Bộ nhớ động trong C++

Cho đến nay, trong các chương trình của chúng ta, tất cả những phần bộ nhớ chúng ta có thể sử dụng là các biến các mảng và các đối tượng khác mà chúng ta đã khai báo. Kích cỡ của chúng là cố định và không thể thay đổi trong thời gian chương trình chạy. Nhưng nếu chúng ta cần một lượng bộ nhớ mà kích cỡ của nó chỉ có thể được xác định khi chương trình chạy, ví dụ như trong trường hợp chúng ta nhận thông tin từ người dùng để xác định lượng bộ nhớ cần thiết.

Giải pháp ở đây chính là *bộ nhớ động*, C++ đã tích hợp hai toán tử `new` và `delete` để thực hiện việc này

Toán tử new và new[ ]

Để có thể có được bộ nhớ động chúng ta có thể dùng toán tử `new`. Theo sau toán tử này là tên kiểu dữ liệu và có thể là số phần tử cần thiết được đặt trong cặp ngoặc vuông. Nó trả về một con trỏ trỏ tới đầu của khối nhớ vừa được cấp phát. Dạng thức của toán tử này như sau:

```
pointer = new type
```

hoặc

```
pointer = new type [elements]
```

Biểu thức đầu tien được dùng để cấp phát bộ nhớ chứa một phần tử có kiểu `type`. Lệnh thứ hai được dùng để cấp phát một khối nhớ (một mảng) gồm các phần tử kiểu `type`.

Ví dụ:

```
int * bobby;
bobby = new int [5];
```

trong trường hợp này, hệ điều hành dành chỗ cho 5 phần tử kiểu **int** trong bộ nhớ và trả về một con trỏ trỏ đến đầu của khối nhớ. Vì vậy lúc này **bobby** trỏ đến một khối nhớ hợp lệ gồm 5 phần tử **int**.

Image not found or type unknown

Bạn có thể hỏi tôi là có gì khác nhau giữa việc khai báo một mảng với việc cấp phát bộ nhớ cho một con trỏ như chúng ta vừa làm. Điều quan trọng nhất là *kích thước* của một mảng phải là một hằng,

điều này giới hạn kích thước của mảng đến kích thước mà chúng ta chọn khi thiết kế chương trình trong khi đó cấp phát bộ nhớ động cho phép cấp phát bộ nhớ trong quá trình chạy với kích thước bất kì.

Bộ nhớ động nói chung được quản lí bởi hệ điều hành và trong các môi trường đa nhiệm có thể chạy một lúc vài chương trình có một khả năng có thể xảy ra là hết bộ nhớ để cấp phát. Nếu điều này xảy ra và hệ điều hành không thể cấp phát bộ nhớ như chúng ta yêu cầu với toán tử **new**, một con trỏ null (zero) sẽ được trả về. Vì vậy các bạn nên kiểm tra xem con trỏ trả về bởi toán tử **new** có bằng null hay không:

```
int * bobby;
bobby = new int [5];
if (bobby == NULL) {
  // error assigning memory.  Take measures.
};
```

Toán tử delete

Vì bộ nhớ động chỉ cần thiết trong một khoảng thời gian nhất định, khi nó không cần dùng đến nữa thì nó sẽ được giải phóng để có thể cấp phát cho các nhu cầu khác trong tương lai. Để thực hiện việc này ta dùng toán tử delete, dạng thức của nó như sau:

```
delete pointer;
```

hoặc

```
delete [] pointer;
```

Biểu thức đầu tiên nên được dùng để giải phóng bộ nhớ được cấp phát cho một phần tử và lệnh thứ hai dùng để giải phóng một khối nhớ gồm nhiều phần tử (mảng). Trong hầu hết các trình dịch cả hai biểu thức là tương đương mặc dù chúng là rõ ràng là hai toán tử khác nhau.

```
// rememb-o-matic
#include <iostream.h>
#include <stdlib.h>

int main ()
{
  char input [100];
  int i,n;
  long * l, total = 0;
  cout << "How many numbers do you want to type in? ";
  cin.getline (input,100); i=atoi (input);
```

```
  l= new long[i];
  if (l == NULL) exit (1);
  for (n=0; n<i; n++)
  {
    cout << "Enter number: ";
    cin.getline (input,100); l[n]=atol (input);
  }
  cout << "You have entered: ";
  for (n=0; n<i; n++)
    cout << l[n] << ", ";
  delete[] l;
  return 0;
}
```

```
How many numbers do you want to type in? 5
Enter number : 75
Enter number : 436
Enter number : 1067
Enter number : 8
Enter number : 32
You have entered: 75, 436, 1067, 8, 32,
```

NULL là một hằng số được định nghĩa trong thư viện C++ dùng để biểu thị con trỏ null. Trong trường hợp hằng số này chưa định nghĩa bạn có thể tự định nghĩa nó:

```
#define NULL 0
```

Dùng 0 hay `NULL` khi kiểm tra con trỏ là như nhau nhưng việc dùng `NULL` với con trỏ được sử dụng rất rộng rãi và điều này được khuyến khích để giúp cho chương trình dễ đọc hơn.

