



CHƯƠNG 7

Kiểu Cấu Trúc

GIẢNG VIÊN

TS. Hà Ngọc Long

Nội dung chính

1. Khái niệm cấu trúc

- Khái niệm

2. Khai báo cấu trúc

- Khai báo kiểu cấu trúc
- Khai báo biến cấu trúc
- Định nghĩa kiểu dữ liệu với **typedef**

3. Xử lý dữ liệu cấu trúc

- Truy cập các trường dữ liệu
- Phép gán giữa các biến cấu trúc
- Một số ví dụ

Ví dụ: Quản lý thí sinh thi Đại học (1/3)

- ❑ Để quản lý cần lưu trữ các thông tin
 - Số báo danh: Số nguyên không dấu
 - Họ tên sinh viên: Chuỗi ký tự không quá 30
 - Khối thi: Ký tự (A,B,C..)
 - Tổng điểm 3 môn thi: kiểu thực
- Do vậy với mỗi sinh viên cần các biến

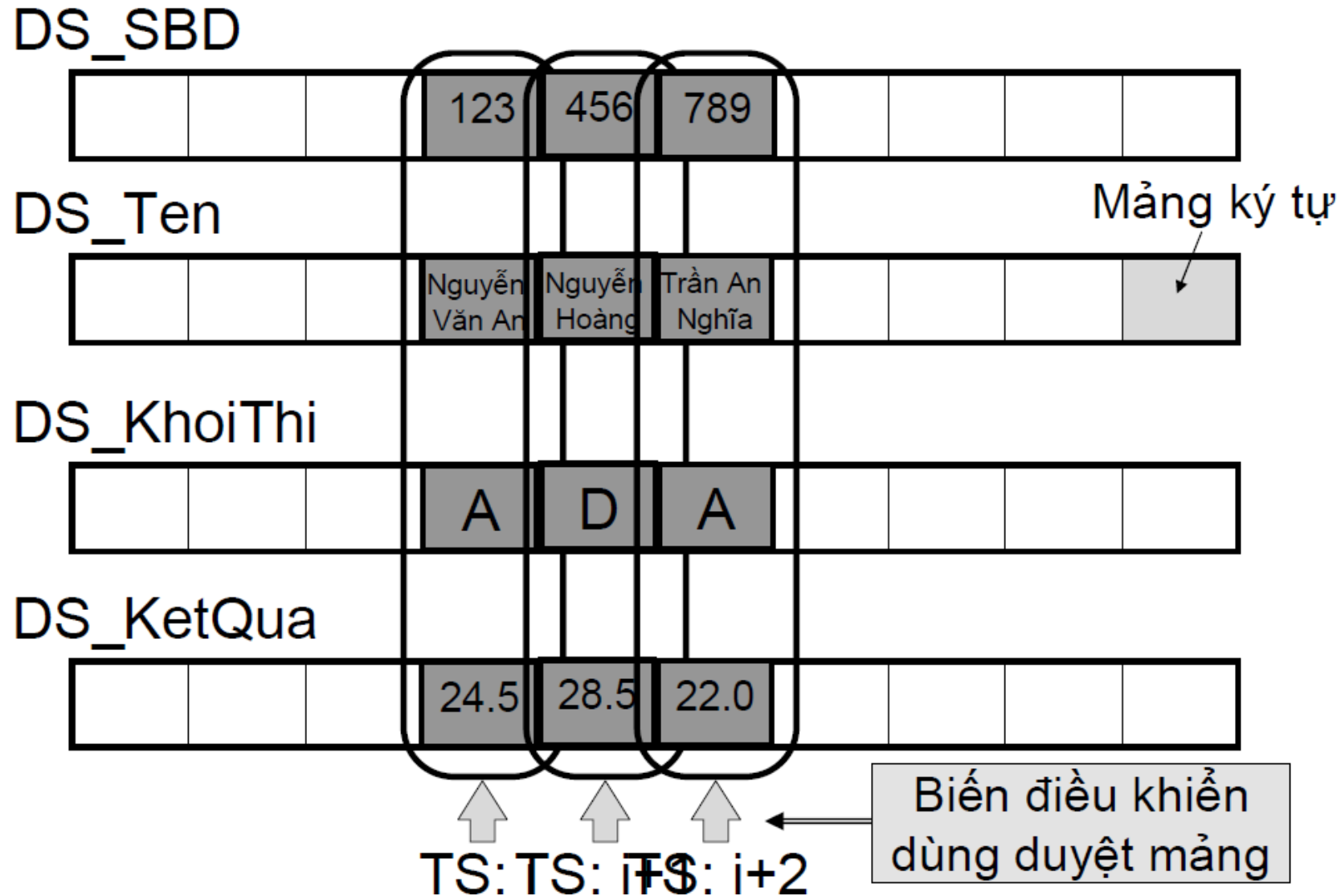
<code>unsigned</code>	<code>SBD;</code>
<code>char</code>	<code>Ten[30];</code>
<code>char</code>	<code>KhoiThi;</code>
<code>float</code>	<code>KetQua;</code>

Ví dụ: Quản lý thí sinh thi Đại học (2/3)

Để quản lý danh sách (dưới 1000) thí sinh dự thi, cần nhiều mảng rời rạc

```
#define MAX      1000  
  
unsigned        DS_SBD[MAX];  
  
char            DS_Ten[MAX][30];  
  
char            DS_KhoiThi[MAX];  
  
float           DS_KetQua[MAX];
```

Ví dụ: Quản lý thí sinh thi Đại học (3/3)



Vấn đề & Giải pháp

- Dùng nhiều mảng
- - Khó quản lý, dễ nhầm lẫn
- - Không thể hiện cấu trúc thông tin dành cho từng thí sinh



Khái niệm Cấu trúc

- **Cấu trúc** là *kiểu dữ liệu phức hợp*, do người dùng *tự định nghĩa*
 - Kiểu cấu trúc bao gồm nhiều thành phần có thể thuộc các kiểu dữ liệu khác nhau
 - Các thành phần: gọi là trường dữ liệu (**field**)
 - Các thành phần, không được truy nhập theo chỉ số (như mảng) mà theo tên của trường.
- Có thể coi một biến cấu trúc là một tập hợp của một hay nhiều biến rời rạc, thường có kiểu khác nhau thành một biến có một tên duy nhất để dễ dàng quản lý và sử dụng

Ví dụ

1) Kết quả học tập của sinh viên

- TenSV: Chuỗi ký tự
- MaSV: Chuỗi số/ số nguyên
- Điểm: Số thực

2) Điểm trong mặt phẳng

- Tên điểm: Ký tự (A, B, C..)
- Hoành độ: Số thực
- Tung độ: Số thực

Nội dung chính

1. Khái niệm cấu trúc

- Khái niệm

2. Khai báo cấu trúc

- Khai báo kiểu cấu trúc
- Khai báo biến cấu trúc
- Định nghĩa kiểu dữ liệu với **typedef**

3. Xử lý dữ liệu cấu trúc

- Truy cập các trường dữ liệu
- Phép gán giữa các biến cấu trúc
- Một số ví dụ

Khai báo kiểu cấu trúc

```
struct Tên_kiểu_cấu_trúc {  
    <Khai báo các trường dữ liệu>  
};
```

- **struct:** từ khóa, cho phép người dùng khai báo kiểu cấu trúc
- **Tên_kiểu_cấu_trúc:** Tên của kiểu cấu trúc do người dùng tự định nghĩa
 - Tuân theo nguyên tắc đặt tên đối tượng trong C
- **Khai báo các trường dữ liệu:** Danh sách các khai báo thành phần (*trường:field*) của cấu trúc
 - Giống khai báo biến
 - Các trường có thể có kiểu bất kỳ

Ví dụ

Thẻ sinh viên

Số hiệu:...(Chuỗi ký tự)..

