Bài 8: Cấp phát động

Tất cả những biến mà chúng ta từng sử dụng cho đến bây giờ đều được tự động tạo ra bởi trình biên dịch ngôn ngữ C. Phương pháp hoàn toàn đơn giản dễ hiểu. Nhưng vẫn có một cách thủ công hơn để tạo ra các biến, gọi là "cấp phát động".

Một trong những lợi ích chính của việc cấp phát động là cho phép chương trình dự trữ sẵn một không gian bắt buộc để lưu trữ mảng trong bộ nhớ. Cho đến bây giờ, những mảng mà chúng ta tạo ra đều đã được cố định kích thước trong code. Sau bài học này chúng ta sẽ biết cách làm việc với mảng một cách linh hoạt hơn.

Bạn bắt buộc phải nắm rõ cách làm việc với con trỏ để có thể hiểu được bài học này. Nếu bạn vẫn còn lăn tăn thì tôi khuyên bạn nên dành chút thời gian xem lại những kiến thức của các bài học về con trỏ trước khi bắt đầu.

Khi bạn khai báo một biến, có nghĩa là chúng ta đang yêu cầu máy tính cấp phát bộ nhớ:

C code:

```
int number = 0;
```

Khi chương trình nhận được dòng code như trên thì sẽ xảy ra những điều sau:

1. Chương trình sẽ yêu cầu hệ điều hành (Windows, Linux, Mas Os …) cho phép sử dụng bộ nhớ.
2. Hệ điều hành sẽ tiếp nhận yêu cầu và cho chương trình biết nơi nào có thể lưu trữ các biến (nó sẽ cho chương trình địa chỉ bộ nhớ mà nó dự trữ sẵn từ trước).
3. Khi hàm kết thúc công việc của nó thì biến cũng đồng thời bị xóa khỏi bộ nhớ. Chương trình của bạn sẽ nói với hệ điều hành rằng "Tao không cần mượn không gian bộ nhớ của mày ở địa chỉ này nữa, cảm ơn nhiều!". Chuyện không đơn giản là nói lời cảm ơn với hệ điều hành mà bạn phải thấy rõ không phải ai khác, chính hệ điều hành điều khiển bộ nhớ.

Cho đến thời điểm này mọi thứ đều diễn ra một cách tự động. Mỗi khi khai báo biến, hệ điều hành tự động được chương trình liên hệ để yêu cầu cấp phát bộ nhớ.

Vậy chúng ta có thể làm điều này bằng tay một cách thủ công không? Không phải vì chúng ta thích tự làm khó mình với những thứ phức tạp (kể cả khi nó có hấp dẫn đi nữa), mà vì có đôi khi ta bắt buộc phải tự làm như vậy.

Trong chương này chúng ta sẽ:

1. Tìm hiểu về các chức năng của bộ nhớ (vâng, một lần nữa) để biết kích thước của biến sẽ thay đổi thế nào tùy thuộc vào kiểu của nó (int/float/char/double…)
2. Sau đó chúng ta đi thẳng vào vấn đề chính, làm thế nào để yêu cầu hệ điều hành cấp phát bộ nhớ một cách thủ công. Chúng ta sẽ làm cái việc ở đầu bài học đã nêu ra: "cấp phát động".
3. Cuối cùng, chúng ta sẽ xem xem việc cấp phát động này sẽ mang lại lợi ích gì thông qua việc học cách tạo ra mảng mà không biết trước kích thước của nó trong chương trình là bao nhiêu.

Kích thước của các biến:

Tùy thuộc vào loại biến mà bạn đang muốn tạo ra (int/char/double/float…), bạn sẽ cần nhiều hoặc ít bộ nhớ.

Thực tế là để lưu trữ một số chạy từ -128 đến 127 (một biến kiểu *float*), máy tính chỉ cần 1 byte trong bộ nhớ, nó thật sự rất nhỏ.

Tuy nhiên với một biến *int*, nó thường chiếm 4 byte trong bộ nhớ, còn với một biến kiểu *double* sẽ là 8 byte.

Vấn đề là … không phải lúc nào mọi thứ cũng diễn ra đúng như những gì chúng ta nói. Nó còn phụ thuộc vào máy tính nữa: chẳng hạn như biến kiểu *int* của bạn có thể chiếm 8 byte, ai mà biết được ?

Và mục đích của chúng ta là kiểm tra xem mỗi kiểu biến sẽ chiếm bao nhiêu bộ nhớ trên máy tính của bạn.

