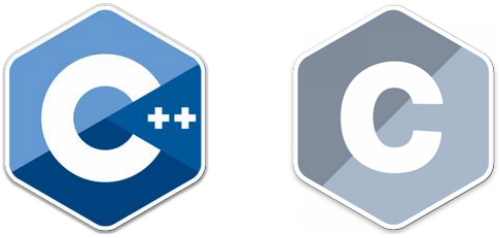
CON TRỎ

Phần 4 – Mảng động và cấp phát động bộ nhớ



[2017 – 04 – 24]

Biên soạn bởi: Nguyễn Trung Thành
https://www.facebook.com/thanh.it95

> “ Học từ cái đáy của thế giới đi lên là cách duy nhất bạn trở thành master. ”
>
> Nguyễn Trung Thành

Mục lục

A. Giới thiệu

1. Điều kiện để học được tài liệu này

- Có kiến thức nền tảng vững chắc với ngôn ngữ C (hoặc C++).
- Đã học xong 3 bài giảng trước đó (con trỏ phần 1, phần 2 và phần 3).

2. Dẫn nhập

Vấn đề đơn giản được đặt ra khi bạn mới học về mảng.

> Khai báo 1 mảng số nguyên có TỐI ĐA 10 phần tử

```
int a[10];
int n = 0; // số lượng phần tử của mảng
```

Giả sử người dùng "rảnh đời", nhập vào mảng có 11 phần tử (n = 11), hay là n = 15, hay là n = 20 thì sao ????

> Chương trình sẽ truy xuất bộ nhớ trái phép (vượt quá phạm vi của mảng) và có thể báo lỗi.

Để tránh trường hợp "người dùng rảnh đời hoặc ngu quá", ta sẽ khai báo 1 mảng thật là bự.

```
int a[100000];
int n = 0; // số lượng phần tử của mảng
```

Tuy nhiên, lỡ người dùng thường xuyên sử dụng khoảng...5 phần tử của mảng, hay là nhiều lắm là sử dụng 20 phần tử của mảng.

➔ Liệu việc khai báo như trên là hợp lý chưa ? Chương trình đang lãng phí bộ nhớ trầm trọng.

Khi khai báo 1 mảng với số lượng phần tử rất lớn, ta đang chiếm dụng nhiều RAM máy tính.

Điều này là hoàn toàn không tốt.

Loại mảng ta khai báo bình thường gọi là ***mảng***.

> Mảng bình thường là khi ta khai báo 1 cái là **cố định số lượng phần tử tối đa** luôn, người dùng lỡ muốn sử dụng nhiều hơn số lượng phần tử tối đa thì….die.

VÌ VẬY NÊN

Ta cần có 1 cơ chế nào đó mà khai báo mảng vừa khớp với nhu cầu sử dụng của người dùng.

➔ **Mảng động**.

B. Mảng động

Mảng động là gì, rất đơn giản, mảng động là mảng có kích thước "phù hợp với nhu cầu người sử dụng".

Người sử dụng muốn n = 20, mảng sẽ có tối đa 20 phần tử.

Người sử dụng muốn n = 10000, mảng sẽ có tối đa 10000 phần tử.

Như vậy, *người lập trình cần phải chủ động xin xỏ bộ nhớ từ hệ điều hành vì chỉ có chủ động thì mới tạo ra mảng đủ số lượng phần tử theo ý người dùng*.

Code giả lập (pseudocode)

```
int main()
{
	// khai báo mảng động tên là a.
	int *a = NULL;

	// số lượng phần tử của mảng động
	int n = 0;

	// yêu cầu người dùng nhập vào n

	// xin xỏ bộ nhớ chứa đủ n phần tử
	// giả sử bộ nhớ này có địa chỉ là 504

	// gán a = 504

	return 0;
}
```

Code thực tế sẽ như sau:

C

```c
#include <stdio.h>
#include <stdlib.h> // malloc

int main()
{
	int *a = NULL;
	int n = 0;

	printf("Nhap so luong phan tu cua mang: ");
	scanf("%d", &n);

	a = (int*)malloc(n * 4);

	// sử dụng a như mảng bình thường

	return 0;
}
```

Giả sử ban đầu ta có bối cảnh như hình dưới.

byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất trống chưa ai sử dụng											

Chương trình hiện ra dòng chữ “Nhap so luong phan tu cua mang: ”. Giả sử người dùng nhập vào n = 3.

