**Dynamic memory allocation**

* Single **:**

1. Công dụng, cách dùng:

* C++ có 3 loại cấp phát bộ nhớ : Tĩnh (cấp phát biến để xài hết chương trình), tự động(cấp phát trong hàm, hết hàm tự xóa biến), và cấp phát động.
* 2 cái cấp phát kia thì nó dễ gây lãng phí bộ nhớ, bị giới hạn bởi bộ nhớ (stack), nên dễ bị lỗi. Từ đó cấp phát động ra đời.
* Cấp phát động: Là cách để yêu cầu bộ nhớ từ hệ điều hành khi cần thiết.(không bị giới hạn bởi hệ điều hành, tất nhiên vẫn bị giới hạn bởi phần cứng)

1. Syntax:

new + <datatype>; (không giới hạn dữ liệu)

* Cơ chế: new operator tạo cho ta 1 vùng nhớ có thể sử dụng và trả về địa chỉ vùng nhớ ấy. Chính vì vậy, ta phải sử dụng con trỏ để lưu địa chỉ vùng nhớ đấy để xài.

VD: int \*ptr=new int;

* Ta có địa chỉ của vùng nhớ đấy. sau đó chỉ cần thao tác trên con trỏ là được.

\*ptr=10000000; (gán giá trị cho vùng nhớ đấy)

* Ta có thể khởi tạo giá trị cho cấp phát bộ nhớ động.

Vd: int \*ptr=new int{5};

1. Delete operator:

* Khi ta không còn sử dụng 1 biến được cấp phát động, ta cần trao quyền sử dụng bộ nhớ của biến đấy cho hệ điều hành:

Sytax: int \*ptr= new int;

delete ptr;

ptr=nullptr; // nhớ gắn nó về gtri null, khi ở gtr null nó sẽ có nghĩa là ptr không chứa bất kỳ địa chỉ nào.

* Như đã nói trên, toán tử delete dereference; không thật sự xóa đi ptr mà nó chỉ trao quyền cho hệ điều hành mà thôi, nên biến trỏ ptr vẫn sử dụng được, nhưng ta nên cho nó về giá trị null để tránh các trường hợp không mong muốn.

1. Dangling pointers:

* Khi delete 1 con trỏ, thì con trỏ này sẽ trỏ vào 1 vùng nhớ chưa được cấp phát (vì vùng nhớ này đã được giao cho HĐH). Con trỏ như vậy gọi là con trỏ lơ lửng, Dangling pointers. Truy cập vào con trỏ lơ lửng thì auto dính lỗi undefined behaviors.
* Khi giải phóng 1 vùng nhớ có nhiều con trỏ trỏ vào, thì bọn trỏ vào auto lơ lửng.
* Đừng trỏ nhiều con trỏ vào 1 vùng nhớ quá, fix bug tê người.

1. Con trỏ null trong cấp phát động:

* Con trỏ null ở trong cấp phát động có nghĩa là “biến trỏ không lưu bất kỳ địa chỉ nào”.
* Có thể dùng để kiểm tra xem con trỏ đã được cấp phát chưa.(đây là lý do tại sao không dùng con trỏ rác, vì con trỏ rác cx chứa địa chỉ, vì vậy nó sẽ không thể được phân biệt với con trỏ cấp phát động).

1. Memory leaks:

* Xảy ra khi ta đã tạo cấp phát động nhưng lại mất địa chỉ vùng cấp phát đó, từ đó vùng đó sẽ bị dư, không thể giải phóng được , gây nặng cho chương trình.
* Ta có thể tắt chương trình để trao quyền bộ nhớ bị mất kiểm soát đó cho HĐH ☺
* Ví dụ:

Void dosomething()

{

Int \*ptr=new int;

}

Khi cho hàm này chạy thì ta thấy biến trỏ ptr sẽ bị xóa khi ra khỏi khối lệnh từ đó ta sẽ đánh mất địa chỉ của vùng dữ liệu new int mới khai báo, ít lần thì k sao, chứ nhiều quá thì dó dư 1 đống vùng dữ liệu từ đó gây nặng ctr thậm chí làm ctr crack.

OR: gán địa chỉ lung tung:

Int value=10;

Int \*ptr=new int;

Ptr=&value;

* Mất mất địa chỉ của new int.

OR: tạo với gán nhiều địa chỉ:

Int \*ptr=new int;

Ptr=new int;

* Mất mất địa chỉ cái vùng của new int phía trên.
* Đừng để mất địa chỉ or coi trao quyền địa chỉ cho vùng dữ liệu khi thấy k dùng nữa rồi gán null cho chắc.
* Array :

1. Syntax:

Length= so phan tu trong mang.

Int \*array=new int[length]

1. Notes:

* Xong thì xài như mảng bình thường.
* Phải khai báo độ dài rõ ràng cho mảng khi khởi tạo.
* Ngoài ra ta có thể khởi tạo 1 mảng có n phần tử 0:

Int \*array=new int[length] ().

* Ta có thể thay đổi kích thước của mảng.
* 2-dimentional array:

1. Syntax:

Int \*\*arr= new int\*[row];

For (int i=0; i< row; i++)

Arr[i]=new int[col];

1. Có thể dùng khai báo mảng 1 chiều để thay thế mảng 2 chiều:

Int get\_single\_index(int row, int col, int col\_max)

{

Return (row\*col\_max)+col;

}

Int row=5;

Int col=10;

Int n=row\*col; //=50

Int \*arr=new int[n];

Vd: A[1,3]=10

Arr[get\_single\_index(1,3,10)]= 10;