CẤU TRÚC TỰ TRỎ VÀ DANH SÁCH LIÊN KẾT

CẤU TRÚC TỰ TRỎ

- **Khái niệm**: là kiểu cấu trúc mà trong các thành phần của nó có 1 trường con trỏ, trỏ đến chính kiểu cấu trúc đó.
- **Ví dụ:**

```
6  struct Sinhvien{
7      string ht;
8      float diem;
9      Sinhvien *tiep;
10 };
```

CẤU TRÚC TỰ TRỎ

- Cách 1

```
typedef struct Têncấutrúc tênbiếncấutrúc;
struct Têncấutrúc {
Khai báo các thuộc tính;
tênbiếncấutrúc *contrỏ;
};
```

- Cách 2

```
struct Têncấutrúc {
Khai báo các thuộc tính;
Têncấutrúc *contrỏ;
};
typedef Têncấutrúc tênbiếncấutrúc;
```

Cách 3

```
typedef struct tênbiếncấutrúc {
Khai báo các thuộc tính;
tênbiếncấutrúc *contrỏ;
};
```

Cách 4

```
struct tênbiếncấutrúc {
Khai báo các thuộc tính;
tênbiếncấutrúc *contrỏ;
};
```

CẤU TRÚC TỰ TRỎ


```
13  typedef struct Sinhvien sv
14  struct Sinhvien{
15      string ht;
16      float diem;
17      sv *tiep;
18  };
```

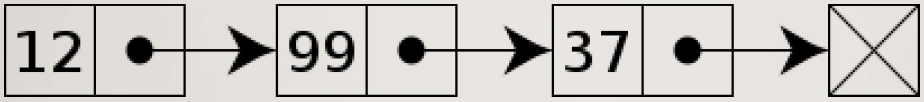
```
28  typedef struct sv
29  {
30      string ht;
31      float diem;
32      sv *tiep;
33  };
```

```
21  struct Sinhvien{
22      string ht;
23      float diem;
24      Sinhvien *tiep;
25  };
26  typedef struct Sinhvien sv
```

```
 6  struct Sinhvien{
 7      string ht;
 8      float diem;
 9      Sinhvien *tiep;
10  };
```

CÁC KIỂU DANH SÁCH LIÊN KẾT

1. Danh sách liên kết đơn (Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử ngay phía sau.

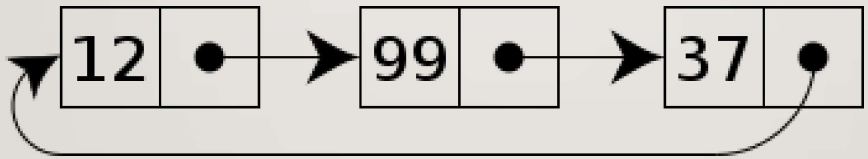


CÁC KIỂU DANH SÁCH LIÊN KẾT

2. Danh sách liên kết đôi (Double linked list): Có sự kết nối 2 chiều giữa phần tử ngay trước với phần tử ngay sau

CÁC KIỂU DANH SÁCH LIÊN KẾT

3. Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.



KHÁI NIỆM DANH SÁCH LIÊN KẾT ĐƠN

- **Danh sách liên kết đơn** (Single Linked List) là một cấu trúc dữ liệu động, nó là một danh sách mà mỗi phần tử đều liên kết với phần tử đứng ngay sau nó trong danh sách.
- Mỗi phần tử trong danh sách liên kết đơn (gọi là một node hay nút) là một cấu trúc gồm 2 thành phần:
  - Thành phần dữ liệu: lưu thông tin về bản thân phần tử đó.
  - Thành phần liên kết: lưu địa chỉ phần tử đứng ngay sau trong danh sách, nếu phần tử đó là phần tử cuối cùng thì thành phần này bằng NULL.

MẢNG VÀ DANH SÁCH LIÊN KẾT ĐƠN

Nội dung	Mảng	Danh sách liên kết
Kích thước	Kích thước cố định Cần chỉ rõ kích thước khi khai báo	Kích thước thay đổi trong quá trình thêm/xóa phần tử
Cấp phát bộ nhớ	Tĩnh. Bộ nhớ được cấp phát trong quá trình biên dịch	Động. Bộ nhớ được cấp phát trong quá trình chạy CT
Thứ tự và sắp xếp	Lưu trữ trên 1 dạy ô nhớ liên tiếp	Lưu trữ trên các ô nhớ ngẫu nhiên
Truy cập	Truy cập được tới phần tử ngẫu nhiên thông qua chỉ số	Truy cập tới phần tử phải duyệt từ đầu đến cuối
Tìm kiếm	Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân	Tìm kiếm tuyến tính

ĐẶC ĐIỂM CỦA DSLK ĐƠN

- Do tính liên kết của phần tử đầu và phần tử đứng ngay sau nó trong DSLK đơn nên nó có các đặc điểm sau:
  - Chỉ cần nắm được phần tử đầu là có thể quản lý được danh sách
  - Muốn truy cập tới phần tử ngẫu nhiên phải duyệt từ đầu đến vị trí đó
  - Chỉ có thể tìm kiếm tuyến tính một phần tử

