School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 25: Principles of Reuse

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

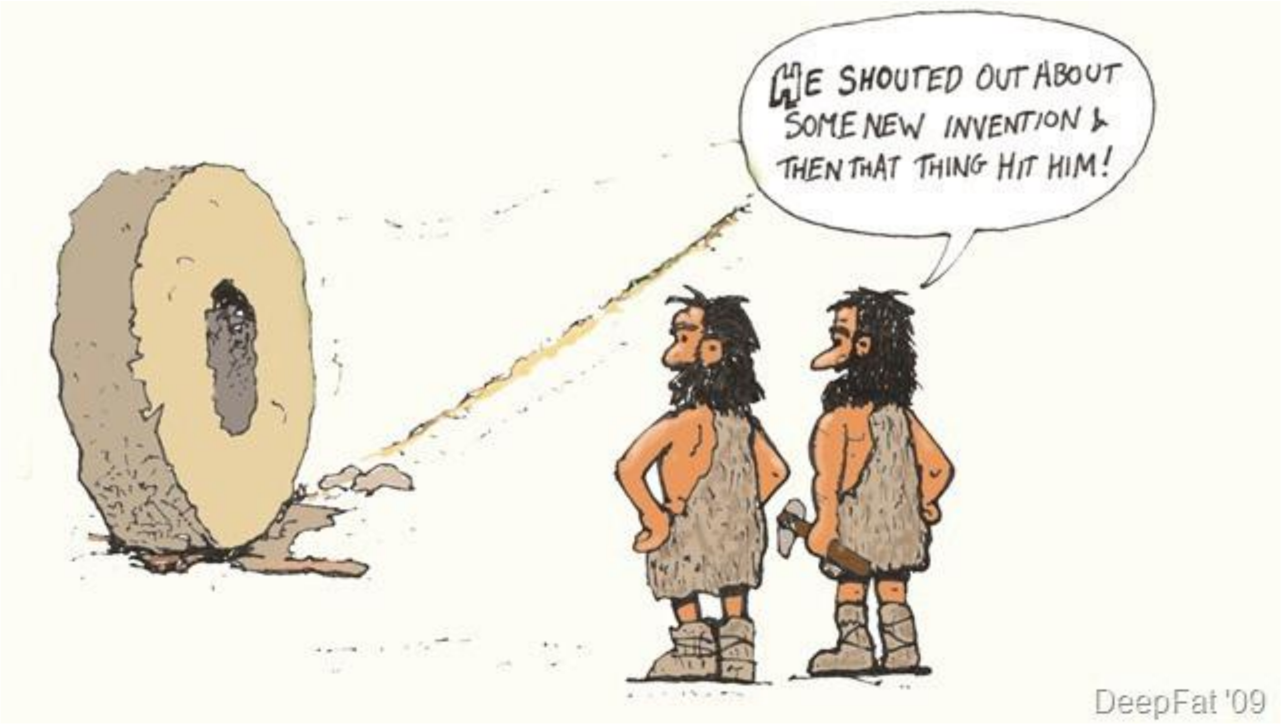
- Last week, we looked at:
  - Encapsulation: How to protect and initialize your objects in C++
  - Debugging as a process, and runtime debugging tools

- Today we’re going to explore some more key concepts of OO programming:
  - Libraries
  - Objects as function parameters
  - Namespaces

  - **Examples of these principles in C++**

How to write scalable code

- **Rule #2: Don't reinvent the wheel.**
  - Integrate high-quality third-party classes into your code.
  - Write reusable code. Assume that others will integrate your classes into their applications.


HE SHOUTED OUT ABOUT SOME NEW INVENTION & THEN THAT THING HIT HIM!
DeepFat '09

Libraries

- Libraries are simply collections of prewritten, precompiled code.
  - Procedural languages provide a sets of functions.
  - Object Oriented languages provide sets of classes.

  - C++ is half and half. ☺

Anyone can create a library…

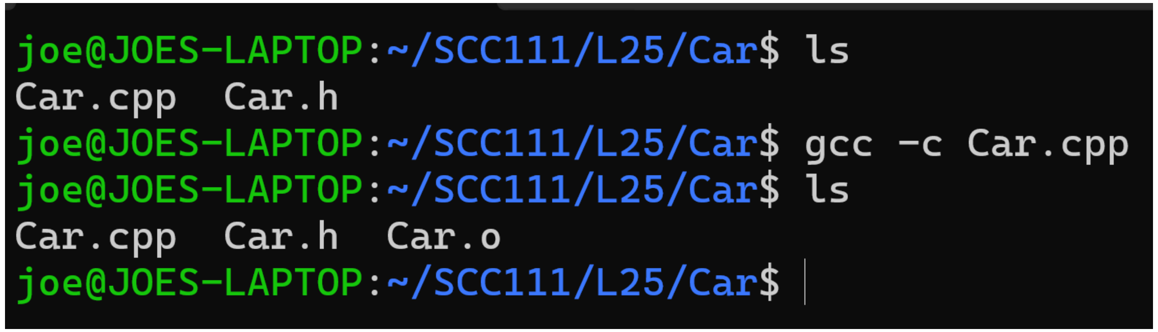
Creating a library

- Creating a C/C++ library is remarkably simple…
  - **Designing a good one is quite hard, but more on that later!**

  - Create your classes, and C functions as normal
  - Separate out your function and class declarations into header files
  - Separate your function/method implementation into .cpp files
  - **Do NOT include a main function.**

  - Compile your code with the **-c** flag.
  - This means "compile only", and does not attempt to create a final, executable program… instead you will get one or more **object** files. (**.o** files)

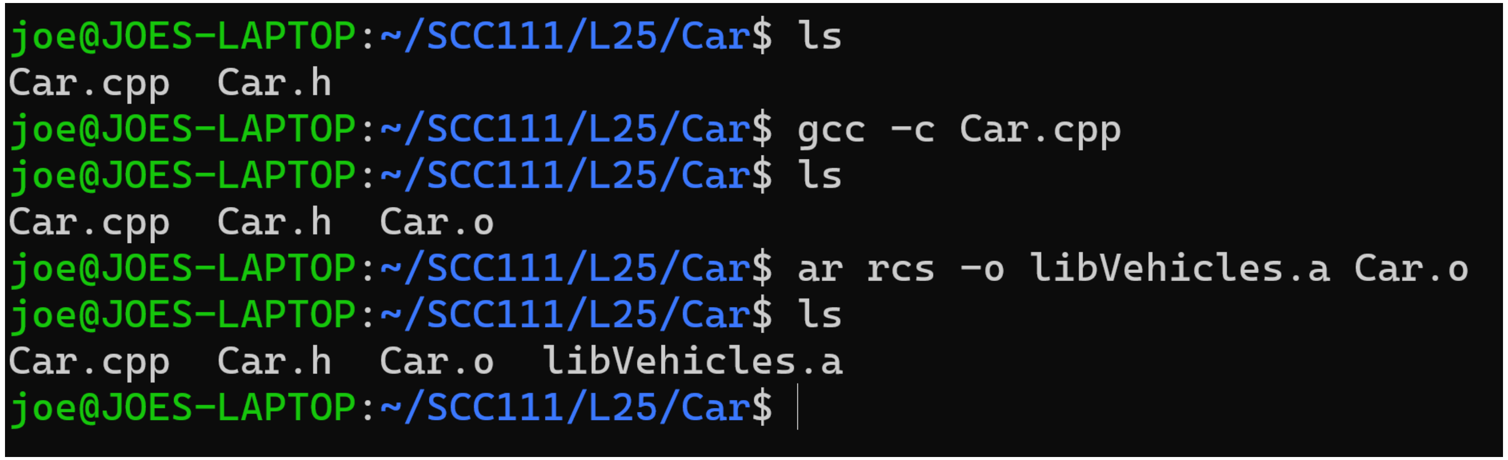
Creating a library: Example



```
joe@JOES-LAPTOP:~/SCC111/L25/Car$ ls
Car.cpp  Car.h
joe@JOES-LAPTOP:~/SCC111/L25/Car$ gcc -c Car.cpp
joe@JOES-LAPTOP:~/SCC111/L25/Car$ ls
Car.cpp  Car.h  Car.o
joe@JOES-LAPTOP:~/SCC111/L25/Car$
```

