**malloc(), calloc(), free() và realloc()**

**1. Cấp Phát Động**

Cấp phát động (Dynamic memory allocation) là một kỹ thuật giúp bạn có thể xin cấp phát một vùng nhớ phù hợp với nhu cầu của bài toán trong lúc thực thi thay vì phải khai báo cố định.

Cấp phát động thường được sử dụng để cấp phát mảng động hoặc sử dụng trong các cấu trúc dữ liệu.

Có nhiều cấu trúc dữ liệu quan trọng dựa trên kỹ thuật này như : Danh sách liên kết, cây nhị phân hay các cấu trúc dữ liệu dạng mảng động.

Khi bạn sử dụng cấp phát động thì vùng nhớ cấp phát sẽ là vùng nhớ heap.

Trong ngôn ngữ lập trình C cung cấp cho bạn 4 hàm trong thư viện để bạn có thể thao tác với việc cấp phát động vùng nhớ và giải phóng vùng nhớ sau khi sử dụng, bao gồm :

* **malloc()**
* **calloc()**
* **free()**
* **realloc()**

**2. Hàm malloc()**

Hàm malloc() viết tắt của từ memory allocation tức là cấp phát động vùng nhớ, hàm này được sử dụng để xin cấp phát khối bộ nhớ theo kích thước byte mong muốn.

Giá trị trả về của hàm là một con trỏ kiểu void, bạn nên ép kiểu sang kiểu dữ liệu mà bạn cần dùng.

Các giá trị trong các ô nhớ được cấp phát là giá trị rác

Cú pháp : ptr = (cast\_type\*)malloc(byte\_size)

Trong đó :

* ptr là con trỏ lưu trữ ô nhớ đầu tiên của vùng nhớ được cấp phát
* cast\_type\* là kiểu con trỏ mà bạn muốn ép kiểu sang
* byte\_size là kích thước theo byte bạn muốn cấp phát

Ví dụ 1:

#include <stdio.h>

#include <stdlib.h>

int main(){

//Cấp phát vùng nhớ tương đương mảng 100 phần tử int

//sizeof(int) = 4

int \*a = (int\*)malloc(100 \* sizeof(int));

//Cấp phát vùng nhớ tương đương mảng 1000 phần tử char

//sizeof(char) = 1

char \*c = (char\*)malloc(1000 \* sizeof(char));

return 0;

}

Trong ví dụ trên sau khi cấp phát động thì con trỏ a và c sẽ lưu giữ địa chỉ của ô nhớ đầu tiên trong các ô nhớ được cấp phát. Bạn có thể sử dụng con trỏ này như mảng cũng được.

Toán tử sizeof trả về kích thước theo byte của kiểu dữ liệu, ví dụ bạn muốn cấp phát vùng nhớ cho 100 phần tử kiểu int thì bạn cần 400 byte. Đơn giản hơn là bạn nhân số phần tử cần cấp phát với kích thước của 1 phần tử.

Trong trường hợp không cấp phát đủ vùng nhớ thì hàm malloc sẽ trả về con trỏ NULL.

Ví dụ 2:

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 10;

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

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

for(int i = 0; i < n; i++){

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

for(int i = 0; i < n; i++){

printf("%d ", a[i]);

}

}

return 0;

}

Output :

Cap phat thanh cong !

28 29 30 31 32 33 34 35 36 37

**3. Hàm calloc()**

Hàm calloc() viết tắt của contiguous allocation tương tự như malloc() sử dụng để cấp phát vùng nhớ động nhưng các giá trị của các vùng nhớ được cấp phát sẽ có giá trị mặc định là 0 thay vì giá trị rác như hàm malloc()

Cú pháp : ptr = (cast\_type\*) calloc(n, element\_size)

Trong đó :

* ptr là con trỏ lưu trữ ô nhớ đầu tiên của vùng nhớ được cấp phát
* cast\_type\* là kiểu con trỏ mà bạn muốn ép kiểu sang
* n là số lượng phần tử bạn muốn cấp phát
* element\_size là kích thước theo byte của 1 phần tử

Ví dụ :

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 10;

int \*a = (int\*)calloc(n, sizeof(int));

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

printf("Mang ban dau : ");

for(int i = 0; i < n; i++){

printf("%d ", a[i]);

}

for(int i = 0; i < n; i++){

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

printf("\nMang sau khi thay doi : ");

for(int i = 0; i < n; i++){

printf("%d ", a[i]);

}

}

return 0;

}

Output :

Cap phat thanh cong !

Mang ban dau : 0 0 0 0 0 0 0 0 0 0

Mang sau khi thay doi : 28 29 30 31 32 33 34 35 36 37

**4.Hàm free()**

Hàm malloc() và calloc() xin cấp phát vùng nhớ nhưng lại không tự giải phóng vùng nhớ mà nó xin cấp phát, hàm free() có chắc năng giải phóng vùng nhớ mà malloc() hoặc calloc() đã xin cấp phát.

Việc sử dụng free() sau khi sử dụng malloc() và calloc() là cần thiết để tránh lãng phí bộ nhớ

Cú pháp : free(ptr)

Ví dụ :

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 10;

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

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

for(int i = 0; i < n; i++){

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

for(int i = 0; i < n; i++){

printf("%d ", a[i]);

}

free(a);

printf("\nGiai phong thanh cong !\n");

}

return 0;

}

Output :

Cap phat thanh cong !

28 29 30 31 32 33 34 35 36 37

Giai phong thanh cong !

**5. Hàm realloc()**

Hàm realloc() viết tắt của re-allocation tức là cấp phát lại, trong trường hợp sử dụng malloc() và calloc() nhưng cần bổ sung thêm bạn sử dụng realloc().

realloc() giúp bạn giữ lại các giá trị trên vùng nhớ cũ và bổ sung thêm vùng nhớ mới với các giá trị rác.

Ví dụ khi bạn đang xin cấp phát 5 phần tử nhưng lại muốn cấp phát lại thành 10 phần tử và giữ nguyên giá trị của 5 phần tử trước đó. Nếu bạn sử dụng malloc() hay calloc() thì 5 phần tử cũ sẽ không còn vì bạn được cấp phát 1 vùng nhớ mới, calloc() thì chỉ bổ sung thêm vùng nhớ cho 5 phần tử mới còn 5 phần tử cũ thì vấn giữ nguyên.

Cú pháp : ptr = (cast\_type\*)realloc(ptr, new\_size)

Ví dụ :

#include <stdio.h>

#include <stdlib.h>

int main(){

int n = 5;

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

if(a == NULL){

printf("Cap phat khong thanh cong !\n");

}

else{

printf("Cap phat thanh cong !\n");

for(int i = 0; i < n; i++){

a[i] = 28 + i; // \*(a + i) = 28 cũng được

}

printf("Mang truoc khi cap phat lai : ");

for(int i = 0; i < n; i++){

printf("%d ", a[i]);

}

a = (int\*)realloc(a, 10);

printf("\nMang sau khi cap phat lai : ");

for(int i = 0; i < 10; i++){

printf("%d ", a[i]);

}

}

return 0;

}

Output :

Cap phat thanh cong !

Mang truoc khi cap phat lai : 28 29 30 31 32

Mang sau khi cap phat lai : 28 29 30 31 32 1313428052 -2116255851 46732 12350032 0