Mảng a là mảng các số nguyên int. Ta biết kiểu int có độ lớn 4 bytes → ta cần xin xỏ n * 4 = 3 * 4 = 12 bytes để lưu nội dung mảng a.

Hàm malloc là hàm có nhiệm vụ “đi xin xỏ bộ nhớ RAM do hệ điều hành cung cấp” (người nông dân đi thuê đất của Nhà nước để làm ăn). Trong đoạn code trên, hàm malloc đã xin xỏ 12 bytes.

byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất trống chưa ai sử dụng											

Hệ điều hành phát hiện từ byte 504 đến byte 515 là đất trống mà lại vừa đủ 12 bytes → hệ điều hành quyết định cho người nông dân thuê 12 ô đất trống này để sử dụng.

Và kết quả là…

byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất của chương trình của mình											

Từ byte 504 đến byte 515 trong bộ nhớ đã thuộc quyền sở hữu của chương trình.

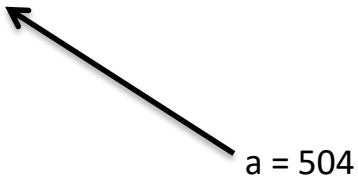
→ Hàm malloc trả về giá trị là 504.

→ a = 504.

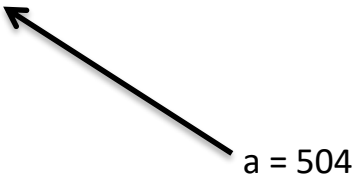
C

```c
int *a = NULL;
int n = 0;

printf("Nhap so luong phan tu cua mang: ");
scanf("%d", &n);

a = (int*)malloc(n * 4);

// sử dụng a như mảng bình thường
```

byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất của chương trình của mình											



byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất của chương trình của mình											



Wow, xin chúc mừng, giờ đây a đã là 1 mảng chứa vừa đủ 3 số nguyên theo ý của người dùng.

Tuy nhiên, nhằm mang tính chuyên nghiệp hơn, ta nên cải tiến hơn như sau:

Thay thế `a = (int*)malloc(n * 4);`

bằng lệnh

```
a = (int*)malloc(n * sizeof(int));
```

sizeof(int) trả về độ lớn của số nguyên int, tức là sizeof(int) = 4. Sau này bạn có mảng các số thực, mảng các struct PhanSo, mảng các struct HocSinh, bạn làm sao mà xác định rõ được đúng không ? Sử dụng sizeof là tốt nhất.

Giải thích hàm malloc:

Nhằm mang tính tổng quát, hàm malloc trả về kiểu void*.

Vì vậy khi ta sử dụng mảng động các số nguyên int, ta cần ép kiểu thành int*.

Và cuối cùng, ta có đoạn code như sau:

C	Chạy chương trình

```c
#include <stdio.h>
#include <stdlib.h> // malloc

int main()
{
	int *a = NULL;
	int n = 0;
	int i;

	printf("Nhap so luong phan tu cua mang: ");
	scanf("%d", &n);

	// XIN XỎ BỘ NHỚ ĐỂ LƯU MẢNG
	a = (int*)malloc(n * sizeof(int));


	// NHẬP MẢNG
	for (i = 0; i < n; i++)
	{
		printf("a[%d] = ", i);
		scanf("%d", &a[i]);
	}


	// XUẤT MẢNG
	printf("Mang vua nhap la: \n");

	for (i = 0; i < n; i++)
		printf("%5d", a[i]);

	printf("\n");

	return 0;
}
```

```
Nhap so luong phan tu cua mang: 4
a[0] = 9
a[1] = 8
a[2] = 7
a[3] = 6
Mang vua nhap la:
    9    8    7    6
```

Wow, thật tuyệt vời. Cuối cùng ta cũng có thể tạo ra mảng “phù hợp với nhu cầu người sử dụng” rồi.