- **The compiler will produce one .o file for each file compiled like this.**
  - It is likely that our library will be modular however…

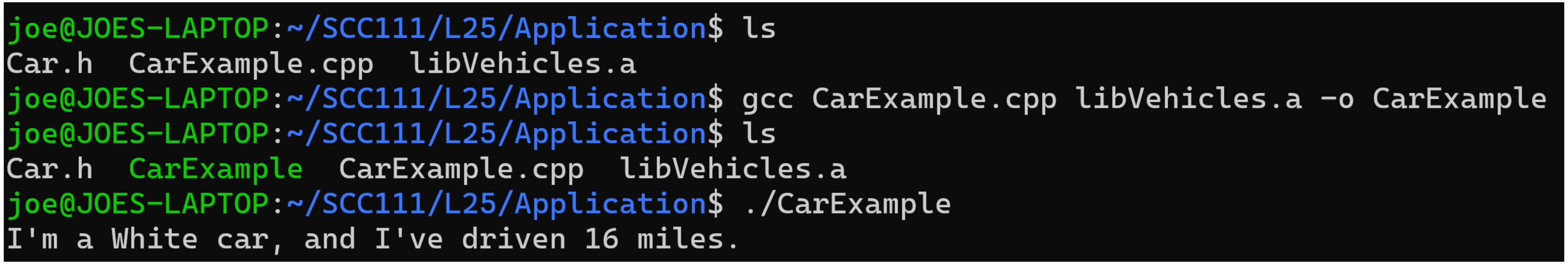
Creating a library: Example 2



```
joe@JOES-LAPTOP:~/SCC111/L25/Car$ ls
Car.cpp  Car.h
joe@JOES-LAPTOP:~/SCC111/L25/Car$ gcc -c Car.cpp
joe@JOES-LAPTOP:~/SCC111/L25/Car$ ls
Car.cpp  Car.h  Car.o
joe@JOES-LAPTOP:~/SCC111/L25/Car$ ar rcs -o libVehicles.a Car.o
joe@JOES-LAPTOP:~/SCC111/L25/Car$ ls
Car.cpp  Car.h  Car.o  libVehicles.a
joe@JOES-LAPTOP:~/SCC111/L25/Car$
```

- **The compiler will produce one .o file for each file compiled like this.**
  - We can group as many .o files as we like into a static library using the ar command… static libraries have **.a** extensions.
  - Just list all the .o files on the command line if you have more than one…

Using a custom library



```
joe@JOES-LAPTOP:~/SCC111/L25/Application$ ls
Car.h  CarExample.cpp  libVehicles.a
joe@JOES-LAPTOP:~/SCC111/L25/Application$ gcc CarExample.cpp libVehicles.a -o CarExample
joe@JOES-LAPTOP:~/SCC111/L25/Application$ ls
Car.h  CarExample  CarExample.cpp  libVehicles.a
joe@JOES-LAPTOP:~/SCC111/L25/Application$ ./CarExample
I'm a White car, and I've driven 16 miles.
```

To use a library, we simply add the precompiled .a file when we compile the application (the thing with a main method in it)

- You can add as many libraries as you like on the command line.
- This provides strong decoupling of library code from application code…
- Different programmers can develop the different parts at different times.

We just built our first LEGO brick!

C++ Standard Libraries

C++ provides many libraries as standard, above those in C!

- **<iostream>** library that deals with basic input and output
  - Imported using *#include <iostream>*
  - It serves as a modern alternative to the *stdio.h* library (*printf* and *scanf*)
- **<string>** library provides the *string* type
- <regex> <queue> <stack> <list> <map> <strstream> <utility>

From C:

- <cstring> provides string related functions (string.h)
- <cmath> provides cos(), sin(), round(), sqrt(), pow() and more
- <cstdlib> provides conversion, memory handling, sorting, search ..

C header:

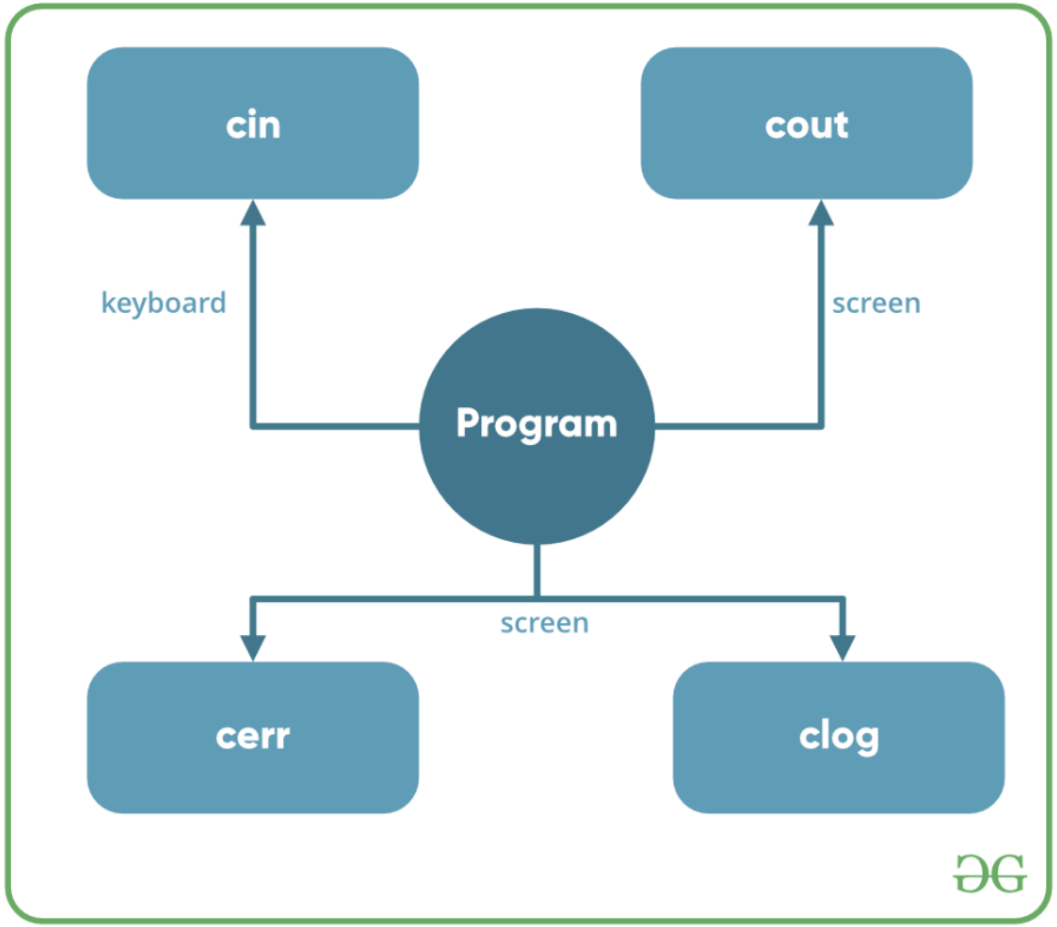
```
#include <stdlib.h>
```

C++ header:

```
#include <cstdlib>
```

iostream: What’s new

- **cin**: Equivalent to scanf for input
- **cout**: Equivalent to printf for output
- **cerr**: Used for printing errors
- **clog:** Employed for logging purposes



iostream Example

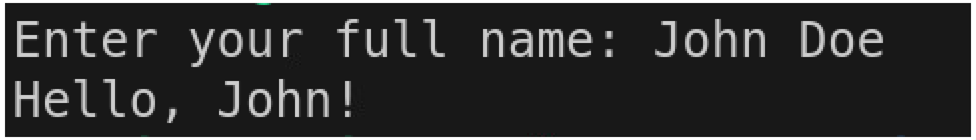
cin, **cout,** and **endl**

```
#include <iostream>

int main() {
    std::cout << "Enter your full name: ";

    char name[50];
    std::cin >> name;

    std::cout << "Hello, " << name << "!" <<
std::endl;
}
```

Equivalent C code:

```
#include <stdio.h>

int main() {
    printf("Enter your full name: ");

    char name[50];
    scanf("%s", name);

    printf("Hello, %s!\n", name);
}
```

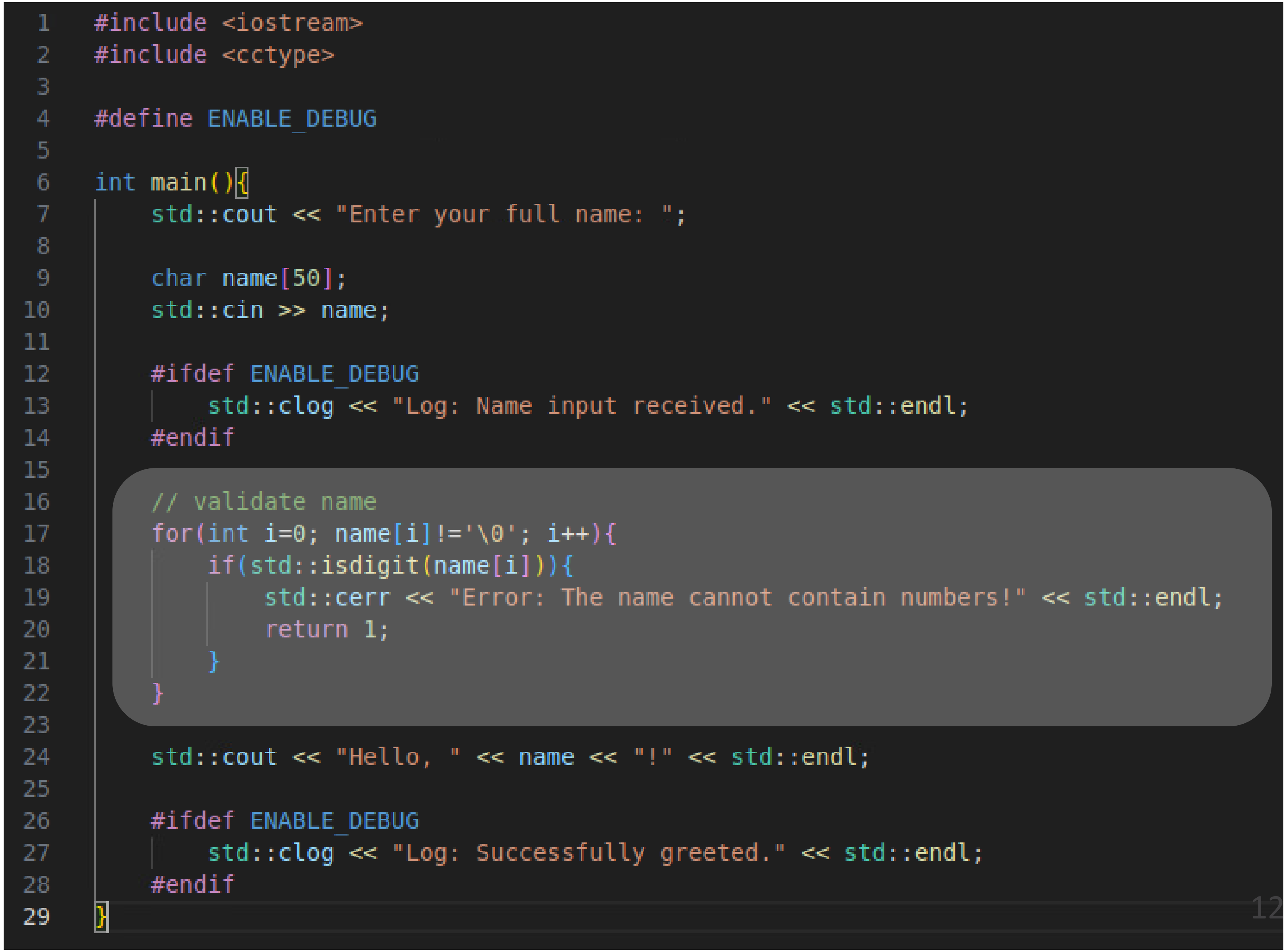
Output:



```
Enter your full name: John Doe
Hello, John!
```

iostream Example (contd.)

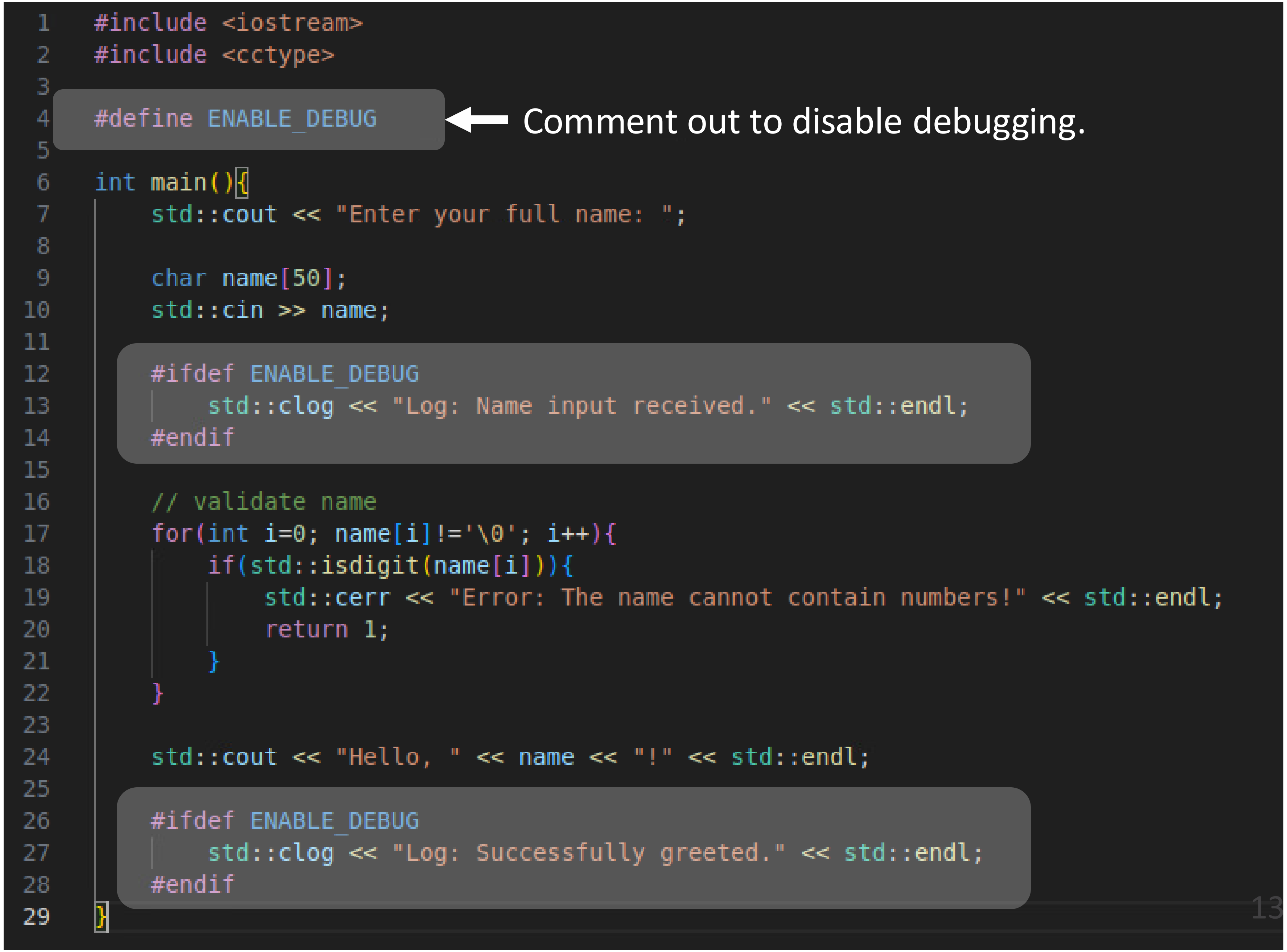
cerr and clog



```cpp
#include <iostream>
#include <cctype>

#define ENABLE_DEBUG

int main(){
    std::cout << "Enter your full name: ";

    char name[50];
    std::cin >> name;

    #ifdef ENABLE_DEBUG
        std::clog << "Log: Name input received." << std::endl;
    #endif

    // validate name
    for(int i=0; name[i]!='\0'; i++){
        if(std::isdigit(name[i])){
            std::cerr << "Error: The name cannot contain numbers!" << std::endl;
            return 1;
        }
    }

    std::cout << "Hello, " << name << "!" << std::endl;

    #ifdef ENABLE_DEBUG
        std::clog << "Log: Successfully greeted." << std::endl;
    #endif
}
```

