C/C++

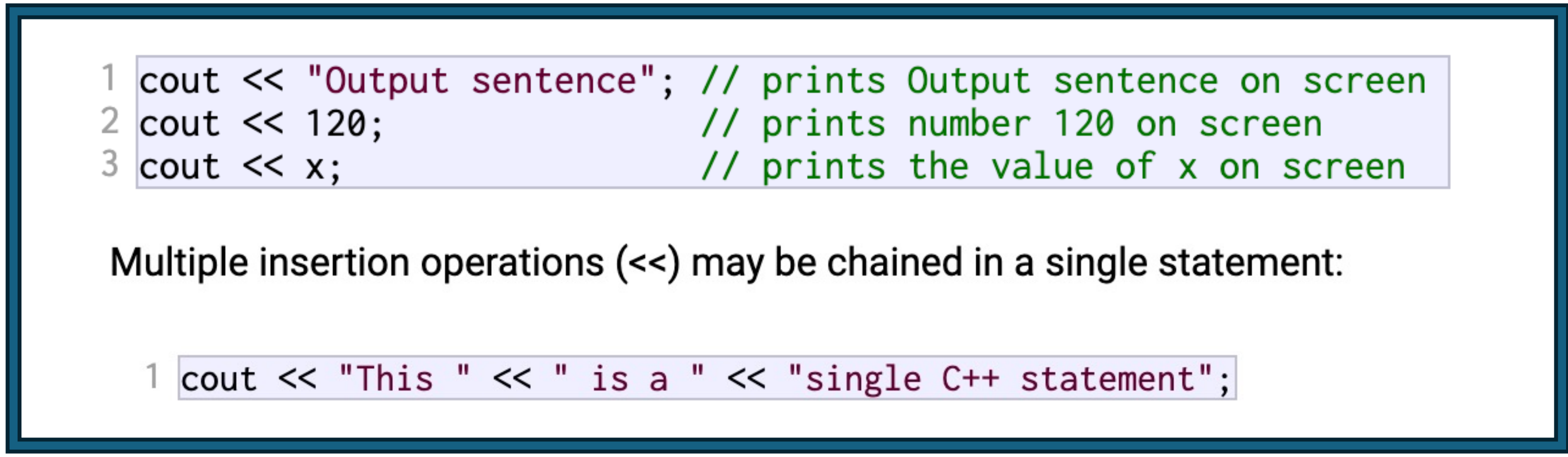
Part 1

Key Concepts

- C++ language syntax
- Writing, compiling and debugging code
- Interactive development environment (IDE)
- C++ classes
- Implementation v. header files
- Public v. private elements of a class
- Linking to external libraries to access helper routines
- Namespaces

C++ Program Structure

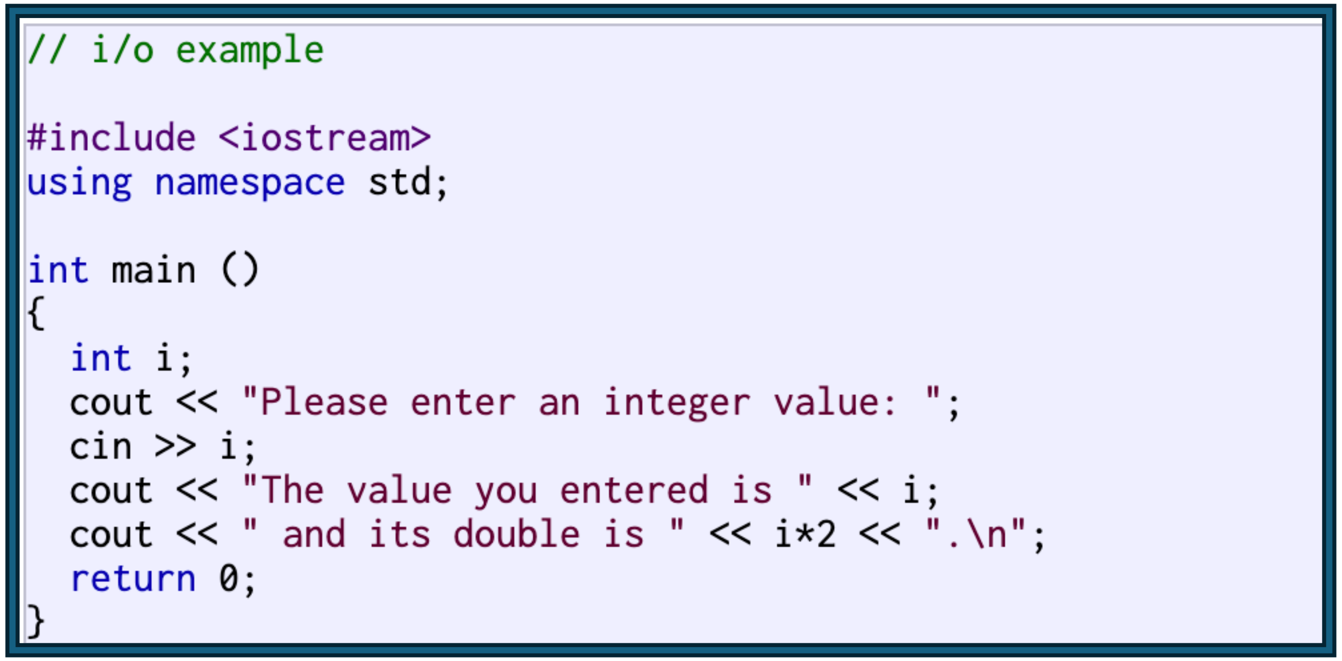
C++: Simple Program



```cpp
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the value of x on screen
```

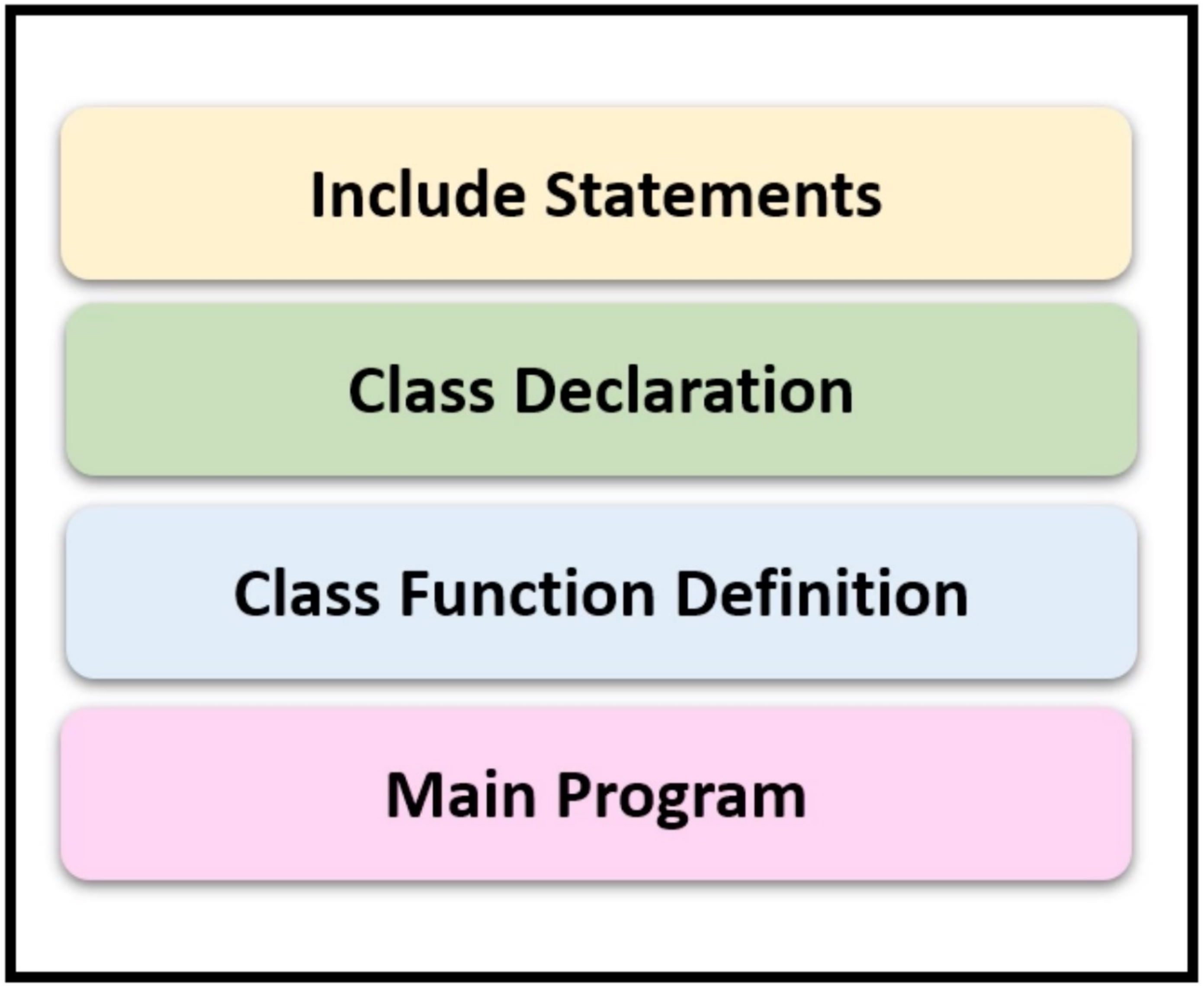
Multiple insertion operations (<<) may be chained in a single statement:

