User Id : weizhe.goh@digipen.edu | Started : 2020.05.29 10:01:17 | Score : 33%

DigiPen

Namespaces

Practice

Intro

- Every struct, variable, or function declared at the global scope must have a unique name.
- The global scope is also referred to as the **global namespace** scope to distinguish it from other namespaces.
- In small programs, this is not a problem, since one person may produce all of these names.
- With very large programs, however, it can be a problem, especially with libraries from other sources.

Example with a problem

- A simple program with 3 global symbols:

```
#include <iostream>

int foo = 1;
int bar = 2;

int Div2(int value) {
  return value / 2;
}

int main(void) {
  std::cout << foo;
  std::cout << bar;
  std::cout << Div2(8);
  return 0;
}
```

- If another source file (.cpp) has foo, bar, or Div2, it will cause problems.

And a solution

- One solution is to make them static:

```
static int foo = 1;
static int bar = 2;
static int Div2(int value) {
  return value / 2;
}
```

but this will limit their use to this file only.

- A better solution is to put them in a unique namespace.

```
namespace DigiPen
{
  int foo = 1;
  int bar = 2;
  int Div2(int value) {
    return value / 2;
  }
}
```

- We need to qualify the symbols in the namespace with the scope resolution operator (::) when use:

```
int main(void) {
  std::cout << DigiPen::foo;
  std::cout << DigiPen::bar;
  std::cout << DigiPen::Div2(8);
  return 0;
}
```

Props

- Namespace definitions do not have to be contiguous:

```
namespace DigiPen {
  int foo = 1;
  int bar = 2;
}
// Lots of other code here ...
namespace DigiPen {
  int Div2(int value)  {
    return value / 2;
  }
}
```

- However, if there are definitions needed by the program, they must still be seen by the compiler before they are used:
- Note that the separate definitions of the same samespace can be in separate files as well. This gives you the flexibility to put the interface for your code into the public files (header files) where your users can see it, and keep the implementation hidden in the .cpp files (.obj, .lib, etc.)

Unnamed Namespaces

- Namespace names are global, so there's no way to protect them from other global names. This is a problem if code uses lots of small namespaces.
- Unnamed namespaces are user to "localize" the use of names in the files where they are defined. (Similar to the static keyword.)

```
#include <iostream>

namespace {
  double sqrt(double x) { return x; }
}

int main(void) {
  std::cout << sqrt(25.0);
  return 0;
}
```

- The unnamed namespace permits access to sqrt without using a scope resolution operator.

Scope Resolution Operator

- When the scope resolution operator is placed before a symbol, it indicates that the symbol should be accessed from the global namespace.

Run

```
#include <iostream>
int foo = 1; // global
int bar = 2; // global
int main(void)
{
  int foo = 10;
    // local foo #1 hides global foo
  int bar = foo;
    // local bar #1 hides global bar
    //   (set to local foo)
  int baz = ::foo;
    // local baz #1 is set to global foo
  if (bar == 10)
  {
    int foo = 100;
      // local foo #2 hides local #1
      //    and global
    bar = foo;
      // local bar #1 is set to local
      //   foo #2
    foo = ::bar;
      // local foo #2 is set to global
      //    bar
  }
  ::foo = foo;
    // global foo is set to local foo #1
  ::bar = ::foo;
    // global bar is set to global foo
  std::cout << "foo is " << foo
    << std::endl; // local foo #1 is 10
  std::cout << "bar is " << bar
    << std::endl; // local bar #1 is 100
  std::cout << "::foo is " << ::foo
    << std::endl; // global foo is 10
  std::cout << "::bar is " << ::bar
    << std::endl; // global bar is 10
  return 0;
}
```

```
foo is 10
bar is 100
::foo is 10
::bar is 10
```

Namespaces Aliases

- To allow unique namespaces and to shorten the names, you can create a namespace alias.

```
// Declare these after the
//   namespace definitions above
namespace AP = AdvancedProgramming;
namespace IP = IntroductoryProgramming;
int main(void)
{
  // Now, use the shorter aliases
  std::cout << AP::foo << std::endl;
  std::cout << IP::foo << std::endl;

  std::cout << AP::f1(8) << std::endl;
  std::cout << IP::Div2(8) << std::endl;
  return 0;
}
```

Using Directives

- A using directive allows you to make all of the names in a namespace visible at once:

Run

```
#include <iostream>

namespace Stuff
{
  int foo = 11; // Stuff::foo
  int bar = 12; // Stuff::bar
  int baz = 13; // Stuff::baz
}

using namespace Stuff;

int main()
{
  std::cout << foo << std::endl;
  std::cout << bar << std::endl;
  std::cout << baz << std::endl;

  return 0;
}
```

```
11
12
13
```

- Using directives are applied only within block where directive is used:

```
int main()
{
  using namespace Stuff;

  std::cout << foo << std::endl;
  std::cout << bar << std::endl;
  std::cout << baz << std::endl;

  return 0;
}
```

- Ambiguity errors are detected when an ambiguous name is referenced, not when the directive is encountered. Qualified names can override the using directive.

Using Directives 2

- Using directives were created to help migrate existing (pre-namespace) code.
- It is not meant to be used to make it "easier" on the programmer (by saving keystrokes).
- Many using directives will cause the global namespace to be polluted, which is the primary purpose of namespaces to begin with.
- It's best to avoid using directives, but may be useful if you are dealing with a lot of legacy code.
- Never use them in header files that are meant to be used by others.
- Note Using directives were designed for backward-compatibility with existing C++ code (which doesn't understand namespaces) to help support older code. They should rarely be used when writing new code.

★ Pop quiz 1 — 1

Check two equivalent codes.

[✓ ■]
```
#include <iostream>
int main(void) {
    std::cout << "Hello World!";
    return 0;
}
```

[✓ ■]
```
#include <iostream>
using namespace std;
int main(void) {
    cout << "Hello World!";
    return 0;
}
```

☐
```
#include <iostream>
namespace std {
  int main(void) {
     cout << "Hello World!";
     return 0;
  }
}
```

★ Pop quiz 2 — 0

What is the output of the following code?

```
#include <iostream>
namespace {
    char c = 'a';
}
char c = 'b';
int main(void) {
    std::cout << ::c;
    return 0;
}
```

✗ [✓ ● a]
✓ ○ b
○ Compilation error

★ Pop quiz 3 — 0

What is the output of the following code?

```
#include <iostream>
namespace first {
    int var = 5;
}
namespace {
    double var = 3.1416;
}
int main(void) {
    using namespace first;
    std::cout << var;
    return 0;
}
```

✗ [✓ ● 5]
○ 3.1416
✓ ○ Compilation error

By signing this document you fully agree that all information provided therein is complete and true in all respects.

Responder sign: