CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Data Structures & Algorithms

DANH SÁCH LIÊN KẾT – LIST

Nội dung

1. Kiểu danh sách

2. Danh sách liên kết đơn.

3. Stack

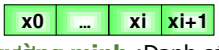
4. Queue

KIỂU DANH SÁCH

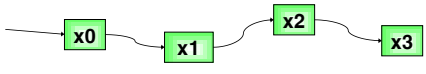
- **Danh sách** = { **các phần tử có cùng kiểu**}
- Danh sách là một ***kiểu dữ liệu tuyến tính*** :
  - Mỗi phần tử ***có nhiều nhất 1 phần tử đứng trước***
  - Mỗi phần tử **có *nhiều nhất 1 phần tử đứng sau***
- Là kiểu dữ liệu quen thuộc trong thực tế :
  - Danh sách học sinh
  - Danh mục sách trong thư viện
  - Danh bạ điện thoại
  - Danh sách các nhân viên trong công ty
  - ...

Các hình thức tổ chức danh sách

- CTDL cho **mỗi phần tử** ?
- Thể hiện **liên kết của các phần tử** ?
- *Hai hình thức cơ bản* :
  - Liên kết **ngầm** : Mảng (array)

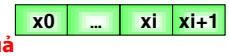


  - Liên kết **tường minh** :Danh sách liên kết (list)



Mảng – Danh sách liên kết ngầm

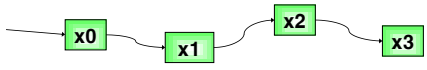
- Mối **liên hệ giữa các phần tử được thể hiện ngầm**:
  - xi : phần tử thứ i trong danh sách
  - xi , xi_{+1} là kế cận trong danh sách
- Phải **lưu trữ liên tiếp các phần tử trong bộ nhớ**
  - công thức xác định địa chỉ phần tử thứ i:
    address(i) = address(1) + (i-1)*sizeof(T)
- **Ưu điểm** : **Truy xuất trực tiếp, nhanh chóng**
- **Nhược điểm**:



  - **Sử dụng bộ nhớ kém hiệu quả**
  - Kích thước **cố định**
  - Các **thao tác thêm vào , loại bỏ không hiệu quả**

Danh sách liên kết tường minh

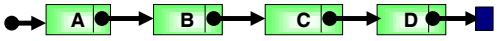
- CTDL cho một phần tử:
  - **Thông tin bản thân**
  - **Địa chỉ của phần tử kế** trong danh sách



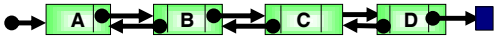
- **Mỗi phần tử là một biến động**
- **Ưu điểm**
  - + **Sử dụng hiệu quả bộ nhớ**
  - + **Linh động về số lượng phần tử**

Các loại danh sách liên kết

- **Danh sách liên kết đơn**: Mỗi phần tử **liên kết với phần tử đứng sau nó** trong danh sách

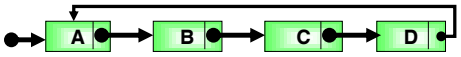


➢ **Danh sách liên kết kép**: Mỗi phần tử **liên kết với phần tử đứng trước và sau nó** trong danh sách

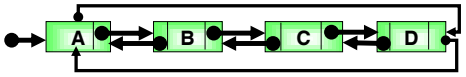


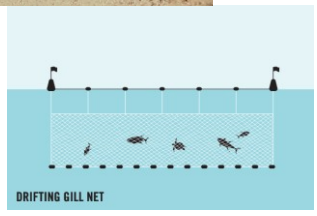
➢ **Danh sách liên Vòng**: **Phần tử cuối danh sách liên với phần tử đầu danh sách**

- **Danh sách liên Vòng:** Phần tử cuối danh sách liên với phần tử đầu danh sách
  - Danh sách liên kết đơn vòng



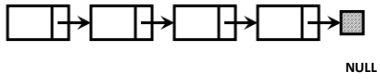
  - Danh sách liên kết đôi vòng





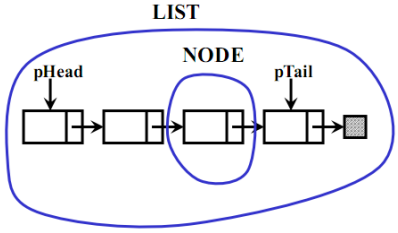
Danh sách liên kết đơn - LIST

Hình ảnh:

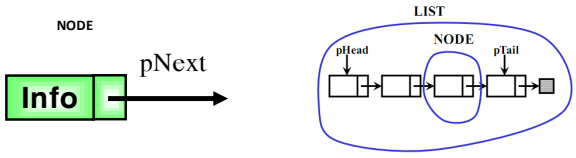


Danh sách liên kết đơn - LIST

Hình ảnh:



Danh sách liên kết đơn - LIST

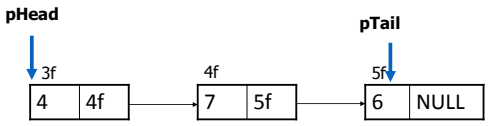



```
11.struct node
12.{
13.    KDL info;
14.    struct node*pNext;
15.};
16.typedef struct node NODE;
```

```
17.struct list
18.{
19.    NODE*pHead;
20.    NODE*pTail;
21.};
22.typedef struct list LIST;
```

13

Danh sách liên kết đơn - LIST



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

14

Danh sách liên kết đơn - LIST

- Cấu trúc dữ liệu của 1 nút trong List đơn

```
struct  node
{
    KDL   Info;   // Lưu thông tin dữ liệu có KDL
    struct  node *pNext; //Lưu địa chỉ của Node đứng sau
};
```

- Cấu trúc dữ liệu của DSLK đơn

```
struct  list
{
    NODE *pHead;//Lưu địa chỉ Node đầu tiên trong List
    NODE *pTail; //Lưu địa chỉ của Node cuối cùng trong
List
};
```

15

Danh sách liên kết đơn - LIST

Ví dụ 1: Hãy khai báo CTDL cho DSLK đơn các **số nguyên**

```
10.struct node
11.{
12.    int info;
13.    struct node*pNext;
14.};
15.typedef struct node NODE;
```

```
16.struct list
17.{
18.    NODE*pHead;
19.    NODE*pTail;
20.};
21.typedef struct list LIST;
```

16

Danh sách liên kết đơn - LIST

Ví dụ 2: Hãy khai báo CTDL cho DSLK đơn các **số thực**

```
10.struct node
11.{
12.    float info;
13.    struct node*pNext;
14.};
15.typedef struct node NODE;
```

```
16.struct list
17.{
18.    NODE*pHead;
19.    NODE*pTail;
20.};
21.typedef struct list LIST;
```

17

Danh sách liên kết đơn - LIST

Ví dụ 3: Hãy khai báo CTDL cho DSLK đơn các **phân số**

```
10.struct phanso
11.{
12.    int tu;
13.    int mau;
14.};
15.typedef struct phanso PHANSO;
```

```
16.struct node
17.{
18.    PHANSO info;
19.    struct node*pNext;
20.};
21.typedef struct node NODE;
```

```
22.struct list
23.{
24.    NODE*pHead;
25.    NODE*pTail;
26.};
27.typedef struct list LIST;
```

18

Danh sách liên kết đơn - LIST

Ví dụ 4: Hãy khai báo CTDL cho DSLK đơn **tọa độ** các điểm trong mặt **phẳng oxy**

```
10.struct diem
11.{
12.    float x;
13.    float y;
14.};
15.typedef struct diem DIEM;
```

```
16.struct node
17.{
18.    DIEM info;
19.    struct node*pNext;
20.};
21.typedef struct node NODE;
```

```
22.struct list
23.{
24.    NODE*pHead;
25.    NODE*pTail;
26.};
27.typedef struct list LIST;
```

19

Các thao tác trên LIST

- ***Khởi tạo DSLK đơn*** - Tạo 1 DSLK đơn rỗng
- **Tạo** 1 node có trường bằng x
- Thêm một **node có khóa x** vào **danh sách**
- **Duyệt danh sách**
- **Hủy một phần tử** trong danh sách
- **Sắp xếp** danh sách liên kết đơn

Khởi tạo DSLK đơn

- Khái niệm: Khởi tạo danh sách liên kết đơn là tạo ra danh sách rỗng không chứa node nào hết.
- Định nghĩa hàm

```
1. void Init(LIST &ℓ)
2. {
3.     ℓ.pHead = NULL;
4.     ℓ.pTail = NULL;
5. }
```

21

Kiểm tra DSLK đơn rỗng

- Khái niệm: Kiểm tra danh sách liên kết đơn rỗng là hàm trả về giá trị 1 khi danh sách rỗng. Trong tình huống danh sách không rỗng thì hàm sẽ trả về giá trị 0.
- Định nghĩa hàm

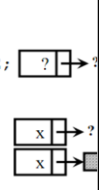
```
1. int IsEmpty(LIST ℓ)
2. {
3.     if (ℓ.pHead==NULL)
4.         return 1;
5.     return 0;
6. }
```

22

Tạo NODE cho DSLK đơn

- **Khái niệm: Tạo node cho danh sách liên kết đơn là xin cấp phát bộ nhớ có kích thước bằng với kích thước của kiểu dữ liệu NODE để chứa thông tin đã được biết trước.**
- Định nghĩa hàm trừu tượng

```
1. NODE* GetNode(KDL x)
2. {
3.     NODE *p=new NODE;
4.     if (p==NULL)
5.         return NULL;
6.     p->info = x;
7.     p->pNext = NULL;
8.     return p;
9. }
```



23

Tạo NODE cho DSLK đơn

Ví dụ 1: Định nghĩa hàm tạo một node cho DSLK đơn các số nguyên để chứa thông tin đã được biết trước

```
1. NODE* GetNode(int x)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)
5.         return NULL;
6.     p->info = x;
7.     p->pNext = NULL;
8.     return p;
```