CÁC PHÉP TOÁN TRÊN DSLK ĐƠN

- Tạo nút
- Tạo DSLK
- Thêm phần tử vào DSLK
- Xóa phần tử trong DSLK
- Tìm kiếm phần tử trong DSLK
- Duyệt DSLK

TẠO NÚT (NODE)

- **Định nghĩa nút**

```
struct Nut
{
   Kieudulieu  Dulieu;
   Nut *tiep ;
};
```

- Khai báo trên sẽ được sử dụng cho mọi nút trong danh sách liên kết.
- Trường Dulieu sẽ chứa giá trị
- Trường tiep sẽ là con trỏ để trỏ đến nút kế tiếp

TẠO NÚT (NODE)

Tạo nút mới:

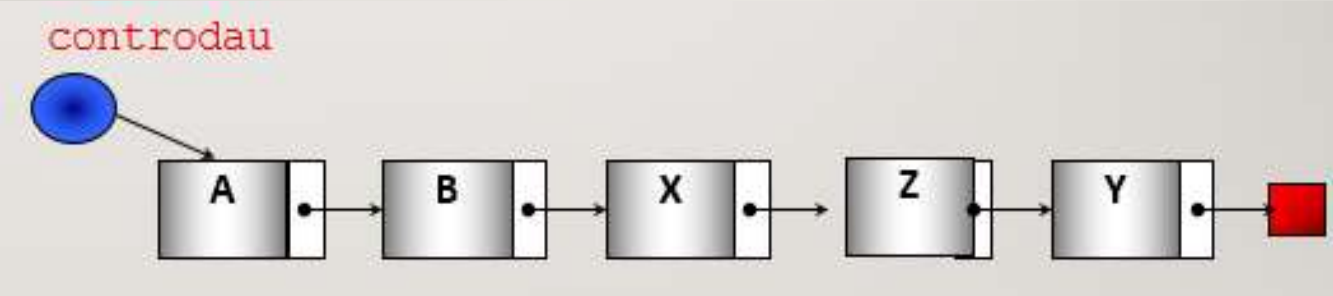
- Thực hiện cấp phát động nút mới.
- Khởi tạo giá trị ban đầu cho nút
- Nút vừa tạo chưa có liên kết với phần tử nào cả. Do đó, phần liên kết của nó bằng NULL
- Trả về địa chỉ con trỏ, trỏ đến nút mới.

```
struct Nut
{
    Kieudulieu  Dulieu;
    Nut *tiep;
};
```

```
Nut *Taonut(Kieudulieu  Giatri)
{
  Nut  *nutmoi = new Nut;
  nutmoi -> dulieu = Giatri;
  nutmoi-> tiep = NULL;
  return nutmoi;
}
```

TẠO DANH SÁCH LIÊN KẾT ĐƠN

- Vì thành phần tạo nên DSLK đơn là **nút** nên cần quản lý chúng bằng cách biết được phần tử đầu DSLK.
- Vì mỗi phần tử đều liên kết với phần tử kế tiếp, vì vậy cần dùng 1 con trỏ lưu trữ địa chỉ phần tử đầu (head)

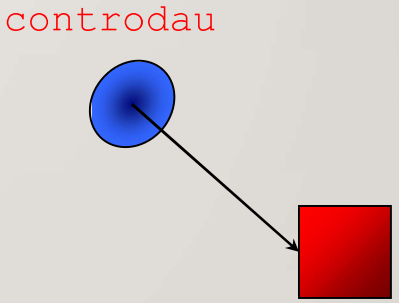


TẠO DANH SÁCH LIÊN KẾT ĐƠN

- Khi mới tạo danh sách, danh sách sẽ không có phần tử nào. Do đó, phần tử đầu không trỏ vào đâu cả nên chúng được gán bằng NULL.

- **Hàm tạo DS rỗng (chưa có nút nào)**

```
void Taods(Nut * &controdau){
   controdau = NULL;
}
```



THÊM NÚT VÀO DANH SÁCH

- **Thêm một nút vào danh sách**: Có 3 vị trí thêm
  - Thêm vào đầu danh sách
  - Thêm vào cuối danh sách
  - Thêm vào sau nút q trong danh sách
- Lưu ý trường hợp danh sách ban đầu rỗng

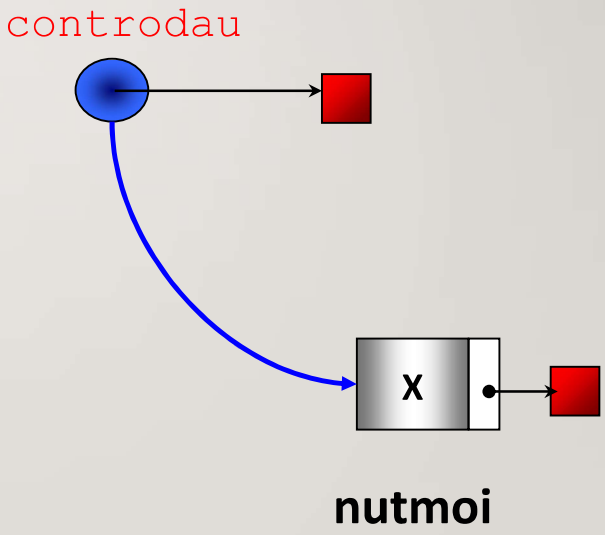
THÊM NÚT VÀO DANH SÁCH

- **Thêm vào đầu DSLK đơn**

a. Nếu DS ban đầu rỗng:

