

Bài 16: BIẾN TOÀN CỤC TRONG C++ (GLOBAL VARIABLES IN C++)

Xem bài học trên website để ủng hộ Kteam: [Biến toàn cục trong C++ \(Global variables in C++\)](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Ở bài học trước, bạn đã nắm được [TẦM VỰC VÀ BIẾN CỤC BÔ TRONG C++ \(Local variables\)](#), phạm vi hoạt động và vòng đời của một biến trong một chương trình.

Hôm nay, mình sẽ hướng dẫn về phần **Biến toàn cục trong C++ (Global variables)** và những kinh nghiệm khi sử dụng biến toàn cục trong lập trình.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- [BIẾN TRONG C++ \(Variables in C++\)](#)
- [BIẾN CỤC BÔ TRONG C++ \(Local variables\)](#)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Tổng quan về tầm vực của biến
- Biến toàn cục (global variables)
- Sử dụng biến toàn cục (non-const) là nguy hiểm

- Khi nào cần sử dụng biến toàn cục (non-const)

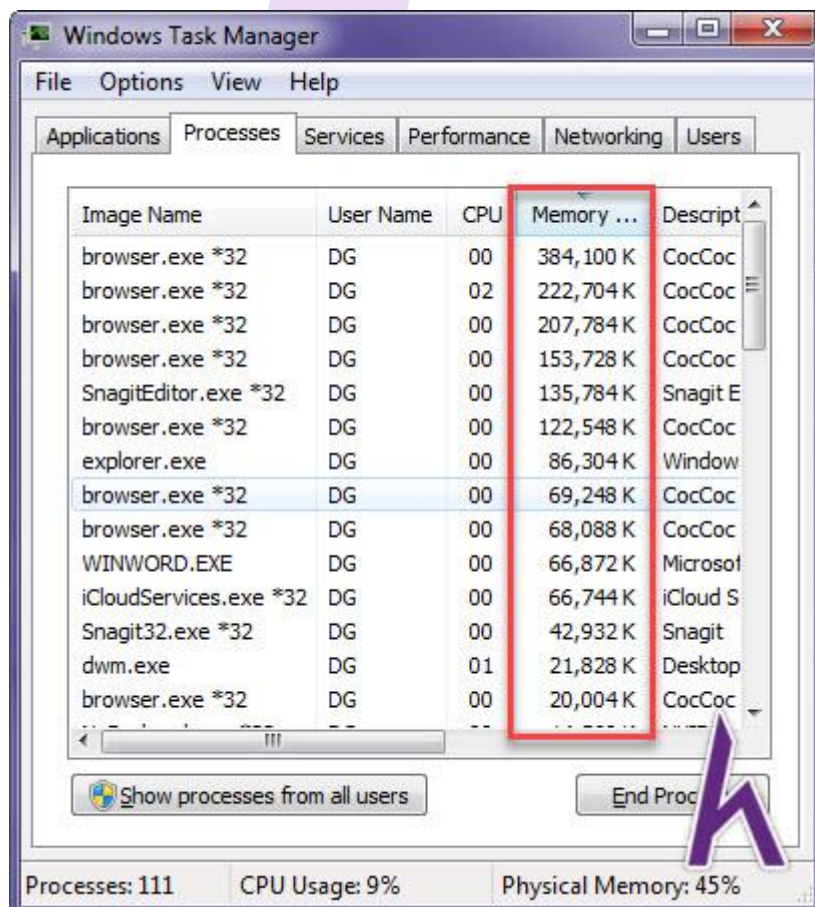
Tổng quan về tầm vực của biến

Trong bài học [BIẾN TRONG C++ \(Variables in C++\)](#), bạn đã biết được cách khai báo, khởi tạo và sử dụng một biến trong chương trình ra sao.

```
// Khai báo biến số nguyên nVarName  
int nVarName;
```

Khi chương trình được chạy, đến dòng lệnh này, **một vùng trong bộ nhớ RAM sẽ được cấp cho biến nVarName** này.

Ví dụ: RAM sẽ phải cấp phát vùng nhớ khi mỗi chương trình thực thi.



Windows Task Manager - Processes tab

Image Name	User Name	CPU	Memory ...	Descript
browser.exe *32	DG	00	384,100 K	CocCoc
browser.exe *32	DG	02	222,704 K	CocCoc
browser.exe *32	DG	00	207,784 K	CocCoc
browser.exe *32	DG	00	153,728 K	CocCoc
SnagitEditor.exe *32	DG	00	135,784 K	Snagit E
browser.exe *32	DG	00	122,548 K	CocCoc
explorer.exe	DG	00	86,304 K	Window
browser.exe *32	DG	00	69,248 K	CocCoc
browser.exe *32	DG	00	68,088 K	CocCoc
WINWORD.EXE	DG	00	66,872 K	Microsof
iCloudServices.exe *32	DG	00	66,744 K	iCloud S
Snagit32.exe *32	DG	00	42,932 K	Snagit
dwm.exe	DG	01	21,828 K	Desktop
browser.exe *32	DG	00	20,004 K	CocCoc

Processes: 111 CPU Usage: 9% Physical Memory: 45%

Vậy câu hỏi đặt ra là: "**Khi nào vùng nhớ của biến nVarName trong RAM được giải phóng?**" Khi trả lời được câu hỏi này, bạn có thể giúp chương trình của mình sử dụng bộ nhớ một cách khoa học hơn. Bài học hôm nay sẽ giúp bạn trả lời câu hỏi đó.