Người sử dụng nhập n = 10 cũng được, nhập n = 1000 cũng chơi được luôn.

Tuy nhiên bạn vẫn chưa xong.

Nếu người nông dân đã chủ động xin xỏ đất để sử dụng thì cũng phải chủ động trả lại đất cho Nhà nước.

Câu lệnh "free" trong ngôn ngữ C chính là hành động trả lại bộ nhớ cho hệ điều hành.

C

```c
#include <stdio.h>
#include <stdlib.h> // malloc, free

int main()
{
	int *a = NULL;
	int n = 0;

	printf("Nhap so luong phan tu cua mang: ");
	scanf("%d", &n);

	a = (int*)malloc(n * sizeof(int));

	// sử dụng a như mảng bình thường

	free(a);

	return 0;
}
```

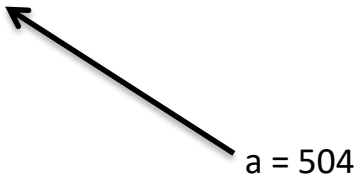
Giả sử ta có a = 504.

Đoạn code trên sẽ gọi free(a), tức là free(504).

Hệ điều hành sẽ tìm đến địa chỉ 504, phát hiện đây là đất thuê của chương trình. Hệ điều hành sẽ lấy lại đất đã thuê. Như vậy chương trình KHÔNG CÒN QUYỀN SỬ DỤNG ĐẤT TẠI ĐỊA CHỈ 504 NỮA.

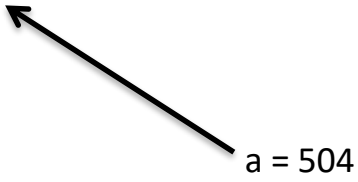
Trước khi free:

byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất của chương trình của mình											



Sau khi free:

byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509	byte 510	byte 511	byte 512	byte 513	byte 514	byte 515
Đất của nông dân khác			Đất tự do, không còn thuộc quyền sử dụng của mình											



Bạn nghĩ xem, sau khi free mà ta có lệnh a[0] = 7 thì chuyện gì xảy ra ? Có phải giống như "nông dân sử dụng đất trái phép" đúng không ?

Vì vậy, thao tác tốt là sau khi free thì ta gán a = NULL.

Đoạn code hoàn chỉnh của ta như sau:

C	Chạy chương trình

```c
#include <stdio.h>
#include <stdlib.h> // malloc, free

int main()
{
	int *a = NULL;
	int n = 0;
	int i;

	printf("Nhap so luong phan tu cua mang: ");
	scanf("%d", &n);

	// XIN XỎ BỘ NHỚ ĐỂ LƯU MẢNG
	a = (int*)malloc(n * sizeof(int));


	// NHẬP MẢNG
	for (i = 0; i < n; i++)
	{
		printf("a[%d] = ", i);
		scanf("%d", &a[i]);
	}


	// XUẤT MẢNG
	printf("Mang vua nhap la: \n");

	for (i = 0; i < n; i++)
		printf("%5d", a[i]);

	printf("\n");


	// GIẢI PHÓNG BỘ NHỚ
	free(a);
	a = NULL;

	return 0;
}
```

```
Nhap so luong phan tu cua mang: 4
a[0] = 9
a[1] = 8
a[2] = 7
a[3] = 6
Mang vua nhap la:
    9    8    7    6
```

Nhìn đơn giản chứ không đơn giản cho lắm đúng không ☺.