Có một cách rất dễ để kiểm tra đó là dùng *sizeof ( )*.

Không giống như những gì chúng ta từng học trước đây, nó không phải là một hàm (function), mà là một chức năng cơ bản của ngôn ngữ C, bạn chỉ cần đặt đối tượng bạn muốn kiểm tra kích thước vào trong cặp dấu ( ) để thấy được điều bạn muốn.

Để biết kích thước của một biến kiểu *int* chúng ta sẽ làm như sau:

C code:

```
sizeof(int)
```

Tại thời điểm bắt đầu biên dịch, nó sẽ được thay thế bởi những con số: đó là số bộ nhớ mà một biến kiểu *int* sẽ chiếm trong bộ nhớ máy tính của bạn, sizeof (int) là 4, nghĩa là nó chiếm 4 byte

trong bộ nhớ. Theo lý thuyết thì nó cũng sẽ có giá trị tương tự nhưng đây không phải là một quy luật không đổi. Chúng ta cùng kiểm tra bằng cách dùng *printf* để hiển thị giá trị, VD:

C code:

```
printf ("char : %d byte\n", sizeof(char));
printf ("int : %d byte\n", sizeof(int));
printf ("long : %d byte\n", sizeof(long));
printf ("double : %d byte\n", sizeof(double));
```

Màn hình console sẽ hiển thị:

```
char: 1 byte
int: 4 byte
dài: 4 byte
Double: 8 byte
```

Tôi đã không kiểm tra tất cả các kiểu biến mà chúng ta đã biết. Tôi nhường phần đó cho bạn để kiểm tra kích thước các kiểu biến khác.

Bạn sẽ nhận thấy rằng kiểu *long* và *int* chiếm cùng một dung lượng bộ nhớ. Việc tạo một biến kiểu *long* cũng sẽ chiếm 4 byte như khi tạo một biến *int.*

 Thật ra thì kiểu *long* chính là *long int,* tương tự như kiểu *int.* Đơn giản là nó cũng chỉ tạo ra thêm một cái tên mới chứ không có gì nhiều, chỉ vậy thôi! Trước đây khi bộ nhớ máy tính vẫn chưa tốt như bây giờ thì những cái tên khác nhau của kiểu biến thực sự rất hữu dụng cho máy tính của chúng ta. Các lập trình viên ngày trước đã luôn phải suy nghĩ chọn kiểu biến phù hợp nhất để tiết kiệm tối đa bộ nhớ.

Ngày nay thì dung lượng bộ nhớ máy tính thực sự đã rất lớn và vấn đề này đã không còn quá quan trọng. Nhưng vẫn có những người thích tạo ra những chương trình chiếm ít bộ nhớ nhất có thể. Tôi nghĩ rằng đó là các chương trình cho điện thoại di động, robot …

 Vậy chúng ta có thể biết kích thước của những kiểu biến tùy chỉnh do mình tạo ra (chẳng hạn như đối với “cấu trúc” – structure)

Câu trả lời là có! *sizeof ( )* cũng hoạt động với cấu trúc (structure)!

C code:

```
typedef struct Toado Toado;
struct Toado
{
   int x;
   int y;
};

int main (int argc, char *argv[ ])
{
   printf ("Toado : %d byte\n", sizeof(Toado));

   return 0;
}
```

Console:

```
Toado : 8 byte
```

Với những cấu trúc chứa nhiều biến thành phần thì sẽ chiếm nhiều bộ nhớ hơn. Quá là hợp lý luôn đúng không ?

Một cách mới để kiểm tra.

Cho tới thời điểm này thì mô hình bộ nhớ của tôi vẫn chưa được rõ ràng lắm. Và bây giờ chúng ta sẽ làm rõ mọi thứ, cuối cùng thì ta cũng có thể biết chính xác kích thước của từng loại biến.