- Tạo 1 nút mới
- Con trỏ đầu trỏ vào phần tử này

```
Controdau = nutmoi;
```



THÊM NÚT VÀO DANH SÁCH

- **Thêm vào đầu DSLK đơn**

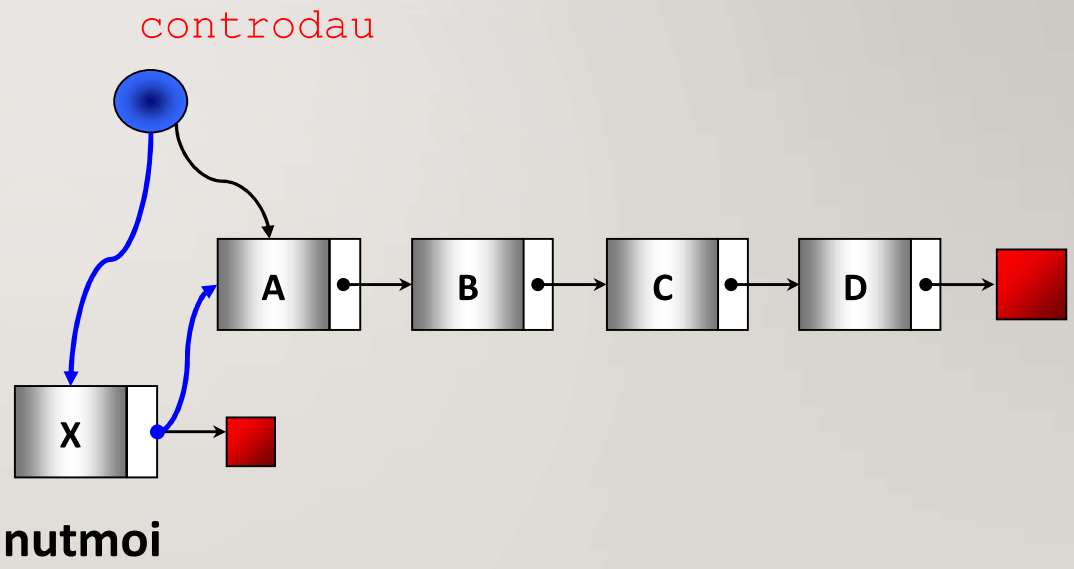
b. Nếu DS ban đầu khác rỗng:

- Cho liên kết của nút mới trỏ tới nút đầu DS cũ

  nutmoi -> **tiep**= **`controdau`**;

- Cập nhật lại con trỏ đầu trỏ tới nút mới

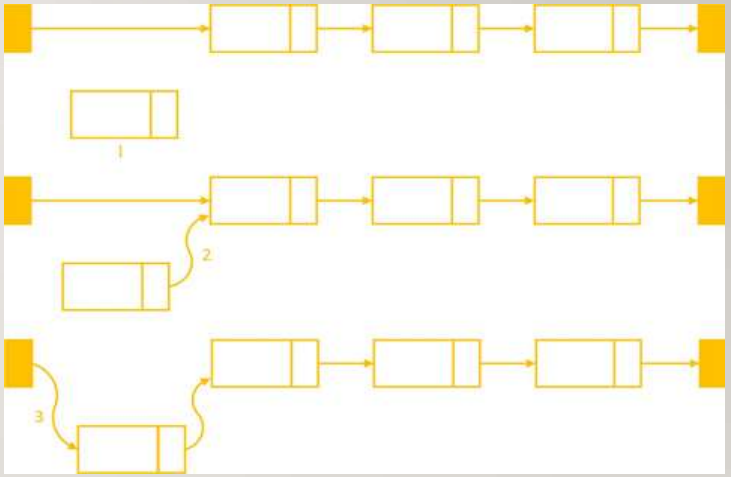
  Controdau = **nutmoi** ;



THÊM NÚT VÀO DANH SÁCH


```
void Themvaodau(Nut *&controdau, Nut *p)
{
    if (controdau == NULL)
    {
      controdau = p;

    }
    else
    {
       p ->tiep = controdau;
       controdau = p;
    }
}
```



