

# NỘI DUNG

## CẤU TRÚC DỮ LIỆU ĐỘNG

# Biến Tĩnh

- Được khai báo tường minh, có tên gọi
- Tồn tại trong phạm vi khai báo
- Được cấp phát trong stack
- Kích thước không đổi => không tận dụng hiệu quả bộ nhớ
- Ví dụ : `int x,y;`  
`char c;`  
`float f[5];`
- Khi biết chắc nhu cầu sử dụng đối tượng trước khi thực sự xử lý : dùng biến không động

# Ví Dụ Hạn Chế Của Biến Tĩnh

- Tổ chức danh sách lớp học

- Dùng mảng tĩnh :

```
typedef      struct
{
    char ten[20];
    int  maso;
}Hocvien;
```

```
Hocvien      danhsach[50];
```

- Số lượng học viên  $< 50 \Rightarrow$  lãng phí
- Số lượng học viện  $> 50 \Rightarrow$  thiếu chỗ !

# Biến Động

- Không được khai báo tường minh, không có tên gọi
- Xin khi cần, giải phóng khi sử dụng xong
- Được cấp phát trong heap
- Linh động về kích thước
- Vấn đề : biến động không có tên gọi tường minh, làm sao thao tác ?

# Kiểu con trỏ

- Kiểu con trỏ dùng lưu địa chỉ của một đối tượng dữ liệu khác.
- Biến thuộc kiểu con trỏ Tp là biến mà giá trị của nó là địa chỉ của một vùng nhớ ứng với một biến kiểu T, hoặc là giá trị NULL.
- Khai báo trong C :

```
typedef int *intpointer;  
intpointer p;
```
- Bản thân biến con trỏ là không động
- Dùng biến con trỏ để lưu giữ địa chỉ của biến động => truy xuất biến động thông qua biến con trỏ

# Các thao tác trên kiểu con trỏ

- Tạo ra một biến động và cho con trỏ 'p' chỉ đến nó:
  - `void* malloc(size);`
  - `void* calloc(n,size);`
  - `new` // hàm cấp phát bộ nhớ trong C++
- Hủy một biến động do p chỉ đến :
  - Hàm `free(p)` huỷ vùng nhớ cấp phát bởi hàm `malloc` hoặc `calloc` do p trỏ tới
  - Hàm `delete p` huỷ vùng nhớ cấp phát bởi hàm `new` do p trỏ tới

# Sử dụng biến tĩnh, con trỏ và biến động

```
int x;
```

```
x = 5;
```

**Biến không động x**

5

```
int *p;
```

```
p = new(int);
```

```
*p = 5
```

**Biến con trỏ p**

0xFF

0xFF

5

**Biến động có địa chỉ 0xFF**

# 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
  - Liên kết tường minh : Danh sách liên kết

# Danh sách liên kết ngầm(mảng)

- Mỗi liên hệ giữa các phần tử được thể hiện ngầm:

- $x_i$  : phần tử thứ  $i$  trong danh sách 

$x_0$	...	$x_i$	$x_{i+1}$
-------	-----	-------	-----------
- $x_i, x_{i+1}$  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  $i$ :  

$$\text{address}(i) = \text{address}(1) + (i-1) * \text{sizeof}(T)$$

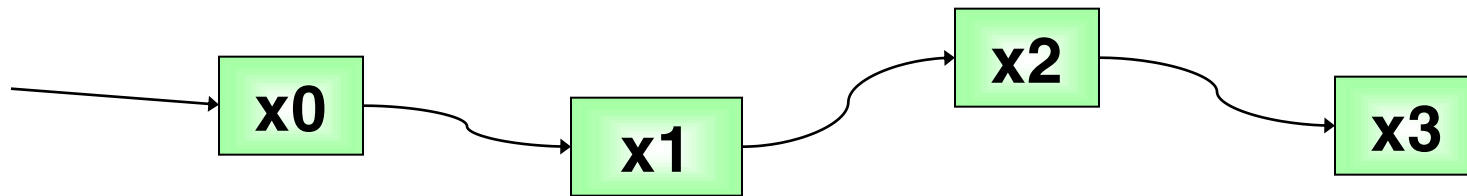
- **Ưu điểm** : Truy xuất trực tiếp, nhanh chóng

- **Nhược điểm**:

- Sử dụng bộ nhớ kém hiệu quả
- Kích thước cố định
- Các thao tác thêm<sub>10</sub>vào , loại bỏ không hiệu quả

# Liên kết tường minh (Danh sách liên kết)

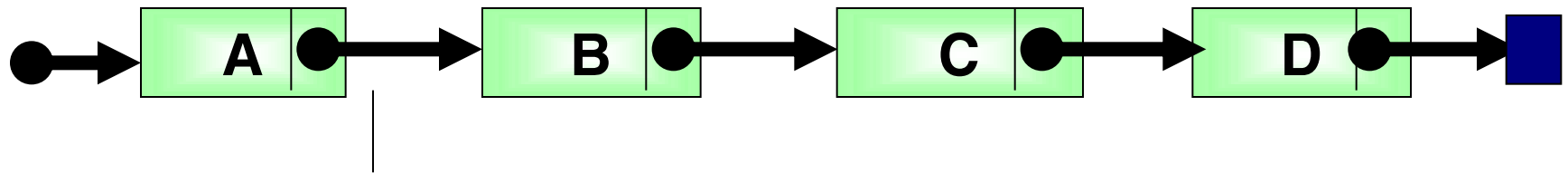
- CTDL cho một phần tử
  - Thông tin bản thân
  - Địa chỉ của phần tử kế trong danh sách



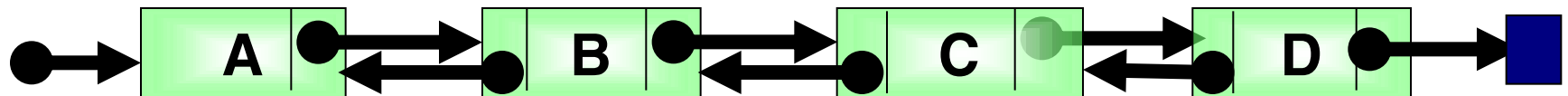
- Mỗi phần tử là một biến động
- **Ưu điểm**
  - + Sử dụng hiệu quả bộ nhớ
  - + Linh động về số lượng phần tử

# 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



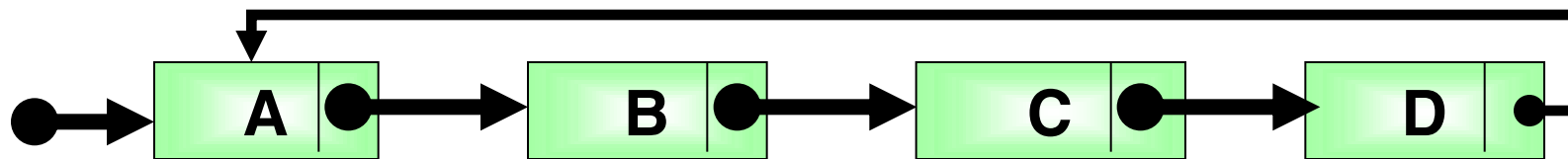
- **Danh sách liên kết kép:** Mỗi phần tử liên kết với phần tử đứng trước và sau nó trong danh sách



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

# Các loại danh sách liên kết (tt)

- 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