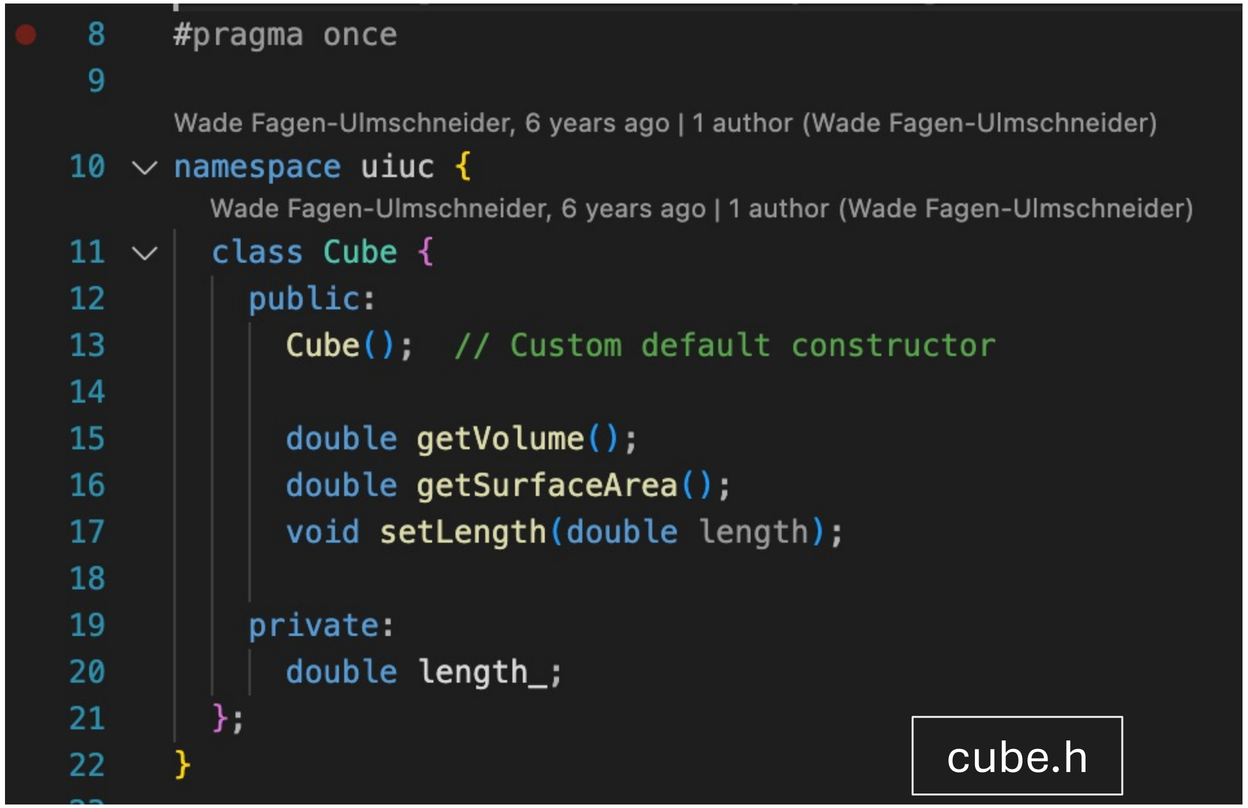
```cpp
cout << "This " << " is a " << "single C++ statement";
```



```cpp
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

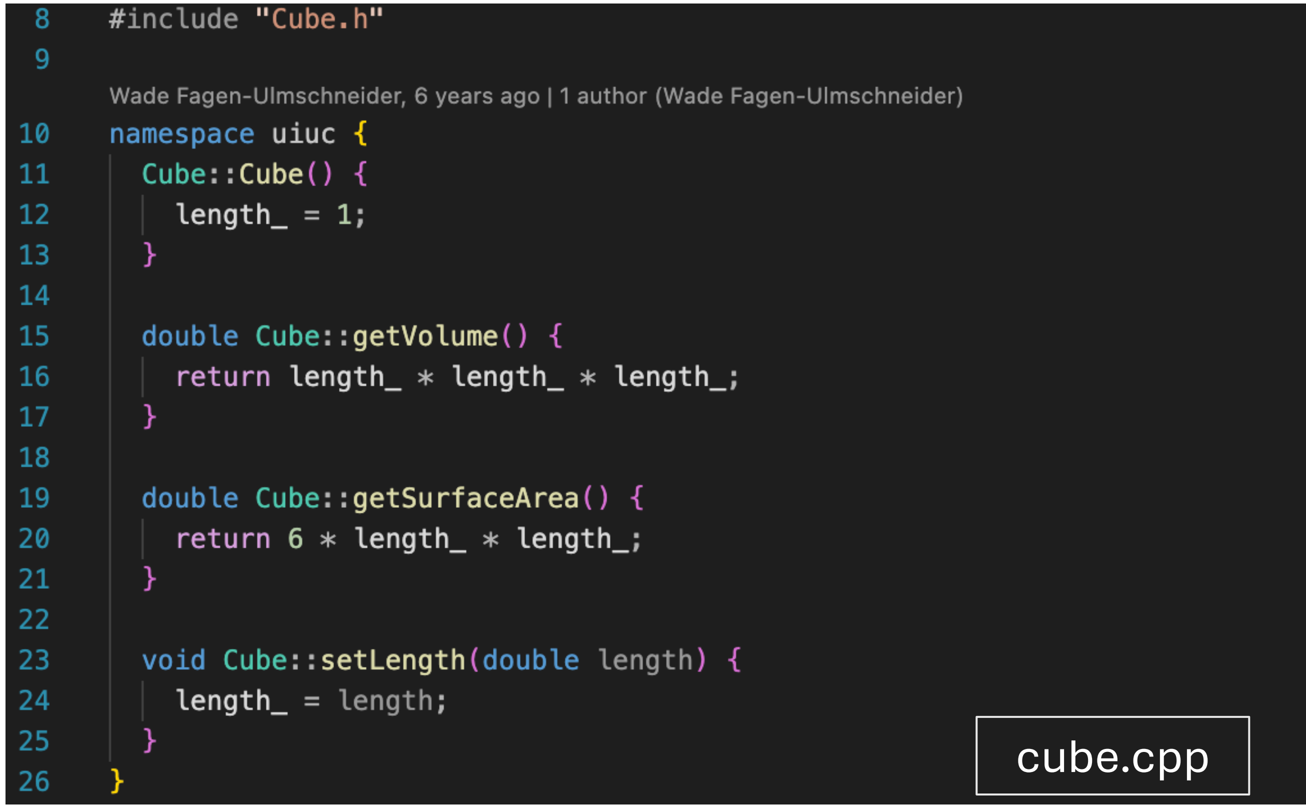
Structure of C++ Program





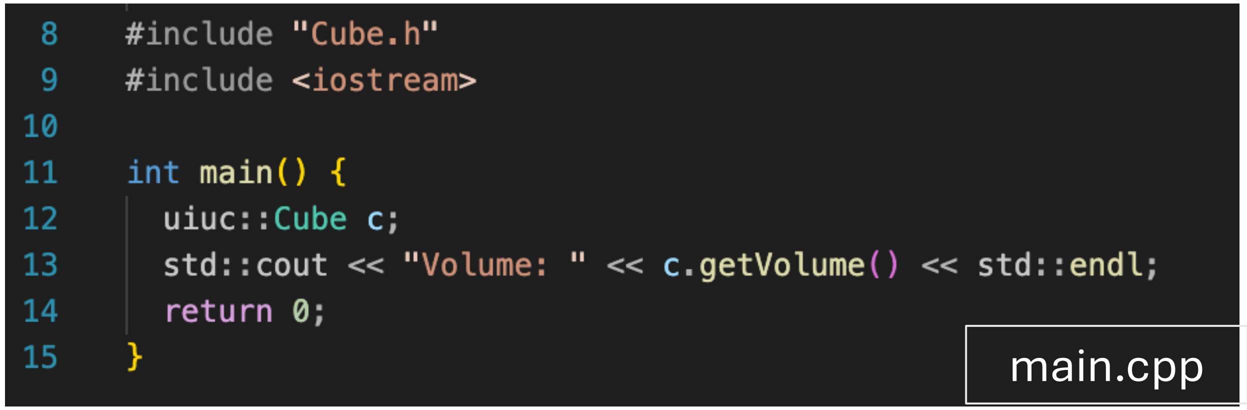
```cpp
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}
```

cube.h



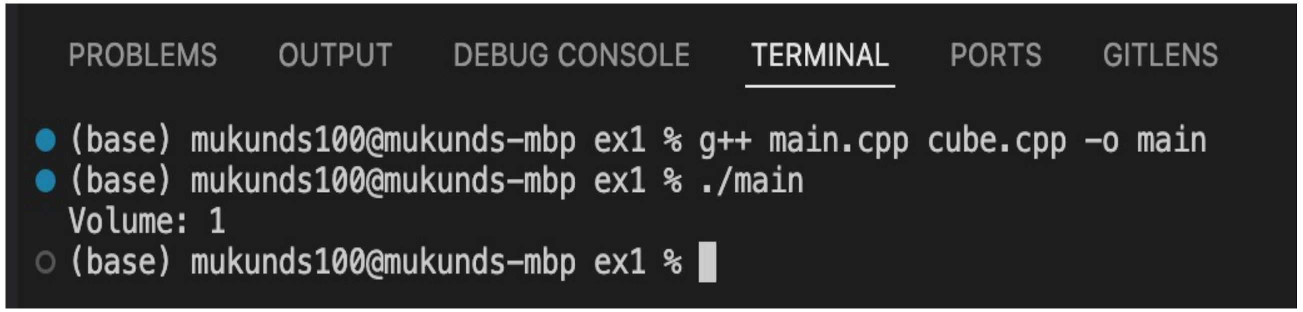
```cpp
#include "Cube.h"