Tên sinh viên: (Chuỗi ký tự)

Năm sinh:...(Số nguyên)...

Khóa:.....(Số nguyên).....

Lớp:..... :(Chuỗi ký tự). ...

```
struct SinhVien{  
    char SHSV[10];  
    char Ten[30];  
    int NS;  
    int Khoa;  
    char Lop [10];  
};
```

Point2D

Hoành độ (x)...(Số thực)..

Tung độ (y).....(Số thực)..

```
struct Point{  
    float x, y;  
};
```

Khai báo biến cấu trúc

- Khai báo **kiểu** cấu trúc nhằm tạo định nghĩa toàn thể cho các cấu trúc sẽ được dung sau này
 - Không cung cấp không gian nhớ cho kiểu
- Khai báo **biến** cấu trúc nhằm yêu cầu chương trình tạo vùng nhớ để lưu trữ các dữ liệu cho biến cấu trúc
 - Chứa dữ liệu của các trường của cấu trúc

Cú pháp khai báo

- Tồn tại định nghĩa kiểu cấu trúc

```
struct Kiểu_cấu_trúc Tên_biến;
```

- Khai báo trực tiếp

```
struct {  
    <Khai báo các trường dữ liệu>  
} Tên_biến;
```

- Kết hợp với khai báo kiểu

```
struct Kiểu_cấu_trúc {  
    <Khai báo các trường dữ liệu>  
} Tên_biến;
```

Ví dụ

- Tồn tại định nghĩa kiểu cấu trúc

```
struct SinhVien SV1, SV2, Thu_khoa;
```

- Khai báo trực tiếp

```
struct {  
    float x, y; //Tọa độ trên mặtP  
}A, B;          //Khai báo 2 điểm A, B
```

- Kết hợp với khai báo kiểu

```
struct Point_3D {  
    float x, y, z; //Tọa độ ko gian  
}A, B;
```

Chú ý (1/3)

- Các cấu trúc có thể được khai báo lồng nhau

```
struct diem_thi {  
    float dToan, dLy, dHoa;  
}  
  
struct thi_sinh{  
    char SBD[10];  
    char ho_va_ten[30];  
    struct diem_thi ket_qua;  
} thi_sinh_1, thi_sinh_2;
```

Chú ý (2/3)

- Có thể khai báo trực tiếp các trường dữ liệu của một cấu trúc bên trong cấu trúc khác

```
struct thi_sinh{  
    char SBD[10];  
    char ho_va_ten[30];  
    struct{  
        float dToan, dLy, dHoa;  
    } ket_qua;  
} thi_sinh_1, thi_sinh_2;
```

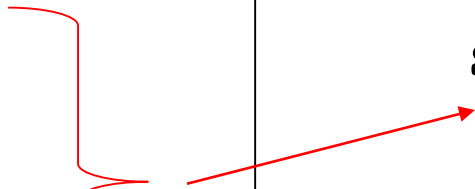

Chú ý (3/3)

- Có thể gán giá trị khởi đầu cho một biến cấu trúc, theo nguyên tắc như kiểu mảng. Ví dụ:

```
struct Date{
    int day;
    int month;
    int year;
};

struct SinhVien{
    char Ten[20];
    struct Date NS;
} SV = {"Tran Anh", 20, 12, 1990 };
```

```
struct SinhVien{
    char Ten[20];
    struct Date{
        int day;
        int month;
        int year;
    } NS;
} SV = {"Tran Anh", 20, 12, 1990 };
```



**Printf => SV.Ten;
SV.NS.day**

Định nghĩa kiểu dữ liệu với typedef (1/2)

```
typedef <tên_cũ> <tên_mới>;
```

Mục đích

- Đặt tên mới đồng nghĩa với tên của một kiểu dữ liệu đã được định nghĩa
 - Thường được sử dụng cho kiểu cấu trúc
 - Giúp cho khai báo trở nên quen thuộc và ít bị sai hơn

Ví dụ

```
typedef char Str80[80] ;  
typedef long mask;  
Str80 str="Bonjour tout le monde !";  
mask a, b;
```

Định nghĩa kiểu dữ liệu với typeof (2/2)

- Thường được kết hợp với kiểu cấu trúc để khai báo một ***bí danh*** cho một cấu trúc => *Giúp khai báo trở nên quen thuộc và ít bị sai hơn*

```
typedef struct { //Định nghĩa một cấu trúc
```

```
    char SHSV[10];
```

```
    char Ten[30];
```

```
    int NS;
```

```
    int Khoa;
```

```
    char Lop [10];
```

```
} SinhVien; //Đặt tên cho cấu trúc là SinhVien
```

```
SinhVien SV; //Tạo một biến cấu trúc
```

Chú ý (1/2)

- Cho phép đặt tên mới trùng với tên cũ

Ví dụ:

```
struct point_3D{  
    float x, y, z;  
}  
struct point_3D M;  
typedef struct point_3D  
point_3D;  
point_3D N;
```

```
typedef struct {  
    float x, y, z;  
}point_3D;  
point_3D M;  
point_3D N;
```

Chú ý (2/2)

```
typedef struct point_2D {  
    float x, y;  
}point_2D, diem_2_chieu,  
ten_bat_ki;  
point_2D X;  
diem_2_chieu Y;  
ten_bat_ki Z;
```

- **Chú ý:**

point_2D, diem_2_chieu, ten_bat_ki là các tên cấu trúc,
không phải tên biến

Nội dung chính

1. Khái niệm cấu trúc

- Khái niệm

2. Khai báo cấu trúc

- Khai báo kiểu cấu trúc
- Khai báo biến cấu trúc
- Định nghĩa kiểu dữ liệu với **typedef**

3. Xử lý dữ liệu cấu trúc

- Truy cập các trường dữ liệu
- Phép gán giữa các biến cấu trúc
- Một số ví dụ

Truy cập các trường dữ liệu

- **Cú pháp**

tên_biến_cấu_trúc.tên_trường

- **Lưu ý**

- Dấu “.” là toán tử truy cập vào trường dữ liệu trong cấu trúc
- Nếu trường dữ liệu là một cấu trúc => sử dụng tiếp dấu “.” để truy cập vào thành phần mức sâu hơn

Ví dụ 1

```
#include <stdio.h>
void main(){
    struct{
        char Ten[20];
        struct Date{
            int day;
            int month;
            int year;
        } NS;
    } SV = {"Tran Anh", 20, 12, 1990 };
    printf(" Sinh vien %s (%d/%d/%d)",
        SV.Ten, SV.NS.day, SV.NS.month, SV.NS.year);
}
```

Sinh vien Tran Anh (20/12/1990)

Ví dụ 2 : Biểu diễn điểm 2D (1/2)

- Xây dựng một cấu trúc biểu diễn điểm trong không gian 2 chiều. Nhập giá trị cho một biến kiểu cấu trúc này, sau đó hiển thị giá trị các trường dữ liệu của biến này ra màn hình.
 - Cấu trúc: tên điểm, tọa độ x, tọa độ y
 - Nhập, hiển thị từng trường của biến cấu trúc như các biến dữ liệu khác.