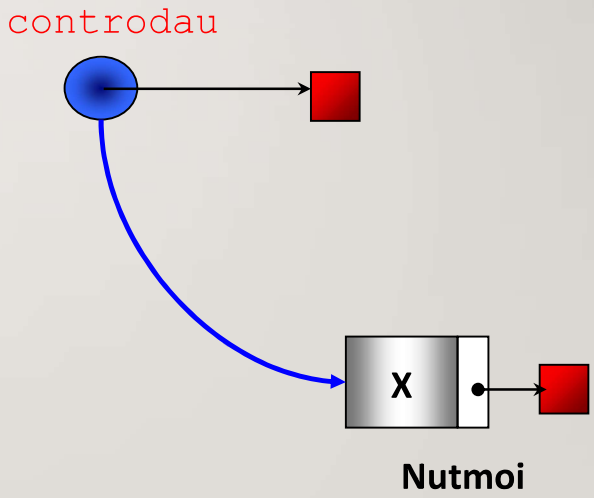
THÊM NÚT VÀO DANH SÁCH

- **Thêm vào cuối DSLK đơn**

  a. Nếu DS ban đầu rỗng:

  - Tạo 1 nút mới
  - Con trỏ đầu trỏ vào phần tử này

```
Controdau = nutmoi;
```



THÊM NÚT VÀO DANH SÁCH

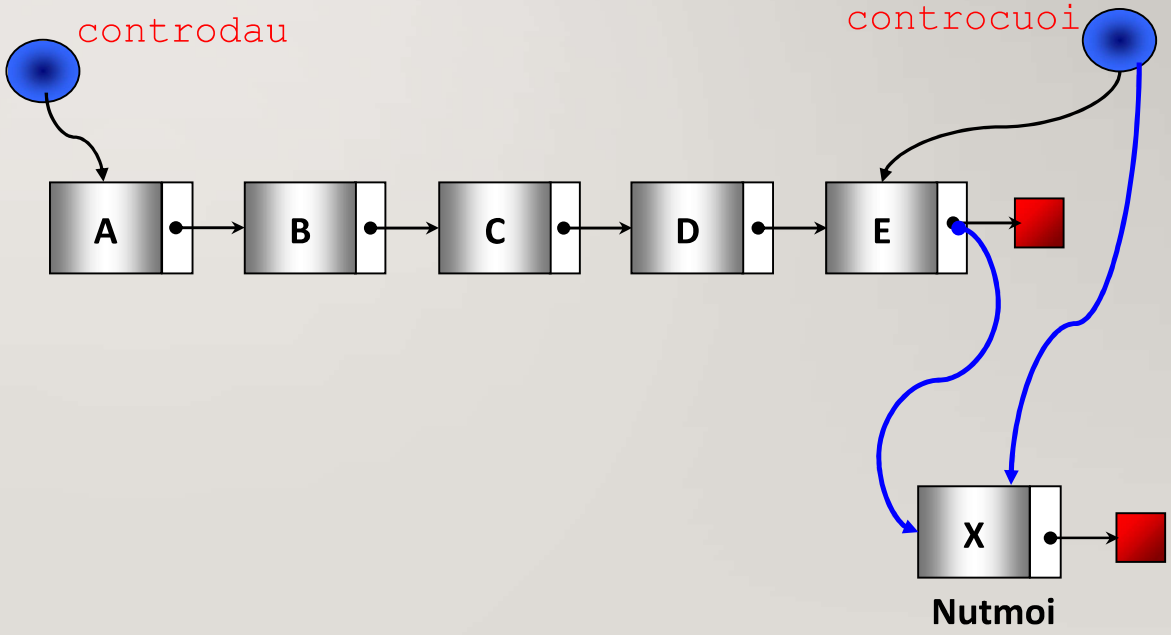
- **Thêm vào cuối DSLK đơn**

b. Nếu DS ban đầu khác rỗng:

- Dùng controcuoi trỏ đến nút cuối của DSLK đơn.
- Cho liên kết của con trỏ cuối trỏ tới nút mới

  controcuoi -> **tiep**= `nutmoi`;
- Cập nhật lại con trỏ cuối

  controcuoi = **nutmoi** ;

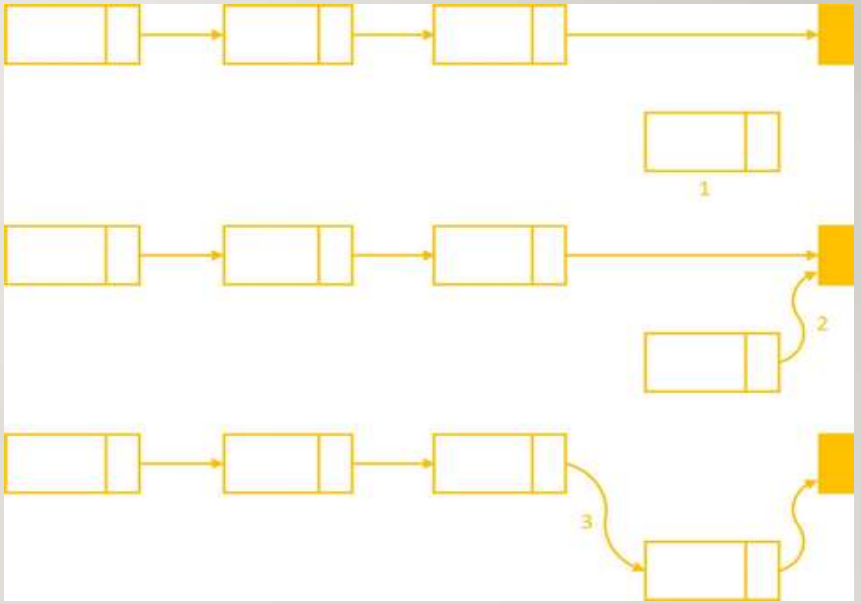


THÊM PHẦN TỬ VÀO DANH SÁCH


```
//chay controcuoi den nut cuoi cua DS
Nut *controcuoi;
 controcuoi=controdau;
 while (controcuoi-> tiep!=NULL)
    controcuoi=controcuoi->tiep;
```

THÊM PHẦN TỬ VÀO DANH SÁCH


```
void Themvaocuoi(Nut *controdau, Nut *p)
{
 Nut *controcuoi; //la con tro tro den nut cuoi của DS
if (controdau == NULL) //DS rỗng
 {
      controdau = p;
 }
 else
 {
      controcuoi->tiep = p;
      controcuoi=p;
  }
}
```



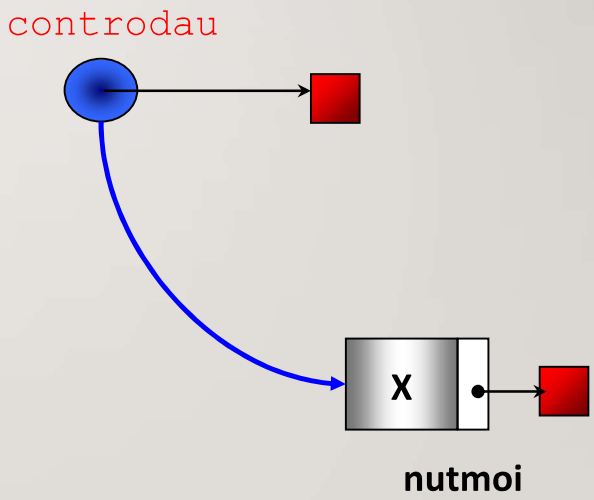
THÊM NÚT VÀO DANH SÁCH

- **Thêm vào sau nút q trong DSLK đơn**

a. Nếu DS ban đầu rỗng:

- Tạo 1 nút mới
- Con trỏ đầu trỏ vào phần tử này

```
controdau = nutmoi;
```



THÊM NÚT VÀO DANH SÁCH

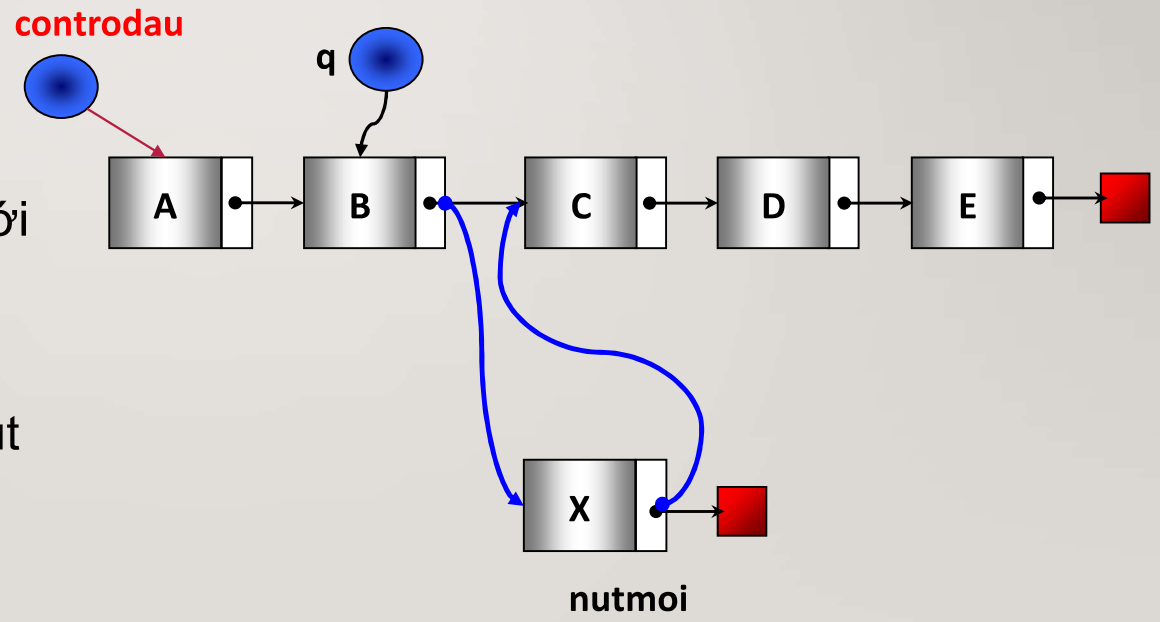
- **Thêm vào sau nút q trong DSLK đơn**

b. Nếu DS ban đầu khác rỗng:

- Tạo 1 nút mới
- Chèn nút mới vào sau q
  - Cho liên kết ở nút mới trỏ tới liên kết của q