C	C++

`#include <stdio.h>`
`#include <stdlib.h> // malloc, free`
 	
`int main()`
`{`
`int *a = NULL;`
`int n = 0;`
 	
`printf("Nhap so luong phan tu cua mang: ");`
`scanf("%d", &n);`
 	
`// XIN XỎ BỘ NHỚ ĐỂ LƯU MẢNG có n số nguyên`
`a = (int*)malloc(n * sizeof(int));`
 	
`// sử dụng mảng`
 	
`// GIẢI PHÓNG BỘ NHỚ`
`free(a);`
`a = NULL;`
 	
`return 0;`
`}`
	`#include <iostream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`int *a = NULL;`
	`int n = 0;`
	
	`cout << "Nhap so luong phan tu cua mang: ";`
	`cin >> n;`
	
	`// XIN XỎ BỘ NHỚ ĐỂ LƯU MẢNG có n số nguyên`
	`a = new int[n];`
	
	`// sử dụng mảng`
	
	`// GIẢI PHÓNG BỘ NHỚ`
	`delete[] a;`
	`a = NULL;`
	
	`return 0;`
	`}`

Khi sử dụng mảng động thì bạn cần lưu ý 2 vấn đề:

- ***Chủ động*** xin xỏ bộ nhớ vừa đủ để sử dụng.
- ***Chủ động*** giải phóng bộ nhớ (KHÔNG ĐƯỢC QUÊN).

C. Một số vấn đề mở rộng

1. Vì sao cần giải phóng bộ nhớ

Ở phần trước, mình rất nhấn mạnh 2 chữ "**chủ động**", tức là cá nhân người lập trình phải tự gọi hàm malloc để xin xỏ bộ nhớ và phải tự gọi hàm free để giải phóng bộ nhớ.

Nếu mình quên giải phóng bộ nhớ thì có chuyện gì xảy ra không ?

Với những người lập trình cơ bản thì sẽ không có chuyện gì xảy ra, vì họ chỉ quan tâm làm được đúng bài tập là ok.

Nhưng với những người lập trình chuyên nghiệp thì điều này gây hậu quả rất ghê gớm.

Không giải phóng bộ nhớ cũng như việc nông dân không trả lại đất cho Nhà nước vậy. Cho dù họ chết đi thì đất của họ vẫn còn, không ai làm gì được → nếu như có quá nhiều nông dân như vậy, sau 1 thời gian ngắn thì sẽ hết đất sử dụng → gây ra rất nhiều vấn đề. Điều này cũng giống như khi máy bạn có RAM quá ít hoặc bạn sử dụng hết RAM của máy tính, chương trình sẽ chạy chậm lại, mọi thứ thật tồi tệ.

Chỉ khi chúng ta reset lại mọi thứ (ví dụ như khởi động lại máy tính) hoặc Nhà nước có cơ chế "đi khảo sát tình hình của nhân dân, xem thử có nông dân nào qua đời mà chưa trả lại đất không" thì may ra bộ nhớ có thể được thu hồi.

➔ Do đó, bạn hãy luôn luôn nhớ 1 chuyện: có xin xỏ bộ nhớ thì phải có giải phóng bộ nhớ, trả lại đất trống cho người khác sử dụng tiếp, vậy mới là hành động văn minh lịch sự nhé.

2. Cấp phát bình thường vs cấp phát động

Mảng ban đầu thật là đơn giản, mới vào khai báo là cố định luôn số lượng phần tử tối đa.

```
int a[20];
```

Mảng động thì ngược lại, phải xin xỏ bộ nhớ rồi lại phải giải phóng bộ nhớ.

```
int *a = NULL;
a = (int*)malloc(20 * sizeof(int));
free(a);
a = NULL;
```

Vậy rốt cuộc bản chất của sự khác biệt là gì đây ?

Chúng ta sẽ tổng quát hóa lên, thay vì "mảng bình thường" và "mảng động" thì tổng quát lên thành "cấp phát bình thường và cấp phát động".

Cấp phát bình thường

> Người nông dân xin đất của Nhà nước có giấy tờ hợp lệ. Trong giấy tờ ghi rõ thời hạn trả đất. Khi đến thời hạn là Nhà nước cưỡng ép thu hồi lại đất của nông dân.

Cấp phát động bộ nhớ

> Người nông dân xin đất của Nhà nước có giấy tờ hợp lệ. Nhà nước "tin tưởng" nên trong giấy tờ KHÔNG ghi rõ thời hạn trả đất. Vì vậy Nhà nước không biết khi nào nông dân trả lại đất → người nông dân cần phải "là người có đạo đức, có văn hóa" và phải chủ động trả lại đất khi sử dụng xong.

Chúng ta cùng nhau đi sâu hơn nhé ☺

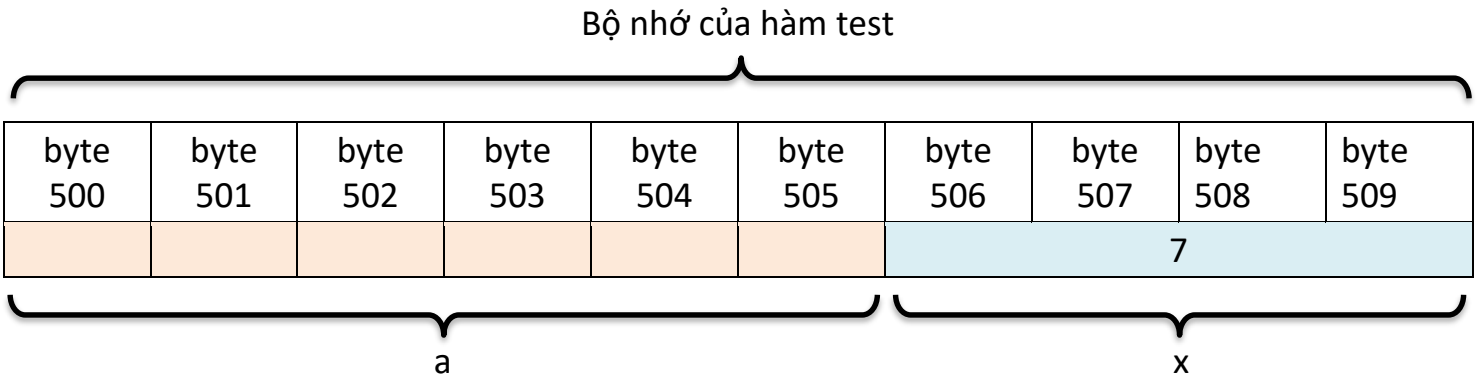
Cấp phát bình thường

Cho đoạn code như sau:

```
1   void test(int x)
2   {
3           char a[6];
4   }
5
6   int main()
7   {
8           test(7);
9
10          return 0;
11  }
```

Ban đầu chạy hàm main, tại dòng code số 8 thì hàm main sẽ gọi test(7).

Hệ điều hành sẽ **cấp phát bộ nhớ** cho hàm test. Cụ thể hơn, hàm test bao gồm biến x và mảng char có 6 phần tử.



(Bạn nhìn vào đoạn code ở trên) Bên trong hàm test, khi chạy đến dòng code số 4 là kết thúc. Đây chính là "***thời hạn sử dụng đất của nông dân***". Khi đã đến thời hạn rồi thì Nhà nước sẽ cưỡng chế thu hồi lại đất. Lúc này hệ điều hành sẽ thu hồi lại toàn bộ bộ nhớ của hàm test.

