# Bài 66. Cấp phát bộ nhớ động trong C

Bởi **Nguyễn Văn Hiếu** -

This entry is part 64 of 69 in the series Học C Không Khó

Nói đến con trỏ không thể không nhắc tới **cấp phát bộ nhớ động** và **giải phóng bộ nhớ** cho biến con trỏ trong ngôn ngữ C. Trong bài viết này, Lập trình không khó sẽ làm rõ vấn đề cấp phát bộ nhớ động sử dụng `malloc()`, `calloc()`, `free()` và `realloc()` trong C. Đây là các hàm giúp các bạn cấp phát và giải phóng bộ nhớ khi làm việc với con trỏ trong ngôn ngữ C.



NỘI DUNG BÀI VIẾT

1. Tại sao cần cấp phát bộ nhớ động?
2. Cấp phát bộ nhớ động trong C
   2.1. Sử dụng hàm malloc()
   2.2. Sử dụng hàm calloc()
   2.3. Sử dụng hàm free()
   2.4. Ví dụ sử dụng malloc() và free()
   2.5. Ví dụ sử dụng calloc() và free()
   2.6. Sử dụng hàm realloc()
   2.7. Ví dụ sử dụng hàm realloc()
3. Cấp phát động trong C++
4. Cấp phát động mảng 1 chiều
5. Tài liệu tham khảo

## Tại sao cần cấp phát bộ nhớ động?

Như bạn đã biết, mảng là một tập hợp của các phần tử nằm liên tiếp nhau trên bộ nhớ và có cùng kiểu dữ liệu. Khi khai báo mảng, bạn phải chỉ định rõ kích thước tối đa (số lượng phần tử tối đa). Và sau khi khai báo, bạn không thể thay đổi kích thước của mảng – **Cấp phát tĩnh**.

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**.

21

Bảng dưới đây so sánh giúp bạn sự khác biệt giữa cấp phát bộ nhớ động và tĩnh.

| Cấp phát bộ nhớ tĩnh | Cấp phát bộ nhớ động |
|---|---|
| Bộ nhớ được cấp phát trước khi chạy | Bộ nhớ được cấp phát trong quá trình chạy |

| | |
|---|---|
| chương trình (trong quá trình biên dịch) | 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 |

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.

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).

# Cấp phát bộ nhớ động trong C

Để cấp phát vùng nhớ động cho biến con trỏ trong ngôn ngữ C, bạn 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.

## Sử dụng hàm malloc()

Từ **malloc** là đại diện cho cụm từ memory allocation (dịch: cấp phát bộ nhớ).

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 của hàm `malloc()`:**

```
0
1 ptr = (castType*) malloc(size);
2
```

Ví dụ:

```
0
1 ptr = (int*) malloc(100 * sizeof(int));
```

```
2
```

Ví dụ trên thực hiện cấp phát cho việc lưu trữ 100 số nguyên. Giả sử `sizeof int` là 4, khi đó lệnh dưới đây thực hiện cấp phát 400 bytes. Khi đó, con trỏ `ptr` sẽ có giá trị là địa chỉ của **byte dữ liệu đầu tiên** trong khối bộ nhớ vừa cấp phát.

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

## Sử dụng hàm calloc()

Từ `calloc` đại diện cho cụm từ contiguous allocation (dịch: cấp phát liên tục).

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.

Hàm `calloc()` nhận vào 2 tham số là **số ô nhớ** muốn khởi tạo và **kích thước của 1 ô nhớ**.

**Cú pháp của hàm `calloc()`:**

```
0
1 ptr = (castType*)calloc(n, size);
2
```

Ví dụ:

```
0
1 ptr = (int*) calloc(100, sizeof(int));
2
```

Trong ví dụ trên, hàm `calloc()` thực hiện cấp phát 100 ô nhớ liên tiếp và mỗi ô nhớ có kích thước là số byte của kiểu `int`. Hàm này cũng trả về con trỏ chứa giá trị là địa chỉ của byte đầu tiên trong khối bộ nhớ vừa cấp phát.