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Cube::Cube() {
    length_ = 1;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

cube.cpp



```cpp
#include "Cube.h"
#include <iostream>

int main() {
  uiuc::Cube c;
  std::cout << "Volume: " << c.getVolume() << std::endl;
  return 0;
}
```

main.cpp



```
(base) mukunds100@mukunds-mbp ex1 % g++ main.cpp cube.cpp -o main
(base) mukunds100@mukunds-mbp ex1 % ./main
Volume: 1
(base) mukunds100@mukunds-mbp ex1 %
```

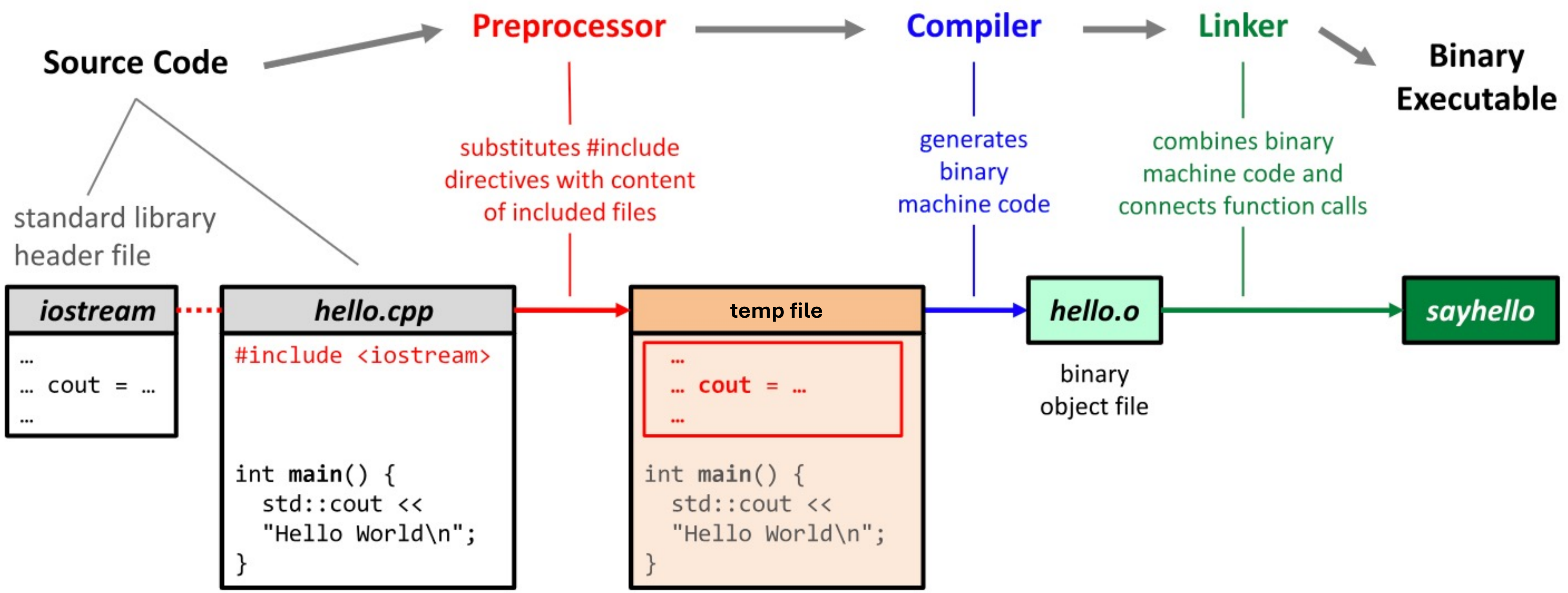
Java vs C++ Architecture

```
 Java Architecture
+---------------------+
|                     |
|   Source Code       |
| (e.g., Hello.java)  |
|                     |
+---------------------+
          |
          |
          v
+---------------------+
|                     |
|   Java compiler     |
|                     |
+---------------------+
          |
          |
          v
+----------------------+
|                      |
|      Byte code       |
| (e.g., Hello.class)  |
|                      |
+----------------------+
          |
          |
          v
+----------------------+
|Java Virtual Machine  |
|        (JVM)         |
|  +---------------+   |
|  |      OS       |   |
|  |  +---------+  |   |
|  |  |   CPU   |  |   |
|  |  +---------+  |   |
|  +---------------+   |
+----------------------+
```

```
C++ Language Architecture
+---------------------+
|                     |
|   Source Code       |
| (e.g., hello.cpp)   |
|                     |
+---------------------+
          |
          |
          v
+---------------------+
|      Compiler       |
+---------------------+
          |
          |
          v
+---------------------+
|     Object code     |
|  (e.g., hello.obj)  |
+---------------------+
          |
          v
+---------------------+
|       Linker        |
+---------------------+
          |
          v
+---------------------+
|     Machine code    |
+---------------------+
          |
          v
  +-----------------+
  |       OS        |
  | +------------+  |
  | |   CPU      |  |
  | +------------+  |
  +-----------------+
```

C++: Compilation process

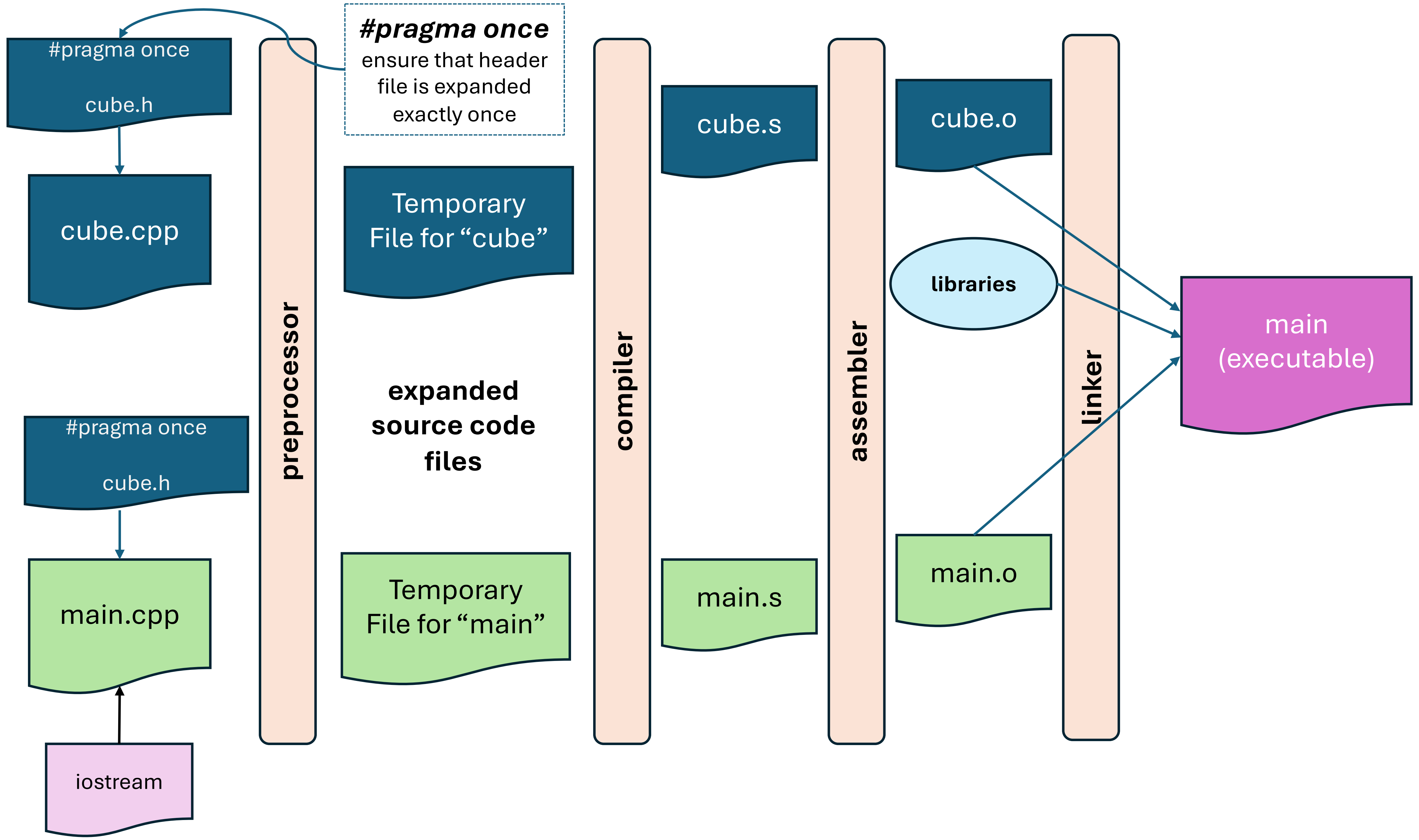
```
#include <iostream>

int main( ) {
      std::cout << “Hello World\n”;
      return 0;
}
```