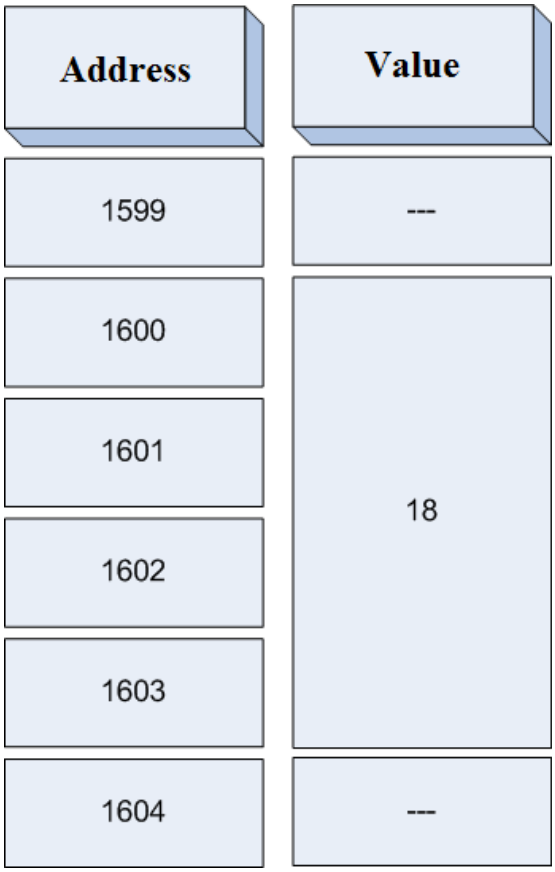
Nếu bạn khai báo một biến kiểu *int*:

C code:

```
int number = 18;
```

… và sizeof (int) đã chỉ ra rằng chúng ta sẽ mượn 4 byte của máy tính, sau đó biến sẽ chiếm 4 byte này của bộ nhớ.

Giả sử biến được cấp phát cho địa chỉ 1600 trong bộ nhớ. Chúng ta cùng xem hình sau để thấy rõ hơn:



Bạn thấy rõ ràng ở đây là biến kiểu *int* chiếm 4 byte trong bộ nhớ.

Nó bắt đầu từ địa chỉ 1600 (địa chỉ mà biến được chỉ định lúc đầu) và kết thúc ở địa chỉ 1603. Biến tiếp theo sẽ không thể lưu vào địa chỉ nào trong 4 địa chỉ trên, nó sẽ được bắt đầu từ 1604.

Nếu chúng ta làm điều tương tự với một biến kiểu *char*, biến sẽ chỉ chiếm một byte trong bộ nhớ như hình sau:

Hãy thử tưởng tượng chúng ta có một mảng các biến kiểu *int.*

Mỗi một ô trong mảng sẽ chiếm 4 byte. Vậy nếu mảng của chúng ta có 100 ô thì sao:

C code:

```
int mang[100];
```

Vậy chính xác là chúng ta sẽ chiếm 4 x 100 = 400 byte trong bộ nhớ.

Vậy thậm chí mảng không chứa gì trong đó thì nó vẫn chiếm trước 400 byte bộ nhớ sao?

Dĩ nhiên rồi! Các không gian bộ nhớ đã được dự trữ sẵn, không có một chương trình nào khác có thể chạm vào nó (ngoại trừ sự tác động của bạn). Một khi biến đã được khởi tạo, nó sẽ ngay lập tức chiếm một vùng trong bộ nhớ.

Lưu ý nếu bạn tạo một mảng kiểu *Toado* thì:

C code:

```
Toado mang[100];
```

… Lần này chúng ta sẽ sử dụng 8 x 100 = 800 byte trong bộ nhớ.

Điều quan trọng là bạn phải cố gắng hiểu những tính toán nhỏ của phần sau.

Cấp phát động.

Nào bây giờ chúng ta sẽ tìm hiểu sâu về nó. Tôi sẽ nhắc lại mục tiêu của chúng ta: học cách yêu cầu cấp phát bộ nhớ bằng cách thủ công.

Chúng ta cần khai báo thư viện <*stdlib.h*>. Nếu bạn tin những gì tôi khuyên thì tốt nhất là nên khai báo thư viện này trong tất cả các chương trình của bạn. Thư viện này chứa 2 hàm mà chúng ta cần đến:

- malloc ("Memory allocation", nghĩa là "cấp phát bộ nhớ động"): hàm sẽ yêu cầu hệ điều hành để được sử dụng bộ nhớ máy tính.
- free ("free", nghĩa là "giải phóng"): hàm sẽ giải phóng vùng nhớ đã được hệ điều hành chỉ định cho yêu cầu cấp phát bộ nhớ của chúng ta trước đó, từ lúc này, các chương trình khác có thể tự do sử dụng vùng nhớ đó.