24

Tạo NODE cho DSLK đơn

Ví dụ 2: Định nghĩa hàm tạo một node cho DSLK đơn các số thực để chứa thông tin đã được biết trước

```
1. NODE* GetNode(float x)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)
5.         return NULL;
6.     p->info = x;
7.     p->pNext = NULL;
8.     return p;
9. }
```

25

Tạo NODE cho DSLK đơn

Ví dụ 3: Định nghĩa hàm tạo một node cho DSLK đơn các phân số chứa thông tin đã được biết trước

```
1. NODE* GetNode(PHANSO x)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)
5.         return NULL;
6.     p->info = x;
7.     p->pNext = NULL;
8.     return p;
9. }
```

26

Tạo NODE cho DSLK đơn

Ví dụ 4: Định nghĩa hàm tạo một node cho DSLK đơn các điểm trong hệ tọa độ oxy chứa thông tin đã được biết trước

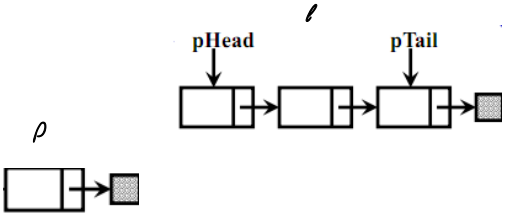
```
1. NODE* GetNode(DIEM P)
2. {
3.     NODE *p = new NODE;
4.     if (p==NULL)
5.         return NULL;
6.     p->info = P;
7.     p->pNext = NULL;
8.     return p;
9. }
```

27

Thêm một phần tử vào List đơn

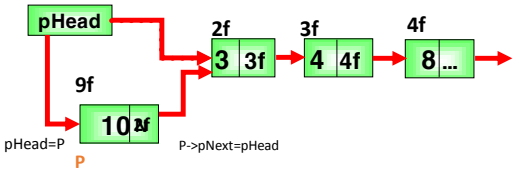
- **Nguyên tắc thêm:** Khi thêm 1 phần tử vào List thì **có làm cho pHead, pTail thay đổi**?
- **Các vị trí cần thêm 1 phần tử vào List:**
  - Thêm vào **đầu List đơn**
  - Thêm vào **cuối List đơn**
  - Thêm vào **sau 1 phần tử q** trong list

Thêm một NODE vào đầu List đơn



29

Thêm một NODE vào đầu List đơn



30

Thêm một NODE vào đầu List đơn

```
void AddHead(LIST &l, Node* p)
{
	if (l.pHead==NULL)
	{
		l.pHead = p;
		l.pTail = l.pHead;
	}
	else
	{
		p->pNext = l.pHead;
		l.pHead = p;
	}
}
```

Nhập List đơn từ bàn phím

Nhập list đơn từ bàn phím là lần lượt nhập các thông tin của từng node trong danh sách.

- Yêu cầu người dùng nhập vào **số node của list**.
- **Nhập vào từng node** trong n của list
  - **Tạo ra** một node
  - **Nhập giá trị info** vào node vừa tạo (**GetNode()**).
  - Thêm node vào list bằng cách thêm vào đầu (**addHead()**).

32

Nhập List đơn từ bàn phím

Nhập list đơn từ bàn phím là lần lượt nhập các thông tin của từng node trong danh sách.

```
void input(List &l)
{
    int n;
    cout<<"Nhap vao so phan tu cua list";
    cin>>n;
    Init(l);
    for(int i=1;i<=n;i++)
    {
        KDL x;
        Nhap(x);
        NODE* p = GetNode(x);
        if(p!=NULL)
            AddHead(l,p)

    }
}
```

33

Nhập List đơn từ bàn phím

VD 1: Nhập List các **số nguyên**

```
10.struct node
11.{
12.    int info;
13.    struct node*pNext;
14.};
15.typedef struct node NODE;
16.struct list
17.{
18.    NODE*pHead;
19.    NODE*pTail;
20.};
21.typedef struct list LIST;
```

34

Nhập List đơn từ bàn phím

VD 1: Nhập List các **số nguyên**

```
1. void Init(LIST&ℓ)
2. {
3.     ℓ.pHead = NULL;
4.     ℓ.pTail = NULL;
5. }
6. NODE* GetNode(int x)
7. {
8.     NODE *p = new NODE;
9.     if (p==NULL)
10.        return NULL;
11.    p->info = x;
12.    p->pNext = NULL;
13.    return p;
14.}
```

35

Nhập List đơn từ bàn phím

VD 1: Nhập List các **số nguyên**

```
11. void AddHead(LIST&ℓ, NODE*p)
12.{
13.    if(ℓ.pHead==NULL)
14.        ℓ.pHead = ℓ.pTail = p;
15.    else
16.    {
17.        p->pNext = ℓ.pHead;
18.        ℓ.pHead = p;
19.    }
20.}
```