iostream Example (contd.)

cerr and **clog**



```
#include <iostream>
#include <cctype>

#define ENABLE_DEBUG

int main(){
    std::cout << "Enter your full name: ";

    char name[50];
    std::cin >> name;

    #ifdef ENABLE_DEBUG
        std::clog << "Log: Name input received." << std::endl;
    #endif

    // validate name
    for(int i=0; name[i]!='\0'; i++){
        if(std::isdigit(name[i])){
            std::cerr << "Error: The name cannot contain numbers!" << std::endl;
            return 1;
        }
    }

    std::cout << "Hello, " << name << "!" << std::endl;

    #ifdef ENABLE_DEBUG
        std::clog << "Log: Successfully greeted." << std::endl;
    #endif
}
```

Comment out to disable debugging.

Namespaces

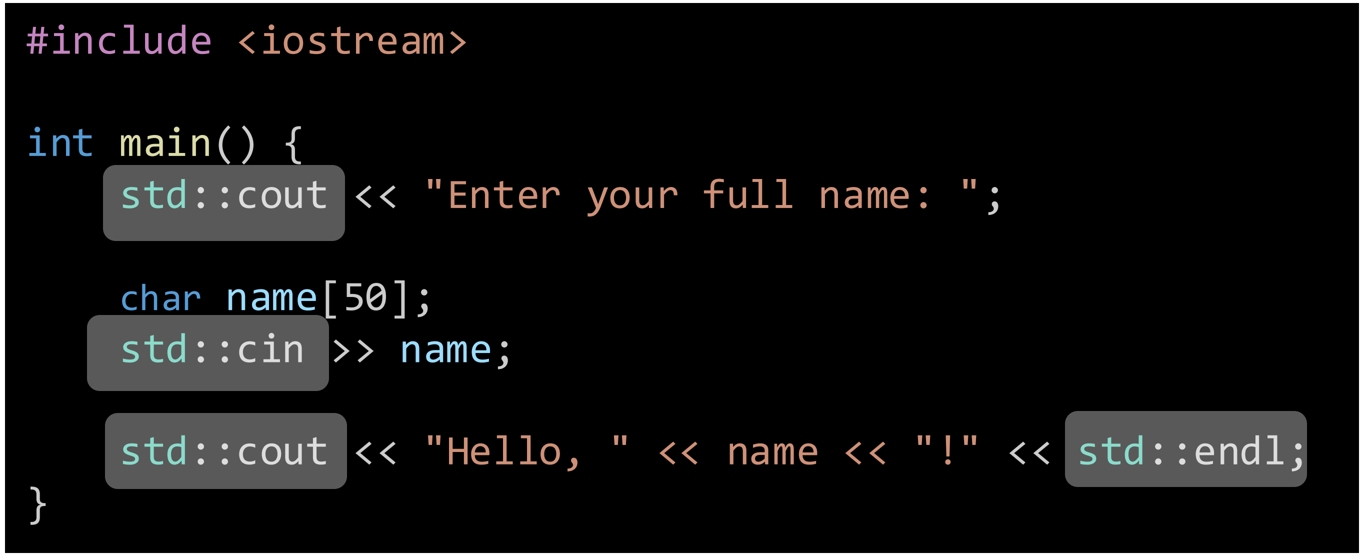
- The C++ standard library uses a namespace call **std**.

  C header:

```
#include <stdlib.h>
```

  C++ header:

```
#include <cstdlib>
```



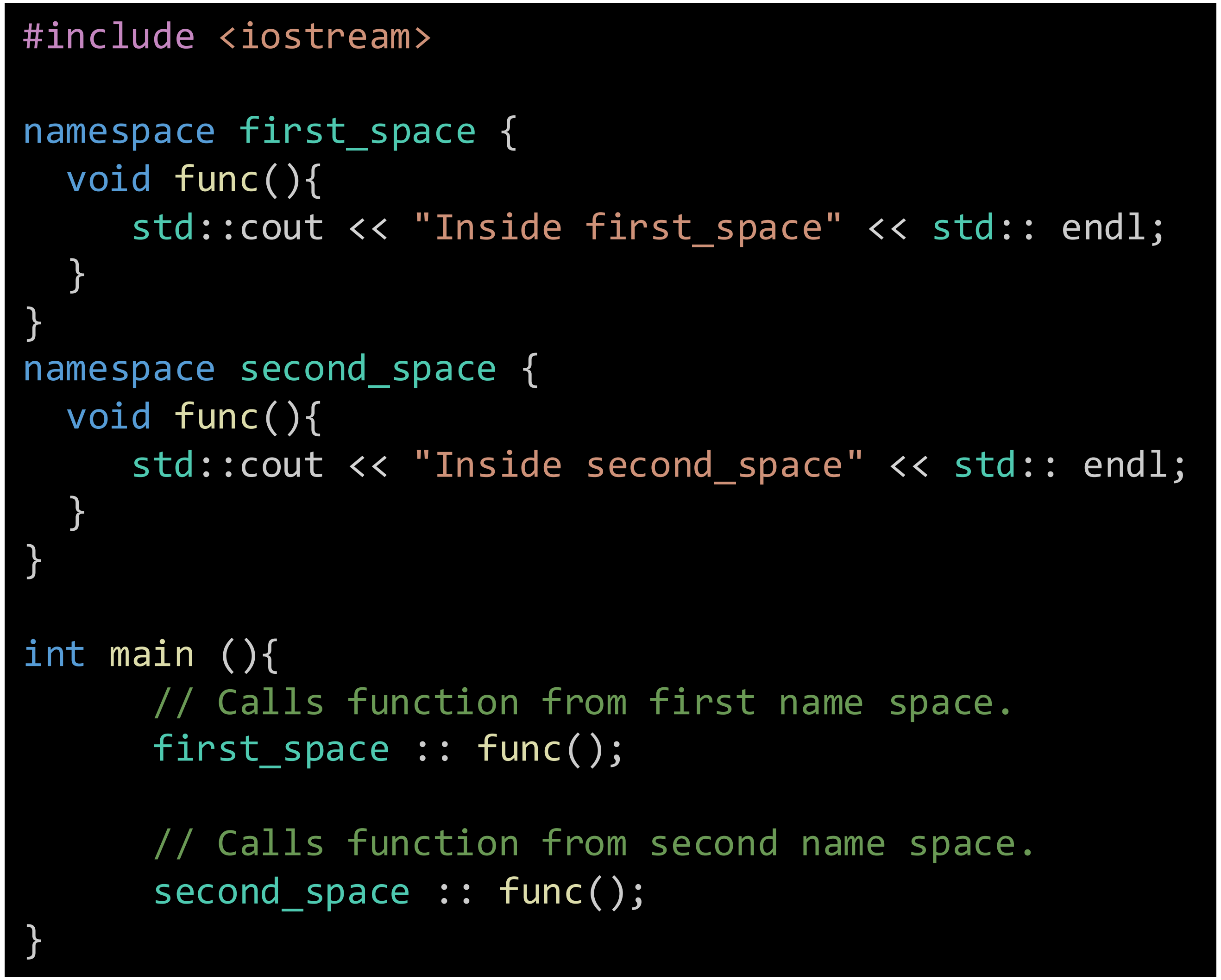
```
#include <iostream>

int main() {
    std::cout << "Enter your full name: ";

    char name[50];
    std::cin >> name;

    std::cout << "Hello, " << name << "!" << std::endl;
}
```