Khi nói về biến, có 2 khái niệm quan trọng bạn cần biết:

- **Phạm vi của biến:** Xác định nơi bạn có thể truy cập vào biến.
- **Thời gian tồn tại của biến:** Xác định nơi nó được tạo ra và bị hủy.

Phạm vi của biến được phân làm 2 loại:

- **Biến cục bộ (Local variables)**
- **Biến toàn cục (Global variables)**

Trong bài học trước, mình đã chia sẻ cho các bạn về [BIẾN CỤC BỘ TRONG C++ \(Local variables\)](#). Hôm nay, mình sẽ nói về **biến toàn cục (global variables)** và những kinh nghiệm khi sử dụng nó.

Biến toàn cục (Global variables)

Các biến khai báo **bên ngoài của khối lệnh** được gọi là **biến toàn cục**.

Biến toàn cục có **thời gian tĩnh**, nghĩa là chúng được tạo ra khi chương trình bắt đầu và bị hủy khi nó kết thúc. Các biến toàn cục có **phạm vi tập tin (file scope)**, hay gọi là "**phạm vi toàn cầu**" (**global scope**) hoặc "**phạm vi không gian tên toàn cầu**" (**global namespace scope**).

Định nghĩa biến toàn cục (Defining global variables)

Theo quy ước, **biến toàn cục (global variables)** được khai báo ở **đầu của một tập tin**, bên dưới `#include`.

Ví dụ:

```
#include <iostream>
```

```
using namespace std;

// Variables declared outside of a block are global variables
int g_x;           // global variable g_x
const int g_y(2); // global variable g_y

void doSomething()
{
    // global variables can be seen and used everywhere in program
    g_x = 3;
    cout << g_y << "\n";
}

int main()
{
    doSomething();

    // global variables can be seen and used everywhere in program
    g_x = 5;
    cout << g_y << "\n";

    return 0;
}
```

Outputs:



Trong chương trình trên, `g_x` và `g_y` là 2 biến toàn cục, có thể truy cập 2 biến này ở bất kỳ đâu trong file mà nó được định nghĩa. Ở đây là 2 hàm `doSomething()` và `main()`.

Phân biệt biến cục bộ và biến toàn cục

Tương tự như việc một biến bên trong một khối lệnh lồng nhau có thể có cùng tên với một biến ở khối lệnh bên ngoài, **biến cục bộ trùng tên** với biến toàn cục sẽ **ẩn các biến toàn cục** trong khối lệnh của biến cục bộ được định nghĩa.

Tuy nhiên, bạn có thể sử dụng **toán tử phân giải phạm vi (scope operator ::)** để thông báo cho trình biên dịch biết đó là biến cục bộ hay biến toàn cục.

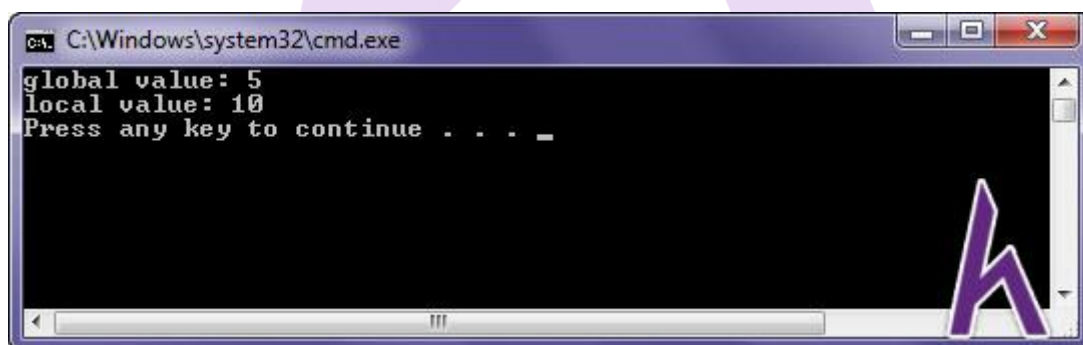
Ví dụ:

```
int value(6); // global variable

int main()
{
    int value = 9; // hides the global variable value
    value++; // increments local value, not global value
    ::value--; // decrements global value, not local value

    cout << "global value: " << ::value << "\n";
    cout << "local value: " << value << "\n";
    return 0;
} // local value is destroyed
```

Outputs:



Chú ý: Cần **tránh** việc đặt tên **biến toàn cục (global variables)** và **biến cục bộ (local variables)** trùng nhau. Theo quy ước, nên đặt

tiền tố "g_" trước các biến toàn cục (global variables). Điều này vừa giúp **xác định các biến toàn cục** cũng như **tránh đặt tên xung đột** với các biến cục bộ.