36

Nhập List đơn từ bàn phím

VD 1: Nhập List các số nguyên

```
16 void input(List &l)
17 {
18     int n;
19     cout<<"Nhap vao so phan tu cua list";
20     cin>>n;
21     Init(l);
22     for(int i=1;i<=n;i++)
23     {
24         int x;
25         cin>>x;
26         NODE* p = GetNode(x);
27         if(p!=NULL)
28             AddHead(l,p)
29
30     }
31 }
```

37

Nhập List đơn từ bàn phím

VD 2: Nhập List các phân số

```
10. struct phanso
11. {
12.     int tu;
13.     int mau;
14. };
15. typedef struct phanso PHANSO;
16. struct node
17. {
18.     PHANSO info;
19.     struct node*pNext;
20. };
21. typedef struct node NODE;
22. struct list
23. {
24.     NODE*pHead;
25.     NODE*pTail;
26. };
27. typedef struct list LIST;
```

38

Nhập List đơn từ bàn phím

VD 2: Nhập List các phân số

```
1. void Init(LIST&l)
2. {
3.     l.pHead = NULL;
4.     l.pTail = NULL;
5. }
6. NODE* GetNode(PHANSO x)
7. {
8.     NODE *p = new NODE;
9.     if (p==NULL)
10.         return NULL;
11.    p->info = x;
12.    p->pNext = NULL;
13.    return p;
14. }
```

39

Nhập List đơn từ bàn phím

VD 2: Nhập List các phân số

```
void nhap(PHANSO &x)
{
    cout<<"Nhap tu";
    cin>>x.tu;
    cout<<"\n Nhap mau";
    cin>>x.mau;
}
```

40

Nhập List đơn từ bàn phím

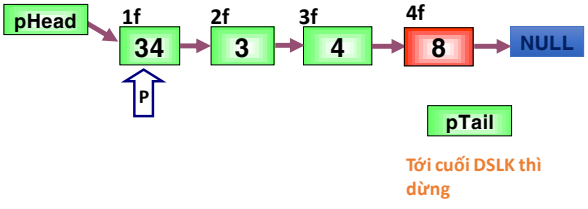
VD 2: Nhập List các phân số

```
void input(List &l)
{
    int n;
    cout<<"Nhap vao so phan tu cua list";
    cin>>n;
    Init(l);
    for(int i=1;i<=n;i++)
    {
        PHANSO x;
        Nhap(x);
        NODE* p = GetNode(x);
        if(p!=NULL)
            AddHead(l,p)
    }
}
```

41

Duyệt DSLK đơn

- Khái niệm: duyệt danh sách liên kết đơn là thăm qua tất cả các node mỗi node một lần.



Tới cuối DSLK thì dừng

42

Duyệt DSLK đơn

- Bước 1:

  p = pHead;// p lưu địa chỉ của phần tử đầu trong List

- Bước 2:

  Trong khi (danh sách chưa hết) thực hiện

  + xử lý phần tử p

  + p=p->pNext;// qua phần tử kế

Duyệt DSLK đơn

- Định nghĩa hàm trừu tượng

```
KDL <Tên Hàm>(LIST ℓ)
{
    ...
    NODE*p = ℓ.pHead;
    while(p!=NULL)
    {
        ...
        p = p->pNext;
    }
    ...
}
```

44

Các thao tác duyệt list đơn

- **In ra** danh sách liên kết đơn – In ra giá trị Info của DSLK đơn.
- **Tìm kiếm** node có **trường Info** thỏa mãn điều kiện.

45

Các thao tác duyệt list đơn

Ví dụ 1: **In ra** danh sách liên kết các **số nguyên**

```
struct node
{
    int info;
    struct node*pNext;
};
typedef struct node NODE;
struct list
{
    NODE*pHead;
    NODE*pTail;
};
typedef struct list LIST;
```

46

Các thao tác duyệt list đơn

Ví dụ 1: **In ra** danh sách liên kết các **số nguyên**

```
void XUAT(List l)
{
    NODE* p= l.pHead;
    while(p!=NULL)
    {
        cout<<p->info<<endl;
        p=p->pNext;
    }
}
```

47

Các thao tác duyệt list đơn

Ví dụ 2: Định nghĩa hàm tính **tổng các số lẻ** trong dslk đơn các số nguyên

- Định nghĩa hàm

```
int TongLe(LIST ℓ)
{
    int s = 0;
    NODE*p = ℓ.pHead;
    while(p!=NULL)
    {
        if(p->info%2!=0)
            s = s + p->info;
        p = p->pNext;
    }
    return s;
}
```

48

Chương trình đầu tiên với List đơn

- Tạo cấu trúc node và list tương ứng (**struct node, struct list**)
- Tạo ra một list rỗng (**Init()**).
- Yêu cầu người dùng nhập vào số node của list. Nhập vào từng node trong n của list (**inPut()**)
  - Tạo ra một node
  - Nhập giá trị info vào node vừa tạo (**GetNode()**).
  - Thêm node vào list bằng cách thêm vào đầu (**addHead()**).
- In ra dslk đơn vừa tạo (**xuat()**).