- Namespaces provide **space** where we can **define or declare** identifiers. i.e. variables, methods and classes.
- A namespace is designed to differentiate functions, classes, variables with the **same name** available in **different libraries**
- In essence, a namespace also defines a **scope**.

Namespaces - example

- We can now differentiate two identifiers with the same name based on their namespace

- C++ uses the double colon notation, just like it does for methods in classes.

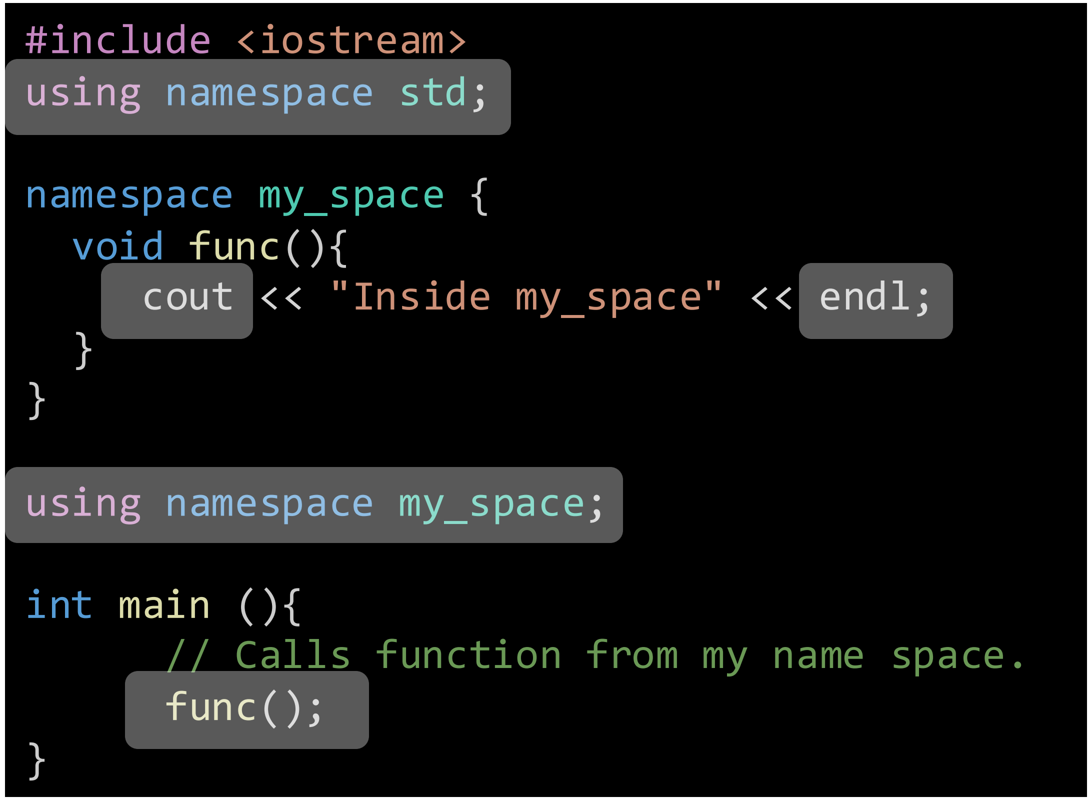


```
#include <iostream>

namespace first_space {
  void func(){
     std::cout << "Inside first_space" << std:: endl;
  }
}
namespace second_space {
  void func(){
     std::cout << "Inside second_space" << std:: endl;
  }
}

int main (){
      // Calls function from first name space.
      first_space :: func();

      // Calls function from second name space.
      second_space :: func();
}
```

Namespaces – example 2

The ‘using’ keyword allows the functions, classes and variables in a given namespace to be used implicitly…

- Take care not to "pollute" namespaces by doing this though.
- Imagine what would happen if you create a namespace and library with a global variable called **i**
- That would be **VERY** annoying to your fellow software developers, right?



```
#include <iostream>
using namespace std;

namespace my_space {
  void func(){
     cout << "Inside my_space" << endl;
  }
}

using namespace my_space;

int main (){
      // Calls function from my name space.
      func();
}
```

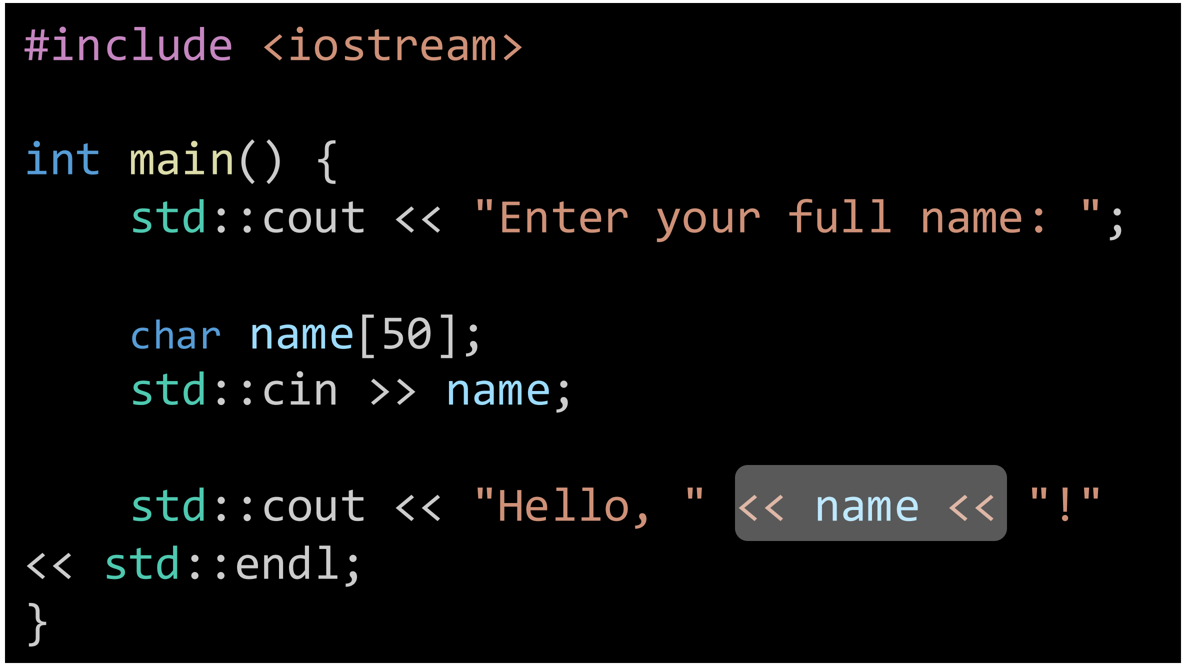
The string Class

In C, we represent strings as a null terminated sequence of chars in memory and use a pointer to the start of that memory to refer to it (char *)…