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507	byte 508	byte 509
Đất tự do không ai sử dụng									

Cấp phát động

Cho đoạn code như sau:

```
1   void test(int x)
2   {
3           char *a = NULL;
4           a = (char*)malloc(6 * sizeof(char));
5
6           a[0] = 65;
7
8           free(a);
9           a = NULL;
10  }
11
12  int main()
13  {
14          test(7);
15
16          return 0;
17  }
```

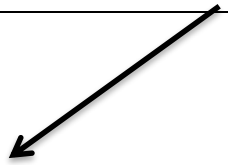
Ban đầu chạy hàm main, tại dòng code số 14 thì hàm main sẽ gọi test(7).

Hệ điều hành sẽ **cấp phát bộ nhớ** cho hàm test. Cụ thể hơn, hàm test bao gồm biến x và con trỏ a. Với kiến trúc x86 (32-bit) thì con trỏ a sẽ chiếm 4 bytes.

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
a = NULL				x = 7			

Tại dòng code số 4: hàm malloc đi xin xỏ đất để sử dụng. Đất được xin xỏ sẽ "không có thời hạn sử dụng". Hệ điều hành phát hiện byte 200 đến byte 205 có vừa đủ 6 ô nhớ liên tục mà không ai sử dụng → hệ điều hành cấp quyền sử dụng đất, hàm malloc trả về giá trị là 200.

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
a = 200				x = 7			

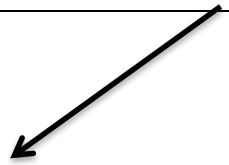


byte 200	byte 201	byte 202	byte 203	byte 204	byte 205
dữ liệu của mảng a					

```
1   void test(int x)
2   {
3           char *a = NULL;
4           a = (char*)malloc(6 * sizeof(char));
5
6           a[0] = 65;
7
8           free(a);
9           a = NULL;
10  }
11
12  int main()
13  {
14          test(7);
15
16          return 0;
17  }
```

Tại dòng code số 6: gán a[0] = 65 (tức là gán a[0] = kí tự 'A' đó).

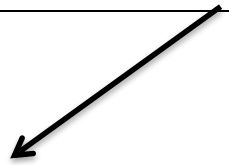
byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
a = 200				x = 7			



byte 200	byte 201	byte 202	byte 203	byte 204	byte 205
65					

Tại dòng code số 8: thể hiện đây là người nông dân văn minh có đạo đức. Người nông dân sử dụng mảng a xong thì bây giờ trả lại đất cho Nhà nước.

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
a = 200				x = 7			



byte 200	byte 201	byte 202	byte 203	byte 204	byte 205
Đất tự do không ai sử dụng					

```
1   void test(int x)
2   {
3           char *a = NULL;
4           a = (char*)malloc(6 * sizeof(char));
5
6           a[0] = 65;
7
8           free(a);
9           a = NULL;
10  }
11
12  int main()
13  {
14          test(7);
15
16          return 0;
17  }
```

Tại dòng code số 9: gán a = NULL (tức là a không trỏ vào cái gì nữa).

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
a = NULL				x = 7			

byte 200	byte 201	byte 202	byte 203	byte 204	byte 205
Đất tự do không ai sử dụng					

Tại dòng code số 10: kết thúc hàm test, hệ điều hành thu hồi lại bộ nhớ của hàm test.

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
Đất tự do không ai sử dụng							

byte 200	byte 201	byte 202	byte 203	byte 204	byte 205
Đất tự do không ai sử dụng					

Vậy là giờ bạn đã hiểu được sự khác nhau giữa cấp phát bộ nhớ bình thường và cấp phát động rồi chứ ?

Bạn hãy rà soát lại các bước chạy trong phần cấp phát động. GIẢ SỬ CHÚNG TA QUÊN GIẢI PHÓNG BỘ NHỚ (quên gọi hàm free) thì chuyện gì xảy ra ?

byte 500	byte 501	byte 502	byte 503	byte 504	byte 505	byte 506	byte 507
Đất tự do không ai sử dụng							

Đất của nông dân

byte 200	byte 201	byte 202	byte 203	byte 204	byte 205
65					

Kể cả khi kết thúc hàm test và kết thúc chương trình, vùng đất 200 đến 205 vẫn thuộc quyền sử hữu của nông dân. Từ đó về sau không ai đụng vào được nữa mặc dù chương trình đã kết thúc, người nông dân đã qua đời.