Ví dụ 2 : Biểu diễn điểm 2D (2/2)

```
#include<stdio.h>
#include<conio.h>
typedef struct{
    char ten[5];
    int x,y;
}toado;
void main(){
    toado t;
    printf("Nhap thong tin toa do\n");
    printf("Ten diem: ");gets(t.ten);
    printf("Toa do x: ");scanf("%d",&t.x);
    printf("Toa do y: ");scanf("%d",&t.y);
    printf("Gia tri cac truong\n");
    printf("%-5s%3d%3d\n",t.ten,t.x,t.y);
    getch();
}
```

```
Nhap thong tin toa do
Ten diem: A
Toa do x: 12
Toa do y: 6
Gia tri cac truong
A      12  6
```

Phép gán giữa các biến cấu trúc

- Muốn sao chép dữ liệu từ biến cấu trúc này sang biến cấu trúc khác cùng kiểu
 - Gán lần lượt từng trường trong hai biến cấu trúc
 - C cung cấp phép gán hai biến cấu trúc cùng kiểu:
`biến_cấu_trúc_1 = biến_cấu_trúc_2;`

Ví dụ (1/2)

- **Ví dụ:** Xây dựng cấu trúc gồm họ tên và điểm TĐC của sinh viên
 - a, b, c là 3 biến cấu trúc.
 - Nhập giá trị cho biến a.
 - Gán $b=a$,
 - gán từng trường của a cho c.
 - So sánh a, b và c ?

Ví dụ (2/2)

```
#include<stdio.h>
#include<conio.h>
typedef struct{
    char hoten[20];
    int diem;
}sinhvien;
void main(){
    sinhvien a,b,c;
    printf("Nhap thong tin sinh vien\n");
    printf("Ho ten: "); gets(a.hoten);
    printf("Diem:"); scanf("%d",&a.diem);
    b = a;
    strcpy(c.hoten,a.hoten);
    c.diem = a.diem;
```

```
printf("Bien a: ");
printf("%-20s%3d\n",a.hoten,a.diem);
printf("Bien b: ");
printf("%-20s%3d\n",b.hoten,b.diem);
printf("Bien c: ");
printf("%-20s%3d\n",c.hoten,c.diem);
getch();
}
```

```
Nhap thong tin sinh vien
Ho ten: LE VAN ANH
Diem:10
Bien a: LE VAN ANH          10
Bien b: LE VAN ANH          10
Bien c: LE VAN ANH          10
```

Bài tập

- 1) **Lập trình đọc vào một danh sách không quá 100 sinh viên** gồm: Họ tên, năm sinh
 1. Đưa ra DS những sinh viên sinh năm 2000
 2. Đưa ra DSSV đã sắp xếp theo thứ tự ABC
- 2) **Lập trình đọc vào DS thí sinh** gồm Họ tên, điểm thi 3 môn Toán, Lý, Hóa, kết thúc nhập khi gặp sinh viên có tên rỗng
 1. Đọc tiếp vào một điểm chuẩn; đưa ra danh sách thí sinh trúng tuyển (không có điểm liệt - 0)
 2. Đưa ra thí sinh cao điểm nhất
 3. Tìm điểm chuẩn, nếu chỉ lấy K SV, K nhập vào. Nếu có nhiều người bằng điểm nhau; loại cả

Giải bài 1

```
#include <stdio.h>
#include <string.h>
typedef struct{
    char Ten[30];
    int NS;
}SinhVien;
void main(){
    SinhVien DS[100], SV;
    int N,i,j;
    printf("Nhap So sinh vien : ");
    scanf("%d",&N);
    fflush(stdin);
    for ( i=0; i < N; i++ ){
        fflush(stdin);
        printf("Nhap du lieu cho sinh vien %d: \n", i+1);
        printf("Ho ten : "); gets(DS[i].Ten);
        printf("Nam sinh :");scanf("%d", &DS[i].NS);
    }
    printf("\n\n DANH SACH SINH VIEN\n\n");
```

```
for(i = 0; i < N; i ++){
    if(DS[i].NS == 2000) //Chỉ in SV sinh năm 2000
        printf("%s\n",DS[i].Ten);
    for(i = 0; i < N - 1; i ++){
        for(j = i+1; j < N; j ++){
            if(strcmp(DS[i].Ten,DS[j].Ten) > 0){
                SV= DS[i];
                DS[i]=DS[j];
                DS[j] = SV;
            }
        }
        printf("\n\n DANH SACH SINH VIEN DA SAP XEP\n\n");
    }
    for(i = 0; i < N; i ++){
        printf("%s\n",DS[i].Ten);
    }
} //main
```

Kết quả bài 1

```
Nhap So sinh vien : 5
Nhap du lieu cho sinh vien 1:
Ho ten : NGUYEN THI BUOI
Nam sinh :1999
Nhap du lieu cho sinh vien 2:
Ho ten : LE THI XINH
Nam sinh :2000
Nhap du lieu cho sinh vien 3:
Ho ten : TRAN THI MO
Nam sinh :2001
Nhap du lieu cho sinh vien 4:
Ho ten : LE THI DEP
Nam sinh :2000
Nhap du lieu cho sinh vien 5:
Ho ten : NGUYEN VAN ANH
Nam sinh :2000
```

DANH SACH SINH VIEN

```
LE THI XINH
LE THI DEP
NGUYEN VAN ANH
```

DANH SACH SINH VIEN DA SAP XEP

```
LE THI DEP
LE THI XINH
NGUYEN THI BUOI
NGUYEN VAN ANH
TRAN THI MO
```


Giải bài 2 (1/3)

```
#include <stdio.h>
#include <string.h>
typedef struct{
    char Ten[30];
    struct{
        int T, L, H, S;
    } DT;
} SinhVien;
void main(){
    SinhVien DS[100], TK, SV;
    int N,i,j,K;
    float C;
    N = 0;
```

Giải bài 2 (2/3)

```
do{
    printf("\nNhap DL cho sv thu %d\n",N+1);
    printf("Ten SV : "); gets(DS[N].Ten);
    if(strlen(DS[N].Ten)==0)
        break;
    else{
        printf("Diem thi T L H cua SV %s : ",DS[N].Ten);
        scanf("%d%d%d",&DS[N].DT.T,&DS[N].DT.L,&DS[N].DT.H);
        fflush(stdin);
        DS[N].DT.S = DS[N].DT.T + DS[N].DT.L + DS[N].DT.H;
        N++;
    }
}while(1);
printf("\n\n DANH SACH SINH VIEN\n\n");
printf(" Ten SV Toan Ly Hoa Tong \n");
for(i = 0; i < N; i ++){
    printf("%-20s%5d%5d%5d%6d\n",DS[i].Ten, DS[i].DT.T,
        DS[i].DT.L,DS[i].DT.H,DS[i].DT.S);
}
```

Giải bài 2 (3/3)

```
printf("\n\nDiem Chuan : ");scanf("%f",&C);
printf("\n\n DANH SACH SINH VIEN TRUNG TUYEN \n\n");
for(i = 0; i < N; i ++)
    if( (DS[i].DT.S >= C) &&
        (DS[i].DT.T*DS[i].DT.L*DS[i].DT.H > 0))
        printf("%s\n",DS[i].Ten);
TK = DS[0];
for(i = 1; i < N; i ++)
    if(DS[i].DT.S > TK.DT.S)
        TK = DS[i];
for(i = 0; i < N; i ++)
    if(DS[i].DT.S == TK.DT.S)
        printf("\n\n THU KHOA: %s \n\n",TK.Ten);
printf("\nSo nguoi trung tuyen:"); scanf("%d",&K);
for(i = 0; i < N - 1; i ++)
    for(j = i+1; j < N; j ++)
        if(DS[i].DT.S < DS[j].DT.S ){
            SV= DS[i];
            DS[i]=DS[j];
            DS[j] = SV;
        }
```

```
while((K>0)&&(DS[K-1].DT.S==DS[K].DT.S)) K--;
if(K>0){
    printf("Diem Chuan La : %4d",DS[K-1].DT.S);
    printf("\n\n Danh Sach sinh vien trung tuyen \n\n");
    for(i=0; i < K; i++)
        printf("%s\n",DS[i].Ten);
} //main
```