Source Code
Preprocessor
Compiler
Linker
Binary Executable
standard library header file
substitutes #include directives with content of included files
generates binary machine code
combines binary machine code and connects function calls
iostream
hello.cpp
temp file
hello.o
sayhello
binary object file

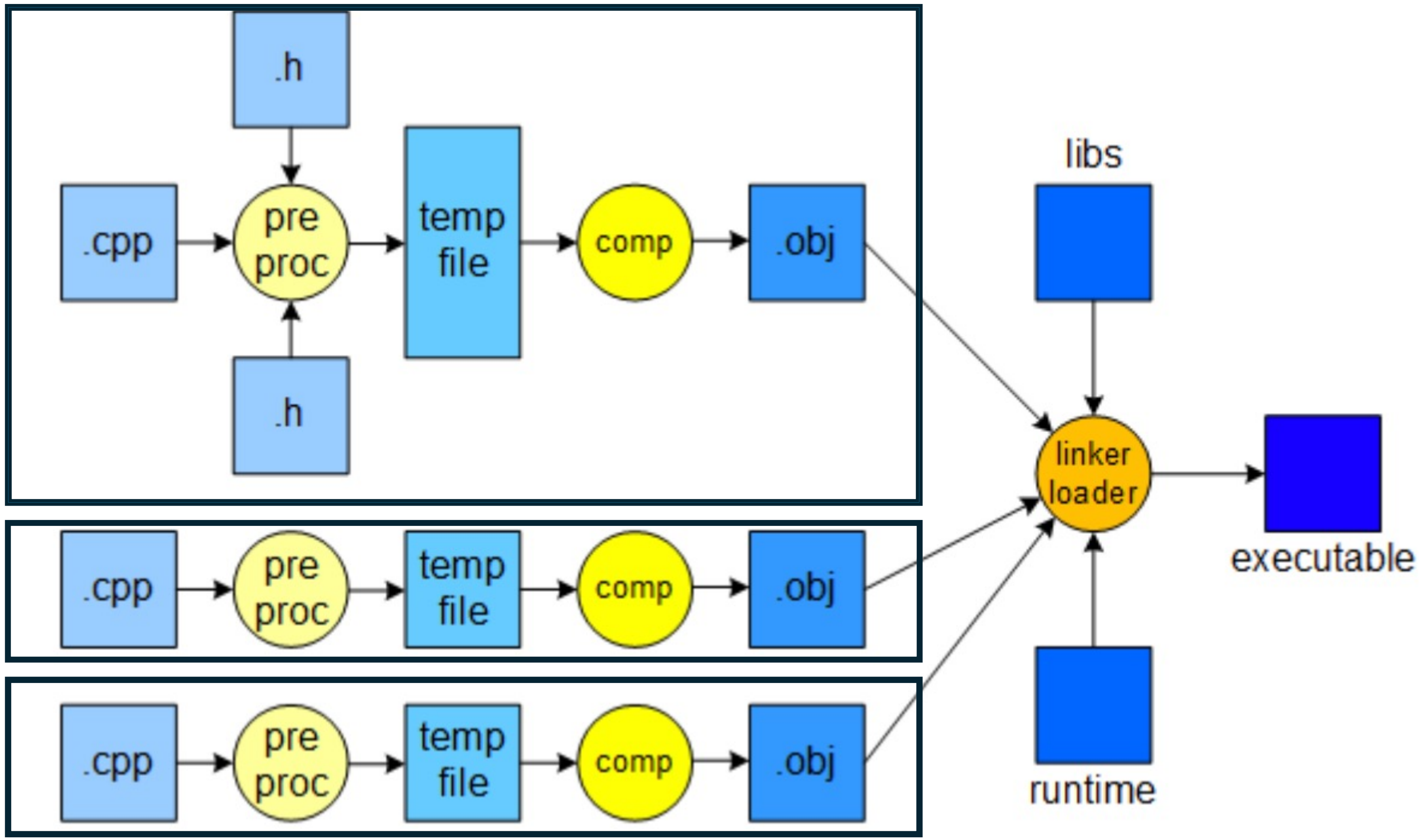
```
iostream:
…
… cout = …
…
```

```
hello.cpp:
#include <iostream>


int main() {
  std::cout <<
  "Hello World\n";
}
```

```
temp file:
…
… cout = …
…
int main() {
  std::cout <<
  "Hello World\n";
}
```


#pragma once
cube.h
cube.cpp
#pragma once
cube.h
main.cpp
iostream
preprocessor
#pragma once
ensure that header file is expanded exactly once
Temporary File for "cube"
expanded source code files
Temporary File for "main"
compiler
cube.s
main.s
assembler
cube.o
libraries
main.o
linker
main (executable)

C++: Compilation process

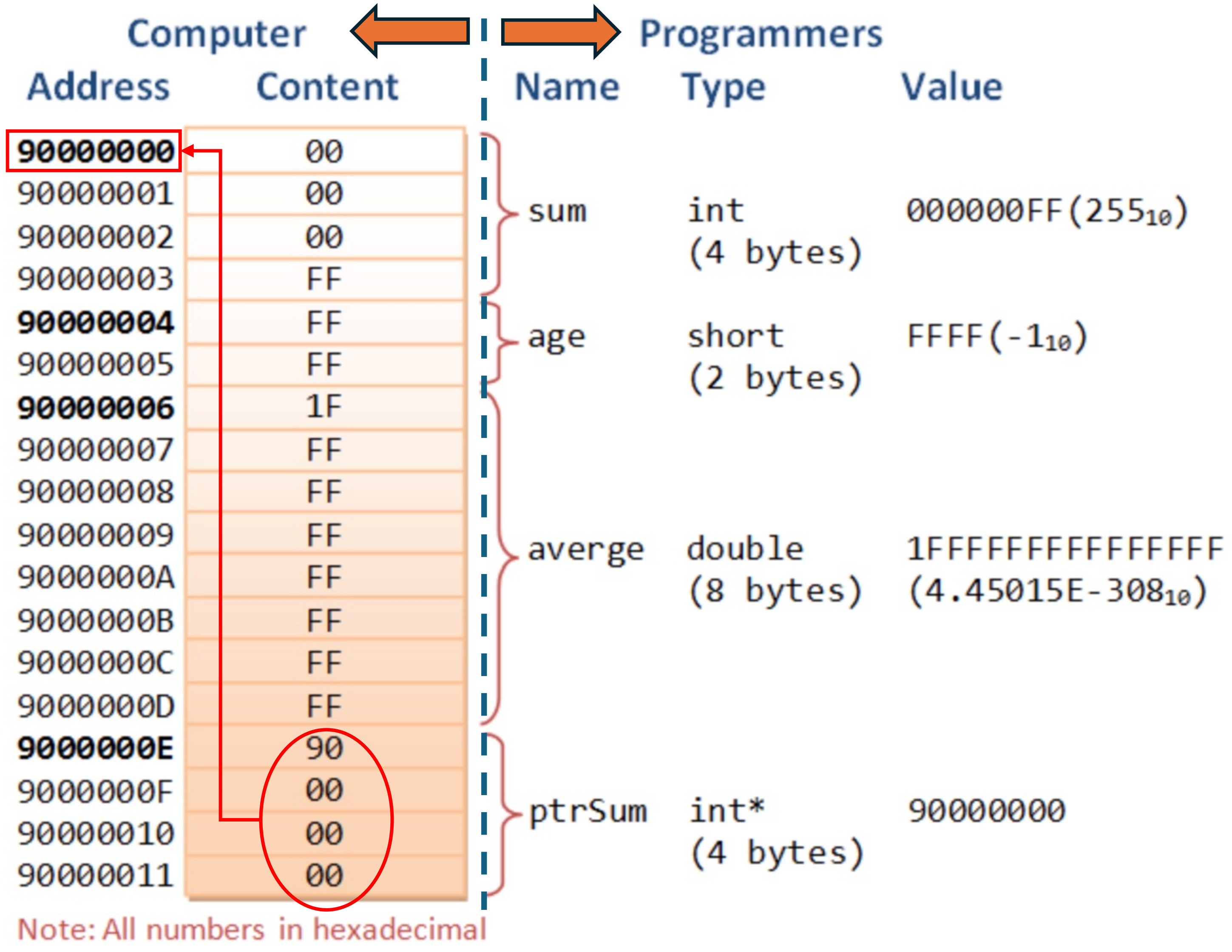


Variables

Variables & Memory

```
#include <iostream>
int main() {
    int sum = 255;
    short age = -1;
    double average = 0x1FFFFFFFFFFFFFFF;
    int* ptrSum = 0x90000000;

    cout << "address of sum: " << &sum <<"\n";
    cout << "address of age: " << &age  <<"\n";
    cout << "address of ptrSum: << &ptrSum;
    return 0;
}
```

- Memory address cannot be programmatically determined in **Java**, only in **C/C++** .