49

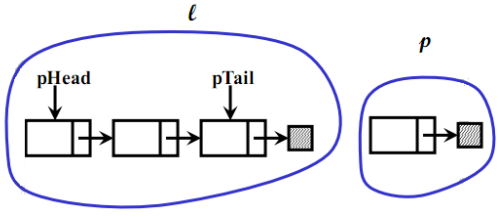
Chương trình đầu tiên với List đơn

Bài toán: Viết chương trình thực hiện các yêu cầu sau:

- Nhập dslk đơn các số nguyên.
- Tính tổng các giá trị trong dslk đơn.
- Xuất dslk đơn.

50

Thêm một node vào cuối List đơn



51

Thêm một node vào cuối List đơn

- Ta cần thêm **nút p** vào **cuối list đơn**

Bắt đầu:

Nếu **List rỗng** thì

+ pHead = p;
+ pTail = pHead;

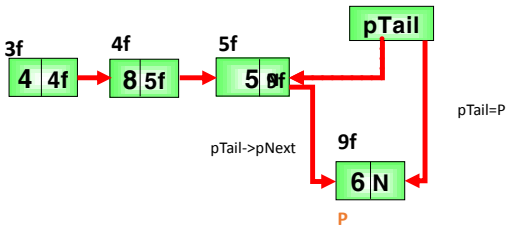
Ngược lại

+ pTail->pNext=p;
+ pTail=p

Thêm một node vào cuối List đơn

```
void AddTail(LIST &l, Node *p)
{
    if (l.pHead==NULL)
    {
        l.pHead = p;
        l.pTail = l.pHead;
    }
    else
    {
        l.pTail->Next = p;
        l.pTail = p;
    }
}
```

Thêm một node vào cuối List đơn



Thêm node p vào sau node q

➢Ta cần thêm nút p vào sau nút q trong list đơn

Bắt đầu:

Nếu (q!=NULL) thì

B1: p->pNext = q->pNext

B2:

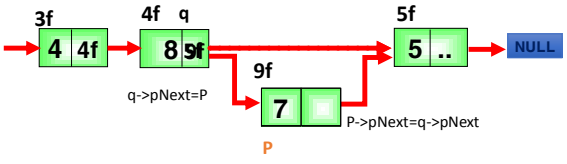
+ q->pNext = p

+ nếu q = pTail thì pTail=p

Thêm node p vào sau node q

```
void InsertAfterQ(List &l, Node *p, Node *q)
{
  if(q!=NULL)
  {
      p->pNext=q->pNext;
      q->pNext=p;
      if(l.pTail==q)
      l.Tail=p;
  }
  else
      AddHead(l,p);// thêm p vào đầu list
}
```

Thêm node p vào sau node q



Hủy phần tử trong List đơn

➢**Nguyên tắc**: Phải **cô lập phần tử** cần hủy trước hủy.

➢Các vị trí cần hủy

- **Hủy phần tử đứng** đầu List
- Hủy phần tử **có khoá bằng x**
- **Huỷ phần tử đứng sau q** trong danh sách liên kết đơn

➢Ở phần trên, các phần tử trong DSLK đơn được **cấp phát vùng nhớ động bằng hàm new**, thì sẽ được **giải phóng vùng nhớ bằng hàm delete**.

Hủy phần tử **đầu** trong List

➢ Bắt đầu:

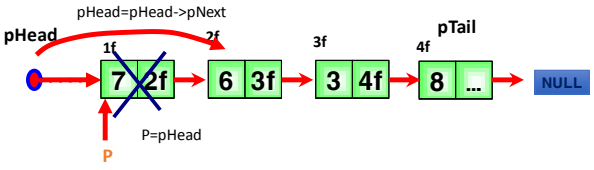
- Nếu (pHead!=NULL) thì
- B1: p=pHead
- B2:
  + pHead = pHead->pNext // p->pNext
  + delete (p)
- B3:
  Nếu pHead==NULL thì pTail=NULL
  // Danh sách ban đầu chỉ có 1 phần tử

Hủy phần tử **đầu** trong List

➢Hủy được hàm trả về 1, ngược lại hàm trả về 0

```
int  RemoveHead(List &l, int &x)
 {     Node *p;
       if(l.pHead!=NULL)
       {       p=l.pHead;
               x=p->Info; //lưu Data của nút cần hủy
               l.pHead=l.pHead->pNext; // l.pHead = p->pNext
               delete p;
               if(l.pHead==NULL)
                       l.pTail=NULL;
               return 1;
       }
       return 0;
 }
```

Hủy phần tử **đầu** trong List



Hủy phần tử **sau phân tử q** trong List

➢Bắt đầu

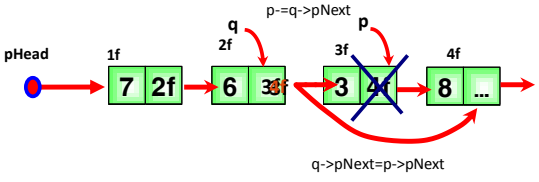
Nếu (q!=NULL) thì //q tồn tại trong List