→ Lãng phí bộ nhớ.

Bạn tưởng tượng 1 máy tính mà có nhiều người nông dân vô đạo đức như vậy thì RAM sẽ ngày càng cạn kiệt vì càng ngày bị chiếm dụng, ***chỉ có vay mượn chớ không có trả đất***.

→ Đây là lý do vì sao mình yêu cầu bạn cần phải ***chủ động*** trả lại đất sau khi bạn ***chủ động*** mượn đất.

Trong trường hợp "cấp phát bộ nhớ bình thường" thì Nhà nước có ghi chú lại thời hạn trả đất, vì vậy Nhà nước tự động thu hồi đất.

Wow, cái gì cũng có cái giá của nó nhỉ, mảng linh động số lượng phần tử thì phải chú ý cẩn thận hơn so với mảng bình thường.

D. Tổng kết

Hành vi mượn đất mà không trả đất (xin xỏ bộ nhớ mà không giải phóng bộ nhớ) gọi là **MEMORY LEAKS (rò rỉ bộ nhớ)**. Bạn có thể Google để tìm hiểu thêm nhé ☺. Đoạn code ở dưới minh họa vấn đề memory leaks khi không giải phóng bộ nhớ.

C	C++

`#include <stdio.h>`
`#include <stdlib.h>`
 	
`int main()`
`{`
`  int *a = NULL;`
`  a = (int*)malloc(3 * sizeof(int));`
 	
`  return 0;`
`}`
	`#include <iostream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  int *a = NULL;`
	`  a = new int[3];`
	
	`  return 0;`
	`}`

Bộ nhớ bình thường của chương trình do **hệ điều hành chủ động** cấp phát và thu hồi là bộ nhớ STACK.

Bộ nhớ do **chương trình chủ động** xin xỏ (ví dụ như malloc, calloc, new,..) là bộ nhớ HEAP hoặc FREE-STORE.

Ví dụ:

```
1   void test(int x)
2   {
3   	char *a = NULL;
4   	a = (char*)malloc(6 * sizeof(char));
5   
6   	a[0] = 65;
7   
8   	free(a);
9   	a = NULL;
10  }
11  
12  int main()
13  {
14  	test(7);
15  
16  	return 0;
17  }
```

Trong hàm test, biến x và biến con trỏ a là nằm trong STACK. Bộ nhớ được cấp phát bởi hàm malloc thì nằm trong HEAP.

Code minh họa cấp phát động bộ nhớ:

C

```c
#include <stdio.h>
#include <stdlib.h> // malloc, free

int main()
{
	int *a = NULL;
	int n = 0;

	printf("Nhap so luong phan tu cua mang: ");
	scanf("%d", &n);

	// XIN XỎ BỘ NHỚ ĐỂ LƯU MẢNG có n số nguyên
	a = (int*)malloc(n * sizeof(int));


	// sử dụng mảng


	// GIẢI PHÓNG BỘ NHỚ
	free(a);
	a = NULL;

	return 0;
}
```

C++

```cpp
#include <iostream>
using namespace std;

int main()
{
	int *a = NULL;
	int n = 0;

	cout << "Nhap so luong phan tu cua mang: ";
	cin >> n;

	// XIN XỎ BỘ NHỚ ĐỂ LƯU MẢNG có n số nguyên
	a = new int[n];


	// sử dụng mảng


	// GIẢI PHÓNG BỘ NHỚ
	delete[] a;
	a = NULL;

	return 0;
}
```

Ôn lại kiến thức vừa học luôn nhé ☺. Trong hàm main thì ta có con trỏ a, biến n nằm trong bộ nhớ STACK.

