Lớp lưu trữ (Storage Class) trong C++

Lớp lưu trữ (Storage Class) định nghĩa phạm vi và vòng đời của biến và/hoặc các hàm bên trong một chương trình C++. Chúng thường đứng trước kiểu dữ liệu mà chúng tác động. Dưới đây là các lớp lưu trữ có thể được sử dụng trong C++:

- auto
- register
- static
- extern
- mutable

Lớp lưu trữ auto trong C++

Lớp lưu trữ **auto** trong C++ là lớp lưu trữ mặc định cho tất cả biến cục bộ trong C++:

```
{
   int mount;
   auto int month;
}
```

Ví dụ trên định nghĩa hai biến với cùng lớp lưu trữ, auto chỉ có thể được sử dụng bên trong các hàn, ví dụ: cho các biến nội bộ.

Lớp lưu trữ register trong C++

Lớp lưu trữ **register** trong C++ được sử dụng để định nghĩa các biến cục bộ mà nên được lưu giữ trong một thanh ghi thay vì RAM. Nghĩa là, biến có kích cỡ tối đa bằng với kích cỡ thanh ghi (thường là 1 từ) và không thể có toán tử một ngôi '&' được áp dụng tới nó (vì không có địa chỉ bộ nhớ).

```
{
   register int  miles;
}
```

Lớp lưu trữ register nên chỉ được dùng cho các biến yêu cầu truy cập nhanh như các biến đếm (counters). Cũng cần chú ý rằng, một biến định nghĩa với 'register' không có nghĩa là biến đó được lưu trữ trong thanh ghi. Tức là nó có thể được lưu trữ trong thanh ghi phụ thuộc vào phần cứng và giới hạn thực thi.

Lớp lưu trữ static trong C++

Lớp lưu trữ **static** trong C++ nói với compiler để giữ một biến cục bộ tồn tại trong toàn bộ thời gian sống của chương trình thay vì tạo và hủy biến mỗi lần nó vào và ra khỏi phạm vi biến. Vì vậy, các biến có static cho phép nó duy trì giá trị giữa các lần gọi hàm.

Lớp lưu trữ static cũng có thể được áp dụng cho các biến toàn cục (global). Khi áp dụng cho biến toàn cục, nó nói với trình biên dịch rằng, phạm vi của biến toàn cục bị giới hạn trong tập tin mà nó được khai báo.

Trong C++, khi static được sử dụng trên thành viên dữ liệu của lớp, nó gây ra: chỉ có một bản sao của thành viên đó được chia sẻ bởi tất cả đối tượng trong lớp của nó.

```
#include <iostream>

// Function declaration
void func(void);

static int count = 10; /* Global variable */

main()
{
    while(count--)
    {
       func();
    }
    return 0;
}
// Function definition
void func( void )
{
```

```
    static int i = 5; // local static variable
    i++;
    std::cout << "i is " << i ;
    std::cout << " and count is " << count << std::endl;
}
```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

```
i is 6 and count is 9
i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0
```

Lớp lưu trữ extern trong C++

Lớp lưu trữ **extern** trong C++ được dùng để cung cấp một tham chiếu của một biến toàn cục được nhìn thấy bởi TẤT CẢ các file chương trình. Khi bạn sử dụng 'extern', biến không thể được khởi tạo, khi nó trỏ tới tên biến tại một vị trí lớp lưu trữ mà đã được định nghĩa trước đó.

Khi bạn có nhiều file và bạn định nghĩa một biến hay hàm toàn cục trong một file và cũng muốn dùng nó trong các file khác, thì **extern** được dùng trong file khác để cung cấp tham chiếu của biến hay hàm được định nghĩa. Cần nhớ rằng, **extern** dùng để khai báo một biến hay hàm toàn cục trong file khác.

Lớp lưu trữ extern được dùng phổ biến khi có hai hoặc nhiều file chia sẻ cùng biến hay hàm toàn cục. Xem ví dụ với hai file sau:

File đầu tiên: main.cpp

```
#include <iostream>
```

```
int count ;
extern void write_extern();

main()
{
   count = 5;
   write_extern();
}
```

File thứ hai: support.cpp

```
#include <iostream>

extern int count;

void write_extern(void)
{
   std::cout << "Count is " << count << std::endl;
}
```

Ở đây, từ khóa *extern* đang được sử dụng để khai báo count trong file khác. Bây giờ biên dịch hai file này như sau:

```
$g++ main.cpp support.cpp -o write
```

Nó sẽ tạo chương trình **write** có thể thực thi, bạn thử thực thi write và kiểm tra kết quả như sau:

```
$./write
5
```

Lớp lưu trữ mutable trong C++

Lớp lưu trữ **mutable** trong C++ chỉ áp dụng cho các đối tượng class, sẽ được bàn luận trong chương sau. Nó cho phép một thành viên của một đối tượng để override (ghi đè). Đó là, một thành viên là mutable có thể được sửa đổi bởi một hàm thành viên const.