- B1: p=q->pNext;// p là phần tử cần hủy
- B2: Nếu (p!=NULL) thì // q không phải là phần tử cuối
  - + q->pNext=p->pNext;// tách p ra khỏi xâu
  - + nếu (p== pTail) // nút cần hủy là nút cuối
    pTail=q;
  - + delete p;// hủy p

Hủy phần tử **sau phân tử q** trong List

```
int RemoveAfterQ(List &l, Node *q, int &x)
{     Node *p;
      if(q!=NULL)
      {       p=q->pNext; //p là nút cần xoá
              if(p!=NULL) // q không phải là nút cuối
              {       if(p==l.pTail) //nút cần xoá là nút cuối cùng
                              l.pTail=q;// cập nhật lại pTail
                      q->pNext=p->pNext; x=p->Info;
                      delete p;
              }
              return 1;
      }
      else
         return 0;
}
```

Hủy phần tử **sau phân tử q** trong List



Hủy phần tử **có khóa x**

Bước 1:

Tìm phần tử p có khoá bằng x, và q đứng trước p

Bước 2:

Nếu (p!=NULL) thì //tìm thấy phần tử có khoá bằng x

Hủy p ra khỏi List bằng cách hủy phần tử đứng sau q

Ngược lại

Báo không tìm thấy phần tử có khoá

Hủy phần tử **có khóa x**

```
int RemoveX(List &l, int x)
{ Node *p,*q = NULL; p=l.Head;
  while((p!=NULL)&&(p->Info!=x)) //tìm
  {    q=p;
       p=p->Next;
  }
  if(p==NULL) //không tìm thấy phần tử có khoá bằng x
       return 0;
  if(q!=NULL)//tìm thấy phần tử có khoá bằng x
       DeleteAfterQ(l,q,x);
  else //phần tử cần xoá nằm đầu List
       RemoveHead(l,x);
  return 1;
}
```

Tìm 1 phần tử trong DSLK đơn

➢Tìm tuần tự (hàm trả về), các bước của thuật toán tìm nút có Info bằng x trong list đơn

Bước 1: p=pHead;// địa chỉ của phần tử đầu trong list đơn

Bước 2:

Trong khi p!=NULL và p->Info!=x

p=p->pNext;// xét phần tử kế

Bước 3:

+ Nếu p!=NULL thì p lưu địa chỉ của nút có Info = x

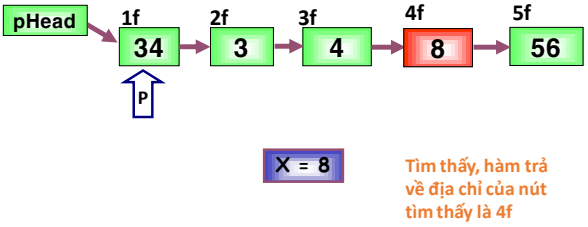
+ Ngược lại : Không có phần tử cần tìm

Tìm 1 phần tử trong DSLK đơn

➢ Hàm tìm phần tử có Info = x, hàm trả về địa chỉ của nút có Info = x, ngược lại hàm trả về NULL

```
Node *Search(LIST l, Data x)
{
  Node      *p;
  p = l.pHead;
  while((p!= NULL)&&(p->Info != x))
      p = p->pNext;
  return p;
}
```

Tìm 1 phần tử trong DSLK đơn



Hủy DSLK đơn

▪Bước 1:

Trong khi (danh sách chưa hết) thực hiện

•B11:

p = pHead;

pHead = pHead->pNext;// cập nhật pHead

•B12:

Hủy p

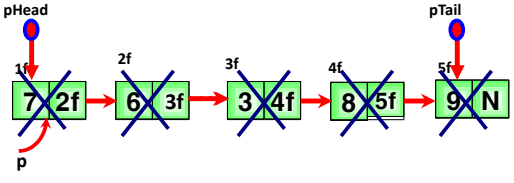
▪Bước 2:

pTail = NULL;// bảo toàn tính nhất quán khi xâu rỗng

Hủy DSLK đơn

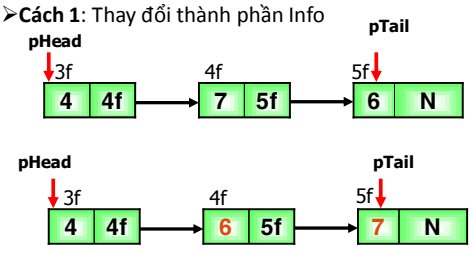
```
void RemoveList(List &l)
{
    Node *p;
    while(l.pHead!=NULL)//còn phần tử trong List
    {
        p = l.pHead;
        l.pHead = p->pNext;
        delete p;
    }
}
```