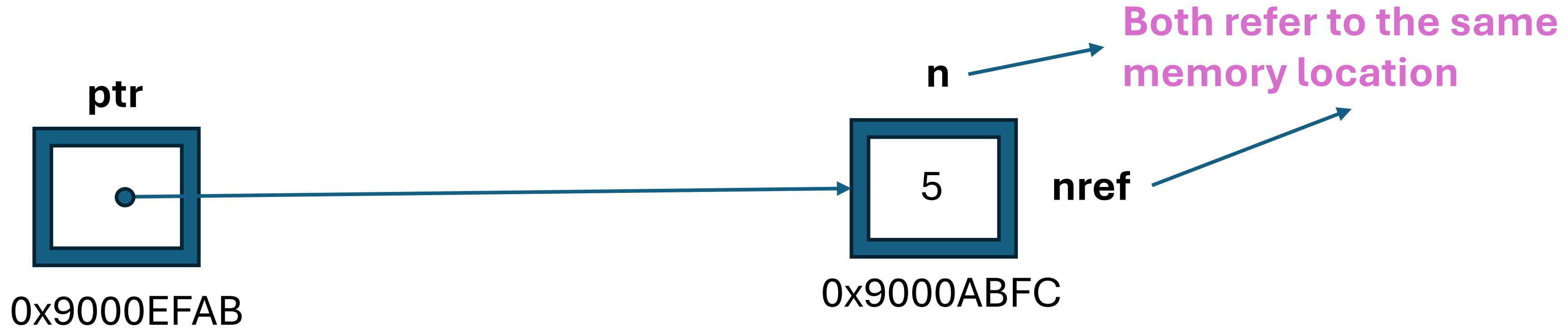
Variables: Pointers and References

- **Pointer** is a variable whose **value** is a **memory address.**

int n = 5;

*int * ptr = &n;*

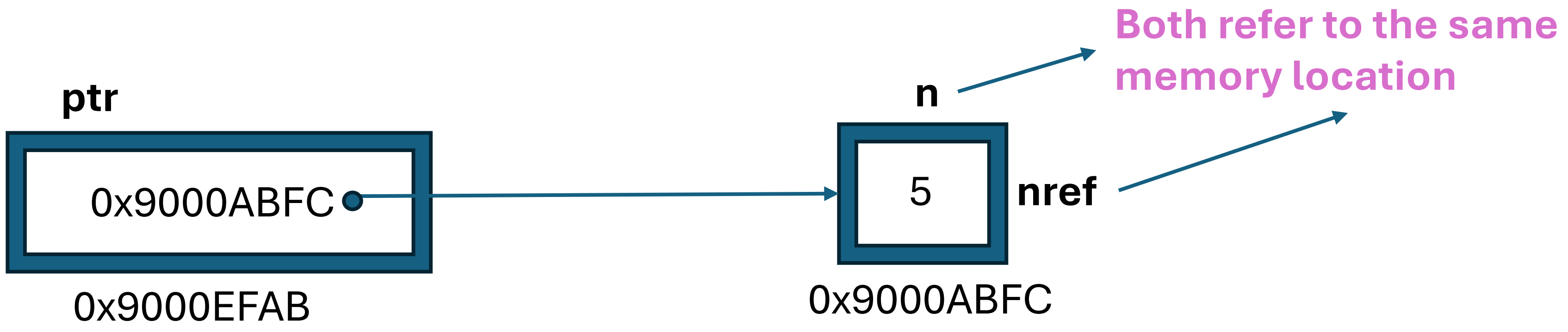
int & nref = n;



- **Reference is alias for a variable**
- Read reference and pointers from right to left
  - int * p is read as “p is pointer to integer variable”
    - pointer means “holds a memory address”
  - int & nref is read as “nref is reference to integer variable”

Variables: Pointers and References

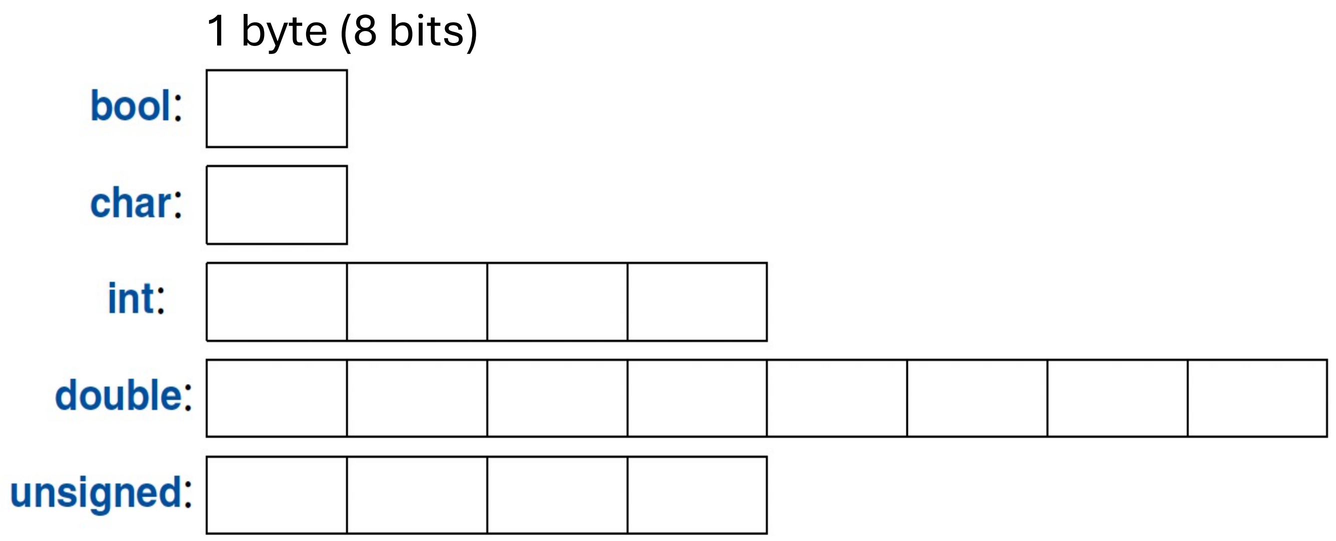
```
int  n = 5;

int * ptr = &n;

int & nref = n;
```



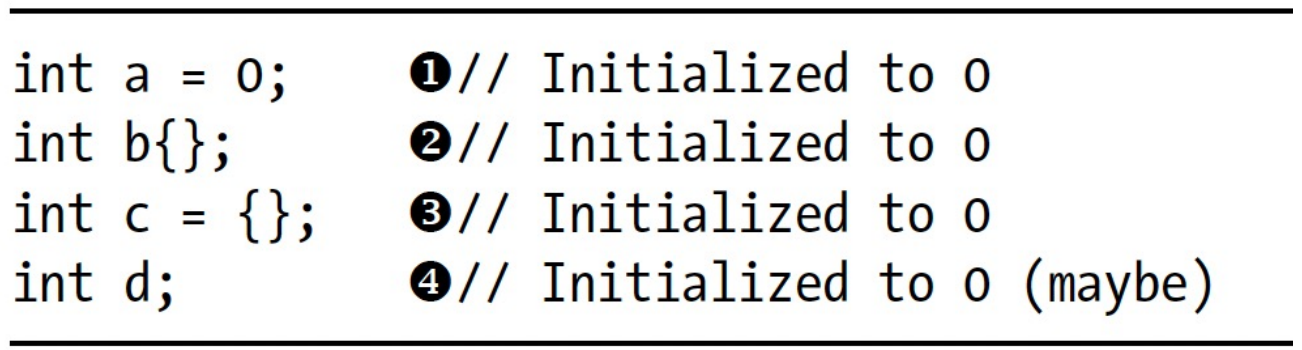
- cout << n << "\n; $\Longrightarrow$ print the contents of variable ***n*** $\Longrightarrow$ ***5***
- cout << *ptr << "\n; $\Longrightarrow$ print the **contents of variable *whose address it holds*** $\Longrightarrow$ ***5***
- cout << ptr << "\n"; $\Longrightarrow$ print the contents of variable ***ptr*** $\Longrightarrow$ **0x9000ABFC**
- cout << &ptr<< "\n"; $\Longrightarrow$ print the address variable ***ptr*** $\Longrightarrow$ ***?????***
- cout << &n << "\n"; $\Longrightarrow$?????

