NAMESPACES AND FUNCTION OVERLOADING

Name collisions and how to avoid them

Florian Warg, Max Staff

April 27, 2017

NAME COLLISIONS

- a project uses libraries A and B
- A and B offer functions with similar names and arguments

```
/* Library A */
digest hash(block input) { return md5(input); }
/* Library B */
digest hash(block input) { return sha1(input); }
/* User code */
hash(my_message);
```

- the compiler cannot deduce which function you want to call
- this is called a name collision
- it will cause a compiler error

NAMESPACES

- libraries use namespaces to avoid name collisions
- namespaces can contain functions, types and objects
- symbols can be accessed with ns::symbol

```
namespace LibA {
    digest hash(block input);
}
namespace LibB {
    digest hash(block input);
}
/* call hash from Lib A */
LibA::hash(my_message);
/* call hash from Lib B */
LibB::hash(my_message);
```

NESTED NAMESPACES

- namespaces can also contain other namespaces
- this can be used to organize software into packages

```
namespace Lib {
    namespace crypto {
        digest hash(block input) { ... }
    }
}
Lib::crypto::hash(my_message);
```

SPLIT NAMESPACES

- namespaces can be opened and closed at any point
- symbols inside the same namespace are visible to each other

```
namespace A {
    int x;
} /* namespace A */

namespace A {
    int y;
} /* namespace A */
```

IMPORT SYMBOLS

- symbols can be imported into the local namespace
- achieved with a ”using directive”

```
1 #include <string>
2 using std::string;
3 int main() {
4     std::string s1;
5     string s1; // imported std::string as string
6 }
```

IMPORT NAMESPACES

- whole namespaces can be imported, too
- this will import all symbols

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main() {
5     string name; // std::string
6     cin >> name; // std::cin
7     cout << "Hello, " << name << "\n";
8 }
```

BEST PRACTICE

- global namespace should have as few symbols as possible
- never import symbols in header files
- full symbol name is easier to understand in code reviews
- import single symbols rather than namespaces
- in production code almost always use full names

SCOPES

- each name in C++ has restricted validity (function, namespace)
- a scope is the portion of source code where a certain name (e.g. variable) is present
- begins at declaration
- ends at closing curly brace
- what about custom scopes?

BLOCK SCOPE

```
1  int main() {
2      int a = 0; ++a;
3      {
4          int a = 1;
5          a = 42;
6      }
7      std::cout << a << "\n"; // ?
8  }
9  int b = a; // Error?
```

FUNCTION OVERLOADING

- two functions to achieve the same goal for different datatypes
- example: outputting to std::cout
- optional parameters with default values

DEFAULT ARGUMENTS

```
1  float round(float a, float b = 1) {
2      return a - (a % b);
3  }
4
5  round(10.5);
6  round(11.3579, 0.1);
7  round(12345f, 10f);
8
9  // this is incorrect:
10 float round(float b = 1, float a) {
11     return a - (a % b);
12 }
```

DIFFERENT AMOUNT OF ARGUMENTS

```
1  float distance(float a, float b) {
2      return std::sqrt(a*a + b*b);
3  }
4
5  float distance(float a, float b, float c) {
6      return std::sqrt(a*a + b*b + c*c);
7  }
8
9  distance(5, 3);
10 distance(5, 3, 7);
```

DIFFERENT TYPE OF ARGUMENTS

```
1  float round(float a, float b = 1) {
2      return a - (a % b);
3  }
4
5  int round(int a, int b = 1) {
6      return a - (a % b);
7  }
8
9  round(5.7);
10 round(4.222, 0.1);
11 round(355);
12 round(12345, 10);
```