21

## 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:

```
0
1 free(ptr); // ptr là con trỏ
2
```

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 mà mình đã nhắc tới ở trên. Thử code này xem sao (bảo đảm là máy bạn sẽ bị treo và chỉ còn cách ấn nút nguồn thôi, bạn có thể để cấp phát size nhỏ hơn và theo dõi thay đổi qua Task Manager):

```
0
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5      for(;;){
6          printf("\nCap phat %d bytes!", 1000000 * sizeof (int));
7          int *ptr = (int*) malloc (1000000 * sizeof (int));
8      }
9  }
10
```

## Ví dụ sử dụng malloc() và free()

Trong ví dụ dưới đây, chúng ta sẽ sử dụng hàm `malloc()` để cấp phát động `n * sizeof int` byte và sử dụng xong sẽ dùng `free()` để giải phóng.

```
0
1  #include <stdio.h>
2  // Thư viện này cần để cấp phát bộ nhớ động
3  #include <stdlib.h>
4
5  int main()
6  {
7      int n, i, *ptr, sum = 0;
8      printf("Nhap so luong phan tu: ");
9      scanf("%d", &n);
10     ptr = (int *)malloc(n * sizeof(int));
11
12     // Nếu không thể cấp phát,
13     // hàm malloc sẽ trả về con trỏ NULL
14     if (ptr == NULL)
15     {
16         printf("Co loi! khong the cap phat bo nho.");
17         exit(0);
18     }
19     printf("Nhap cac gia tri: ");
20     for (i = 0; i < n; ++i)
21     {
22         scanf("%d", ptr + i);
23         sum += *(ptr + i);
24     }
25     printf("Tong = %d", sum);
26
27     // Giải phóng vùng nhớ cho con trỏ
28     free(ptr);
29     return 0;
30 }
31
```

## Ví dụ sử dụng calloc() và free()

Trong ví dụ này, chúng ta sẽ dùng `calloc()` để cấp phát n ô nhớ liên tiếp và mỗi ô nhớ có kích thước là `sizeof int`. Lưu ý là hàm `calloc()` sẽ chậm hơn `malloc()` một chút do nó phải thêm bước khởi tạo các ô nhớ có giá trị bằng 0. Do đó, tùy thuộc bạn cần hiệu năng hay cần khởi tạo giá trị ban đầu mà sử dụng hàm cấp phát thích hợp.

```
0
1  #include <stdio.h>
2  // Thư viện này cần để cấp phát bộ nhớ động
3  #include <stdlib.h>
4
5  int main()
6  {
7      int n, i, *ptr, sum = 0;
8      printf("Nhap so luong phan tu: ");
9      scanf("%d", &n);
10     ptr = (int *)calloc(n, sizeof(int));
11
```

```
12      // Nếu không thể cấp phát,
13      // hàm calloc sẽ trả về con trỏ NULL
14      if (ptr == NULL)
15      {
16          printf("Co loi! khong the cap phat bo nho.");
17          exit(0);
18      }
19      printf("Nhap cac gia tri: ");
20      for (i = 0; i < n; ++i)
21      {
22          scanf("%d", ptr + i);
23          sum += *(ptr + i);
24      }
25      printf("Tong = %d", sum);
26
27      // Giải phóng vùng nhớ cho con trỏ
28      free(ptr);
29      return 0;
30  }
31
```

## Sử dụng hàm realloc()

Nếu việc cấp phát bộ nhớ động không đủ hoặc cần nhiều hơn mức đã cấp phát, bạn có thể thay đổi kích thước của bộ nhớ đã được cấp phát trước đó bằng cách sử dụng hàm `realloc()`.

**Cú pháp của `realloc()`:**

```
0
1  ptr = realloc(ptr, n);
2
```

Hàm này thực hiện cấp phát vùng nhớ mới cho con trỏ `ptr`. Vùng nhớ mới đó sẽ có kích thước mới là `n` bytes.

Hàm này cũng trả về con trỏ chứa giá trị là địa chỉ của byte đầu tiên trong vùng nhớ mới. Hàm này sẽ cố gắng mở rộng số ô nhớ ra phía sau nếu có thể để giữ nguyên giá trị của con trỏ ban đầu. Trong trường hợp phải đổi sang một vùng nhớ khác, hàm `realloc()` cũng sẽ mang theo giá trị đã có ở vùng nhớ cũ sang vùng nhớ mới và giải phóng luôn vùng nhớ cũ (*đọc thêm tài liệu số 2*). Trong trường hợp không thể, nó sẽ trả về con trỏ `NULL` giống như `malloc()` và `calloc()`.

