



# CẤP PHÁT BỘ NHỚ ĐỘNG

---

- ❑ Bộ nhớ Heap và Stack
- ❑ Biến tĩnh và biến động
- ❑ Cơ chế tạo biến động
- ❑ Một số cấu trúc dữ liệu động



## 5.1. Bộ nhớ Heap và bộ nhớ Stack

- Bản chất đều cùng là vùng nhớ được tạo ra và lưu trữ trong RAM khi chương trình được thực thi.
  - Bộ nhớ Stack được dùng để lưu trữ các biến cục bộ trong hàm, tham số truyền vào hàm, địa chỉ trả về của hàm.
  - Bộ nhớ Heap được dùng để lưu trữ vùng nhớ cho các biến con trỏ được cấp phát động bởi các hàm malloc - calloc - realloc (trong C) hoặc từ khóa new (trong C++, Java,...).



# Bộ nhớ Stack

---

## ■ Đặc điểm

- Kích thước của bộ nhớ Stack là cố định, tùy thuộc vào từng hệ điều hành.
- Vùng nhớ Stack được quản lý bởi hệ điều hành, dữ liệu được lưu trong Stack sẽ tự động hủy khi hàm thực hiện xong công việc của mình.
- Bộ nhớ Stack cố định nên nếu chương trình sử dụng quá nhiều bộ nhớ vượt quá khả năng lưu trữ của Stack chắc chắn sẽ xảy ra tình trạng tràn bộ nhớ Stack.



# Bộ nhớ Heap

---

## ■ Đặc điểm

- Kích thước của bộ nhớ Heap là không cố định, có thể tăng giảm
- Vùng nhớ Heap được quản lý bởi lập trình viên, dữ liệu trong Heap sẽ không bị hủy khi hàm thực hiện xong. Do vậy, phải tự tay hủy vùng nhớ bằng câu lệnh free/ delete.
- Nếu liên tục cấp phát vùng nhớ mà không giải phóng thì có thể sẽ bị lỗi tràn vùng nhớ Heap



# Heap hay Stack

---

- Khi không biết chính xác cần bao nhiêu vùng nhớ là đủ để lưu trữ dữ liệu trong khi chương trình đang chạy thì dùng bộ nhớ Heap (ví dụ điển hình là cấp phát động của mảng), còn lại thì sử dụng bộ nhớ Stack.
- Hoặc khi dữ liệu quá lớn vượt quá khả năng của Stack thì nên dùng Heap.



## 5.2. Biến tĩnh và biến động

- Các biến sử dụng trong các chương trình trước đây là biến tĩnh (static)
  - 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.

**int n, a[50]; char c;**

➡ Khi biết chắc nhu cầu sử dụng đối tượng trước khi thực sự xử lý : dùng biến không động

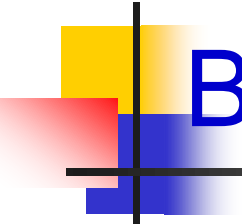
# Ví dụ hạn chế của biến tĩnh

- Tổ chức danh sách lớp học

```
struct ngaysinh{  
    int Ngay , Thang, Nam;  
};  
struct sinhvien {  
    int MaSV;  
    char TenLop[10];  
    char HoTen[30];  
    float Diem;  
    ngaysinh NS;  
};  
struct sinhvien Hocvien[30];
```

Nếu số lượng học viên <30  
=>lãng phí

Nếu số lượng học viên > 30  
=>thiếu chỗ !



# Biến động (dynamic)

## ■ Đặc điểm

- 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

## ■ Biến động không có tên gọi tường minh, làm sao thao tác:

- 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





# So sánh biến tĩnh và biến động

Biến tĩnh	Biến động
- Có tên gọi	- Không có tên gọi
- Truy nhập giá trị của biến bằng tên biến	- Truy nhập giá trị của biến bằng địa chỉ của biến
- Được cấp vùng nhớ chứa giá trị ngay khi chương trình chạy	- Chưa được cấp vùng nhớ chứa giá trị ngay khi chương trình chạy
- Không điều chỉnh được kích thước vùng nhớ chứa giá trị	- Điều chỉnh được kích thước vùng nhớ chứa giá trị
- Truy nhập dữ liệu gián tiếp qua tên biến	- Truy nhập dữ liệu gián tiếp qua biến <b>con trỏ</b>



## 5.3. Cơ chế tạo biến động

- Hàm **malloc()** là một trong các hàm được sử dụng thường xuyên nhất để thực hiện việc cấp phát bộ nhớ từ vùng nhớ còn tự do.