Bài tập

3) Lập trình thực hiện các công việc sau

- Đọc vào từ bàn phím một danh sách thuốc gồm
 - Tên thuốc (chuỗi không quá 20 ký tự)
 - Năm hết hạn
 - Số lượng còn
 - Đơn giá

Kết thúc nhập khi gặp thuốc có tên “* * *”

- Đưa danh sách thuốc ra màn hình
- Đưa ra danh sách các thuốc đã hết hạn
- Xóa khỏi danh sách những thuốc đã hết hạn. Đưa danh sách mới ra màn hình
- Tính tổng giá trị các thuốc đã hết hạn

Bài tập

4) Cho một danh sách thành tích thi đấu bóng đá của 32 đội tuyển bao gồm: Tên đội bóng, số bàn thắng, số bàn thua, số thẻ đỏ, số thẻ vàng

Viết chương trình thực hiện:

- Nhập dữ liệu vào từ bàn phím
- Nhập vào tên đội bóng,
 - đưa ra thành tích của đội này
 - Nếu không tồn tại, thông báo: không tìm thấy
- Tính và đưa ra màn hình số điểm của các đội nếu
 - Mỗi bàn thắng được tính 10 điểm
 - Mỗi bàn thua bị phạt 5 điểm, mỗi thẻ vàng trừ 2 điểm, thẻ đỏ trừ 5 điểm



CHƯƠNG 8 CON TRỎ VÀ DANH SÁCH LIÊN KẾT ĐƠN

GIẢNG VIÊN

TS. Hà Ngọc Long

Danh Sách Liên Kết Đơn

Danh Sách Liên Kết (1/3)

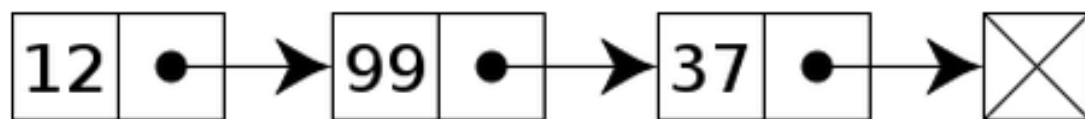
❑ Tại sao phải dùng danh sách liên kết (DSLK)?

– Nếu muốn chèn vào 1 phần tử vào mảng ?

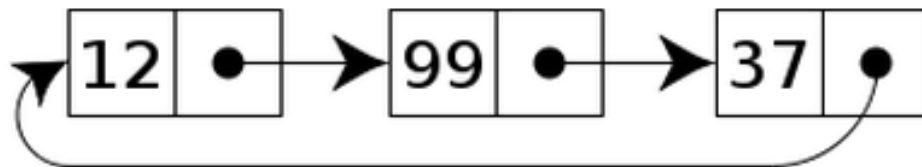
- Chi phí là $O(n)$

– Muốn xóa một phần tử trong mảng ?

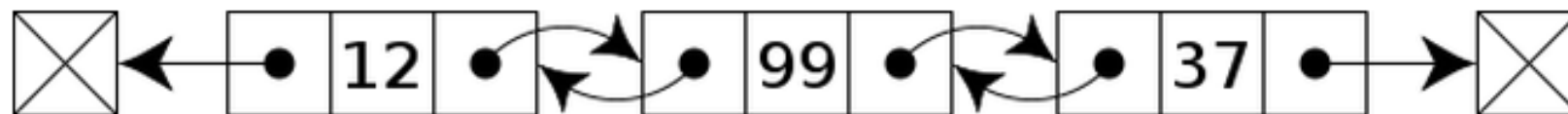
- Chi phí $O(n)$.



Danh sách liên kết đơn



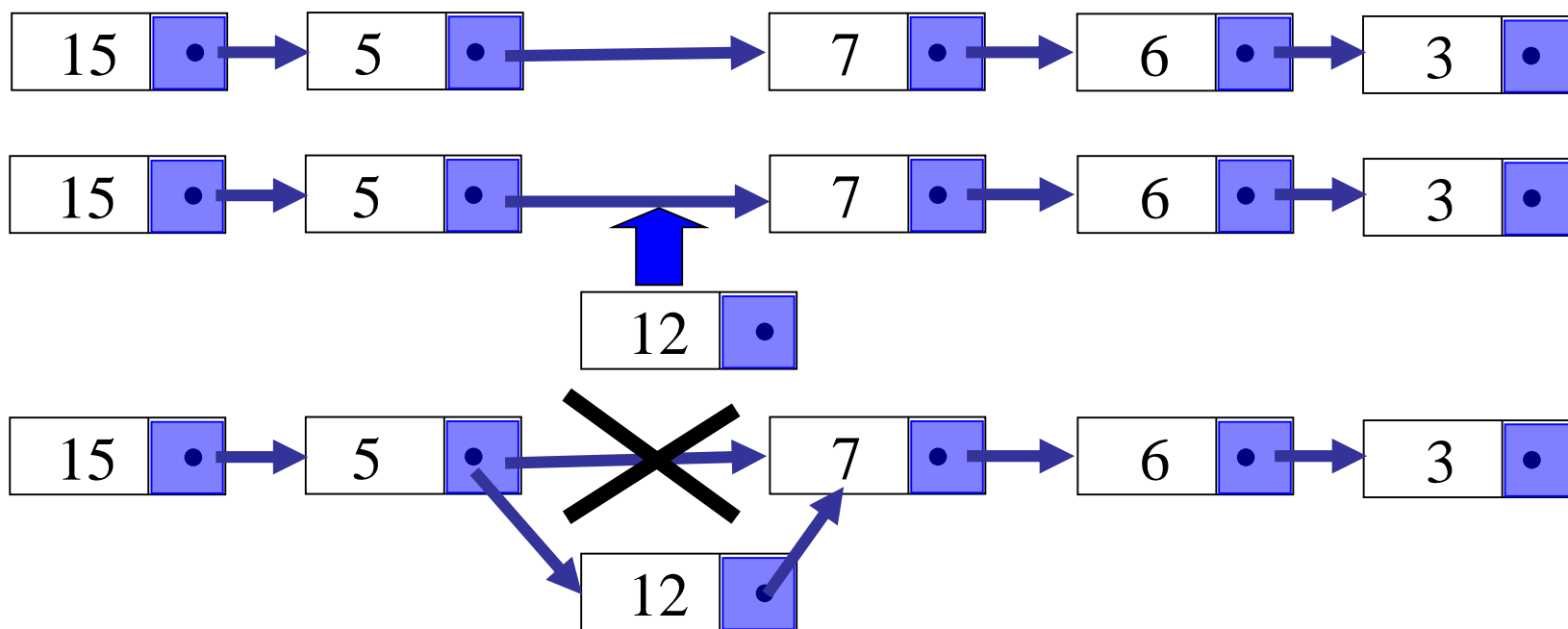
Danh sách vòng



Danh sách liên kết đôi

Danh Sách Liên Kết (2/3)

- Ta tách rời các phần tử của mảng, và kết nối chúng lại với nhau bằng một “móc xích”



Danh Sách Liên Kết (3/3)

- Một dãy tuần tự các nút (Node)
- Giữa hai nút có một tham chiếu
- Các nút không cần phải lưu trữ liên tiếp nhau trong bộ nhớ
- Có thể mở rộng tùy ý (chỉ giới hạn bởi dung lượng bộ nhớ)
- Thao tác Chèn/Xóa không cần phải dịch chuyển phần tử
- Quản lý danh sách bởi con trỏ đầu Head
- Có thể truy xuất đến các phần tử khác thông qua con trỏ liên kết.

