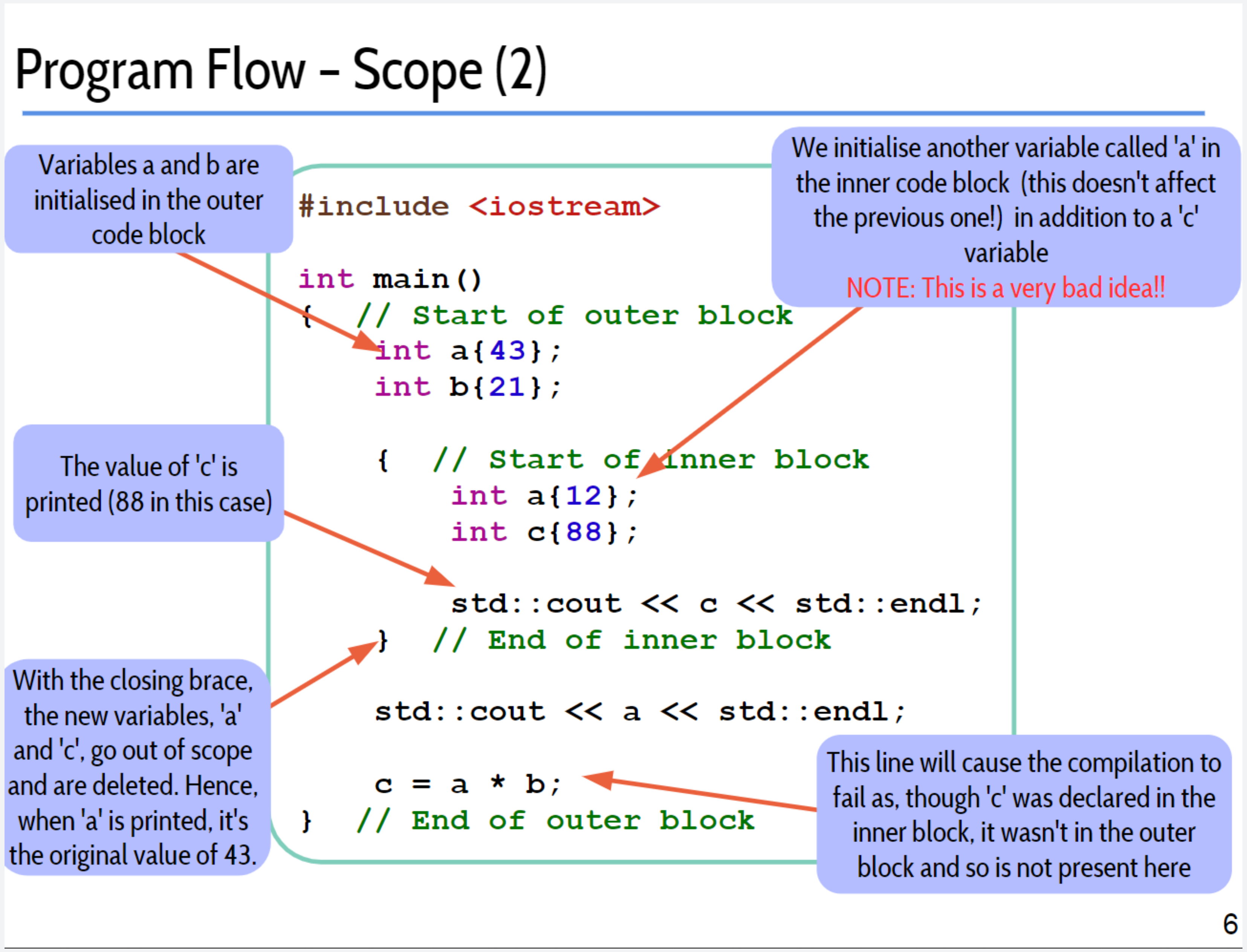
University of BRISTOL

C++ Namespaces

Luke Kreczko

Reminder: Scopes in C++

- On day 1 you've encountered scopes:



Program Flow – Scope (2)

```
#include <iostream>

int main()
{   // Start of outer block
    int a{43};
    int b{21};

    {   // Start of inner block
        int a{12};
        int c{88};

        std::cout << c << std::endl;
    }   // End of inner block

    std::cout << a << std::endl;

    c = a * b;
}   // End of outer block
```

Variables a and b are initialised in the outer code block

We initialise another variable called 'a' in the inner code block (this doesn't affect the previous one!) in addition to a 'c' variable
NOTE: This is a very bad idea!!

