Bài: Biến cục bộ trong C++ (Local variables in C++)

Xem bài học trên website để ủng hộ Kteam: Biến cục bộ trong C++ (Local 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 CƠ BẢN VỀ CHUỖI KÝ TỰ TRONG C++ (std::string).

Hôm nay, mình sẽ hướng dẫn về phần **Biến cục bộ trong C++ (Local variables)** và những kinh nghiệm khi sử dụng biến cục bộ 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++)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Tổng quan về tầm vực của biến
- Biến cục bộ (Local variables)

Tổng quan về tầm vực của biến

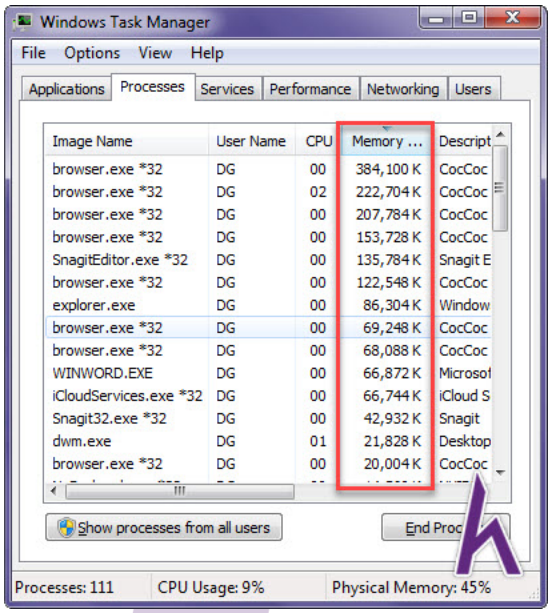
Trong bài học BIẾN TRONG C++ (Variables in C++), bạn đã biết được cách khai báo, khởi tạo và sử dụng một biến trong chương trình ra sao.

C++:

```
// 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.



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 này, mình sẽ chia sẽ với các bạn về biến cục bộ và những kinh nghiệm khi sử dụng nó.

Biến cục bộ (Local variables)

Biến được định nghĩa bên trong một khối lệnh (block) được gọi là các **biến cục bộ (Local variables).**

- Các **biến cục bộ có thời gian tự động**, có nghĩa là chúng **được tạo tại thời điểm định nghĩa**, và **bị hủy khi ra khỏi khối lệnh** mà biến đó được định nghĩa.
- Các biến cục bộ có **phạm vi bên trong khối lệnh** (còn được gọi là **phạm vi cục bộ**), nghĩa là sẽ **không truy cập được biến khi ở bên ngoài khối lệnh**.

Ví dụ: Ở chương trình bên dưới, n và d là 2 biến được định nghĩa bên trong hàm *main*(), và 2 biến này đều bị hủy khi hàm *main*() kết thúc.

C++:

```
int main()
{
     int n(6);         // n created and initialized here
     double d(9.0);    // d created and initialized here

     cout << "Enter a value for n: ";
     cin >> n;
     cout << "You entered: " << n << endl;

     return 0;
} // n and d go out of scope and are destroyed here
```

Vấn đề 1: Biến được định nghĩa **bên trong khối lệnh** lồng nhau **bị hủy** ngay **sau khi các khối bên trong kết thúc.**

Ví dụ:

C++:

```
int main() // outer block
{
     int n(6); // n created and initialized here

     { // begin nested block
          double d(9.0); // d created and initialized here
     } // d goes out of scope and is destroyed here

     // d can not be used here because it was already destroyed!

     return 0;
} // n goes out of scope and is destroyed here
```

Vấn đề 2: Khối lệnh lồng nhau được coi là một phần của khối lệnh bên ngoài, nơi biến được định nghĩa. Do đó, các biến được định nghĩa trong khối lệnh bên ngoài có thể được nhìn thấy bên trong một khối lệnh lồng nhau.

Ví dụ:

C++:

```
int main()
{ // start outer block
     int x(6);

     { // start nested block
          int y(9);
          // we can see both x and y from here
          cout << x << " + " << y << " = " << x + y;
     } // y destroyed here