Mỗi khi muốn thực hiện việc cấp phát bộ nhớ động theo cách thủ công, bạn nên thực hiện lần lượt 3 bước sau:

1. Gọi hàm *malloc* để yêu cầu cấp phát bộ nhớ.
2. Kiểm tra giá trị trả về của hàm *malloc* để biết hệ điều hành có cấp phát bộ nhớ thành công hay không.
3. Sau khi sử dụng xong bạn phải tiến hành giải phóng bộ nhớ bằng hàm *free*. Nếu chúng ta không làm thao tác này, chương trình của bạn sẽ dễ gặp phải vấn đề tràn bộ nhớ, khi được hoàn tất chương trình của bạn sẽ chiếm một dung lượng bộ nhớ khổng lồ mà trong đó có những vùng nhớ bị sử dụng không cần thiết.

Ba bước này có gợi lại cho bạn những gì đã học về thao tác với tập tin không? Có đấy! Về nguyên tắc, nó gần như giống với những gì bạn đã học khi thao tác với các tập tin: đầu tiên nó sẽ được cấp phát bột nhớ, sau đó kiểm tra xem việc cấp phát có thành công không, và khi đã được cấp phát nó sẽ sử dụng vùng bộ nhớ đó, sử dụng xong thì giải phóng vùng nhớ cho chương trình khác có thể sử dụng tiếp.

Hàm malloc: Yêu cầu cấp phát bộ nhớ.

Đây là prototype của hàm *malloc,* nó trông khá ngộ:

C code:

```
void* malloc(size_t soByteCansudung);
```

Hàm chỉ cần 1 tham số: đó là số byte cần sử dụng. Vì vậy, chỉ cần sử dụng *sizeof (int)* trong tham số này để dự trữ đủ không gian lưu trữ một biến kiểu *int.*

Nhưng hình như có gì đó hơi lạ về giá trị trả về của hàm: nó trả về 1 void* …! Nếu các bạn còn nhớ về những gì đã học ở bài học về hàm (function), tôi đã từng nói với bạn void có nghĩa là trả về "không gì cả", và chúng ta sử dụng kiểu void này để chỉ định hàm này sẽ không trả về giá trị nào hết.

Vậy ở đây chúng ta có 1 hàm sẽ trả về "con trỏ rỗng" sao? Điều này có ổn không? Xem ra các lập trình viên khá hài hước nhỉ.

Bạn cứ bình tĩnh, có lý do hết đấy. Thực tế là, hàm này sẽ trả về một con trỏ đến địa chỉ mà hệ điều hành đã cấp phát cho biến của bạn. Nếu hệ điều hành đã cấp cho biến của bạn ô nhớ ở địa chỉ 1600 thì nó sẽ trả về giá trị con trỏ tương ứng với địa chỉ 1600.

Vấn đề là hàm *malloc* không biết được kiểu biến mà bạn muốn tạo. Thực tế là bạn đưa cho hàm một tham số: số lượng byte cần sử dụng trong bộ nhớ. Nếu nó là 4 byte, nó có thể là biến kiểu *int* hoặc *long int* chẳng hạn.

Vì *malloc* không biết nên trả về giá trị kiểu nào, nó trả về kiểu void*. Nó sẽ là một con trỏ đến bất kỳ kiểu nào cũng được. Chúng ta cũng có thể xem nó là một con trỏ linh động.

Nào chúng ta thực hành thôi.

Nếu tôi muốn vui vẻ một chút (e hèm!), tôi sẽ tự mình tạo một biến kiểu *int* trong bộ nhớ, tôi đang đề cập đến việc sử dụng *malloc* với *sizeof (int)* byte trong bộ nhớ.

Tôi nhận được kết quả của *malloc* là con trỏ đến *int.*

C code:

```
int* capphatBonho = NULL; // Tao mot con tro kieu int

capphatBonho = malloc(sizeof(int)); // Nhung dia chỉ danh cho con tro da duoc danh rieng
```

Kết thúc đoạn code trên, capphatBonho là một con đang chứa địa chỉ mà bạn muốn hệ điều hành dự trữ sẵn vùng nhớ cho bạn, vd như địa chỉ 1600.

Kiểm tra con trỏ.

Hàm *malloc* đã gọi một biến capphatBonho để chứa địa chỉ con trỏ mà hệ điều hành dành riêng cho bạn. Có 2 trường hợp có thể xảy ra:

- Nếu việc cấp phát thành công, con trỏ của chúng ta sẽ mang giá trị ứng với địa chỉ đã được máy tính chỉ định.
- Nếu việc cấp phát không thành công, con trỏ sẽ mang giá trị NULL.