  Nutmoi->tiep = q->tiep
  - Cho liên kết của q trỏ tới nút mới

  q->tiep = nutmoi



THÊM NÚT VÀO DANH SÁCH

- **Thêm vào sau nút q trong DSLK đơn 1 nút p**

```
void Themvaosau(Nut *controdau, Nut *q,  Nut * p)
{
      if (q!=NULL)
      {
            p->tiep = q->tiep;
            q->tiep = p;
      }
}
```

BÀI TẬP

Bài 2:

- Viết chương trình tạo 1 danh sách liên kết đơn dùng để lưu trữ n số nguyên x, với n và x nhập từ bàn phím.
- Danh sách liên kết trên được tạo nên từ các nút bằng cách thêm vào **đầu** danh sách.
- Viết hàm tính và thông báo ra màn hình tổng các số trong danh sách trên.
- *Chú ý: Trong chương trình sử dụng hàm Tạo nút, tạo danh sách rỗng, thêm 1 nút vào đầu danh sách.*

BÀI TẬP

Bài 3:

- Viết chương trình tạo 1 danh sách liên kết đơn dùng để lưu trữ n số nguyên x, với n và x nhập từ bàn phím.
- Danh sách liên kết trên được tạo nên từ các nút bằng cách thêm vào **cuối** danh sách.
- Đếm số lượng các số chẵn trong danh sách trên.
- *Chú ý: Trong chương trình sử dụng hàm Tạo nút, Tạo danh sách rỗng, Thêm 1 nút vào cuối danh sách.*

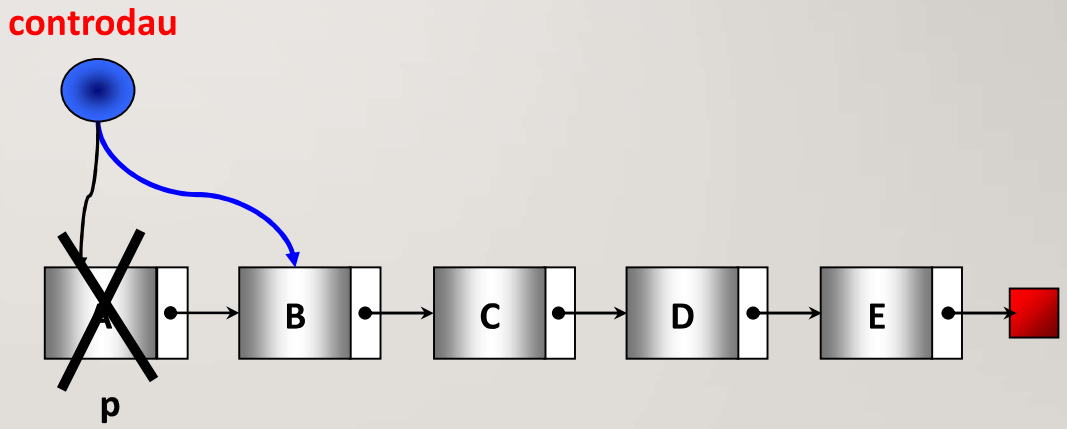
XÓA KHỎI DSLK 1 NÚT

- Xóa một nút của danh sách liên kết:
  - Xóa nút đầu của danh sách
  - Xóa nút sau nút q trong danh sách
  - Xóa nút có giá trị k (hoặc xóa 1 nút trỏ bởi con trỏ p)

XÓA KHỎI DSLK 1 NÚT

- **Xóa nút đầu của DSLK**
  - Gán p =controdau
  - Cho controdau trỏ vào nút sau nút p: controdau=p->tiep

  (hoặc controdau=controdau->tiep)
  - Giải phóng vùng nhớ mà p trỏ tới: `delete p`



XÓA KHỎI DSLK 1 NÚT

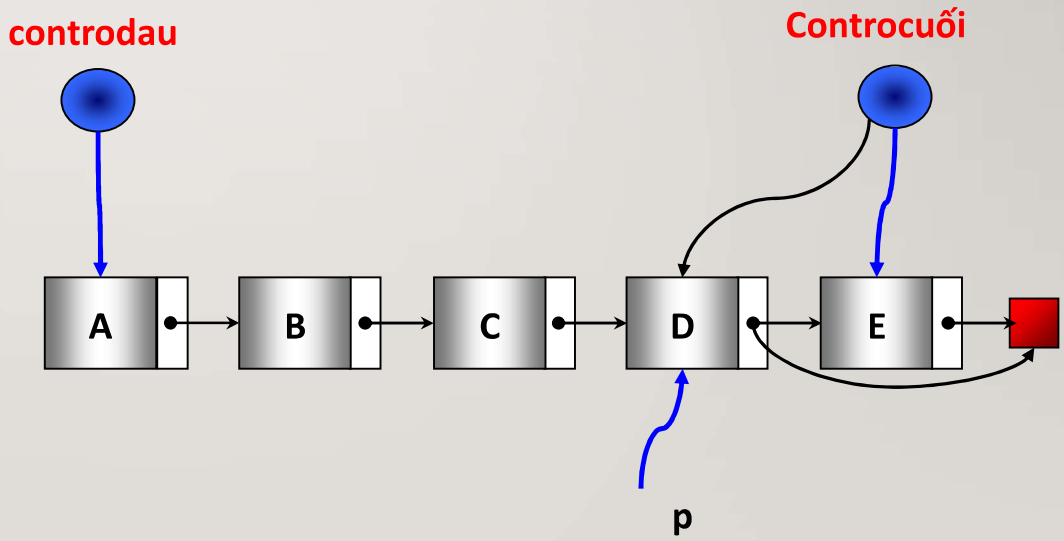

```
void  Xoadauds(Nut *controdau)
{  if controdau = NULL
        cout<<“Danh sach rong”;
   else
        {Nut* p= controdau;
        controdau = p->tiep;        //hoặc controdau = controdau->tiep;
        delete p;
        }
}
```