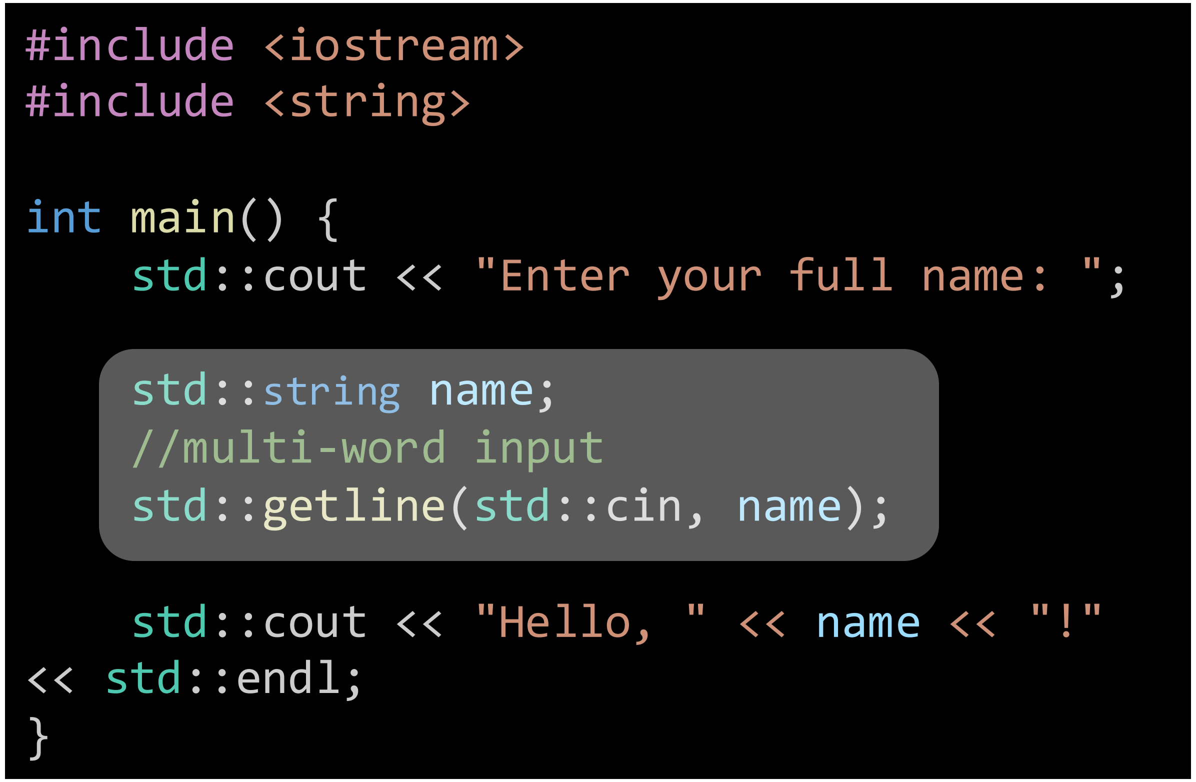
- This can be a bit limiting however…
  - We either used fixed length arrays
  - Or need to deal with dynamic memory allocation (malloc/free)

- C++ wraps this functionality in the **string** class.
- The **string** class definition can be imported using **#include <string>**
- **string** provides access to a wide range of methods useful for string manipulation

string Example

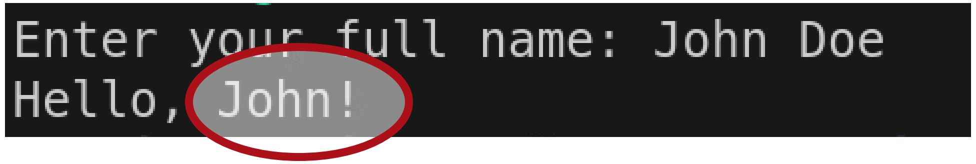


```
#include <iostream>

int main() {
    std::cout << "Enter your full name: ";

    char name[50];
    std::cin >> name;

    std::cout << "Hello, " << name << "!"
<< std::endl;
}
```

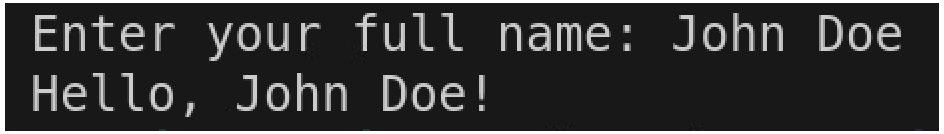


```
#include <iostream>
#include <string>

int main() {
    std::cout << "Enter your full name: ";

    std::string name;
    //multi-word input
    std::getline(std::cin, name);

    std::cout << "Hello, " << name << "!"
<< std::endl;
}
```

Outputs:



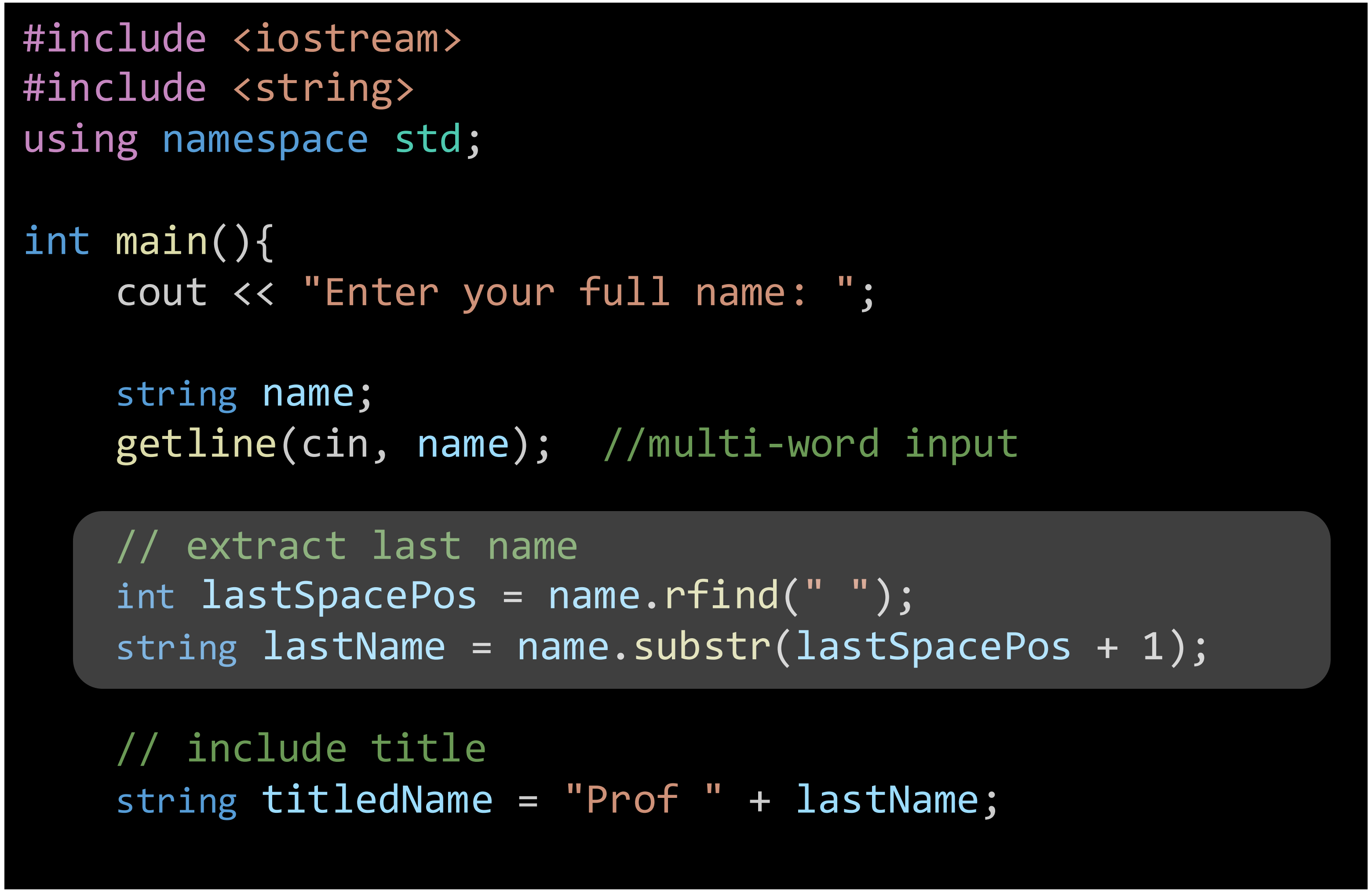
```
Enter your full name: John Doe
Hello, John!
```



```
Enter your full name: John Doe
Hello, John Doe!
```

