6.x — Chapter 6 summary and quiz

ALEX SEPTEMBER 21, 2021

Quick review

We covered a lot of material in this chapter. Good job, you're doing great!

A compound statement or block is a group of zero or more statements that is treated by the compiler as if it were a single statement. Blocks begin with a `{` symbol, end with a `}` symbol, with the statements to be executed placed in between. Blocks can be used anywhere a single statement is allowed. No semicolon is needed at the end of a block. Blocks are often used in conjunction with `if statements` to execute multiple statements.

User-defined namespaces are namespaces that are defined by you for your own declarations. Namespaces provided by C++ (such as the `global namespace`) or by libraries (such as `namespace std`) are not considered user-defined namespaces.

You can access a declaration in a namespace via thescope resolution operator (::). The scope resolution operator tells the compiler that identifier specified by the right-hand operand should be looked for in the scope of the left-hand operand. If no left-hand operand is provided, the global namespace is assumed.

Local variables are variables defined within a function (including function parameters). Local variables have block scope, meaning they are in-scope from their point of definition to the end of the block they are defined within. Local variables have automatic storage duration, meaning they are created at the point of definition and destroyed at the end of the block they are defined in.

A name declared in a nested block canshadow or name hide an identically named variable in an outer block. This should be avoided.

Global variables are variables defined outside of a function. Global variables have file scope, which means they are visible from the point of declaration until the end of the file in which they are declared. Global variables have static duration, which means they are created when the program starts, and destroyed when it ends. Avoid dynamic initialization of static variables whenever possible.

An identifier's linkage determines whether other declarations of that name refer to the same object or not. Local variables have no linkage. Identifiers with internal linkage can be seen and used within a single file, but it is not accessible from other files. Identifiers with external linkage can be seen and used both from the file in which it is defined, and from other code files (via a forward declaration).

Avoid non-const global variables whenever possible. Const globals are generally seen as acceptable. Use inline variables for global constants if your compiler is C++17 capable.

Local variables can be given static duration via thestatic keyword.

Using statements (including using declarations and using directives) can be used to avoid having to qualify identifiers with an explicit namespace. These should generally be avoided.

Finally, C++ supports unnamed namespaces, which implicitly treat all contents of the namespace as if it had internal linkage. C++ also supports inline namespaces, which provide some primitive versioning capabilities for namespaces.

Quiz time

Question #1

Fix the following program:

```
#include <iostream>

int main()
{
 std::cout << "Enter a positive number: ";
 int num{};
 std::cin >> num;


 if (num < 0)
  std::cout << "Negative number entered.  Making
positive.\n";
  num = -num;

 std::cout << "You entered: " << num;

 return 0;
}
```

Show Solution

Question #2

Write a file named constants.h that makes the following program run. If your compiler is C++17 capable, use inline constexpr variables. Otherwise, use normal constexpr variables.

main.cpp:

```
#include <iostream>
#include "constants.h"

int main()
{
 std::cout << "How many students are in your class? ";
 int students{};
 std::cin >> students;


 if (students > constants::max_class_size)
  std::cout << "There are too many students in this
class";
 else
  std::cout << "This class isn't too large";

 return 0;
}
```

Show Solution

Question #3

Complete the following program by writing the passOrFail() function, which should return true for the first 3 calls, and false thereafter. Do this without modifying the main() function.

Show Hint

```
#include <iostream>

int main()
{
 std::cout << "User #1: " << (passOrFail() ? "Pass\n" :
"Fail\n");
 std::cout << "User #2: " << (passOrFail() ? "Pass\n" :
"Fail\n");
 std::cout << "User #3: " << (passOrFail() ? "Pass\n" :
"Fail\n");
 std::cout << "User #4: " << (passOrFail() ? "Pass\n" :
"Fail\n");
 std::cout << "User #5: " << (passOrFail() ? "Pass\n" :
"Fail\n");

 return 0;
}
```

The program should produce the following output:

```
User #1: Pass
User #2: Pass
User #3: Pass
User #4: Fail
User #5: Fail
```

Show Solution

Next lesson
7.1 Control flow introduction

Back to table of contents

Previous lesson
6.13 Unnamed and inline namespaces

DP N N FOUT

Newest

X

X