XÓA KHỎI DSLK 1 NÚT

- **Xóa nút cuối của DSLK**
  - Giả sử controcuoi trỏ vào nút cuối của DSLK
  - Gán p =controdau và chạy p đến trước nút cuối cùng.
  - Làm cho p trở thành nút cuối của DS: p->tiep=NULL
  - Giải phóng vùng nhớ mà controcuoi trỏ tới :

```
delete controcuoi
```

  - Cập nhật lại cotrocuoi: controcuoi=p

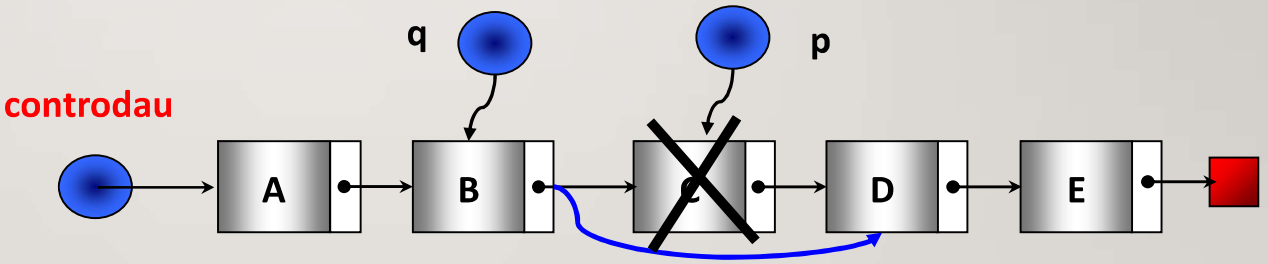


XÓA KHỎI DSLK 1 NÚT

- **Xóa nút sau nút q trong DSLK đơn**
  - Điều kiện để có thể xóa được nút sau q là:
    - q phải khác NULL (q !=NULL)
    - nút sau q phải khác NULL (q->tiep !=NULL)
  - Có các thao tác:
    - Gọi p là nút sau q
    - Cho vùng tiep của q trỏ vào nút đứng sau p
    - Nếu p là phần tử cuối thì q sẽ là phần tử cuối.
    - Giải phóng vùng nhớ mà p trỏ tới

XÓA KHỎI DSLK 1 NÚT

- q->tiep= p->tiep
- Giải phóng vùng nhớ mà p đang trỏ tới: delete p;



XÓA KHỎI DSLK 1 NÚT


```
void Xoasauq(Nut *controdau, Nut *q )
{
        if (q !=NULL && q->tiep !=NULL)
        {
                Nut *p = q->tiep;
                q->tiep = p->tiep;
                delete p;
        }
        else cout <<“ Khong co nut sau q”;
}
```

XÓA KHỎI DSLK 1 NÚT

- **Xóa nút trỏ bởi con trỏ p trong DSLK đơn**
  - Tìm được nút q đứng trước p
  - Đưa về bài toán xóa nút đứng sau q

DUYỆT DANH SÁCH LIÊN KẾT ĐƠN

- Là thao tác thường được thực hiện khi có nhu cầu xử lý các phần tử của DSLK hoặc khi cần lấy thông tin từ các phần tử của DSLK như:
  - Đếm các phần tử của danh sách
  - Tìm tất cả các phần tử thoả điều kiện nào đó.
  - …

DUYỆT DANH SÁCH LIÊN KẾT ĐƠN

- Bước 1: p = controdau; *//Cho p trỏ đến phần tử đầu danh sách*
- Bước 2: Trong khi (Danh sách chưa hết) thực hiện:
  - B2.1 : Xử lý phần tử được trỏ bởi p
  - B2.2 : p=p->tiep; *// Cho p trỏ tới phần tử kế*