## Ví dụ sử dụng hàm realloc()

Trong ví dụ dưới đây, ta sẽ sử dụng hàm `realloc()` để tái phân bổ lại bộ nhớ. Như trong ví dụ dưới đây thì việc cấp phát không phải di chuyển sang vùng nhớ khác mà chỉ mở rộng ra phía sau.

21

```
0
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
```

```
    int *ptr, i , n1, n2;
    printf("Nhap so luong phan tu: ");
    scanf("%d", &n1);
    ptr = (int*) malloc(n1 * sizeof(int));
    printf("Dia chi cua vung nho vua cap phat: %u", ptr);

    printf("\nNhap lai so luong phan tu: ");
    scanf("%d", &n2);
    // phân bổ lại vùng nhớ
    ptr = (int*) realloc(ptr, n2 * sizeof(int));
    printf("Dia chi cua vung nho duoc cap phat lai: %u", ptr);
    // giải phóng
    free(ptr);
    return 0;
}

```

Kết quả chạy:

```
Nhap so luong phan tu: 2
Dia chi cua vung nho vua cap phat: 1993360
Nhap lai so luong phan tu: 100
Dia chi cua vung nho duoc cap phat lai: 1993360

```

# Cấp phát động trong C++

Mục này không phải kiến thức của C, nhưng mình cũng sẽ ghi lại để các bạn học luôn hoặc tìm hiểu, phân biệt và sử dụng trong C++ khi cần.

Để cấp phát động trong C++ bạn sử dụng toán tử `new` và giải phóng bộ nhớ sử dụng `delete`. Ví dụ:

```

#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int num;
    cout << "Nhap so luong sinh vien: ";
    cin >> num;
    float* ptr;

    // Cấp phát `num` ô nhớ kiểu float
    ptr = new float[num];
    cout << "Nhap diem GPA." << endl;
    for (int i = 0; i < num; ++i)
    {
        cout << "Sinh vien " << i + 1 << ": ";
        cin >> *(ptr + i);
    }
    cout << "\nHien thi diem GPA cua sinh vien." << endl;
    for (int i = 0; i < num; ++i) {
        cout << "Sinh vien " << i + 1 << " :" << *(ptr + i) << endl;
    }

```

```
23      // giải phóng bộ nhớ của `ptr`
24      delete [] ptr;
25      return 0;
26  }
27
```

Sau đây mình sẽ hướng dẫn bạn cách sử dụng cấp phát động vào mảng 1 chiều.

