BUỔI 18: MẢNG ĐỘNG -- FULL HOUSE

1. **Định nghĩa:** Đôi khi kích thước của mảng bạn khai báo có thể **không đủ sài**. Để giải quyết vấn đề này, bạn có thể **cấp phát thêm** bộ nhớ theo cách thủ công trong thời gian chạy chương trình. Đó cũng chính là khái niệm **cấp phát động trong C**.

2. So sánh với mảng tĩnh:

Cấp phát bộ nhớ tĩnh	Cấp phát bộ nhớ động
Bộ nhớ được cấp phát trước khi chạy chương trình (trong quá trình biên dịch)	Bộ nhớ được cấp phát trong quá trình chạy chương trình.
Không thể cấp phát hay phân bổ lại bộ nhớ trong khi chạy chương trình	Cho phép quản lý, phân bổ hay giải phóng bộ nhớ trong khi chạy chương trình
Vùng nhớ được cấp phát và tồn tại cho đến khi kết thúc chương trình	Chỉ cấp phát vùng nhớ khi cần sử dụng tới
Chương trình chạy nhanh hơn so với cấp phát động	Chương trình chạy chậm hơn so với cấp phát tĩnh
Tốn nhiều không gian bộ nhớ hơn	Tiết kiệm được không gian bộ nhớ sử dụng

3. Ưu và nhược điểm

- **Ưu điểm:** Ưu điểm chính của việc sử dụng cấp phát động là giúp ta tiết kiệm được không gian bộ nhớ mà chương trình sử dụng. Bởi vì chúng ta sẽ chỉ cấp phát khi cần dùng và có thể giải phóng vùng nhớ đó ngay sau khi sử dụng xong.
- **Nhước điểm:** Nhược điểm chính của cấp phát động là bạn phải tự quản lý vùng nhớ mà bạn cấp phát. Nếu bạn cứ cấp phát mà quên giải phóng bộ nhớ thì chương trình của bạn sẽ tiêu thụ hết tài nguyên của máy tính dẫn đến tình trạng tràn bộ nhớ (memory leak).

4. Các hàm cần biết

a. **Sử dụng hàm malloc()**

- Định nghĩa: Hàm **malloc()** thực hiện cấp phát bộ nhớ bằng cách chỉ định **số byte** cần cấp phát. Hàm này trả về con trỏ kiểu **void** cho phép chúng ta có thể ép kiểu về bất cứ kiểu dữ liệu nào.
- Cú pháp: ptr = (castType*) malloc(size);
- Ví dụ: ptr = (int*) malloc(100 * sizeof(int));

b. Sử dụng hàm calloc()

- Định nghĩa: Hàm **malloc()** khi cấp phát bộ nhớ thì vùng nhớ cấp phát đó không được khởi tạo giá trị ban đầu. Trong khi đó, hàm **calloc()** thực hiện cấp phát bộ nhớ và khởi tạo tất cả các ô nhớ có giá trị bằng 0.
- Cú pháp: ptr = (castType*)calloc(n, size);
- Ví dụ: ptr = (int*) calloc(100, sizeof(int));

c. **Sử dụng hàm free()**

- Định nghĩa: Dùng để giải phóng bộ nhớ. Giải phóng ở đây có nghĩa là trả lại vùng nhớ đó cho hệ điều hành và hệ điều hành có thể sử dụng vùng nhớ đó vào việc khác nếu cần.
- Cú pháp: **free**(ptr);

d. **Sử dụng hàm realloc()**

- Định nghĩa: hàm **realloc**() để tái phân bổ lại bộ nhớ, việc cấp phát không phải di chuyển sang vùng nhớ khác mà chỉ mở rộng ra phía sau.
- Cú pháp:

```
ptr=(castType*)realloc(ptr,size);
                                                    );
```

e. Ví dụ:

- **Vd 1**: Tổng kết, nhập, xuất mảng động

```c
#include <stdlib.h>

int main() {
    int n, i, *ptr;
    scanf("%d", &n);
    ptr = (int *)malloc(n * sizeof(int));
//    ptr = (int *)calloc(n, sizeof(int));

    if (ptr == NULL) {
        printf("Co loi! khong the cap phat bo nho.");
        exit(0);
    }
    for (i = 0; i < n; ++i) {
        scanf("%d", &ptr[i]);
    }

    for (i = 0; i < n; ++i) {
        printf("%d ", ptr[i]);
    }

    int n2;
    scanf("%d",&n2);
    ptr = (int*) realloc(ptr, n2 * sizeof(int));

//    realloc(ptr, 0);
    free(ptr);
    return 0;
}
```

- **Vd 2**: Tính tổng các phần tử của mảng

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

int main() {
    int n, i, *ptr, sum = 0;
    scanf("%d", &n);
    ptr = (int *)malloc(n * sizeof(int));
//    ptr = (int *)calloc(n, sizeof(int));

    if (ptr == NULL) {
        printf("Co loi! khong the cap phat bo nho.");
        exit(0);
    }
    for (i = 0; i < n; ++i) {
//        scanf("%d", ptr + i);
        scanf("%d", &ptr[i]);
    }


    for (i = 0; i < n; ++i) {
//        printf("%d ", *(ptr + i));
        printf("%d ", ptr[i]);
        sum += *(ptr + i);
    }
    printf("\nTong = %d\n", sum);

//    realloc(ptr, 0);
    free(ptr);
    return 0;
}
```

5. Mảng 2 chiều

a. Khai báo

```c
int x, y;
scanf("%d%d", &x,&y);

int **a = (int **)malloc(x * sizeof(int *));
for (int i = 0; i < x; ++i) {
    a[i] = (int *)malloc(y * sizeof(int));
}
```

b. **Nhập mảng**

```
257 void NhapMaTran(int **a, int x, int y) {
258     for (int i = 0; i < x; ++i)
259         for (int j = 0; j < y; ++j) {
260             scanf("%d", &a[i][j]);
261 //          scanf("%d", (*(a+i)+j));
262         }
263 }
264
265 int main() {
266     ...
267     ...
268
269     NhapMaTran(a, x, y);
270
271     ...
272 }
```

c. **Xuất mảng**

```
277 void XuatMaTran(int **a, int x, int y) {
278     for (int i = 0; i < x; ++i) {
279         for (int j = 0; j < y; ++j)
280             printf("%d ", a[i][j]);
281 //          printf("%d ", *(*(a+i)+j));
282         printf("\n");
283     }
284 }
285 int main() {
286     ...
287     ...
288
289     XuatMaTran(a, x, y);
290
291     ...
```

d. Giải phóng

```
NhapMaTran(a, x, y);
XuatMaTran(a, x, y);

for (int i = 0; i < x; ++i) {
    free(a[i]);
}
free(a);
```

e. Tổng quát

```
#include <stdio.h>
#include <stdlib.h>

void NhapMaTran(int **a, int x, int y) {
    for (int i = 0; i < x; ++i)
        for (int j = 0; j < y; ++j) {
            scanf("%d", &a[i][j]);
//          scanf("%d", (*(a+i)+j));
        }
}

void XuatMaTran(int **a, int x, int y) {
    for (int i = 0; i < x; ++i) {
        for (int j = 0; j < y; ++j)
            printf("%d ", a[i][j]);
//          printf("%d ", *(*(a+i)+j));
        printf("\n");
    }
}
int main() {
    int x, y;
    scanf("%d%d", &x,&y);

    int **a = (int **)malloc(x * sizeof(int *));
    for (int i = 0; i < x; ++i) {
        a[i] = (int *)malloc(y * sizeof(int));
    }

    NhapMaTran(a, x, y);
    XuatMaTran(a, x, y);

    for (int i = 0; i < x; ++i) {
        free(a[i]);
    }
    free(a);
    return 0;
}
```