Việc cấp phát bộ nhớ thất bại rất hiếm thấy nhưng nó có xảy ra, chẳng hạn bạn yêu cầu 34GB bộ nhớ RAM, khó mà yêu cầu máy tính đáp ứng cho bạn được.

Chúng ta sẽ gặp một hàm cơ bản mà bạn chưa từng thấy ở những bài trước: *exit ( )*. Nó sẽ ngừng chương trình ngay lập tức. Nó cần một tham số: đó chính là giá trị mà chương trình phải trả về (chính xác hơn thì đây là giá trị trả về của hàm *main ( )*).

C code:

```
int main (int argc, char *argv[ ])
{
   int* capphatBonho = NULL;

   capphatBonho = malloc(sizeof(int));
   if (capphatBonho == NULL) // Neu viec cap phat bo nho ko thanh cong
   {
      exit(0); // Chung ta se ngung chuong trinh ngay lap tuc
   }

   // Chuong trinh se tiep tuc hoat dong neu moi viec dien ra thuan loi

   return 0;
}
```

Nếu giá trị con trỏ khác NULL, chương trình có thể tiếp tục hoạt động, ngược lại nó sẽ hiển thị thông báo lỗi hoặc ngừng chương trình ngay lập tức. Nguyên nhân có thể là do dung lượng bộ nhớ bạn yêu cầu vượt quá khả năng của máy tính.

Hàm free: Giải phóng bộ nhớ.

Cũng giống như khi ta sử dụng *fclose* để đóng tập tin khi không còn sử dụng đến, chúng ta sẽ sử dụng hàm *free* để giải phóng những vùng bộ nhớ không còn cần sử dụng nữa.

C code:

```
void free(void* contro);
```

Chúng ta cần hàm *free* để giúp giải phóng các vùng địa chỉ của bộ nhớ.

Vì vậy chúng ta sẽ gửi cho nó tham số là những con trỏ, VD như con trỏ capphatBonho trong ví dụ lúc nãy.

Sau đây là toàn bộ chương trình của chúng ta từ đầu đến giờ.

Nhìn chung thì cách hoạt động cũng không có quá nhiều điểm khác biệt so với những gì ta đã học khi thao tác với tập tin.

C code:

```
int main (int argc, char *argv[ ])
{
   int* capphatBonho = NULL;

   capphatBonho = malloc(sizeof(int));
   if (capphatBonho == NULL) // chung ta kiem tra xem bo nho da duoc cap phat chua
   {
      exit(0); // Error: co loi va chuong trinh se bi ngung ngay lap tuc
   }

   // Bo nho da duoc cap phat va san sang de su dung
   free(capphatBonho); // Chung ta ko can su dung bo nho nua, giai phong no thoi

   return 0;
}
```

Ví dụ cụ thể.

Chúng ta sẽ sắp xếp lại một số kiến thức bạn đã được học trước đây: Hỏi tuổi của người dùng và in nó ra.

Điều khác biệt duy nhất so với những gì chúng ta đã từng làm trước đó là lần này các biến sẽ được phân bổ một cách thủ công (còn gọi là cấp phát động) chứ không phải tự động như trước đây.

Vì vậy cho nên những dòng code nhìn sẽ phức tạp hơn. Nhưng các bạn hãy cố gắng hiểu được nó đi, điều này thật sự quan trọng đấy:

C code:

```
int main (int argc, char *argv[ ])
{
   int* capphatBonho = NULL;

   capphatBonho = malloc(sizeof(int)); // Cap phat bo nho
   if (capphaBonho == NULL)
   {
      exit(0);
   }

   // Su dung bo nho
   printf ("Ban bao nhieu tuoi ? ");
   scanf ("%d", capphatBonho);
   printf ("Ban %d tuoi\n", *capphatBonho);

   free(capphatBonho); // Giai phong bo nho

   return 0;
}
```

Console:

```
Ban bao nhieu tuoi ? 69
Ban 69 tuoi
```

 Hãy lưu ý: capphatBonho là một con trỏ, cách sử dụng nó khác với cách bạn làm việc với một biến bình thường. Để có được giá trị của con trỏ, bạn phải đặt dấu * trước capphatBonho (xem lại dòng code của *printf* để thấy rõ hơn). Trong khi đó, nếu muốn chỉ ra địa chỉ của con trỏ thì chỉ cần viết tên của nó capphatBonho (xem dòng *scanf* nhé).

