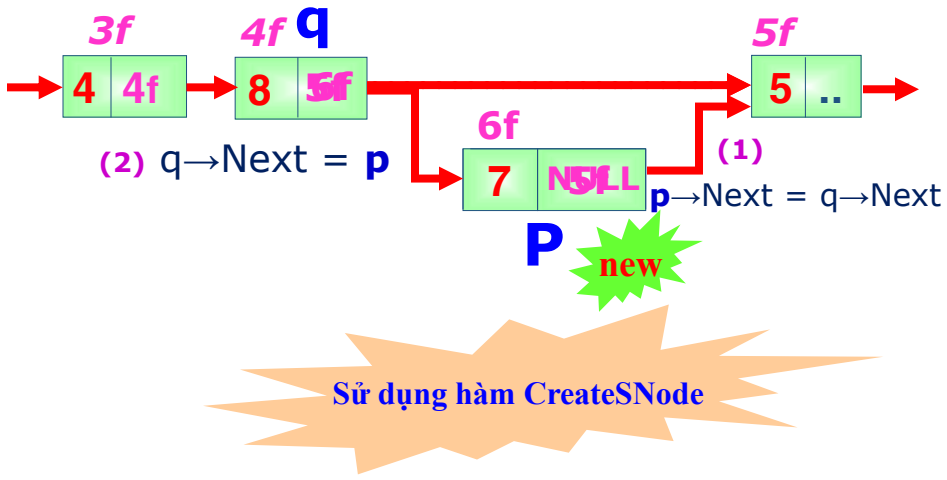
```
int insertTail(SList &sl, SNode* p) {
    if(p == NULL)  return 0;
    if(isEmpty(sl) == 1) {
        sl.Head = p;
        sl.Tail = p;
    }//p làm nút đầu đồng thời là nút cuối
    else {
        sl.Tail→Next = p;       (1)
        sl.Tail = p;            (2)
    }
    return 1;
}
```


Thêm phần tử p vào sau q của danh sách

Giải thuật:

- **Bước 1:** Nếu phần tử q hoặc phần tử muốn thêm p không tồn tại thì không thực hiện.
- **Bước 2:**
  - + Gán p→Next = q→Next;
  - + Gán q→Next = p;
  - + Nếu ***q là phần tử cuối*** thì:
    Gán Tail = p;

Ví dụ minh họa Thêm p vào sau q



Thêm phần tử p vào sau q của danh sách

```
int insertAfter(SList &sl, SNode* q, SNode* p)
{
    if(q == NULL || p == NULL)
        return 0;
    p→Next = q→Next;        (1)
    q→Next = p;             (2)
    if(sl.Tail == q)
        sl.Tail = p;
    return 1;
}
```

Tìm kiếm phần tử có giá trị bằng x

Giải thuật:

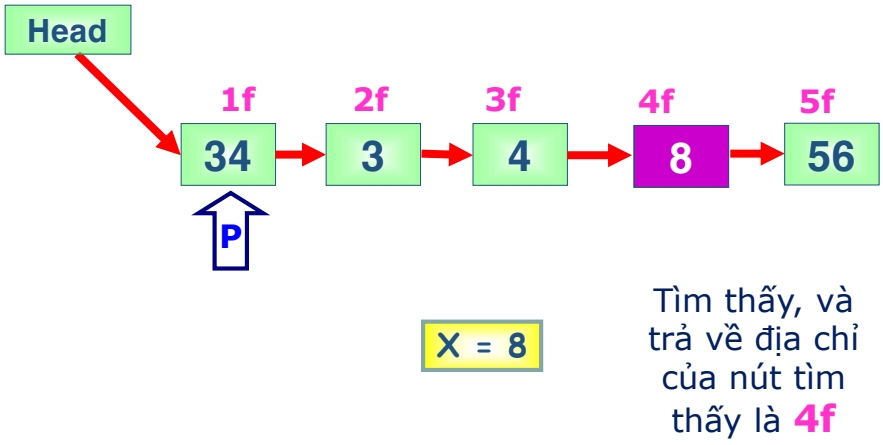
- **Bước 1:** Nếu (Head == NULL) thì: trả về NULL;
- **Bước 2:** Gán p = Head; *//địa chỉ của phần tử đầu trong SList*
- **Bước 3:**