Hủy DSLK đơn



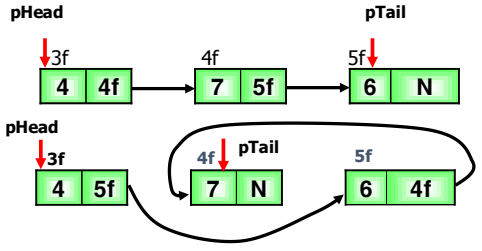
Sắp xếpDSLK đơn

- Có hai cách tiếp cận
- **Cách 1**: Thay đổi thành phần Info



Sắp xếp DSLK đơn

- **Cách 2**: Thay đổi thành phần pNext (thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn)



Sắp xếp DSLK đơn
Ưu – Nhược điểm

- Thay đổi thành phần **Info (dữ liệu)**
  - Ưu: **Cài đặt đơn giản, tương tự như sắp xếp mảng**
  - Nhược:
    - Đòi hỏi thêm vùng nhớ khi hoán vị nội dung của 2 phần tử → chỉ **phù hợp với những xâu có kích thước Info nhỏ**
    - Khi kích thước Info (dữ liệu) lớn chi phí cho việc hoán vị thành phần Info lớn →**làm cho thao tác sắp xếp chậm**
- Thay đổi thành phần pNext
  - Ưu:
    - Kích thước của trường này **không thay đổi**, do đó **không phụ thuộc vào kích thước bản chất dữ liệu** lưu tại mỗi nút → **thao tác sắp xếp nhanh**
  - Nhược: **Cài đặt phức tạp**

Sắp xếp DSLK đơn Selection Sort

```
void SelectionSort(LIST &l)
{
  Node *p,*q,*min;
  p=l.pHead;
  while(p!=l.pTail)
  {
      min=p;
      q=p->pNext;
```

```
while(q!=NULL)
{
      if(q->Info<p->Info)
            min=q;
      q=q->pNext;
      }
      HV(min->Info,p->Info);
      p=p->pNext;
      }
  }
```

Sắp xếp DSLK đơn

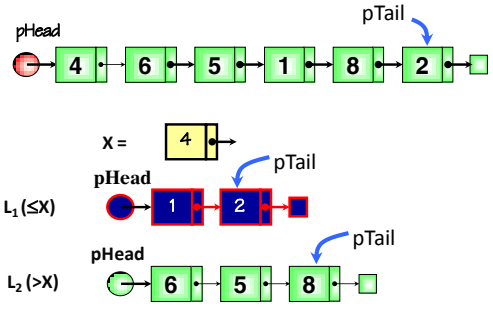
- Các thuật toán sắp xếp xâu (List) bằng các thay đổi thành phần pNext (thành phần liên kết) có hiệu quả cao như:
  - Thuật toán sắp xếp Quick Sort
  - Thuật toán sắp xếp Merge Sort
  - Thuật toán sắp xếp Radix Sort

Sắp xếp DSLK đơn – Quick Sort

- Bước 1:
  Chọn X là phần tử đầu xâu L làm phần tử cầm canh
  Loại X ra khỏi L
- Bước 2:
  Tách xâu L ra làm 2 xâu L_1(gồm các phần tử nhỏ hơn hoặc bằng x) và L_2 (gồm các phần tử lớn hơn X)
- Bước 3: Nếu (L_1 !=NULL) thì QuickSort(L_1)
- Bước 4: Nếu (L_2!=NULL) thì QuickSort(L_2)
- Bước 5: Nối L_1, X, L_2 lại theo thứ tự ta có xâu L đã được sắp xếp

Sắp xếp DSLK đơn – Quick Sort

➢Cho danh sách liên kết gồm các phần tử sau:

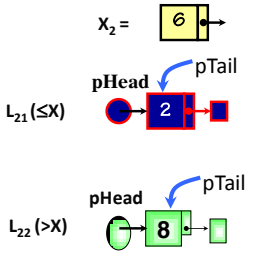


Sắp xếp DSLK đơn – Quick Sort

➢Sắp xếp L_1

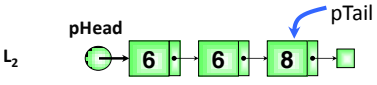
➢Sắp xếp L_2

- Chọn x=6 cầm canh, và tách L_2 thành L_{21} và L_{22}

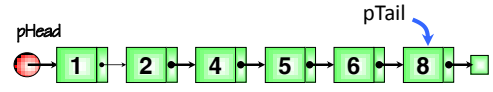


Sắp xếp DSLK đơn – Quick Sort

➢Nối L_{21}, X_2, L_{22} thành L_2



➢Nối L1, X, L2 thành L



Sắp xếp DSLK đơn – Quick Sort

```
void QuickSort(List &l)
{ Node *p,*X;//X lưu địa chỉ của phần tử cầm canh
  List l1,l2;
  if(l.pHead==l.pTail) return;//đã có thứ tự
  CreateList(l1);
  CreateList(l2);
  X=l.pHead;
  l.pHead=X->pNext;
  while(l.pHead!=NULL)//tách L = L1 va L2
  {     p=l.pHead;
        l.pHead=p->pNext;
        p->pNext=NULL;
        if(p->Info<=X->Info)
              AddHead(l1,p);
        else
              AddHead(l2,p);
  }
```