## Cấp phát động mảng 1 chiều

Dưới đây là ví dụ sử dụng cấp phát động cho mảng 1 chiều để các bạn tham khảo và nắm được ứng dụng của cấp phát bộ nhớ động.

```
0
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   void NhapMang(int *arr, int n)
5   {
6       for (int i = 0; i < n; i++)
7       {
8           printf("arr[%d] = ", i);
9           // Do giá trị con trỏ là địa chỉ rồi. Nên bạn sẽ không thấy dấu & quen thuộc r
10          scanf("%d", (arr + i));
11      }
12  }
13
14  void XuatMang(int *arr, int n)
15  {
16      for (int i = 0; i < n; i++)
17      {
18          printf("arr[%d] = %d\n", i, *(arr + i));
19      }
20  }
21
22  void ThemPhanTu(int *a, int &n, int val, int pos)
23  {
24      // Phân bổ lại bộ nhớ đã cấp phát cho con trỏ.
25      // Ta cần thêm 1 ô nhớ cho nó => dùng realloc()
26      a = (int *)realloc(a, (n + 1) * sizeof(int));
27      // Neu pos <= 0 => Them vao dau
28      if (pos < 0)
29      {
30          pos = 0;
31      }
32      // Neu pos >= n => Them vao cuoi
33      else if (pos > n)
34      {
35          pos = n;
36      }
37      // Dich chuyen mang de tao o trong truoc khi them.
38      for (int i = n; i > pos; i--)
39      {
40          *(a + i) = *(a + i - 1);
41      }
42      // Chen val tai pos
43      *(a + pos) = val;
44      // Tang so luong phan tu sau khi chen.
```

```
45      ++n;
46  }
47
48  void XoaPhanTu(int *a, int &n, int pos)
49  {
50      // Mang rong, khong the xoa.
51      if (n <= 0)
52      {
53          return;
54      }
55      // Neu pos <= 0 => Xoa dau
56      if (pos < 0)
57      {
58          pos = 0;
59      }
60      // Neu pos >= n => Xoa cuoi
61      else if (pos >= n)
62      {
63          pos = n - 1;
64      }
65      // Dich chuyen mang
66      for (int i = pos; i < n - 1; i++)
67      {
68          a[i] = a[i + 1];
69      }
70      // Cấp phát lại vùng nhớ, giờ ta chỉ cần n - 1 ô nhớ
71      a = (int *)realloc(a, (n - 1) * sizeof(int));
72
73      // Giam so luong phan tu sau khi xoa.
74      --n;
75  }
76
77  int main()
78  {
79      int *arr;
80      int n;
81      do
82      {
83          printf("Nhap so luong n = ");
84          scanf("%d", &n);
85      } while (n < 1);
86
87      // cấp phát đủ sài
88      arr = (int *)malloc(n * sizeof(int));
89      // arr = (int*) calloc(n, sizeof(int));
90
91      if (arr == NULL)
92      {
93          printf("Khong the cap phat!");
94          exit(0);
95      }
96
97      NhapMang(arr, n);
98      printf("\nMang vua nhap la:\n");
99      XuatMang(arr, n);
100
101     printf("\n=======THEM PHAN TU======\n");
102     int val, pos;
103     printf("\nNhap so can them: ");
104     scanf("%d", &val);
105     printf("\nNhap vi tri muon chen: ");
```

```
106     scanf("%d", &pos);
107     ThemPhanTu(arr, n, val, pos);
108     printf("\nMang sau khi them:\n");
109     XuatMang(arr, n);
110 
111     printf("\n=======XOA PHAN TU======\n");
112     printf("\nNhap vi tri muon xoa: ");
113     scanf("%d", &pos);
114     XoaPhanTu(arr, n, pos);
115     printf("\nMang sau khi xoa:\n");
116     XuatMang(arr, n);
117     // giải phóng
118     free(arr);
119 }
120 
```

Kết quả chạy chương trình:

```
0 
1  Nhap so luong n = 3
2  arr[0] = 1
3  arr[1] = 2
4  arr[2] = 3
5 
6  Mang vua nhap la:
7  arr[0] = 1
8  arr[1] = 2
9  arr[2] = 3
10 
11 =======THEM PHAN TU======
12 
13 Nhap so can them: 4
14 
15 Nhap vi tri muon chen: 4
16 
17 Mang sau khi them:
18 arr[0] = 1
19 arr[1] = 2
20 arr[2] = 3
21 arr[3] = 4
22 
23 =======XOA PHAN TU======
24 
25 Nhap vi tri muon xoa: 4
26 
27 Mang sau khi xoa:
28 arr[0] = 1
29 arr[1] = 2
30 arr[2] = 3
31 
```

# Tài liệu tham khảo

1. C Dynamic Memory Allocation

2. Changing Block Size (The GNU C Library)

**Nguyễn Văn Hiếu**

Sáng lập cộng đồng Lập Trình Không Khó với mong muốn giúp đỡ các bạn trẻ trên con đường trở thành những lập trình viên tương lai. Tất cả những gì tôi viết ra đây chỉ đơn giản là sở thích ghi lại các kiến thức mà tôi tích lũy được.

## BÀI VIẾT HAY

**Bài 1. Giới thiệu khóa học "Học C Bá Đạo"**
21/07/2019

**1000 bài tập lập trình C/C++ có lời giải của thầy Khang**
25/12/2019

**Kiểm tra số nguyên tố sử dụng C/C++ và Java**
15/07/2018

## CHUYÊN MỤC HAY

| | |
|---|---|
| Học C/C++ | 199 |
| Học Python | 48 |
| Học Java | 45 |
| Học Javascript | 37 |
| Khóa học | 34 |
| Chia sẻ | 26 |
| Học Web | 26 |
| Học C# | 25 |

## - BẠN BÈ & ĐỐI TÁC -

Luyện Code - Tự Học Đồ Họa - Cách Học Lập Trình - VNTALKING



21