  Lặp lại trong khi (p != NULL và p→Info != x) thì thực hiện:

  p = p→Next; *//xét phần tử kế sau*

- **Bước 4:** Trả về p; *//nếu (p != NULL) thì p lưu địa chỉ của phần tử có khóa bằng x, hoặc NULL là không có phần tử cần tìm.*

Minh họa tìm phần tử trong DSLK



Tìm kiếm phần tử có giá trị bằng x

```
SNode* findSNode(SList sl, ItemType x)
{  //Hàm trả về con trỏ trỏ đến phần tử muốn tìm
    SNode* p = sl.Head;
    while(p != NULL)
    {
        if(p→Info == x)
            return p; //Tìm thấy, trả về địa chỉ
        p = p→Next;
    }
    return NULL; //Không tìm thấy, trả về NULL
}
```


Tìm kiếm phần tử có giá trị bằng x

```
SNode* findSNode(SList sl, ItemType x)
{  //Hàm trả về con trỏ trỏ đến phần tử muốn tìm
    SNode* p;
    for(p = sl.Head; p != NULL; p = p→Next)
        if(p→Info == x)
            return p; //Tìm thấy, trả về địa chỉ
    return NULL; //Không tìm thấy, trả về NULL
}
```

Thao tác xóa phần tử trong danh sách

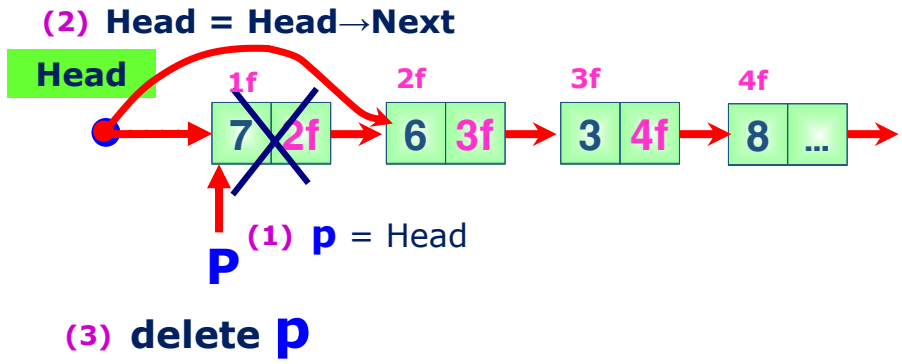
- **Nguyên tắc xóa:**
  - Phải cô lập phần tử cần xóa trước khi xóa nó.
  - Khi xóa 1 phần tử trong SList thì có làm cho các con trỏ Head, Tail thay đổi?
- **Các vị trí cần xóa 1 phần tử trong SList:**
  - Xóa phần tử đầu/cuối SList
  - Xóa phần tử p sau/trước 1 phần tử q trong SList
  - Xóa phần tử có giá trị x bất kỳ trong SList
  - Xóa toàn bộ SList
- **Giải phóng vùng nhớ của nút bằng hàm delete**

Xóa phần tử đầu của danh sách

Giải thuật:

- **Bước 1:** Nếu (Head == NULL) thì: không thực hiện gì cả.
- **Bước 2:** Khi (Head != NULL) thì: thực hiện các công việc sau:
  - + **Bước 2.1:** Gán p = Head;
  - + **Bước 2.2:** Gán Head = Head→Next;
  - + **Bước 2.3:** Nếu (Head == NULL) thì: Gán Tail = NULL;
  - + **Bước 2.4:** Lưu lại thông tin nút bị xóa;
  - + **Bước 2.5:** **delete p**; *//Hủy nút do p trỏ đến (con trỏ p)*