DUYỆT DANH SÁCH LIÊN KẾT ĐƠN

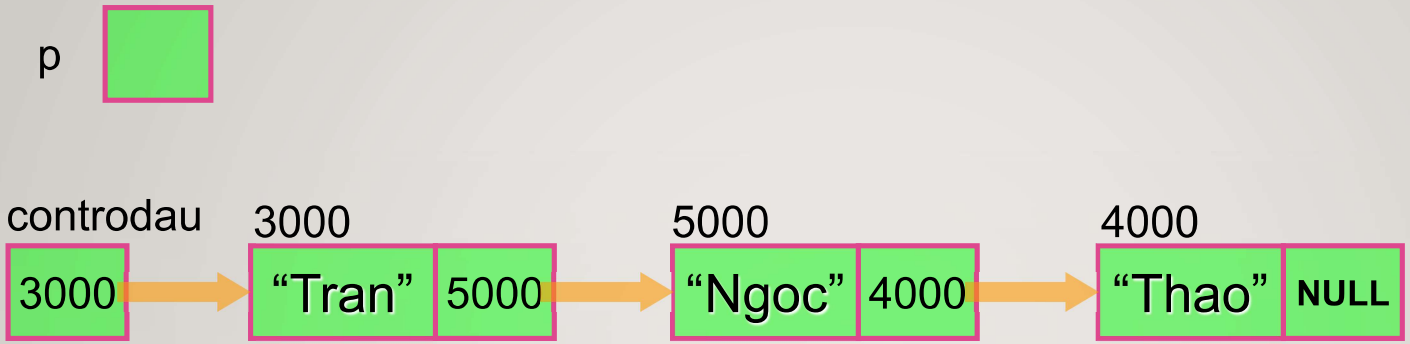
```
void duyetDSLK(Nut *controdau)
{
        Nut *p = controdau;
        while (p!= NULL)
        {
            // xử lý cụ thể p tùy ứng dụng
            p = p->tiep;
        }
}
```

DUYỆT DANH SÁCH LIÊN KẾT ĐƠN

Ví dụ: In các phần tử trong danh sách

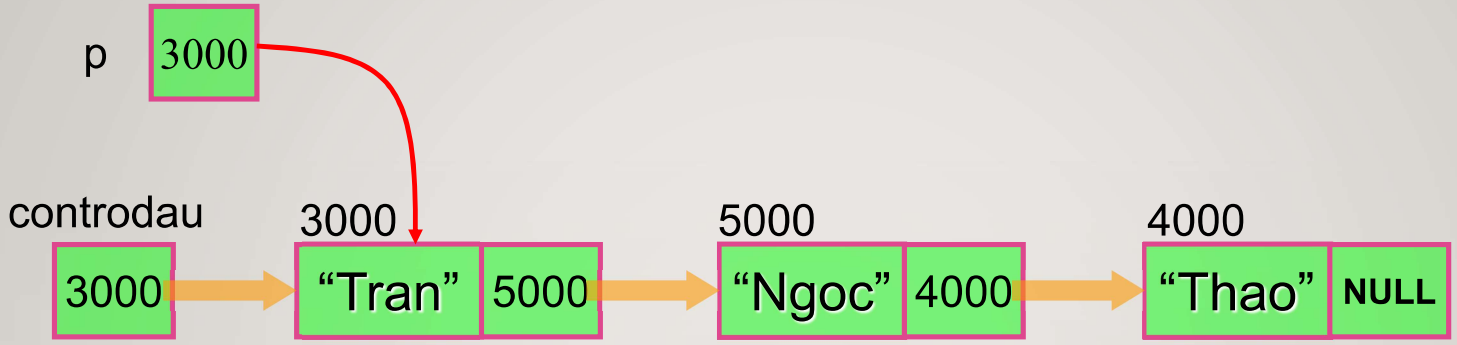
```
void Hienthi(Nut *controdau)
{
    Nut * p=controdau;
    while (p!=NULL)
    {
        cout << p->data << "\t";
        p=p ->tiep;
    }
    cout<<endl;
}
```

DUYỆT DANH SÁCH LIÊN KẾT ĐƠN




```
p = controdau;
    while (p!=NULL)
    {
        cout<<p->data<<“\t”;
        p = p->link;
    }
```

DUYỆT DANH SÁCH LIÊN KẾT ĐƠN




```
p =controdau;
    while (p!=NULL)
    {
        cout<<p->data<<"\t";
        p = p->link;
    }
```

TÌM KIẾM PHẦN TỬ TRONG DSLK ĐƠN

- **Tìm kiếm một phần tử có giá trị = x**


```
Nut *Tim (Nut *controdau, int  x)
{       if (controdau == NULL) return NULL;
        else
        {       Nut* p = controdau;
                while (p!=NULL)
                {   if (p->data == x)
                     return p;
                    else p=p->tiep;
                }
        }
}
```

BÀI TẬP

Bài 4: Cho cấu trúc

```
struct Sinhvien{
    string      ten;
    int         diem;
    Sinhvien    *tro;
};
```

1. Viết chương trình tạo 1 danh sách liên kết đơn dùng để lưu trữ thông tin về n sinh viên bằng cách thêm vào **cuối** danh sách.
2. In ra danh sách n sinh viên. Thông tin của mỗi sinh viên gồm tên và điểm trên cùng 1 dòng.
3. Đưa ra số lượng sinh viên có điểm >8
4. Nhập 1 tên cần tìm từ bàn phím. Tìm xem trong danh sách có sinh viên như vậy không? Nếu có thì xóa sinh viên đó khỏi danh sach.