Những điều này bạn đã được học trong bài về con trỏ rồi. Tuy nhiên, cần phải có một thời gian nhất định để bạn có thể quen với con trỏ, và khả năng nhầm lẫn vẫn rất cao. Trong trường hợp này bạn nên đọc lại một lần bài học về con trỏ, đây là những kiến thức cơ bản rất quan trọng.

Trở lại với đoạn code của chúng ta. Biến kiểu *int* đã được cấp phát động. Suy cho cùng những gì bạn đã viết cũng giống như trước đây khi tiến trình này diễn ra một cách tự động:

C code:

```
int main (int argc, char *argv[ ])
{
   int bienCuatoi = 0; // Cap phat bo nho (qua trinh dien ra hoan toan tu dong)

   // Su dung bo nho
   printf ("Ban bao nhieu tuoi ? ");
   scanf ("%d", &bienCuatoi);
   printf ("Ban %d tuoi\n", bienCuatoi);

   return 0;
} // Giai phong bo nho (dien ra tu dong moi khi ket thuc ham)
```

Tóm lại có 2 cách để tạo một biến, hay có thể nói là yêu cầu máy tính cấp phát bộ nhớ. Hai cách đó là:

- Tự động: Đây là những gì bạn đã từng học trước đây mỗi khi tạo ra một biến.
- Thủ công (cấp phát động): Cách mà tôi dạy cho bạn trong bài học này.

 Tôi không hiểu sao chúng ta phải học cái cách phức tạp này và liệu có cần thiết không?

Đúng là có phức tạp hơn một chút … nhưng nó có vô dụng không? Không! Đôi khi chúng ta buộc phải tự mình phân bổ bộ nhớ, tôi sẽ cho bạn thấy một trường hợp ngay sau đây.

Cấp phát động trong mảng.

Tới bây giờ chúng ta chỉ sử dụng phương pháp "cấp phát động" để tạo những biến bình thường đơn giản. Nhưng nhìn chung mọi người vẫn không thường dùng cách này, phương pháp tự động rõ ràng là đơn giản hơn nhiều.

Có phải bạn đang tự hỏi, vậy thì khi nào chúng ta nên dùng phương pháp này? Thường thì người ta sử dụng phương pháp "cấp phát động" khi tạo ra một mảng (array) mà ta không biết trước kích thước của nó khi chạy chương trình là bao nhiêu.

Ví dụ, tưởng tượng bạn sẽ tạo một chương trình lưu trữ tuổi những người bạn của bạn vào một mảng. Bạn có thể một mảng kiểu *int* để lưu trữ tuổi của họ, như sau:

C code:

```
int tuoiBanbe[15];
```

Nhưng ai dám đảm bảo với bạn rằng người dùng chỉ có 15 người bạn? Có thể nhiều hơn chứ. Khi viết chương trình, bạn không thể nào biết trước được kích thước nào là phù hợp cho mảng của bạn. Bạn chỉ có thể biết khi chương trình đã chạy, và bạn phải hỏi người dùng có bao nhiêu bạn bè.

Lợi ích của "cấp phát động" là ở đó: Chúng ta sẽ hỏi số lượng bạn bè của người dùng trước, và sau đó "cấp phát động" sẽ tạo ra mảng với kích thước chính xác như được yêu cầu (không quá nhỏ mà cũng không quá lớn). Nếu người dùng có 15 người bạn, chúng ta sẽ tạo một mảng với 15 giá trị *int*, tương tự nếu họ có 28 người bạn, nó sẽ tạo ra một mảng với 28 giá trị *int*, …

Như tôi đã từng dạy bạn, ngôn ngữ C không thể tạo ra một mảng với kích thước được chỉ định bằng một biến:

C code:

```
int banbe[soluongBanbe];
```

 Đoạn code trên có thể sẽ hoạt động với một số trình biên dịch trong một số trường hợp cụ thể, nhưng nó được khuyến cáo là không nên sử dụng như vậy.

Lợi ích của "cấp phát động" là nó cho phép ta tạo ra một mảng có kích thước khớp với số lượng bạn bè như biến soluongBanbe, dựa vào đó đoạn code có thể hoạt động ở bất kỳ đâu, với bất kỳ trình biên dịch nào.