Vùng nhớ mà a trỏ đến (được cấp phát bởi malloc hoặc new) thì sẽ nằm ở bộ nhớ HEAP / FREE-STORE.

Ngoài ra, bạn cũng nên thử tìm hiểu về sự khác nhau giữa hàm malloc và hàm calloc.

E. Bài tập

Mặc định là bạn sử dụng mảng động hết nhé. Người dùng sẽ nhập vào số lượng phần tử của mảng và chương trình sẽ cấp phát bộ nhớ vừa đủ để sử dụng.

Tất cả bài tập chỉ cần viết code toàn bộ trong hàm main. Nếu cần thiết bạn viết thêm 1 hàm XuatMang là đủ.

Bài 1. Nhập vào mảng, xuất mảng, tính tổng các phần tử của mảng.

Bài 2. Nhập vào mảng, đảo ngược nội dung của mảng đó.

Bài 3. Ghép mảng.

Nhập vào 2 mảng a và b, tạo ra mảng c bằng cách ghép mảng a và mảng b. Mảng c có VỪA ĐỦ số lượng phần tử.

Ví dụ:

a = { 9, 8, 7 } với na = 3

b = { 5, 3, 4, 6 } với nb = 4

⇨ c = { 9, 8, 7, 5, 3, 4, 6 } với nc = 7.
Mảng c chứa TỐI ĐA 7 phần tử (vừa đủ, không dư).

Bài 4. Bạn hãy chạy thử 2 đoạn code sau, cho giải thích kết quả chạy được, so sánh kết quả chạy được giữa 2 hàm testBinhThuong và testCapPhatDong.

Giải thích bằng cách comment trong đoạn code.

Nếu bạn đang sử dụng ngôn ngữ C:

C

```c
#include <stdio.h>
#include <stdlib.h> // malloc, free

void testBinhThuong()
{
	int a[3];
	int i;

	printf("\nTEST BINH THUONG \n\n");

	printf("&a = %d \n", &a);

	for (i = 0; i < 3; i++)
		printf("&a[%d] = %d \n", i, &a[i]);
}


void testCapPhatDong()
{
	int *a = NULL;
	int i;

	a = (int*)malloc(3 * sizeof(int));

	printf("\nTEST CAP PHAT DONG \n\n");

	printf("&a = %d \n", &a);

	for (i = 0; i < 3; i++)
		printf("&a[%d] = %d \n", i, &a[i]);

	free(a);
	a = NULL;
}


int main()
{
	testBinhThuong();
	testCapPhatDong();

	return 0;
}
```

(Bài 4 – tiếp theo).

Nếu bạn đang sử dụng ngôn ngữ C++:

C++

```
#include <iostream>
using namespace std;

void testBinhThuong()
{
	int a[3];

	cout << "\nTEST BINH THUONG \n\n";

	cout << "&a = " << &a << endl;

	for (int i = 0; i < 3; i++)
		cout << "&a[" << i << "] = " << (int)&a[i] << endl;
}


void testCapPhatDong()
{
	int *a = NULL;

	a = new int[3]; // cấp phát bộ nhớ để lưu 3 số nguyên int

	cout << "\nTEST CAP PHAT DONG \n\n";

	cout << "&a = " << &a << endl;

	for (int i = 0; i < 3; i++)
		cout << "&a[" << i << "] = " << (int)&a[i] << endl;

	delete[] a;
	a = NULL;
}


int main()
{
	testBinhThuong();
	testCapPhatDong();

	return 0;
}
```

F. Lời kết

Sang phần sau, bạn sẽ học gì ?

Bạn sẽ tiếp tục học về mảng động cho đầy đủ.

Bắt đầu lướt qua về “con trỏ đa cấp”.

Có 1 ý niệm về “tham chiếu”.

Hi vọng tài liệu này sẽ giúp ích cho bạn. Cảm ơn bạn đã xem.

Tác giả: Nguyễn Trung Thành
Facebook: https://www.facebook.com/thanh.it95