The value of 'c' is printed (88 in this case)

With the closing brace, the new variables, 'a' and 'c', go out of scope and are deleted. Hence, when 'a' is printed, it's the original value of 43.

This line will cause the compilation to fail as, though 'c' was declared in the inner block, it wasn't in the outer block and so is not present here

6

- C++ symbols (variables, functions, classes) have a scope of validity
- Accessing them outside their scope leads to compilation errors
- Scopes can clash (e.g. variable 'a' defined in a scope and a parent scope)

Namespaces – named scopes, kind of.

The namespace keyword in C++ is used to create a named scope

```
// example_namespace.hh
namespace myspace
{
   void function1();                 // myspace::function1
   constexpr int myConst = 43;   // myspace::myConst
   class MyClass();                  // myspace::MyClass
}
```

Note
Namespaces are defined at the top-level scope or within other namespaces
(i.e. NOT inside functions or classes

You can access this scope my using <name of namespace>::<name of symbol>.
You've encountered an example before, the std:: namespace

```
// main.cc
#include "example_namespace.hh"
int main()
{
   auto c = myspace::MyClass();
   myspace::function1();
   std::cout << myspace::myConst << std::endl;
}
```

What are namespaces good for?

Code is rarely used in isolation – **namespaces can help to separate scopes**

Example:

- Library A and library B defined a constant “pi”
- You want to use both libraries
  - Without namespaces you have two definitions of “pi”, one of which is shadowed – which value are you accessing?
  - With namespaces you have A::pi and B::pi → can pick which one you want

What are namespaces good for?

Larger code bases usually organized in smaller bits – **namespaces can help to organize them**

Example:

```
// my_lib.hh
namespace myspace
{
   void function1();                    // myspace::function1
}
// my_lib/algebra.hh
namespace myspace
{
   namespace algebra
   {
      float sqrt();                   // myspace::algebra::sqrt
   }
}
```

What are namespaces good for?

Sometimes humans need to read the code - **namespaces can help to provide context in a lightweight way (i.e. without using classes)**

Example:

```
// my_lib.hh
namespace solarsystem
{
  namespace venus
  {
    int numberOfSpecies();
  }

  namespace earth
  {
    int numberOfSpecies();
  }
}
```

Note
If you ever find yourself creating a class with *mostly static functions*

→ maybe you are looking for a namespace

Namespaces - Shortcuts

Nested namespaces can lead to long names – “using” is the keyword to select a namespace

```
// main.cc
#include “solar_system.hh” // contains solarsystem::earth::Europe::UK::<city>::checkWeather();
int main()
{
   using namespace solarsystem::earth::Europe::UK; // puts everything in …::UK:: into the current scope

   auto weatherBirmingham = Birmingham::checkWeather();
   auto weatherBristol = Bristol::checkWeather();
   auto weatherWarwick = Warwick::checkWeather();

   std::cout << “Today’s weather in” << std::endl;
   std::cout << “Birmingham” << weatherBirmingham << std::endl;
   std::cout << “Bristol” << weatherBristol << std::endl;
   std::cout << “Warwick” << weatherWarwick << std::endl;
}
```

Alternatively, you can “alias” namespaces:

```
namespace UK = solarsystem::earth::Europe::UK;
```

Summary

- Namespaces can help to
  - Avoid clashes between different libraries
  - Organize code in a larger code base
  - Provide (lightweight) context to improve readability by humans
- Together this enables the **D**on't **R**epeat **Y**ourself (DRY) principle
- If you do not want to type long namespaces you can
  - Merge them with your scope (using namespace mySpace)
  - Alias them: namespace UK = solarsystem::earth::Europe::UK