Chúng ta sẽ dùng hàm *malloc* để yêu cầu cấp phát soluongBanbe * sizeof(int) byte trong bộ nhớ:

C code:

```
banbe = malloc(soluongBanbe * sizeof(int));
```

Đoạn code này sẽ giúp ta tạo ra một mảng kiểu *int* với kích thước đúng theo như số lượng bạn bè của người dùng.

Sau đây là những gì mà chương trình sẽ lần lượt thực hiện:

1. Yêu cầu người dùng cho biết họ có bao nhiêu bạn bè.
2. Tạo ra một mảng có kích thước bằng với số lượng bạn bè của người dùng (bằng cách sử dụng *malloc*)
3. Lần lượt hỏi tuổi của từng người bạn và lưu trữ vào trong mảng.
4. Hiển thị tất cả tuổi đã được lưu vào mảng trước đó.
5. Cuối cùng, vì chúng ta không còn cần đến mảng tuổi của những người bạn này nữa, ta sẽ giải phóng bộ nhớ bằng hàm *free*.

C code:

```c
int main (int argc, char *argv[ ])
{
   int soluongBanbe = 0, i = 0;
   int* tuoiBanbe = NULL; // con tro nay se duoc su dung nhu mot mang sau khi dung malloc

   // Chung ta se yeu cau nguoi dung cho biet so luong ban be cua ho
   printf ("Ban co bao nhieu nguoi ban ? ");
   scanf ("%d", &soluongBanbe);

   if (soluongBanbe > 0) // Phai co it nhat mot nguoi ban (xin chia buon neu ko co ai ^^!)
   {
      tuoiBanbe = malloc(soluongBanbe * sizeof(int)); // Phan phoi bo nho cho mang
      if (tuoiBanbe == NULL) // Kiem tra xem viec cap phat bo nho co thanh cong ko?
      {
         exit(0); // Chuong trinh ngung lai ngay lap tuc
      }

      // Yeu cau nhap tuoi tung nguoi ban
      for (i = 0 ; i < soluongBanbe ; i++)
      {
         printf ("Nguoi ban thu %d bao nhieu tuoi ? ", i + 1);
         scanf ("%d", &tuoiBanbe[i]);
      }

      // Lan luot hien thi tuoi cua ban be
      printf ("\n\nTuoi cua ban be ban la :\n");
      for (i = 0 ; i < soluongBanbe ; i++)
      {
         printf ("%d tuoi\n", tuoiBanbe[i]);
      }

      // Giai phong bo nho da duoc cap phat cho mang boi malloc, no ko con can thiet nua.
      free(tuoiBanbe);
   }

   return 0;
}
```

Console:

```
Ban co bao nhieu nguoi ban ? 5
Nguoi ban thu 1 bao nhieu tuoi ? 18
Nguoi ban thu 2 bao nhieu tuoi ? 19
Nguoi ban thu 3 bao nhieu tuoi ? 20
Nguoi ban thu 4 bao nhieu tuoi ? 21
Nguoi ban thu 5 bao nhieu tuoi ? 22

Tuoi cua ban be ban la :
18 tuoi
19 tuoi
20 tuoi
21 tuoi
22 tuoi
```

Chương trình có vẻ không ứng dụng được gì nhiều trong thực tế nhưng tôi chọn làm nó vì nó khá đơn giản để các bạn có thể hiểu được cách làm việc của hàm *malloc.*

Tôi đảm bảo với bạn rằng ở những bài học sau, chúng ta sẽ có cơ hội sử dụng hàm *malloc* để lưu trữ những thứ thú vị hơn là tuổi của bạn bè người dùng.

Tổng kết:

- Một biến sẽ chiếm không gian bộ nhớ nhiều hay ít là tùy thuộc vào kiểu của nó (int hay double hay char …)
- Người ta có thể biết được số byte mà kiểu biến đó sẽ chiếm trong bộ nhớ bằng cách sử dụng *sizeof ( ).*
- Cấp phát động là hành động dự trữ vùng nhớ cho biến hoặc mảng.
- Việc phân bổ bộ nhớ được thực hiện với hàm *malloc ( )* và điều quan trọng đừng nên quên đó là hãy nhớ giải phóng bộ nhớ bằng hàm *free ( )* ngay sau khi bạn không cần đến vùng nhớ đó nữa.
- Đặc biệt cấp phát động cho phép tạo ra mảng có kích thước được xác định bởi một biến trong khi chạy chương trình.