C++: Fundamental Types

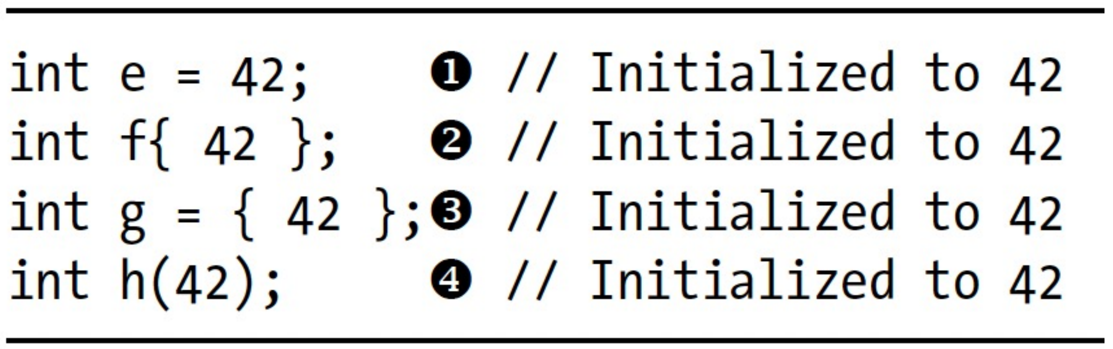
bool	*// Boolean, possible values are true and false*
char	*// character, for example, 'a', 'z', and '9'*
int	*// integer, for example, -273, 42, and 1066*
double	*// double-precision floating-point number, for example, -273.15, 3.14, and 6.626e-34*
unsigned	*// non-negative integer, for example, 0, 1, and 999 (use for bitwise logical operations)*



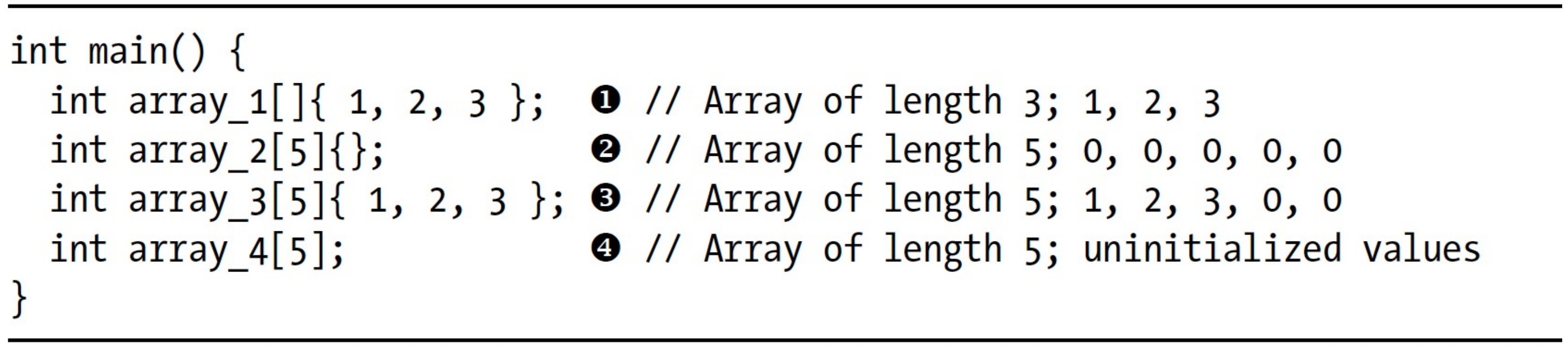
Variable Initialization



```
int a = 0;      ❶// Initialized to 0
int b{};        ❷// Initialized to 0
int c = {};     ❸// Initialized to 0
int d;          ❹// Initialized to 0 (maybe)
```



```
int e = 42;      ❶ // Initialized to 42
int f{ 42 };     ❷ // Initialized to 42
int g = { 42 };❸ // Initialized to 42
int h(42);       ❹ // Initialized to 42
```

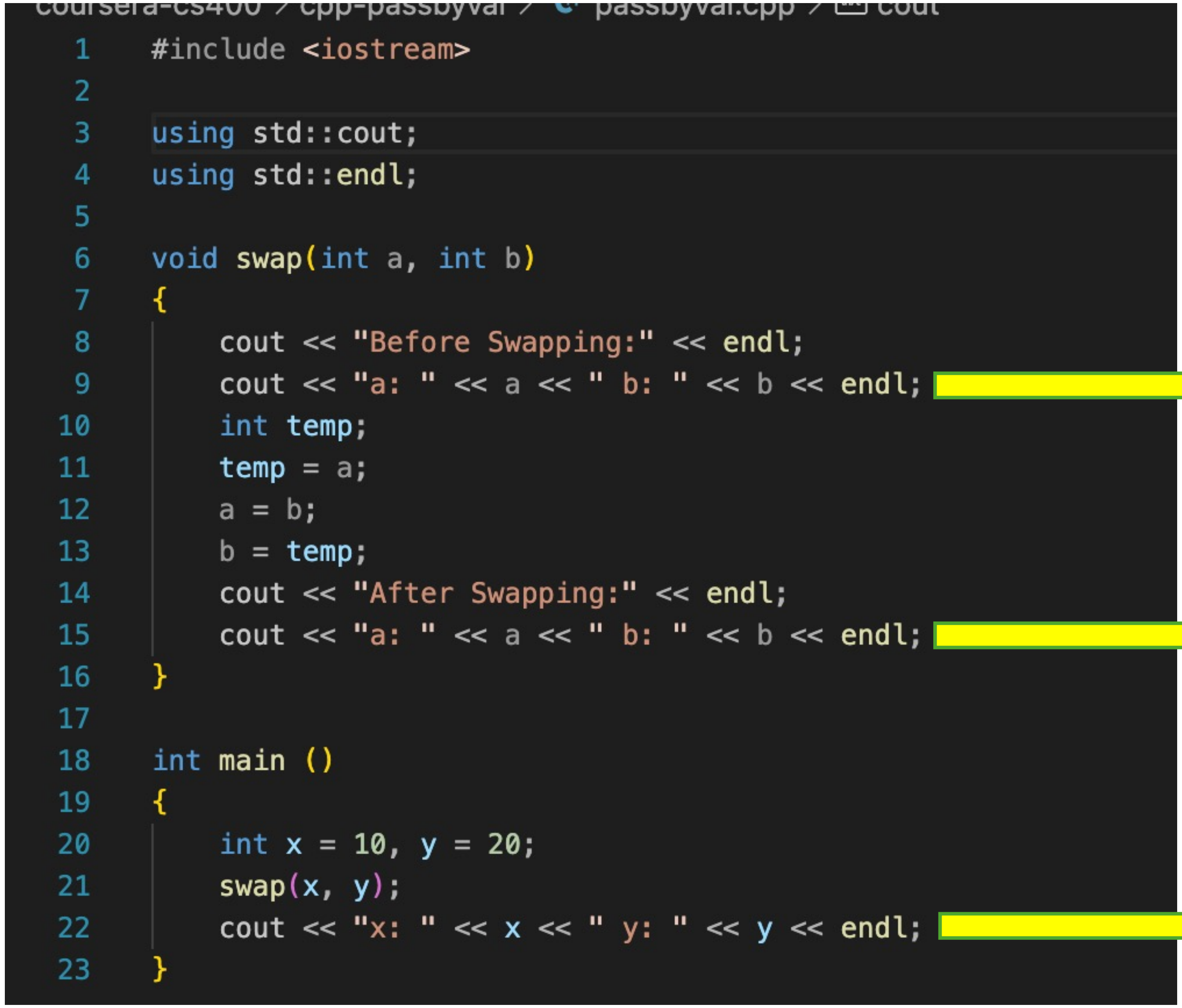


```
int main() {
  int array_1[]{ 1, 2, 3 };  ❶ // Array of length 3; 1, 2, 3
  int array_2[5]{};          ❷ // Array of length 5; 0, 0, 0, 0, 0
  int array_3[5]{ 1, 2, 3 }; ❸ // Array of length 5; 1, 2, 3, 0, 0
  int array_4[5];            ❹ // Array of length 5; uninitialized values
}
```

In C++, **uninitialized variables** have **garbage** data and causes **bugs**.