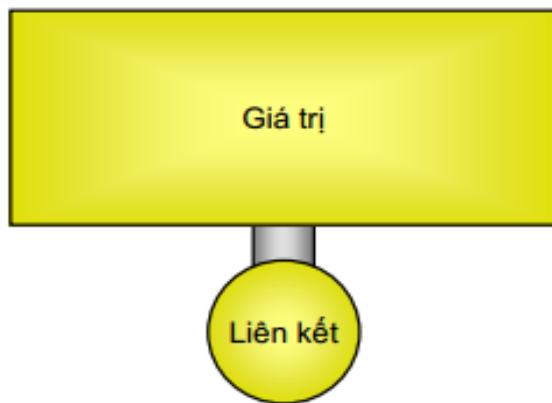
Các Loại Danh Sách Liên Kết

- ❑ Danh sách liên kết đơn (Single-Linked list)
 - Mỗi nút chỉ có 1 con trỏ liên kết (Next)
- ❑ Danh sách liên kết đôi (Double-Linked list)
 - Mỗi nút có 2 con trỏ liên kết (Prev, Next)
- ❑ Danh sách đa liên kết (Multi-Linked list)
 - Mỗi nút có nhiều hơn 2 con trỏ liên kết
- ❑ Danh sách liên kết vòng (Circular-Linked list)
 - Liên kết ở nút cuối cùng của danh sách chỉ đến nút đầu tiên trong danh sách

Danh Sách Liên Kết Đơn (1/2)

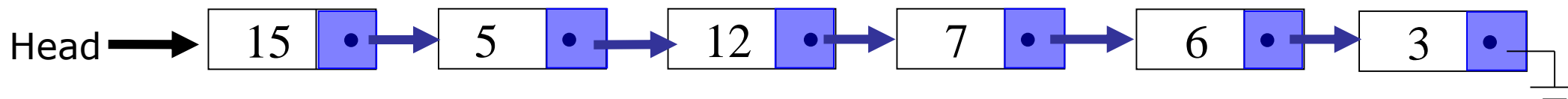
❑ Khái niệm:

- Danh sách móc nối đơn là tập hợp các nút, mỗi nút là một đối tượng có hai thuộc tính được ký hiệu là INFO và LINK.
- Thuộc tính INFO dùng để lưu giữ thông tin (dữ liệu), thuộc tính LINK dùng để lưu địa chỉ của nút đứng sau nó



Danh Sách Liên Kết Đơn (2/2)

- Nút đầu tiên trong danh sách LK đơn gọi là chốt của danh sách (Head).
- Để duyệt DSLK đơn, ta bắt đầu từ chốt (Head), dựa vào trường liên kết để đi sang nút kế tiếp, khi gặp giá trị đặc biệt (nút cuối) thì dừng lại.
 - Quy ước có một danh sách móc nối đơn Head không có nút nào, gọi là danh sách rỗng, khi đó *Head* = *null*.
 - Một nút có địa chỉ được quản lý bởi p thì ở đây quy định p.Info và p.Link là trường *INFO* và *LINK* của nút ấy.

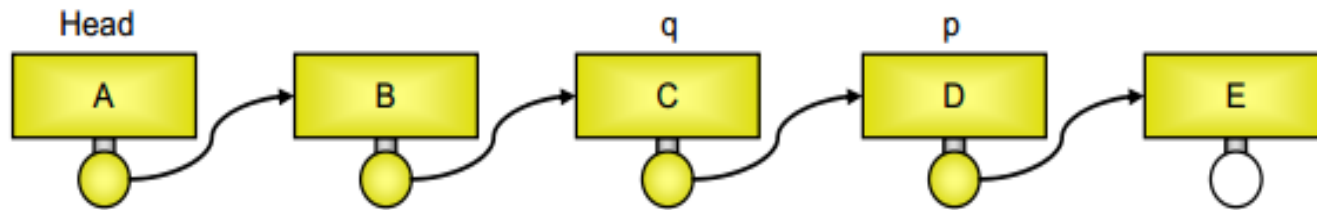


Thao Tác Trên Danh Sách Liên Kết Đơn (1/6)

❑ Kiểm tra danh sách rỗng

```
boolean IsEmptyList(Head){  
    return (Head ==NULL);  
}
```

❑ Đếm số phần tử trong danh sách



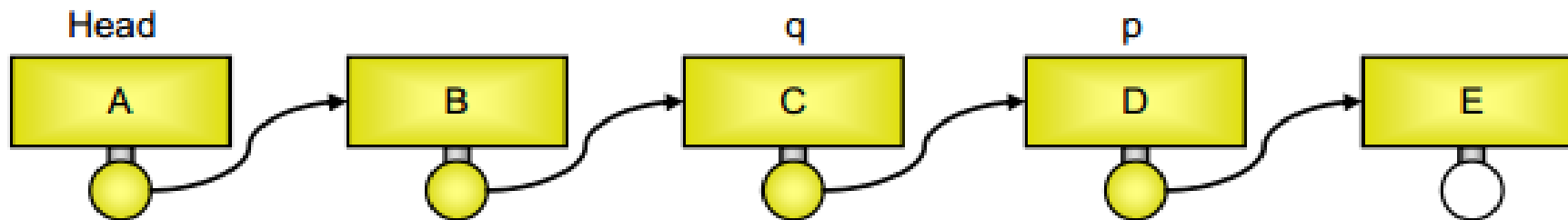
```
int CountNode(Node Head){  
    int count = 0;  
    Node p = Head;  
    while (p != NULL){  
        cout++;  
        p = p.Link;  
    }  
    return cout;  
}
```

Thao Tác Trên Danh Sách Liên Kết Đơn (2/6)

❑ Duyệt danh sách

- Tìm cách thăm tất cả các nút của danh sách và mỗi nút thăm đúng một lần.
- “Thăm nút” ở đây là quản lý được địa chỉ của nút đó, khi đó sẽ truy cập được dữ liệu của nút này.
- Việc truy cập dữ liệu để có những tác động gì là phụ thuộc vào yêu cầu của từng bài toán cụ thể.

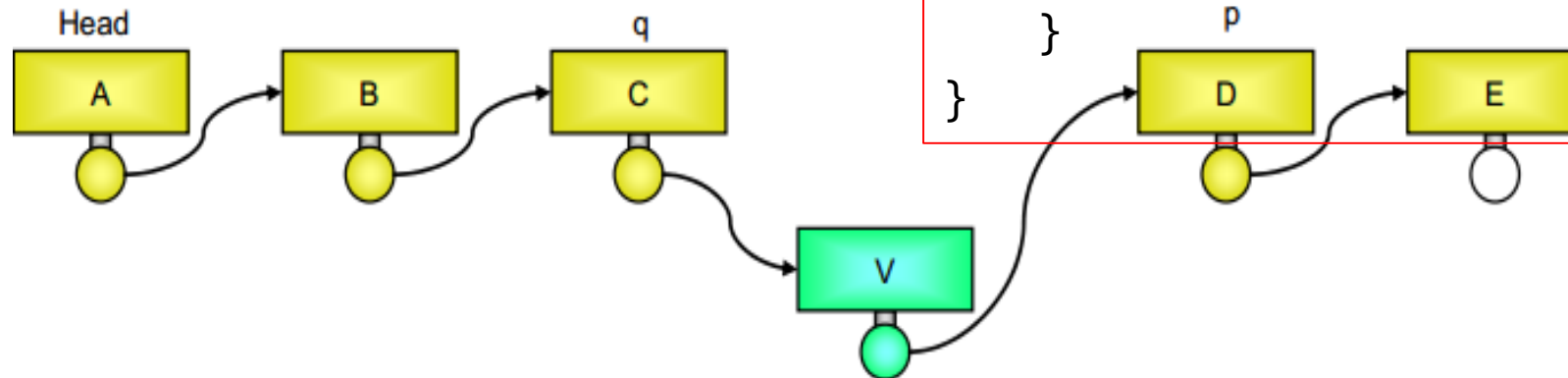
```
Public void DuyệtDS(Head) {  
    p = Head;  
    while (p ≠ null) {  
        <Thăm nút trả bởi p>;  
        p = p.Link;  
    }  
}
```