Bộ nhớ động trong ANSI-C

Toán tử `new` và `delete` là độc quyền C++ và chúng không có trong ngôn ngữ C. Trong ngôn ngữ C, để có thể sử dụng bộ nhớ động chúng ta phải sử dụng thư viện `stdlib.h`. Chúng ta sẽ xem xét cách này vì nó cũng hợp lệ trong C++ và nó vẫn còn được sử dụng trong một số chương trình.

- Hàm malloc

Đây là một hàm tổng quát để cấp phát bộ nhớ động cho con trỏ. Cấu trúc của nó như sau:

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

trong đó nbytes là số byte chúng ta muốn gán cho con trỏ. Hàm này trả về một con trỏ kiểu void*, vì vậy chúng ta phải chuyển đổi kiểu sang kiểu của con trỏ đích, ví dụ:

```
char * ronny;
ronny = (char *) malloc (10);
```

Đoạn mã này cấp phát cho con trỏ ronny một khối nhớ 10 byte. Khi chúng ta muốn cấp phát một khối dữ liệu có kiểu khác char (lớn hơn 1 byte) chúng ta phải nhân số phần tử mong muốn với kích thước của chúng. Thật may mắn là chúng ta có toán tử sizeof, toán tử này trả về kích thước của một kiểu dữ liệu cụ thể.

```
int * bobby;
bobby = (int *) malloc (5 * sizeof(int));
```

Đoạn mã này cấp phát cho bobby một khối nhớ gồm 5 số nguyên kiểu int, kích cỡ của kiểu dữ liệu này có thể bằng 2, 4 hay hơn tùy thuộc vào hệ thống mà chương trình được dịch.

- Hàm calloc:

calloc hoạt động rất giống với malloc, sự khác nhau chủ yếu là khai báo mẫu của nó:

```
void * calloc (size_t nelements, size_t size);
```

nó sử dụng hai tham số thay vì một. Hai tham số này được nhân với nhau để có được kích thước tổng cộng của khối nhớ cần cấp phát. Thông thường tham số đầu tiên (nelements) là số phần tử và tham số thức hai (size) là kích thước của mỗi phần tử. Ví dụ, chúng ta có thể định nghĩa bobby với calloc như sau:

```
int * bobby;
bobby = (int *) calloc (5, sizeof(int));
```

Một điểm khác nhau nữa giữa malloc và calloc là calloc khởi tạo tất cả các phần tử của nó về 0.

- Hàm realloc:

Nó thay đổi kích thước của khối nhớ đã được cấp phát cho một con trỏ.

```
void * realloc (void * pointer, size_t size);
```

Tham số `pointer` nhận vào một con trỏ đã được cấp phát bộ nhớ hay một con trỏ null, và size chỉ định kích thước của khối nhớ mới. Hàm này sẽ cấp phát size byte bộ nhớ cho con trỏ. Nó có thể phải thay đổi vị vị trí của khối nhớ để có thể đủ chỗ cho kích thước mới của khối nhớ, trong trường hợp này nội dung hiện thời của khối nhớ được copy tới vị trí mới để đảm bảo dữ liệu không bị mất. Con trỏ mới trỏ tới khối nhớ được hàm trả về. Nếu không thể thay đổi kích thước của khối nhớ thì

hàm sẽ trả về một con trỏ null nhưng tham số pointer và nội dung của nó sẽ không bị thay đổi.

- Hàm free:

Hàm này giải phóng một khối nhớ động đã được cấp phát bởi `malloc`, `calloc` hoặc `realloc`.

```
void free (void * pointer);
```

Hàm này chỉ được dùng để giải phóng bộ nhớ được cấp phát bởi các hàm malloc, calloc and realloc.

Revision #1
Created 5 October 2019 15:44:10 by Laptrinh.vn
Updated 5 October 2019 15:51:27 by Laptrinh.vn