Ví dụ minh họa hủy nút đầu DS



Xóa phần tử đầu danh sách

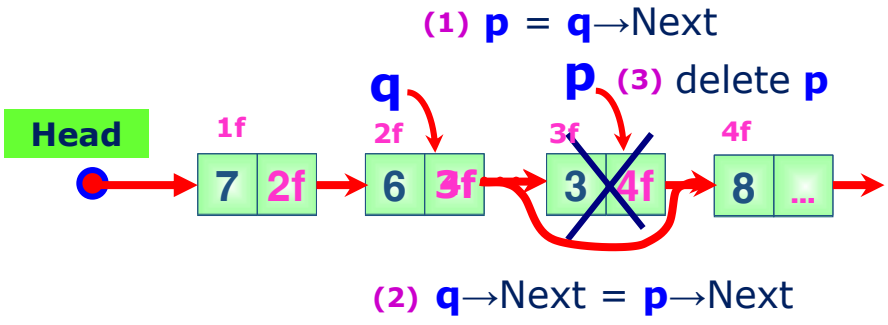
```
int deleteHead(SList &sl, ItemType &x)
{
    if(isEmpty(sl) == 1) return 0;
    SNode* p = sl.Head;          (1)
    sl.Head = sl.Head→Next;  (2)
    if(sl.Head == NULL)
        sl.Tail = NULL;
    x = p→Info; //lưu ItemType của nút cần hủy
    delete p; //Xóa nút p
    return 1; //Xóa thành công
}
```

Xóa phần tử p sau phần tử q của danh sách

Giải thuật:

- **Bước 1:** Nếu (q == NULL hoặc q→Next == NULL) thì: không thực hiện.
- **Bước 2:** Khi (q != NULL và q→Next != NULL) thì: thực hiện các công việc sau:
  - + **Bước 2.1:** Gán p = q→Next;
  - + **Bước 2.2:** Gán q→Next = p→Next;
  - + **Bước 2.3:** Nếu (Tail == p) thì: Gán Tail = q;
  - + **Bước 2.4:** Lưu lại thông tin nút bị xóa;
  - + **Bước 2.5:** **delete p**; *//Hủy nút do p trỏ đến (con trỏ p)*

Ví dụ minh họa hủy nút p sau nút q



Xóa phần tử sau q trong danh sách

```
int deleteAfter(SList &sl, SNode* q, ItemType &x)
{
    if(q == NULL || q→Next == NULL) return 0;
    SNode* p = q→Next;  (1)
    q→Next = p→Next;          (2)
    if(sl.Tail == p)
        sl.Tail = q;
    x = p→Info;
    delete p; //Xóa nút p
    return 1; //Xóa thành công
}
```


Xóa phần tử có khóa x của danh sách

Giải thuật:

- **Bước 1:** Nếu (Head == NULL) thì: không thực hiện gì cả.
- **Bước 2:** Tìm phần tử p có khoá bằng x, và q là phần tử đứng kế trước p.
- **Bước 3:** Nếu (p == NULL) thì: Không có nút chứa x nên không thực hiện.
- **Bước 4:** *//Ngược lại (p != NULL) → nghĩa là tồn tại nút p chứa khóa x*
  - + Nếu (p == Head) thì: *//p là nút đầu danh sách*
    DeleteHead(sl, x); *//Gọi hàm xóa nút đầu*
  - + Ngược lại:
    DeleteAfter(sl, q, x); *//Xóa nút p chứa x kế sau nút q*

Xóa phần tử có khóa x của danh sách

```
int deleteSNodeX(SList &sl, ItemType x) {
    if(isEmpty(sl) == 1)  return 0; //Không thực hiện được
    SNode* p = sl.Head;
    SNode* q = NULL; //sẽ trỏ đến nút kế trước p
    while( (p != NULL) && (p→Info != x) )
    {//vòng lặp tìm nút p chứa x, q là nút kế trước p
        q = p;  p = p→Next;
    }
    if(p == NULL)  return 0; //không tìm thấy phần tử có khóa x
    if(p == sl.Head) //p có khóa bằng x là nút đầu danh sách
        deleteHead(sl, x);
    else //xóa nút p có khóa x nằm kế sau nút q
        deleteAfter(sl, q, x);
    return 1; //Thực hiện thành công
}
```