Thao Tác Trên Danh Sách Liên Kết Đơn (3/6)

❑ Chèn phần tử

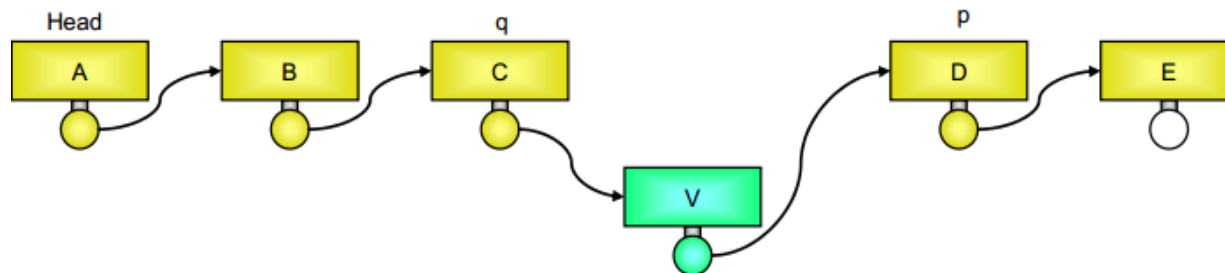
- Muốn chèn thêm một nút chứa giá trị **V** vào sau vị trí của nút *q* nằm trong danh sách liên kết đơn được quản lý bởi Head:
- Tạo ra một nút mới *NewNode* chứa giá trị *V*.
- Điều chỉnh mối nối của *q* và *NewNode* so với *p*.



```
Public void BoSung(Head, q, V) {  
    <Tạo nút mới NewNode có giá trị V>;  
    if (Head ≠ null) {  
        NewNode.Link = q.Link;  
        q.Link = NewNode  
    }  
    else {  
        Head = NewNode;  
        NewNode.Link = null;  
    }  
}
```


Thao Tác Trên Danh Sách Liên Kết Đơn (4/6)

- Trường hợp bổ sung một nút vào cuối danh sách liên kết đơn Head, ta tiến hành theo các bước sau:
 - Tìm nút q là nút cuối cùng của danh sách liên kết đơn Head
 - Tiến hành giải thuật chèn một nút sau nút trở bởi q

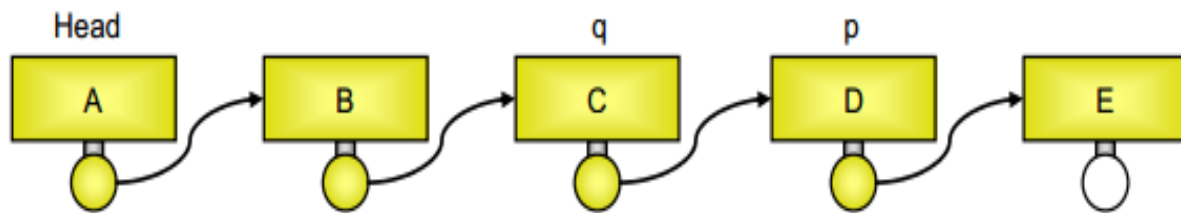


```
Public void BoSungCuoi(Head, V){  
    <Tạo nút mới NewNode có giá trị V>;  
    q = Head;  
    if (Head ≠ null) {  
        while (q.Link ≠ null)  
            q = q.Link;  
        q.Link = NewNode;  
        NewNode.Link = null;  
    }  
    else {  
        Head = NewNode;  
        NewNode.Link = null;  
    }  
}
```

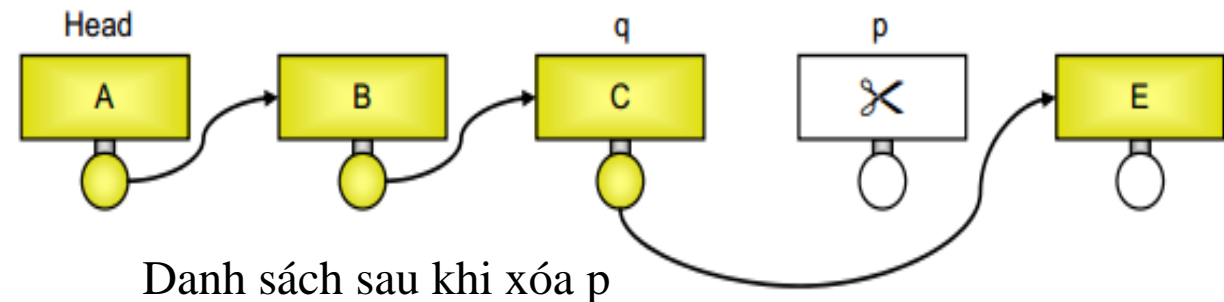
Thao Tác Trên Danh Sách Liên Kết Đơn (5/6)

❑ Xóa phần tử khỏi danh sách nối đơn

- Muốn hủy nút p ra khỏi danh sách nối đơn, ta phải tìm nút q là nút đứng liền trước nút p trong danh sách (nút có liên kết với p).
- Nếu tìm thấy thì chỉnh lại liên kết: q liên kết thẳng với nút liền sau p , khi đó quá trình duyệt danh sách bắt đầu từ Head khi duyệt tới q sẽ nhảy qua không duyệt p nữa. Trên thực tế khi cài đặt bằng biến động và con trỏ, ta nên có tác phong giải phóng bộ nhớ đã cấp cho nút p .

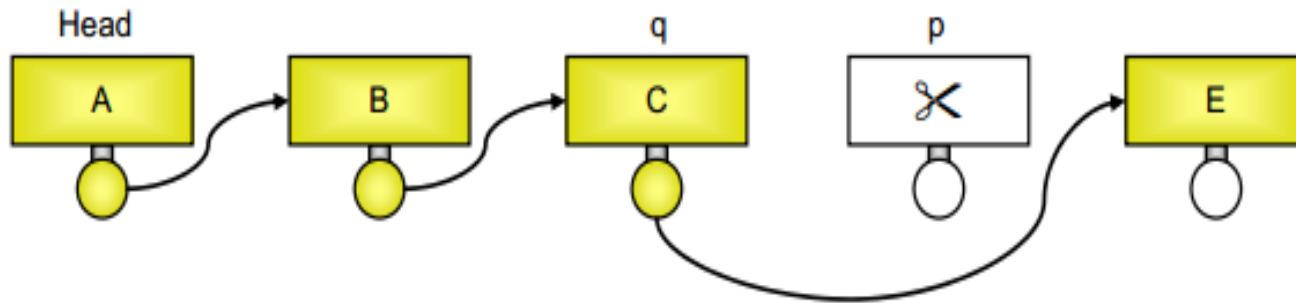


Danh sách ban đầu



Thao Tác Trên Danh Sách Liên Kết Đơn (6/6)

- Nếu không có nút đứng trước p trong danh sách thì tức là $p = \text{Head}$, ta chỉ việc đặt lại Head bằng nút đứng kế tiếp Head (cũ) trong danh sách. Sau đó có thể giải phóng bộ nhớ cấp cho nút p (Head cũ).



```
Public void TimNut(Head, q, p) {  
    q = Head;  
    if (p != Head)  
        while ((q != null) && (q.Link != p))  
            q = q.Link;  
}  
  
Public void LoaiBo(Head, p) {  
    if (Head == p) Head = p.Link;  
    else {  
        TimNut(q, p);  
        q.Link = p.Link;  
    }  
}
```

Xây Dựng DSLK Đơn Bằng C/C++ (1/14)

```
struct Node {  
    int data;  
    Node* next;  
};
```

Tạo Node

```
Node* CreateNode(int init_data) {  
    Node* node = new Node;  
    node->data = init_data; // node.data  
    node->next = NULL;      // node vừa tạo chưa  
    // thêm vào danh sách nên chưa liên kết với phần tử  
    // nào cả nên phần liên kết gán bằng NULL  
    return node;  
}
```

Cập phát động cho node mới

Bình thường, khi các bạn khai báo 1 biến structure thì các bạn có thể truy cập đến thành viên của structure đó thông qua toán tử dot(.). Nhưng đối với một con trỏ trỏ tới một **structure** thì toán tử mũi tên (->) sẽ được sử dụng thay cho toán tử dot (.).