C++ Functions

C++: Parameter Passing

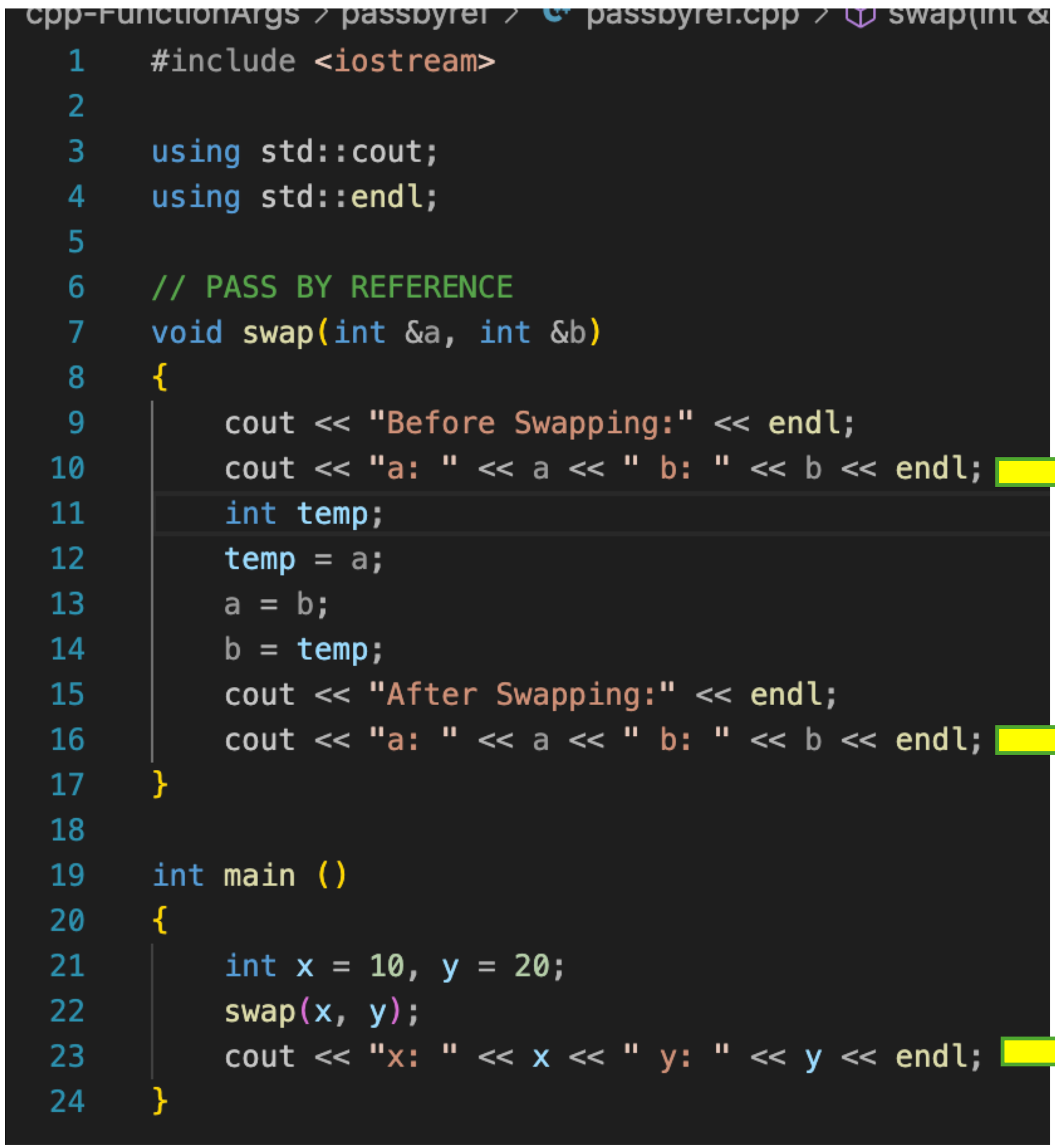


coursera-cs400 > cpp-passbyval > passbyval.cpp > cout

```cpp
#include <iostream>

using std::cout;
using std::endl;

void swap(int a, int b)
{
    cout << "Before Swapping:" << endl;
    cout << "a: " << a << " b: " << b << endl;
    int temp;
    temp = a;
    a = b;
    b = temp;
    cout << "After Swapping:" << endl;
    cout << "a: " << a << " b: " << b << endl;
}

int main ()
{
    int x = 10, y = 20;
    swap(x, y);
    cout << "x: " << x << " y: " << y << endl;
}
```

Pass-by-value

C++: Parameter Passing

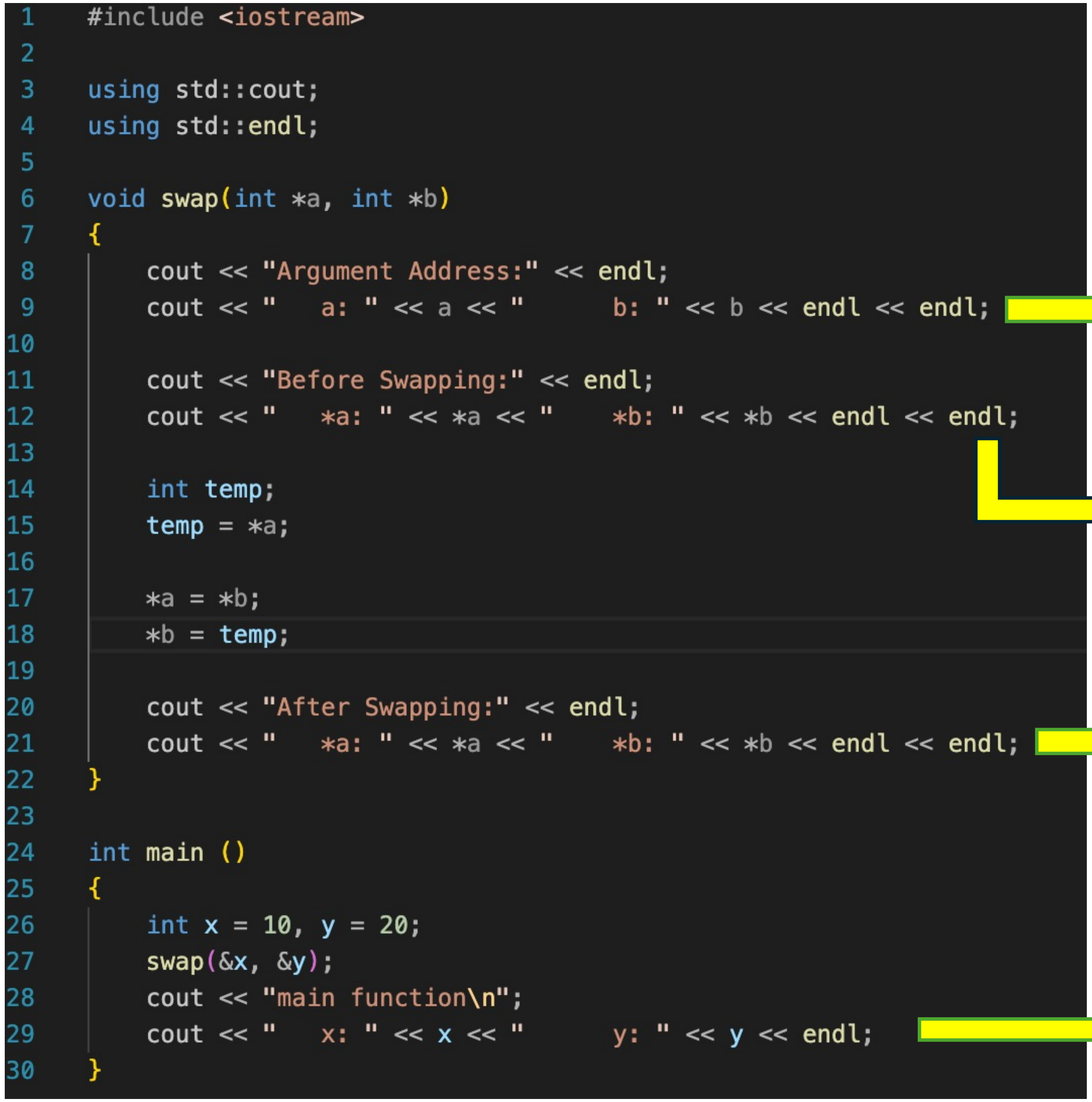


```cpp
#include <iostream>

using std::cout;
using std::endl;

// PASS BY REFERENCE
void swap(int &a, int &b)
{
    cout << "Before Swapping:" << endl;
    cout << "a: " << a << " b: " << b << endl;
    int temp;
    temp = a;
    a = b;
    b = temp;
    cout << "After Swapping:" << endl;
    cout << "a: " << a << " b: " << b << endl;
}

int main ()
{
    int x = 10, y = 20;
    swap(x, y);
    cout << "x: " << x << " y: " << y << endl;
}
```