Xóa phần tử có khóa x của danh sách

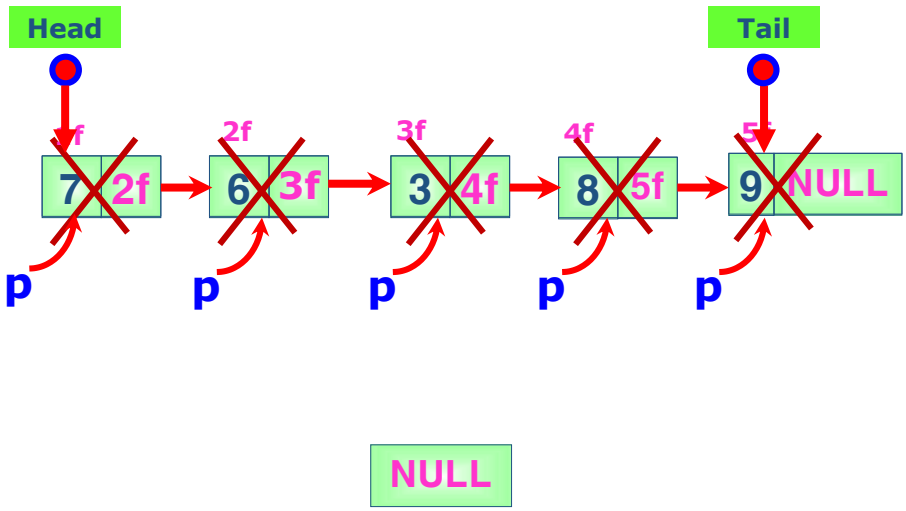
```
int deleteSNodeX(SList &sl, ItemType x) {
    if(isEmpty(sl) == 1)  return 0; //Không thực hiện được
    SNode* p = findSNode(sl, x);
    if(p == NULL) return 0; //Thực hiện không thành công
    if(p == sl.Head)
        deleteHead(sl, x);
    else
    {
        SNode* q = sl.Head; //tìm nút q kế trước nút p
        while(q->Next != p)
            q = q->Next;
        deleteAfter(sl, q, x);
    }
    return 1; //Thực hiện thành công
}
```

Xóa toàn bộ danh sách

Giải thuật:

- **Bước 1:** Nếu (Head == NULL) thì: không thực hiện gì cả.
- **Bước 2:** Lặp lại trong khi (danh sách còn phần tử) thì thực hiện lần lượt những việc sau:
  - + **Bước 2.1:** Gán p = Head;
  - + **Bước 2.2:** Gán Head = Head→Next; *//Cập nhật Head*
  - + **Bước 2.3:** Hủy con trỏ p;
- **Bước 3:** Gán Tail = NULL; *//bảo toàn tính nhất quán khi danh sách rỗng*

Ví dụ minh họa Hủy danh sách



Xóa toàn bộ danh sách

```
void deleteAllSList(SList &sl)
{
    while(sl.Head != NULL)
    {//trong khi còn phần tử trong SList
        SNode *p = sl.Head;
        sl.Head = sl.Head→Next;
        delete p; //Xóa nút p
    }
    sl.Tail = NULL;
}
```

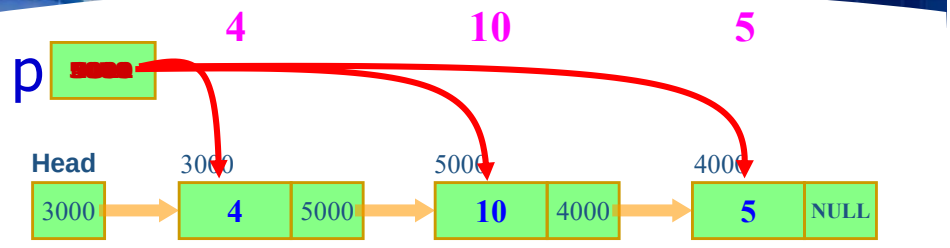

Xử lý danh sách

```
void processSList(SList sl) {
    if(isEmpty(sl) == 1) {
        printf("Danh sách rỗng!");
        return;
    }
    SNode *p = sl.Head;
    while(p != NULL) {
        processSNode(p); //xử lý cụ thể tuỳ trường hợp
        p = p→Next;
    }
}
```