string Example (contd.)

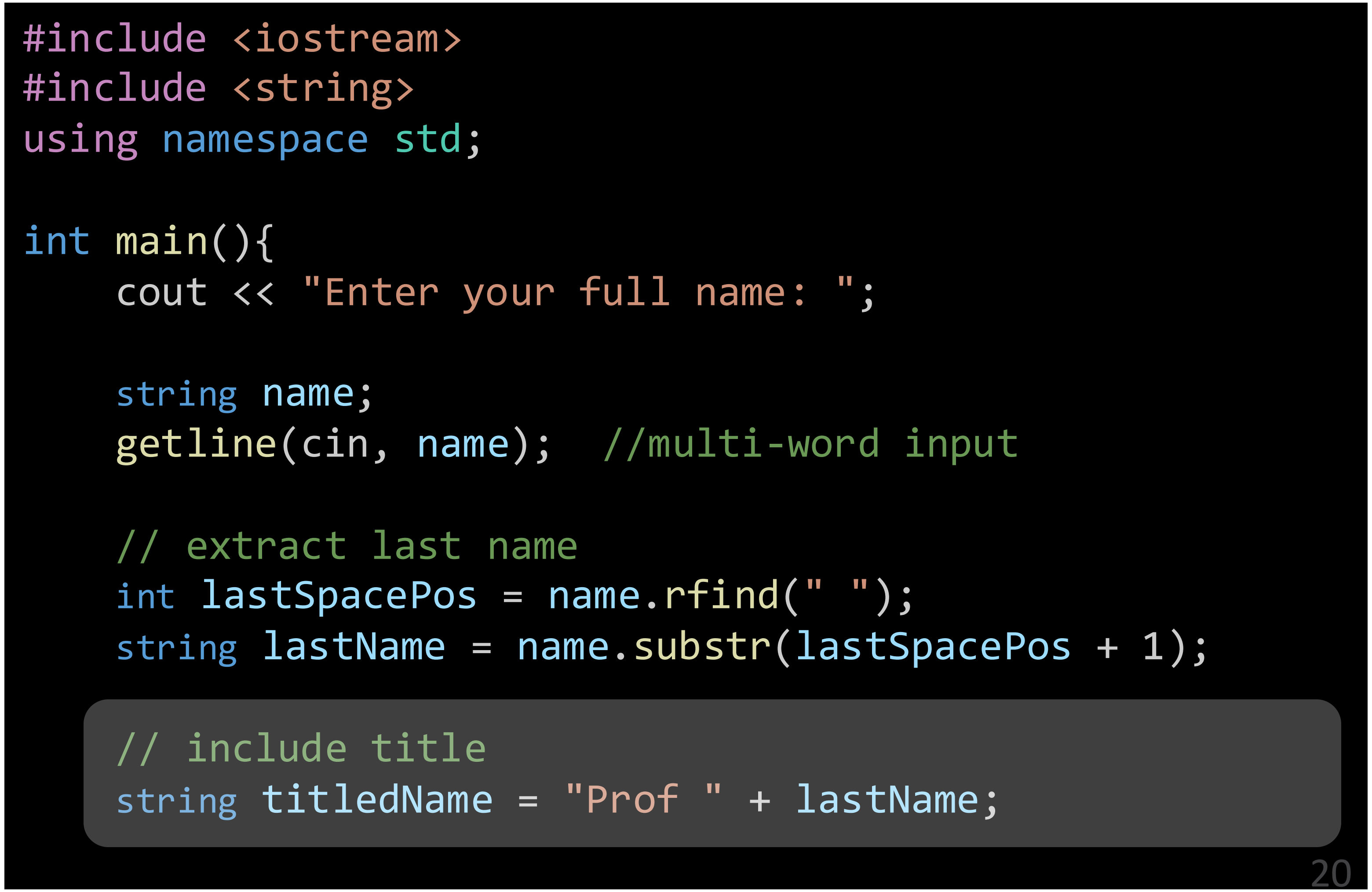
- **find**, **rfind** and **substr**



```
#include <iostream>
#include <string>
using namespace std;

int main(){
    cout << "Enter your full name: ";

    string name;
    getline(cin, name);  //multi-word input

    // extract last name
    int lastSpacePos = name.rfind(" ");
    string lastName = name.substr(lastSpacePos + 1);

    // include title
    string titledName = "Prof " + lastName;
```

string Example (contd.)

