6.4 — Introduction to global variables

ALEX APRIL 20, 2021

In lesson 6.3 -- Local variables, we covered that local variables are variables defined inside a function (or function parameters). Local variables have block scope (are only visible within the block they are declared in), and have automatic duration (they are created at the point of definition and destroyed when the block is exited).

In C++, variables can also be declared *outside* of a function. Such variables are called global variables.

Declaring and naming global variables

By convention, global variables are declared at the top of a file, below the includes, but above any code. Here's an example of a global variable being defined:

```
#include <iostream>

// Variables declared outside of a function are global variables
int g_x {}; // global variable g_x

void doSomething()
{
    // global variables can be seen and used everywhere in the file
    g_x = 3;
    std::cout << g_x << '\n';
}

int main()
{
    doSomething();
    std::cout << g_x << '\n';

    // global variables can be seen and used everywhere in the file
    g_x = 5;
    std::cout << g_x << '\n';

    return 0;
}
// g_x goes out of scope here
```

The above example prints:

```
3
3
5
```

By convention, many developers prefix global variable identifiers with "g" or "g_" to indicate that they are global.

Best practice

Consider using a "g" or "g_" prefix for global variables to help differentiate them from local variables.

Global variables have file scope and static duration

Global variables have file scope (also informally called global scope or global namespace scope), which means they are visible from the point of declaration until the end of the *file* in which they are declared. Once declared, a global variable can be used anywhere in the file from that point onward! In the above example, global variable `g_x` is used in both functions `doSomething()` and `main()`.

Because they are defined outside of a function, global variables are considered to be part of the global namespace (hence the term "global namespace scope").

Global variables are created when the program starts, and destroyed when it ends. This is called static duration. Variables with *static duration* are sometimes called static variables.

Unlike local variables, which are uninitialized by default, static variables are zero-initialized by default.

Global variable initialization

Non-constant global variables can be optionally initialized:

```
int g_x; // no explicit initializer (zero-initialized by
default)
int g_y {}; // zero initialized
int g_z { 1 }; // initialized with value
```

Constant global variables

Just like local variables, global variables can be constant. As with all constants, constant global variables must be initialized.

```
#include <iostream>

const int g_x; // error: constant variables must be initialized
constexpr int g_w; // error: constexpr variables must be initialized

const int g_y { 1 };  // const global variable g_y, initialized with a value
constexpr int g_z { 2 }; // constexpr global variable g_z, initialized with a
value

void doSomething()
{
    // global variables can be seen and used everywhere in the file
    std::cout << g_y << '\n';
    std::cout << g_z << '\n';
}

int main()
{
    doSomething();

    // global variables can be seen and used everywhere in the file
    std::cout << g_y << '\n';
    std::cout << g_z << '\n';

    return 0;
}
// g_y and g_z goes out of scope here

```

Related content

We discuss global constants in more detail in lesson 6.9 -- Sharing global constants across multiple files (using inline variables).

A word of caution about (non-constant) global variables

New programmers are often tempted to use lots of global variables, because they can be used without having to explicitly pass them to every function that needs them. However, use of non-constant global variables should generally be avoided altogether! We'll discuss why in upcoming lesson 6.8 -- Why (non-const) global variables are evil

Quick Summary

```cpp
// Non-constant global variables
int g_x;                // defines non-initialized global variable (zero initialized by default)
int g_x {};             // defines explicitly zero-initialized global variable
int g_x { 1 };          // defines explicitly initialized global variable

// Const global variables
const int g_y;          // error: const variables must be initialized
const int g_y { 2 };    // defines initialized global constant

// Constexpr global variables
constexpr int g_y;      // error: constexpr variables must be initialized
constexpr int g_y { 3 }; // defines initialized global const
```


Next lesson

6.5 Variable shadowing (name hiding)

Back to table of contents

Previous lesson

6.3 Local variables

Notify me about replies: POST COMMENT

DP N N FOUT

Newest