Duyệt in danh sách: *Cách 1*

```
void showSList(SList sl) {
    if(isEmpty(sl) == 1) {
        printf("Danh sách rỗng!");
        return;
    }
    SNode *p = sl.Head;
    while(p != NULL) {
        printf("%4d", p→Info); //Xuất ra màn hình
        p = p→Next;
    }
}
```


Minh họa duyệt in danh sách




```
SNode* p = sl.Head;
while(p != NULL)
{
        printf("%4d", p→Info);
        p = p→Next;
}
```

Duyệt in danh sách: *cách 2*

```
void showSList(SList sl)
{
    if(isEmpty(sl) == 1)
    {
        printf("\nDanh sach rong!");
        return;
    }
    printf("\nNoi dung cua danh sach la: ");
    for(SNode* p=sl.Head; p!=NULL; p=p→Next)
        printf("%4d", p→Info);
}
```


Tìm phần tử lớn nhất

```
int findMax(SList sl)
{//Hàm trả về giá trị phần tử lớn nhất
    int max = sl.Head→Info;
    for(SNode* p = sl.Head→Next;  p != NULL; p =
    p→Next)
        if(max < p→Info)
            max = p→Info;
    return  max;
}
```

Tìm phần tử lớn nhất

```
int findMax(SList sl)
{//Hàm trả về giá trị phần tử lớn nhất
    int  max = sl.Head→Info;
    SNode* p = sl.Head→Next;
    while(p != NULL)
    {
        if(max < p→Info)
            max = p→Info;
        p = p→Next;
    }
    return  max;
}
```


Tìm phần tử lớn nhất

❖**Tìm địa chỉ chứa phần tử lớn nhất:**

```
SNode* findMaxPosition(SList sl)
{//Hàm trả về con trỏ trỏ đến phần tử lớn nhất
    SNode* pos = sl.Head;
    for(SNode* p = sl.Head→Next; p != NULL; p =
    p→Next)
        if(pos→Info < p→Info)
            pos = p;
     return pos;
}
```

Sắp xếp bằng InterchangeSort

```
void interchangeSort_Asc(SList &sl)
{
    SNode *p, *q;
    for(p = sl.Head; p→Next!=NULL; p = p→Next)
        for(q = p→Next; q!=NULL; q = q→Next)
            if(p→Info > q→Info)
                swap(p→Info, q→Info);
}
```


Sắp xếp bằng SelectionSort

```
void selectionSort_Asc(SList &sl)
{
    SNode *q, *p, *min;
    for(p = sl.Head; p→Next!=NULL; p = p→Next)
  {
        min = p;
        for(q = p→Next; q!=NULL; q = q→Next)
            if(min→Info > q→Info)
                min = q;
        swap(p→Info, min→Info);
    }
}
```

Bài tập bổ sung

1. Đếm số Node có trong danh sách
2. Đếm số phần tử dương trong danh sách
3. Đếm số phần tử lớn hơn phần tử kề sau
4. Kiểm tra mọi phần tử trong danh sách có chẵn không?
5. Kiểm tra danh sách có được sắp xếp tăng?

Bài tập bổ sung

6. Tạo danh sách **sl1** chứa các phần tử dương trong danh sách đã cho.
7. Xóa 1 phần tử có khoá là x
8. Thêm phần tử x vào danh sách đã có thứ tự (tăng) sao cho sau khi thêm vẫn có thứ tự (tăng).

Bài tập bổ sung

9. Xóa các phần tử trùng nhau trong danh sách, chỉ giữ lại duy nhất một phần tử (*)
10. Trộn hai danh sách có thứ tự tăng thành một danh sách cũng có thứ tự tăng. (*)
11. Chèn một phần tử có khoá x vào vị trí pos bất kỳ trong danh sách.