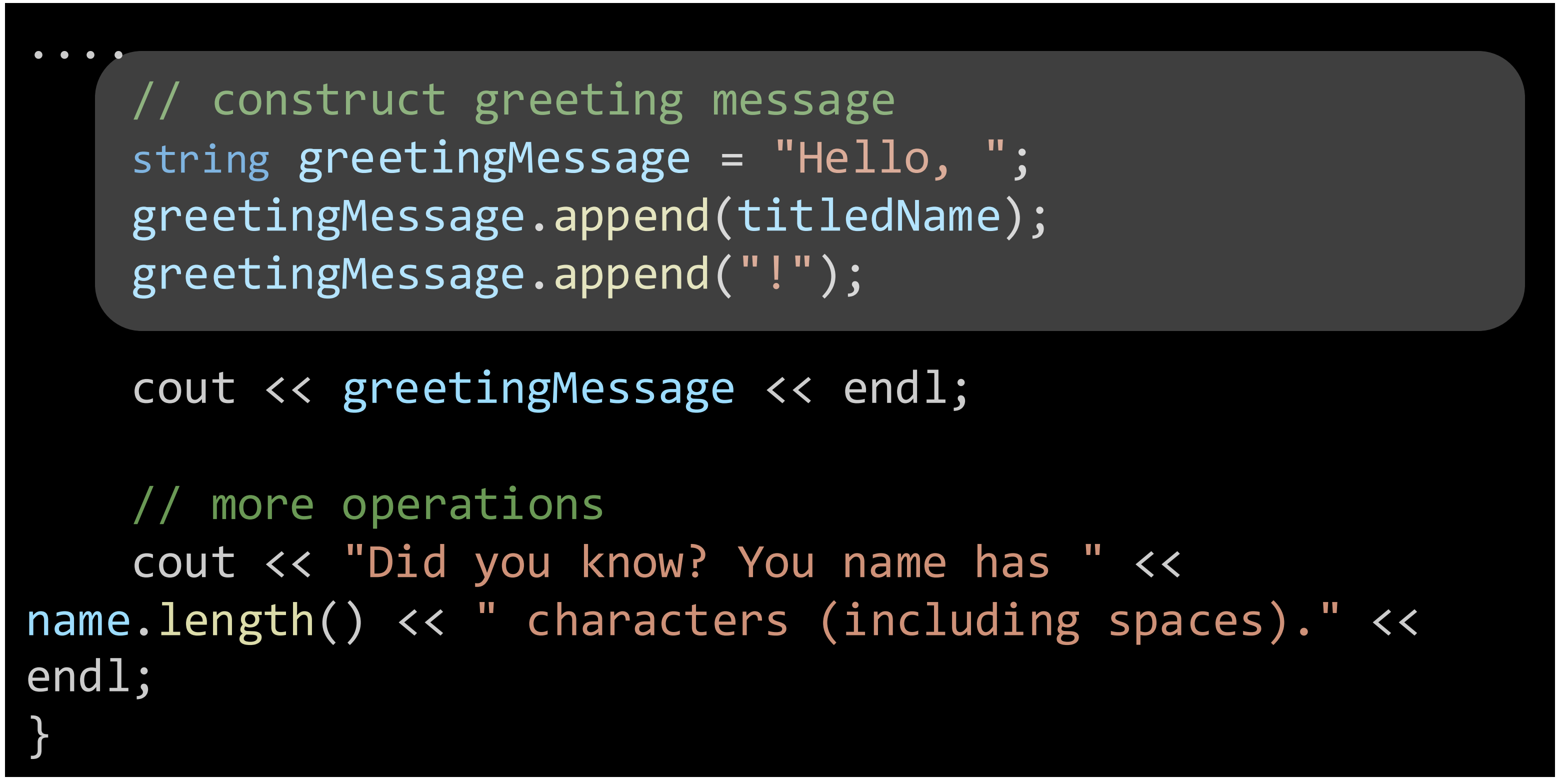
- find, rfind and substr
- **concatenate**



```cpp
#include <iostream>
#include <string>
using namespace std;

int main(){
    cout << "Enter your full name: ";

    string name;
    getline(cin, name);  //multi-word input

    // extract last name
    int lastSpacePos = name.rfind(" ");
    string lastName = name.substr(lastSpacePos + 1);

    // include title
    string titledName = "Prof " + lastName;
```

string Example (contd.)

- **append**



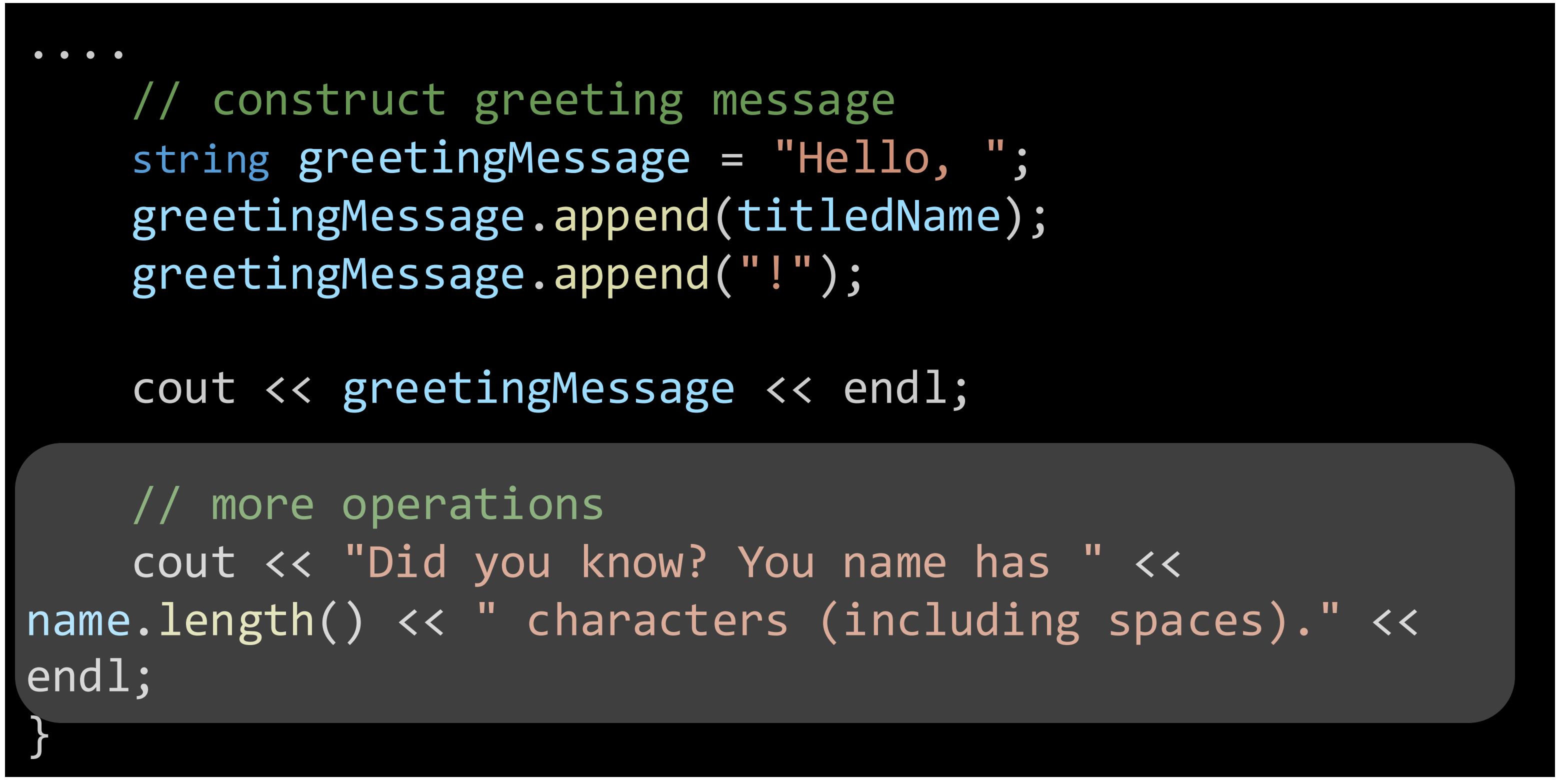
```
....
    // construct greeting message
    string greetingMessage = "Hello, ";
    greetingMessage.append(titledName);
    greetingMessage.append("!");

    cout << greetingMessage << endl;

    // more operations
    cout << "Did you know? You name has " <<
name.length() << " characters (including spaces)." <<
endl;
}
```

string Example (contd.)

- append

- **length**

- Begin, replace, insert, etc.
- Read more: https://cplusplus.com/reference/string/string



```
....
    // construct greeting message
    string greetingMessage = "Hello, ";
    greetingMessage.append(titledName);
    greetingMessage.append("!");

    cout << greetingMessage << endl;

    // more operations
    cout << "Did you know? You name has " <<
name.length() << " characters (including spaces)." <<
endl;
}
```

Summary

- Today we learned about
  - What software libraries really are, and how to create them
  - Some standard C++ libraries and how to use them in practice
  - What namespaces are, and how to use them
  - Strings (*at last!*)