**C (-) Memory**

* Allocate:

**Stactic memory** là vùng nhớ được cấp phát cho biến trước khi chương trình được chạy. có thể được xem là **compile time allocation**.

\*compile time là giai đoạn chuyển source code thành machine code, mà trong quá trình đó có bao gồm việc cấp phát bộ nhớ tĩnh cho các biến được khai báo.

Ví dụ: khi đăng môn học có 20 student apply, tương đương với students[20] và kích thước bộ nhớ cần cấp là 80(20\*4bytes). Nhưng khi bắt đầu khoá học, chỉ còn 12 students so we lost 8 location, tuy nhiên không thay đổi được kích thước mảng => phí. **Vì vậy dynamic allocation is necessary.**

**Dynamic memory** được cấp phát sau khi chương trình được chạy, có thể được xem như là **runtime allocation**.

Sử dụng dynamic có thể giúp kiểm soát được toàn quyền về bộ nhớ động of **anytime** và chỉ có thể truy cập được bằng con trỏ chứ không phải biến.

Việc cấp phát có thể sử dụng các hàm **malloc()** và **calloc()** của **stdlib.h**

Với hàm **malloc()** chỉ có 1 thông số cần thiết lập là size of need.

Hàm **calloc()** có 2 thông số: **amount** là số lượng phần tử cần cấp phát và **size** là kích thước của một phần tử.

Về cơ bản 2 cái này nó cũng giống nhau ☺

**Lưu ý:** ptr được cấp phát bằng malloc() có dữ liệu không biết trước **so chúng ta phải write somethings** ☺. Calloc() thì lại gán tất cả giá trị bằng 0 khi vừa được khởi tạo.

**Another lưu ý:** trong câu lệnh sizeof(), ta có 2 cách viết

01 malloc(sizeof(int)) // ví dụ biến con trỏ được khai báo theo kiểu dữ liệu int

02 malloc(sizeof(\*ptr))

Hai cách viết này tương dương nhau, nhưng ở các viết thứ 2 nếu thay **(\*ptr) thành (ptr)** thì chắc chắn nó sẽ xảy ra lỗi cấp phát sai do. **(\*ptr) là đang lấy kích thước của dữ liệu tại địa chỉ mà con trỏ trỏ đến.** Trong khi (ptr) thì nó sẽ lấy kích thước con trỏ, mặc định là 8 byte **oke nhé ☺**.

* Access

Truy cập đến dynamic memory tương tự như mảng, kiểu dữ liệu của this memory được follow theo kiểu dữ liệu mà pointer được khai báo

Có thể truy cập theo 2 cách đối với phần tử bất kì, ví dụ với phần tử đầu tiên có thể truy cập **ptr[0] = 12;** hoặc **\*ptr = 12**, với phần tử tiếp theo **ptr[1] hoặc \*(ptr+1)** là hai cách truy cập.

* Reallocate

Nếu dữ liệu bộ nhớ ban đầu cấp phát là không đủ, ta có thể tái cấp phát reallocate to make it larger. **Reallocate** dành ra một lượng bộ nhớ, thường là lớn hơn trong khi vẫn giữ được nội dung dữ liệu được lưu vào trước đó.

Cú pháp: **realloc(ptr1, size);**

Trong cú pháp có 2 thông số bao gồm:

01 con trỏ tới vùng nhớ cần được tái kích thước.

02 kích thước được chỉ định (specifies) thay đổi **với đơn vị là byte.**

**Lưu ý:** khi tái cấp phát bộ nhớ mà địa chỉ của con trỏ (mặc định là vị trí ban đầu bị khác đi), thì chúng không an toàn để sử dụng, thay vào đó nên tạo 1 con trỏ mới và gán cho vị trí được cấp phát.

|  |
| --- |
| int \*ptr1, \*ptr2; // Allocate memory ptr1 = malloc(4); // Attempt to resize the memory ptr2 = realloc(ptr1, 8);  // Check whether realloc is able to resize the memory or not if (**ptr2 == NULL**) {   // If reallocation fails   printf("Failed. Unable to resize memory"); } else {   // If reallocation is sucessful   printf("Success. 8 bytes reallocated at address %p \n", ptr2);   ptr1 = ptr2;  // Update ptr1 to point to the newly allocated memory } |

Khi cấp phát bộ nhớ (đương nhiên là đang nói về con trỏ rồi), thì nên kiểm tra con trỏ có NULL hay không, các lỗi khi sử dụng 1 con trỏ chưa được cấp phát thành công gồm:

1. Crash chương trình: dereference(sử dụng để truy cập vùng nhớ), lỗi truy cập bộ nhớ (segmentation fault) dẫn đến suddenly stop.
2. Dữ liệu không hợp lệ: ảnh hưởng đáng tin cậy của chương trình.
3. Khó Debug: khó phát hiện và sửa chữa trong chương trình chính.

Ngoài ra còn nên release ptr mỗi khi sài xong, **cảm ơn.**

* Deallocate: cái này nói đến memory leak và free().