Xây Dựng DSLK Đơn Bằng C/C++ (2/14)

```
struct LinkedList
{
    Node* head;
    Node* tail;
};
```

```
void CreateList(LinkedList& l)
{
    l.head = NULL;
    l.tail = NULL;
}
// Khởi tạo một danh sách liên kết đơn
LinkedList list;
CreateList(list); // Gán head và tail bằng NULL
```

Tạo danh sách liên kết đơn với đầu và đuôi

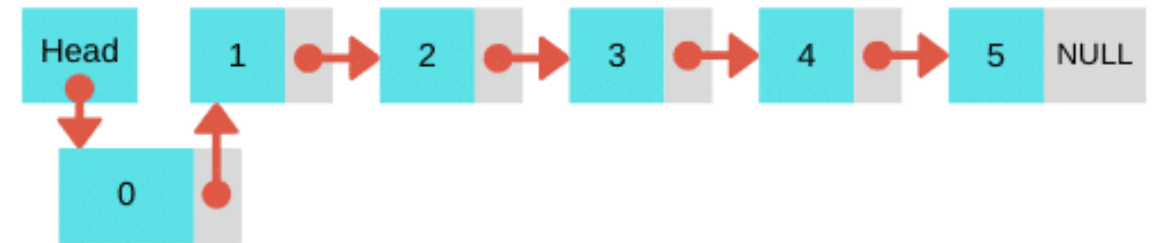
- Ta đã có được thành phần tạo nên danh sách liên kết đơn là node, tiếp theo chúng ta cần quản lý chúng bằng cách biết được phần tử đầu và cuối. Vì mỗi phần tử đều liên kết với phần tử kế vậy nên ta chỉ cần biết phần tử đầu và cuối là có thể quản lý được danh sách này.
- Vậy đơn giản ta cần tạo một cấu trúc lưu trữ địa chỉ phần tử đầu (**head**) và phần tử cuối (**tail**)

Xây Dựng DSLK Đơn Bằng C/C++ (3/14)

```
void AddHead(LinkedList& l, Node* node) {  
    if (l.head == NULL) {  
        l.head = node;  
        l.tail = node;  
    }  
    else {  
        node->next = l.head;  
        l.head = node;  
    }  
}
```

Thêm vào đầu

Add node to head



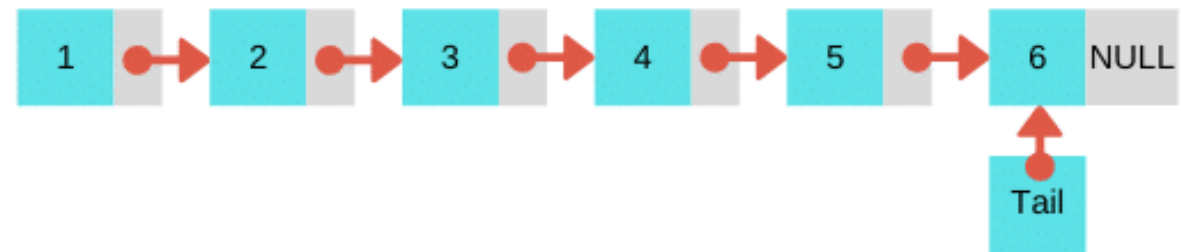
- Để thêm node vào đầu danh sách, đầu tiên ta cần kiểm tra xem danh sách đó có rỗng hay không, nếu danh sách rỗng, ta chỉ cần gán head và tail của danh sách bằng node đó.
- Ngược lại nếu danh sách không rỗng, ta thực hiện trở thành phần liên kết vào head, sau đó gán lại head bằng node mới.

Xây Dựng DSLK Đơn Bằng C/C++ (4/14)

```
void AddTail(LinkedList& l, Node* node) {  
    if (l.head == NULL) {  
        l.head = node;  
        l.tail = node;  
    }  
    else {  
        l.tail->next = node;  
        l.tail = node;  
    }  
}
```

Thêm vào cuối

Add node to tail



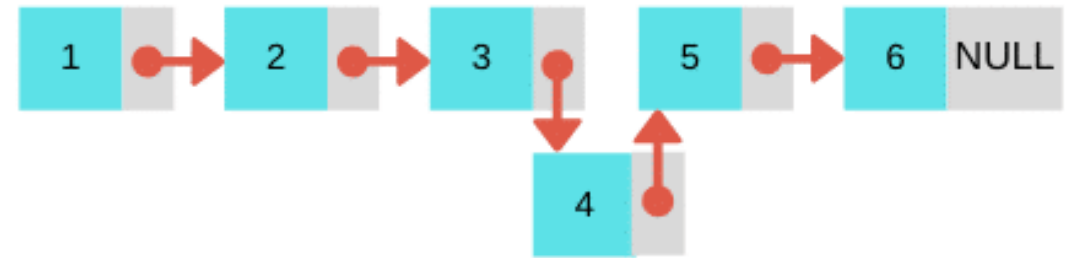
- Tương tự, để thêm node vào cuối danh sách, đầu tiên ta kiểm tra xem danh sách rỗng hay không, rỗng thì gán head và tail đều bằng node mới.
- Nếu không rỗng, ta thực hiện trỏ tail->next vào node mới, sau đó gán lại tail bằng node mới (vì bây giờ node mới thêm chính là tail).

Xây Dựng DSLK Đơn Bằng C/C++ (5/14)

```
void InsertAfterQ(LinkedList& l, Node* p, Node* q) {  
    if (q != NULL) {  
        p->next = q->next;  
        q->next = p;  
        if (l.tail == q)  
            l.tail = p;  
    }  
    else  
        AddHead(l, p);  
}
```

Thêm vào vị trí bất kỳ

Insert After Q



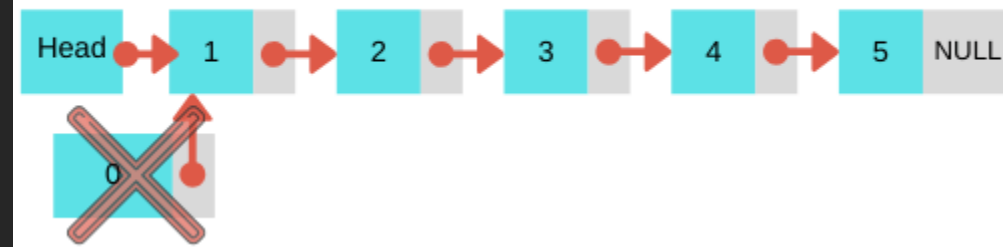
- Để thêm một node p vào sau node q bất kỳ, đầu tiên ta cần kiểm tra xem node q có NULL hay không, nếu node q là NULL tức là danh sách rỗng, vậy thì ta sẽ thêm vào đầu danh sách. Nếu node q không NULL, tức là tồn tại trong danh sách, ta thực hiện trở $p->next = q->next$, sau đó $q->next = p$.
- Tiếp theo chúng ta kiểm tra xem node q trước đó có phải là node cuối hay không, nếu node q là node cuối thì thêm p vào, p sẽ thành node cuối nên ta gán lại $tail = p$.