Sắp xếp DSLK đơn – Quick Sort

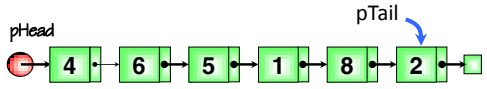
```
QuickSort(l1);//Gọi đệ quy sắp xếp L1
QuickSort(l2);//Gọi đệ quy sắp xếp L2
if(l1.pHead!=NULL)//nối l1, l2 va X vao l
{
      l.pHead=l1.pHead;
      l1.pTail->pNext=X;//nối X vào
}
else
      l.pHead=X;
X->pNext=l2.pHead;
if(l2.pHead!=NULL) //l2 có trên một phần tử
      l.pTail=l2.pTail;
else  //l2 không có phần tử nào
      l.pTail=X;
}
```

Sắp xếp DSLK đơn – Merge Sort

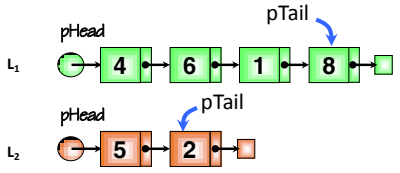
- Bước 1: Phân phối luân phiên từng đường chạy của xâu L vào 2 xâu con L_1 và L_2.
- Bước 2: Nếu L_1 != NULL thì Merge Sort (L_1).
- Bước 3: Nếu L_2 != NULL thì Merge Sort (L_2).
- Bước 4: Trộn L_1 và L_2 đã sắp xếp lại ta có xâu L đã được sắp xếp.
- Không tốn thêm không gian lưu trữ cho các dãy phụ

Sắp xếp DSLK đơn – Merge Sort

➢Cho danh sách liên kết gồm các phần tử sau:



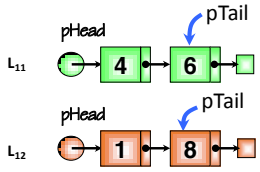
➢Phân phối các đường chạy của L_1 vào L_1, L_2



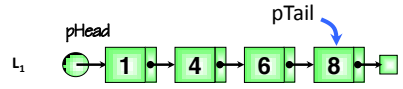
Sắp xếp DSLK đơn – Merge Sort

➢Sắp xếp L_1

- Phân phối các đường chạy L_1 vào L_{11}, L_{12}



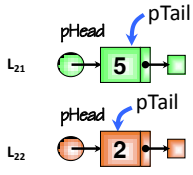
- Trộn L_{11} và L_{12} vào L_1



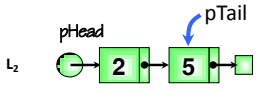
Sắp xếp DSLK đơn – Merge Sort

➢Sắp xếp L_2

- Phân phối các đường chạy của L_2 vào L_{21}, L_{22}

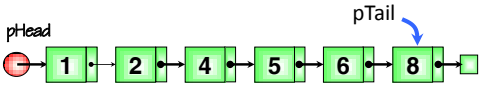


➢Trộn L_{21}, L_{22} thành L_2



Sắp xếp DSLK đơn – Merge Sort

➢Trộn L_1, L_2 thành L



Bài tập

➢Yêu cầu: Thông tin của một sinh viên gồm, mã số sinh viên, tên sinh viên, điểm trung bình.

1. Hãy khai báo cấu trúc dữ liệu dạng danh sách liên kết để lưu danh sách sinh viên nói trên.
2. Nhập danh sách các sinh viên, và thêm từng sinh viên vào đầu danh sách (việc nhập kết thúc khi tên của một sinh viên bằng rỗng)
3. Tìm một sinh viên có trong lớp học hay không
4. Xoá một sinh viên có mã số bằng x (x nhập từ bàn phím)
5. Liệt kê thông tin của các sinh viên có điểm trung bình lớn hơn hay bằng 5.

Bài tập

6. Xếp loại và in ra thông tin của từng sinh viên, biết rằng cách xếp loại như sau:
   - ĐTB <=3.6 : Loại yếu
   - ĐTB>=50 và ĐTB<6.5 : Loại trung bình
   - ĐTB>=6.5 và ĐTB < 7.0: Loại trung bình khá
   - ĐTB>=7.0 và ĐTB <8.0: Loại khá
   - ĐTB>=8.0 và ĐTB < 9.0: Loại giỏi.
   - ĐTB>=9.0 : Loại xuất sắc
7. Sắp xếp và in ra danh sách sinh viên tăng theo điểm trung bình.
8. Chèn một sinh viên vào danh sách sinh viên tăng theo điểm trung bình nói trên, sao cho sau khi chèn danh sách sinh viên vẫn tăng theo điểm trung bình

..vv

Slide được tham khảo từ

- **Slide được tham khảo từ:**
  - Slide CTDL GT, Khoa Khoa Học Máy Tính, ĐHCNTT
  - Slide CTDL GT, Thầy Nguyễn Tấn Trần Minh Khang, ĐH CNTT
  - Congdongcviet.com
  - Cplusplus.com

91

92