Sử dụng biến toàn cục là nguy hiểm

Biến toàn cục là một trong những khái niệm **bị lạm dụng nhất** trong lập trình. Mặc dù nó có vẻ vô hại trong các chương trình nhỏ, nhưng lại thường cực kỳ nguy hiểm ở những chương trình lớn.

Lập trình viên mới thường có thói quen sử dụng rất nhiều biến toàn cục, vì sử dụng nó khá đơn giản, bạn chỉ khai báo một lần và sử dụng nó trong tất cả các hàm của chương trình nếu bạn muốn. Tuy nhiên, đây không phải là một ý tưởng hay.

Nhiều nhà phát triển tin rằng nên **tránh hoàn toàn** việc sử dụng các biến toàn cục (**non-const global variables**). Ở đây **đang nói về biến toàn cục (non-const)**, không phải nói về tất cả.

Tại sao phải tránh sử dụng biến toàn cục?

Lý do lớn nhất khiến **biến toàn cục (non-const) nguy hiểm** vì giá trị của nó có thể được **thay đổi bởi bất cứ hàm nào** mỗi khi hàm đó được gọi. Khiến lập trình viên **khó kiểm soát** được chuyện gì đang xảy ra với biến toàn cục của mình.

Ví dụ:

```
// declare global variable
int g_nMode;

void doSomething()
{
    g_nMode = 2; // set the global g_mode variable to 2
}
```

```
}  
  
int main()  
{  
    // note: this sets the global g_mode variable to 1.  
    // It does not declare a local g_mode variable!  
    g_nMode = 1;  
  
    doSomething();  
  
    // Programmer still expects g_mode to be 1  
    // But doSomething changed it to 2!  
  
    if (g_nMode == 1)  
        cout << "Khong bi thay doi." << endl;  
    else  
        cout << "Bi thay doi." << endl;  
  
    return 0;  
}
```

Outputs:



Trong chương trình trên, chúng ta vừa gán giá trị `g_nMode` là 1, và sau đó gọi hàm `doSomething()`. Có thể bạn sẽ không biết chính xác hàm `doSomething()` có thay đổi giá trị của `g_nMode` hay không và nếu thay đổi thì sẽ thay đổi như thế nào. Do đó bạn sẽ khó mà kiểm soát tốt chương trình của bạn để chúng làm việc như mong muốn, đặc biệt là với những chương trình lớn.

Biến toàn cục (non-const) làm cho mỗi lời gọi hàm **tiềm ẩn những nguy hiểm**, lập trình viên khó kiểm soát được hàm nào đang tác động đến nó! **Biến cục bộ (local variables)** sẽ **an toàn hơn** vì mỗi hàm sẽ **độc lập và không ảnh hưởng** tới nhau.

Ở bài học trước, mình có nói qua 1 nguyên tắc: "**Định nghĩa** các biến cục bộ trong **phạm vi nhỏ nhất có thể**", làm vậy sẽ giảm thiểu phạm vi ảnh hưởng của biến trong chương trình. Ngược lại, biến toàn cục có thể sử dụng bất cứ nơi nào, làm tăng phạm vi ảnh hưởng và khó kiểm soát đối với chương trình lớn.

Biến toàn cục cũng làm cho chương trình của bạn **ít module và kém linh hoạt**.

Mỗi hàm thực hiện **một chức năng khác nhau**, truyền dữ liệu thông qua **tham số (parameter)**, và nó **hoạt động độc lập** với nhau. Một hàm sử dụng biến toàn cục sẽ **không thể tái sử dụng**, và **hoạt động độc lập** được.

Khi nào cần sử dụng biến toàn cục (non-const)

Có nhiều cách để giải quyết vấn đề tránh sử dụng **biến toàn cục (non-const)**. Nhưng trong một số trường hợp, sử dụng đúng đắn của các biến toàn cục (non-const) có thể làm giảm sự phức tạp cho chương trình.

Ví dụ trong trường hợp bạn có những loại dữ liệu muốn sử dụng cho toàn bộ chương trình. (configuration settings, database, ...)

Kết luận

Qua bài học này, bạn đã nắm được **Biến toàn cục trong C++ (Global variables)** và những kinh nghiệm khi sử dụng biến toàn cục trong lập trình. Mình tóm tắt lại 2 nội dung quan trọng các bạn cần nắm trong bài học này:

- **Tránh sử dụng các biến toàn cục (non-const) nếu có thể!** Nếu bạn phải sử dụng chúng, sử dụng chúng một cách hợp lý và thận trọng.
- **Biến hằng toàn cục (const)** là tốt để sử dụng, miễn là bạn sử dụng quy ước đặt tên thích hợp.

Ở bài tiếp theo, bạn sẽ được học về **BIẾN TĨNH TRONG C++ (Static variables)**.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".