Xây Dựng DSLK Đơn Bằng C/C++ (6/14)

```
int RemoveHead(LinkedList& l, int& x) {  
    if (l.head != NULL) {  
        Node* node = l.head;  
        x = node->data; // Lưu giá trị của node head  
        l.head = node->next;  
        delete node;    // Hủy node head  
        if (l.head == NULL)  
            l.tail = NULL;  
        return 1;  
    }  
    return 0;  
}
```

Xóa ở đầu DSLK

Remove head



- Để xóa phần tử ở đầu danh sách, ta kiểm tra xem danh sách đó có rỗng hay không, nếu rỗng, ta không cần xóa, trả về kết quả là 0. Nếu danh sách không rỗng, ta thực hiện lưu node head lại, sau đó gán head bằng next của node head, sau đó xóa node head đi.
- Tiếp theo ta cần kiểm tra xem danh sách vừa bị xóa đi node head có rỗng hay không, nếu rỗng ta gán lại tail bằng NULL luôn sau đó trả về kết quả 1.

Xây Dựng DSLK Đơn Bằng C/C++ (7/14)

```
int RemoveAfterQ(LinkedList& l, Node* q, int& x) {  
    if (q != NULL) {  
        Node* p = q->next;  
        if (p != NULL) {  
            if (l.tail == p)  
                l.tail = q;  
            q->next = p->next;  
            x = p->data;  
            delete p;  
            return 1;  
        }  
        return 0;  
    }  
    return 0;  
}
```

Xóa ở vị trí bất kỳ trong DSLK

Xây Dựng DSLK Đơn Bằng C/C++ (8/14)

```
void PrintList(LinkedList l) {  
    if (l.head != NULL) {  
        Node* node = l.head;  
        while (node != NULL) {  
            cout << node->data << ' ';  
            node = node->next; //  
            Chuyển sang node tiếp theo  
        }  
    }  
}
```

Duyệt danh sách và in

```
Node* GetNode(LinkedList& l, int index) {  
    Node* node = l.head;  
    int i = 0;  
    while (node != NULL && i != index) {  
        node = node->next;  
        i++;  
    }  
    if (i == index && node != NULL)  
        return node;  
    return NULL;  
}
```

Lấy giá trị node bất kỳ

Xây Dựng DSLK Đơn Bằng C/C++ (9/14)

```
Node* Search(LinkedList l, int x) {
    Node* node = l.head;
    while (node != NULL && node->data != x)
        node = node->next;
    if (node != NULL)
        return node;
    return NULL;
}
```

Tìm kiếm phần tử trong DSLK

```
int Length(LinkedList l) {
    int count = 0;
    Node* node = l.head;
    while (node != NULL) {
        count++;
        node = node->next;
    }
    return count;
}
```

Đếm số phần tử trong DSLK

Xây Dựng DSLK Đơn Bằng C/C++ (10/14)

```
void DestroyList(LinkedList& l) {  
    int x;  
    Node* node = l.head;  
    while (node != NULL) {  
        RemoveHead(l, x);  
        node = l.head;  
    }  
    l.tail = NULL;  
}
```

Xóa danh sách

- Để xóa danh sách, ta cần hủy tất cả các node tức là duyệt và hủy từng node. Ở đây mình sẽ dùng lại hàm RemoveHead. Đầu tiên, ta gán một node bằng head, kiểm tra nếu node đó khác NULL thì gọi RemoveHead và gán lại node bằng head tiếp, cứ lặp như vậy cho đến khi node đó NULL thì thôi. Sau khi xóa hết tất cả phần tử thì gán lại tail bằng NULL

Xây Dựng DSLK Đơn Bằng C/C++ (11/14)

```
int main() {  
    // Create a linked list  
    LinkedList list;  
    CreateList(list);  
  
    // Add sample data to list  
    Node* node;  
    for (auto i = 1; i <= 10; i++) {  
        // Create new node with init data is i  
        node = CreateNode(i);  
  
        // Add node to head  
        // List that is added node by AddHead will be reversed  
        //AddHead(list, node);  
  
        // Add node to Tail  
        AddTail(list, node);  
    }  
}
```

Xây Dựng DSLK Đơn Bằng C/C++ (12/14)

```
// Print list
PrintList(list);
cout << endl;

// Get list's length
int len = Length(list);
cout << "Length of list: " << len << endl;

// Get node at index 7
Node* nodeAtIdx7 = GetNode(list, 7);
if (nodeAtIdx7 != NULL)
    cout << "Data at node have idx 7: " << nodeAtIdx7->data << endl;

// Search for 4 in list
Node* search4InList = Search(list, 4);
if (search4InList != NULL)
    cout << "4 was founded" << endl;
else
    cout << "4 not Found" << endl;
```

Xây Dựng DSLK Đơn Bằng C/C++ (13/14)

```
/ Remove node after 4 in list
int x;
int res = RemoveAfterQ(list, search4InList, x);
if (res) {
    cout << "Data of node has been removed: " << x << endl;
    cout << "List after removed: ";
    PrintList(list);
    cout << endl;
}
else
    cout << "Nothing is removed" << endl;

// Insert 2409 after node 4
Node* node2409 = CreateNode(2409);
InsertAfterQ(list, node2409, search4InList);
cout << "List after insert 2409 after 4: ";
PrintList(list);
cout << endl;
```


Xây Dựng DSLK Đơn Bằng C/C++ (14/14)

```
// Remove Head
res = RemoveHead(list, x);
if (res)
{
    cout << "Data of node has been removed: " << x << endl;
    cout << "List after removed head: ";
    PrintList(list);
    cout << endl;
}
else
    cout << "Nothing is removed" << endl;

// Destroy all node
DestroyList(list);

return 0;
}
```

Ưu Và Nhược Của Danh Sách Liên Kết

- **Ưu điểm**

- Tận dụng được không gian nhớ để lưu trữ
 - Các nút không cần lưu trữ kế tiếp nhau
 - Có thể mở rộng kích thước tùy ý (phụ thuộc bộ nhớ)
- Việc thêm vào hay loại bỏ được tiến hành dễ dàng $O(1)$
- Dễ dàng kết nối hay phân rã danh sách

- **Nhược điểm**

- Truy xuất tuần tự từng phần tử

Mảng & Danh Sách Liên Kết

❖ Mảng

- Phải biết trước số phần tử
- Lưu trữ tuần tự
- Khi chèn và xóa phải dịch chuyển các phần tử
- Truy xuất qua chỉ mục

❖ Danh sách liên kết

- Số phần tử tùy biến
- Sử dụng con trỏ
- Khi chèn/xóa chỉ cần thay đổi con trỏ liên kết
- Truy xuất tuần tự

Bài Tập Danh Sách Liên Kết Đơn

- 1) Cho danh sách móc nối đơn có nút đầu tiên trỏ bởi L . Hãy viết các giải thuật giải quyết các công việc sau:
 - Tính số lượng các nút của danh sách
 - Bổ sung nút mới chứa X vào sau nút thứ k , nếu không có nút thứ k thì bổ sung vào sau cùng.
- 2) Dùng danh sách liên kết quản lý sinh viên trong lớp
 - Lưu trữ các thông tin: MSSV, Họ, Tên, Ngày sinh, Điểm tổng kết học kỳ.
- 3) Thực hiện các thao tác: thêm, bớt, sắp xếp sinh viên (theo điểm tổng kết) trong danh sách.

Q&A