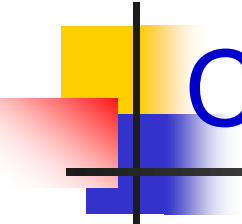
- Cú pháp

`void* malloc (size);` // size: số bytes cần cấp phát

- Ví dụ

```
int *p= (int *)malloc(10*sizeof(int))
```

- Con trỏ có kiểu void có thể ép được sang bất kỳ kiểu dữ liệu nào. Do đó, (int \*)trong ví dụ trên được dùng để ép sang kiểu int.



# Cấp phát bộ nhớ động

- Hàm **calloc** tương tự như **malloc**, nhưng điểm khác biệt chính là mặc nhiên giá trị 0 được lưu vào không gian bộ nhớ vừa cấp phát
- Hàm **calloc** yêu cầu hai tham số
  - Tham số thứ nhất là số lượng các biến cần cấp phát bộ nhớ
  - Tham số thứ hai là kích thước của mỗi biến
  - Cú pháp

```
int *p= (int *)malloc(10, sizeof(int))
```



# Cấp phát thêm/bớt

- Có thể cấp phát lại cho một vùng đã được cấp (thêm/bớt số bytes) bằng cách sử dụng hàm **realloc**, mà không làm mất dữ liệu.
- Hàm **realloc** yêu cầu hai tham số
  - Tham số thứ nhất là con trỏ tham chiếu đến bộ nhớ
  - Tham số thứ hai là tổng số byte muốn cấp phát
  - Cú pháp

```
int *p= (int *)realloc(p, 20*sizeof(int))
```



# Giải phóng bộ nhớ

- Hàm **free()** được sử dụng để giải phóng bộ nhớ khi nó không cần dùng nữa.
- Cú pháp

`void free(void*ptr);`

- Hàm này giải phóng không gian được trả bởi *ptr*, để dùng cho tương lai.
- *ptr* phải được dùng trước đó với lời gọi hàm *malloc()*, *calloc()*, hoặc *realloc()*.

# Ví dụ biến tĩnh và biến động

```
int x;
```

```
x=5;
```

Biến không động x

5

Biến con trỏ p

0xFF

```
int *b;
```

```
b=(int*)malloc(s
```

```
*b=5
```

0xFF

5

Biến động có địa chỉ 0xFF

# Cấp phát bộ nhớ động với new

- Cấp phát bộ nhớ động với hàm new

```
new <kiểu_dữ_liệu>;
```

Ví dụ new int;

Khi tạo biến động gán luôn địa chỉ của nó :

```
int *ptr = new int;
```

```
int *ptr1 = new int(2409); khởi tạo
```

- Thu hồi bộ nhớ động

```
delete <tên_biến_con_trỏ>;
```

```
delete ptr; ptr = nullptr;
```



# Cấp phát bộ nhớ động với new

## ■ Mảng động

```
new <kiểu_dữ_liệu_>[size];
```

```
ví dụ new int[100];
```

```
int *myArr = new int[100];
```

## ■ Truy cập

```
myArr[0] = 1;
```

```
*(myArr + 0) = 1;
```

## ■ Thu hồi bộ nhớ

```
delete[] ptr;
```





# Ví dụ cấp phát bộ nhớ động

- Nhập n phần tử số nguyên từ bàn phím, in kết quả ra màn hình: code\_taobiendong
- Xây dựng dưới dạng hàm
  - Hàm nhập danh sách
  - Hàm In danh sách
  - Hàm sắp xếp
  - ...



# Bài tập

---

- Xây dựng cấu trúc sinh viên bao gồm: Mã sv, Họ tên, Lớp, điểm. Viết chương trình tổ chức dưới dạng hàm:
- Cách 1: dùng mảng
  - Nhập danh sách n sinh viên từ bàn phím
  - In danh sách n sinh viên ra màn hình
  - Sắp xếp danh sách sinh viên theo tăng dần theo điểm
  - Tìm sinh viên có mã sv nhập từ bàn phím
- Cách 2: dùng cấp phát động