     // y can not be used here because it was already destroyed!

     return 0;
} // x is destroyed here
```

Vấn đề 3: Biến được định nghĩa bên trong một khối lệnh chỉ có thể được nhìn thấy trong khối lệnh đó.

Vì mỗi hàm (function) có 1 khối lệnh riêng, các biến trong một hàm (function) không thể được nhìn thấy từ một hàm (function) khác.

Ví dụ:

C++:

```
void someFunction()
{
     int value(4); // value defined here

     // value can be seen and used here

} // value goes out of scope and is destroyed here

int main()
{
     // value can not be seen or used inside this function.

     someFunction();

     // value still can not be seen or used inside this function.

     return 0;
}
```

Vấn đề 4: Hàm (function) có thể có **các biến (variables)** hoặc các **tham số (parameters)** với **tên giống như các hàm (function) khác. Nghĩa là bạn không cần lo lắng về việc đặt tên xung đột giữa hai hàm (function) độc lập.**

Trong ví dụ bên dưới, cả hai hàm có các biến có tên là x và y. Các biến trong từng hàm không nhận thức được sự tồn tại của các biến khác có cùng tên trong các hàm khác.

Ví dụ:

C++:

```
#include <iostream>
using namespace std;

// add's x can only be seen/used within function add()
int add(int x, int y) // add's x is created here
{
     return x + y;
} // add's x is destroyed here

// main's x can only be seen/used within function main()
int main()
{
     int x = 6; // main's x is created here
     int y = 9;

     // the value from main's x is copied into add's x
     cout << add(x, y) << endl;

     return 0;
} // main's x is destroyed here
```

Vấn đề 5: Một biến bên trong một khối lệnh lồng nhau **có thể có cùng tên** với một biến ở khối lệnh bên ngoài. Trong trường hợp này, **biến ở khối lệnh lồng sẽ "ẩn" biến bên ngoài**. Nó được gọi tên ẩn (hiding) hoặc shadowing.

Ví dụ:

C++:

```
#include <iostream>
using namespace std;

int main()
{ // outer block
     int apples(6); // here's the outer block apples

     if (apples >= 6) // refers to outer block apples
     { // nested block
           int apples; // hides previous variable named apples

           // apples now refers to the nested block apples
           // the outer block apples is temporarily hidden

           apples = 9; // this assigns value 10 to nested block apples, not outer block apples

           cout << apples << endl; // print value of nested block apples
     } // nested block apples destroyed

     // apples now refers to the outer block apples again

     cout << apples << endl; // prints value of outer block apples

     return 0;
} // outer block apples destroyed
```

Outputs:

Chú ý: Tránh định nghĩa các biến lồng nhau có cùng tên với các biến bên ngoài khối lệnh.

> **Vấn đề 6:** Nếu một biến chỉ được sử dụng trong một khối lệnh lồng nhau, nó phải được định nghĩa bên trong khối lệnh lồng nhau.

Ví dụ:

C++:

```
#include <iostream>
using namespace std;

int main()
{
     // we're declaring y here because we need it in this outer block later
     int y(6);

     {
           int x;
           cin >> x;
           // if we declared y here, immediately before its actual first use...
           if (x == 9)
                 y = 9;
     } // ... it would be destroyed here

     cout << y; // and we need y to exist here

     return 0;
}
```

Nguyên tắc: Biến phải được **định nghĩa trong phạm vi nhỏ nhất có thể**.

Kết luận

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

- **Biến được định nghĩa bên trong khối lệnh được gọi là các biến cục bộ (local variables)**. Những biến này chỉ có thể được truy cập bên trong các khối lệnh mà nó được định nghĩa (bao gồm các khối lệnh lồng nhau), và bị hủy ngay sau khi các khối lệnh kết thúc.
- **Định nghĩa các biến trong phạm vi nhỏ nhất có thể**. Nếu một biến chỉ được sử dụng trong một khối lệnh lồng nhau, nó phải được định nghĩa bên trong khối lệnh lồng nhau.

Ở bài tiếp theo, bạn sẽ được học về BIẾN TOÀN CỤC TRONG C++ (Global 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**".