# CHƯƠNG IX: CẤP PHÁT BỘ NHỚ ĐỘNG TRONG C

## CẤP PHÁT BỘ NHỚ ĐỘNG TRONG C

### Cấp phát bộ nhớ là gì? Tại sao cần cấp phát động bộ nhớ

#### Cấp phát tĩnh

Chúng ta đã biết, mỗi khi tạo ra một biến nào đó, trình biên dịch sẽ đưa ra 1 địa chỉ để lưu giữ biến đó. Khi chúng ta sử dụng biến có thể truy cập bằng tên biến hoặc con trỏ.  
Việc cấp phát như vậy gọi là **cấp phát tĩnh.**

Khi cấp phát tĩnh, ô nhớ đó sẽ tồn tại từ khi chương trình hoạt động tới khi chương trình kết thúc

Giả sử bạn phải khai báo 1 [mảng](https://khuenguyencreator.com/mang-la-gi-cach-su-dung-mang-lap-trinh-c/) mà chưa rõ phải sử dụng kích thước là bao nhiêu. Vậy thì nếu cấp phát tĩnh bộ nhớ cho mảng đó sẽ xảy ra 2 vấn đề:

* Thiếu kích thước dẫn tới lưu thiếu dữ liệu
* Thừa kích thước dẫn tới lãng phí bộ nhớ

Vậy nên chúng ta phải sử dụng cấp phát động trong trường hợp này.

#### Cấp phát động

**Cấp phát động** bộ nhớ chính là việc cấp phát/giải phóng, thay đổi kích thước bộ nhớ một cách linh hoạt. Giúp chúng ta điều khiển được việc sử dụng bộ nhớ của chương trình.  
Thế nhưng cấp phát động không tốt (không giải phóng ô nhớ) sẽ khiến chương trình gặp lỗi tràn bộ nhớ **Memory leak.**

#### Cấp phát động và cấp phát 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 |

### Cấp phát bộ nhớ động trong C

Để cấp phát vùng nhớ động cho biến con trỏ trong ngôn ngữ C, chúng ta có thể sử dụng hàm **malloc()** hoặc hàm **calloc()**. Sử dụng hàm **free()** để giải phóng bộ nhớ đã cấp phát khi không cần sử dụng, sử dụng **realloc()** để thay đổi (phân bổ lại) kích thước bộ nhớ đã cấp phát trong khi chạy chương trình.

A picture containing text, screenshot, line, diagram

Description automatically generated

#### Sử dụng hàm free

Việc cấp phát bộ nhớ động trong C dù sử dụng **malloc()** hay **calloc()** thì chúng cũng đều không thể tự giải phóng bộ nhớ. Bạn cần sử dụng hàm **free()** để giải phóng vùng nhớ.

Cú pháp:

|  |
| --- |
| free(ptr);  //ptr là con trỏ |

Lệnh này sẽ giải phóng vùng nhớ mà con trỏ **ptr** đã được cấp phát. 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.

Nếu bạn không giải phóng nó thì nó sẽ tồn tại cho tới khi chương trình kết thúc. Điều này sẽ rất nguy hiểm nếu chương trình của bạn liên tục cấp phát các vùng nhớ mới và sẽ gây ra hiện tượng [**tràn bộ nhớ**](https://vi.wikipedia.org/wiki/R%C3%B2_r%E1%BB%89_b%E1%BB%99_nh%E1%BB%9B) mà chúng ta đã nhắc tới ở trên.

#### Sử dụng hàm malloc()

Từ **malloc** là đại diện cho cụm từ **memory allocation** (dịch: cấp phát bộ nhớ). Khi khai báo kiểu **malloc**, các ô nhớ sẽ được giữ nguyên bộ nhớ ban đầu (draf data hay dữ liệu rác).

Hàm malloc được định nghĩa như sau

|  |
| --- |
| void \*malloc(kích cỡ) (kích cỡ này tính bằng byte) |

Kiểu trả về là con trỏ void (không có giá trị), tham số truyền vào là  kích cỡ tính bằng byte

**VD:**

|  |
| --- |
| *int* \*pt = malloc(10\* sizeof(*int*)); |

Cấp phát con trỏ kiểu int, với kích thước là 10\*4 = 40 byte. Vì 1 int có kích thước 4 byte.

Vậy thì khai báo như vậy con trỏ pt sẽ được cấp cho 40byte tương ứng với 10 phần tử trong mảng

Trong trường hợp không thể cấp phát bộ nhớ, nó sẽ trả về một con trỏ **NULL**.

|  |
| --- |
| #include<stdio.h>  int main(){      int \*pt = malloc(10 \* sizeof(int));      pt[0] = 1;      pt[1] = 2;      pt[2] = 3;      pt[3] = 4;      pt[4] = 5;      pt[5] = 6;      pt[6] = 7;      pt[7] = 8;      pt[8] = 9;      pt[9] = 10;      free(pt);  } |

Vì hàm malloc trả về dạng **void** cho phép chúng ta có thể ép kiểu về bất cứ kiểu dữ liệu nào, vậy nên chúng ta nên ép kiểu cho nó trở về đúng kiểu con trỏ mà chúng ta sử dụng, như sau:

**Cú pháp:**

|  |
| --- |
| ptr = (castType\*) malloc(size); |

**VD:**

|  |
| --- |
| int \*ptr;  ptr = (int\*) malloc(100 \* sizeof(int)); |

**VD:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h> // Thư viện này để cấp phát bộ nhớ động  int main(){      int n, i, \*ptr, sum = 0;      printf("Nhap so luong phan tu: ");      scanf("%d", &n);      ptr = (int \*)malloc(n \* sizeof(int));      // Nếu không thể cấp phát,      // hàm malloc sẽ trả về con trỏ NULL      if (ptr == NULL){          printf("Co loi! khong the cap phat bo nho.");          exit(0);      }      for (i = 0; i < n; ++i){          printf("Nhap gia tri %d: ", i+1);          scanf("%d", ptr + i);          sum += \*(ptr + i);      }      printf("Tong = %d", sum);      // Giải phóng vùng nhớ cho con trỏ      free(ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Nhap so luong phan tu: 3**  **Nhap gia tri 1: 1**  **Nhap gia tri 2: 3**  **Nhap gia tri 3: 5**  **Tong = 9** |

#### Sử dụng hàm calloc

Hàm **calloc()** thực hiện cấp phát bộ nhớ và khởi tạo tất cả các ô nhớ có giá trị bằng 0. Vì thế nên hàm calloc sẽ cần thời gian thực thi lâu hợn **malloc()**

Việc cấp phát và sử dụng cũng khá giống malloc() nhưng thay vì kích cỡ cố định là byte, calloc sẽ cho chúng ta thêm 1 tham số truyền vào là size của 1 dữ liệu.

Hàm calloc được định nghĩa như sau:

|  |
| --- |
| void \*calloc(số *phần* *tử*, kích cỡ của 1 phần *tử*) |

Giá trị trả về là [con trỏ](https://khuenguyencreator.com/con-tro-la-gi-cach-su-dung-con-tro-trong-c/) void, tham số truyền vào là số phần tử và kích thước của phần tử.

**Cú pháp:**

|  |
| --- |
| ptr = (castType\*)calloc(num, size); |

Với num là số lượng phần tử, size là kích thước mỗi phần tử.

**VD:**

|  |
| --- |
| char \*pt = calloc(100, sizeof(char));  hoặc  int \*ptr;  ptr = (int\*) calloc(100, sizeof(int)); |

**VD:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h> // Thư viện này để cấp phát bộ nhớ động  int main(){      int n, i, \*ptr, sum = 0;      printf("Nhap so luong phan tu: ");      scanf("%d", &n);      ptr = (int \*)calloc(n, sizeof(int));      // Nếu không thể cấp phát,      // hàm malloc sẽ trả về con trỏ NULL      if (ptr == NULL){          printf("Co loi! khong the cap phat bo nho.");          exit(0);      }      for (i = 0; i < n; ++i){          printf("Nhap gia tri %d: ", i+1);          scanf("%d", ptr + i);          sum += \*(ptr + i);      }      printf("Tong = %d", sum);      // Giải phóng vùng nhớ cho con trỏ      free(ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Nhap so luong phan tu: 3**  **Nhap gia tri 1: 1**  **Nhap gia tri 2: 3**  **Nhap gia tri 3: 5**  **Tong = 9** |

#### Sử dụng hàm realloc

Hàm realloc được sử dụng khi bạn đã cấp phát động bộ nhớ nhưng thấy thiếu, cần cấp phát thêm bộ nhớ để sử dụng.

**Cú pháp:**

|  |
| --- |
| void\* realloc(void\* *ptr*, *size\_t* *size*); |

**Khai báo:**

|  |
| --- |
| ptr = realloc(ptr, n); |

**VD:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h> // Thư viện này để cấp phát bộ nhớ động  int main(){      int n, i, \*ptr, sum = 0;      printf("Nhap so luong phan tu: ");      scanf("%d", &n);      ptr = (int \*)calloc(n, sizeof(int));      // Nếu không thể cấp phát,      // hàm malloc sẽ trả về con trỏ NULL      if (ptr == NULL){          printf("Co loi! khong the cap phat bo nho.");          exit(0);      }      for (i = 0; i < n; ++i){          printf("Nhap gia tri %d: ", i+1);          scanf("%d", ptr + i);          sum += \*(ptr + i);      }      printf("Tong = %d \n", sum);      printf("Them phan tu: ");      int m;      scanf("%d", &m);      ptr = (int\*)realloc(ptr, m\* sizeof(int));      for (i = n; i < n+m; ++i){          printf("Nhap gia tri %d: ", i+1);          scanf("%d", ptr + i);          sum += \*(ptr + i);      }      printf("Tong moi = %d", sum);      // Giải phóng vùng nhớ cho con trỏ      free(ptr);      return 0;  } |

**Kết quả**

|  |
| --- |
| **Nhap so luong phan tu: 3**  **Nhap gia tri 1: 1**  **Nhap gia tri 2: 3**  **Nhap gia tri 3: 5**  **Tong = 9**  **Them phan tu: 2**  **Nhap gia tri 4: 7**  **Nhap gia tri 5: 9**  **Tong moi = 25** |

### Tổng kết

Việc sử dụng phương thức **calloc** sẽ an toàn hơn **malloc** trong [lập trình](https://khuenguyencreator.com/) vì vùng nhớ cấp phát động sẽ được gán giá trị bằng 0 thay vì giá trị rác như **calloc.** Tuy nhiên việc thêm 1 bước gán giá trị các ô nhớ bằng 0 này cũng sẽ khiến nó bị chậm hơn so với malloc do phải thực hiện thêm thao tác.

Sử dụng **realloc** và **free** một cách linh hoạt sẽ giúp các bạn điều khiển sự tăng giảm của bộ nhớ 1 cách dễ dàng