Pass-by-ref

C++: Parameter passing

Pass-by-pointer



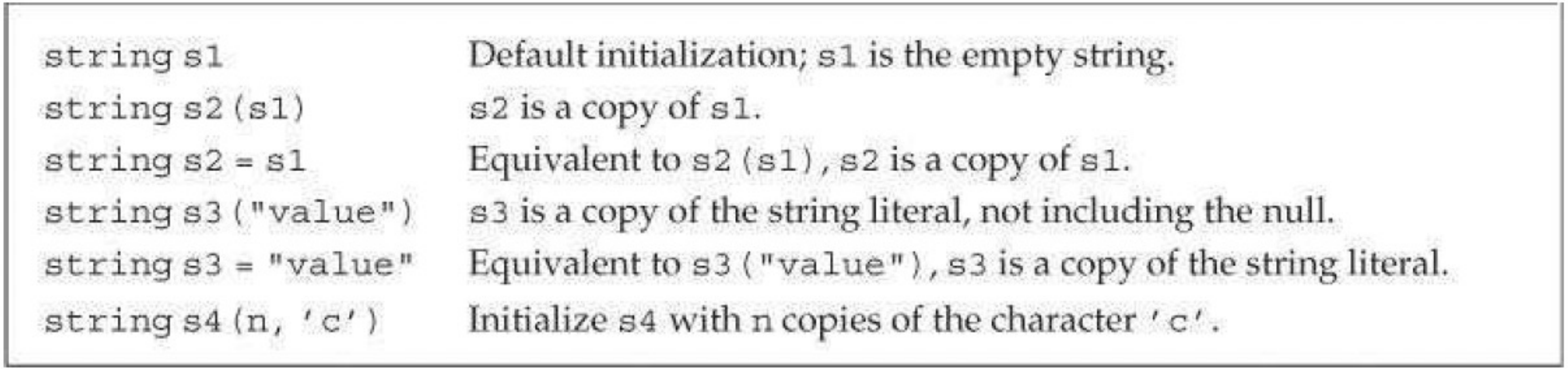
```cpp
#include <iostream>

using std::cout;
using std::endl;

void swap(int *a, int *b)
{
    cout << "Argument Address:" << endl;
    cout << "    a: " << a << "       b: " << b << endl << endl;

    cout << "Before Swapping:" << endl;
    cout << "    *a: " << *a << "    *b: " << *b << endl << endl;

    int temp;
    temp = *a;

    *a = *b;
    *b = temp;

    cout << "After Swapping:" << endl;
    cout << "    *a: " << *a << "    *b: " << *b << endl << endl;
}

int main ()
{
    int x = 10, y = 20;
    swap(&x, &y);
    cout << "main function\n";
    cout << "    x: " << x << "       y: " << y << endl;
}
```

Guidelines for parameter passing

- **Pass by value for small objects**: This is the most efficient way to pass small objects, such as integers and floats
- **Pass by const reference for large objects**: This is the most efficient way to pass large objects, such as arrays, strings, & class objects.
- **Pass by pointer or reference for objects that need to be modified**: This is the only way to pass an object that needs to be **modified** by the function.

Strings

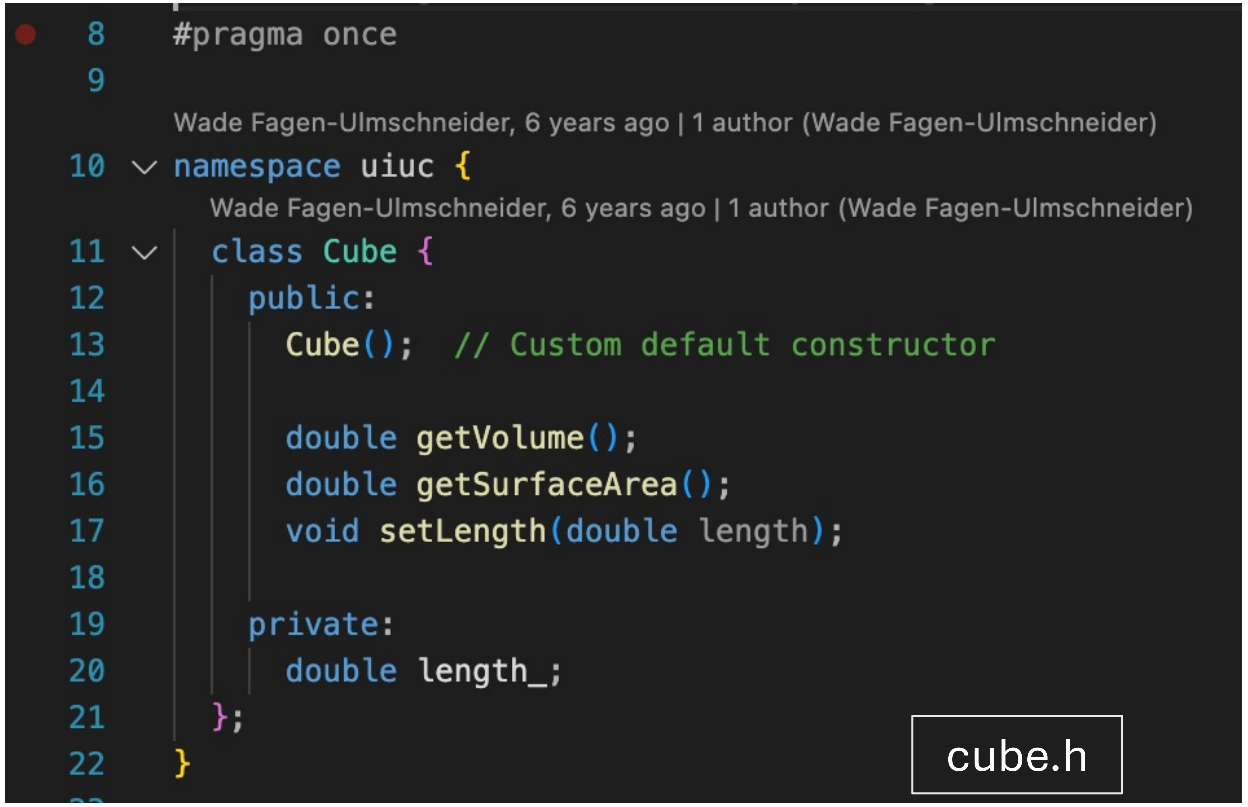
std::string



`string s1`	Default initialization; `s1` is the empty string.
`string s2(s1)`	`s2` is a copy of `s1`.
`string s2 = s1`	Equivalent to `s2(s1)`, `s2` is a copy of `s1`.
`string s3("value")`	`s3` is a copy of the string literal, not including the null.
`string s3 = "value"`	Equivalent to `s3("value")`, `s3` is a copy of the string literal.
`string s4(n, 'c')`	Initialize `s4` with `n` copies of the character `'c'`.

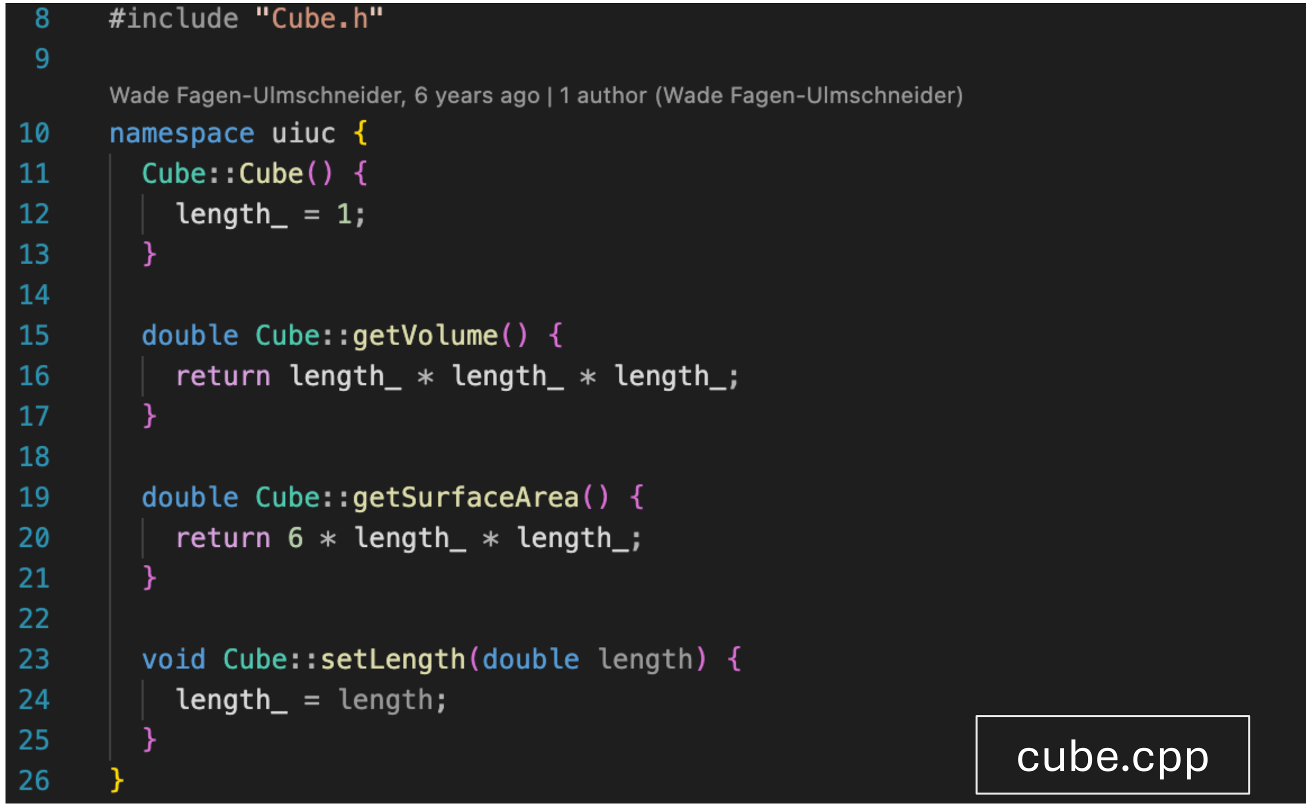
```
#include <string>
using std::string;
using std::cout;

int main(){
        string s1;
        string s3(“value”);
        cout  <<   s3 << endl;
}
```

Class



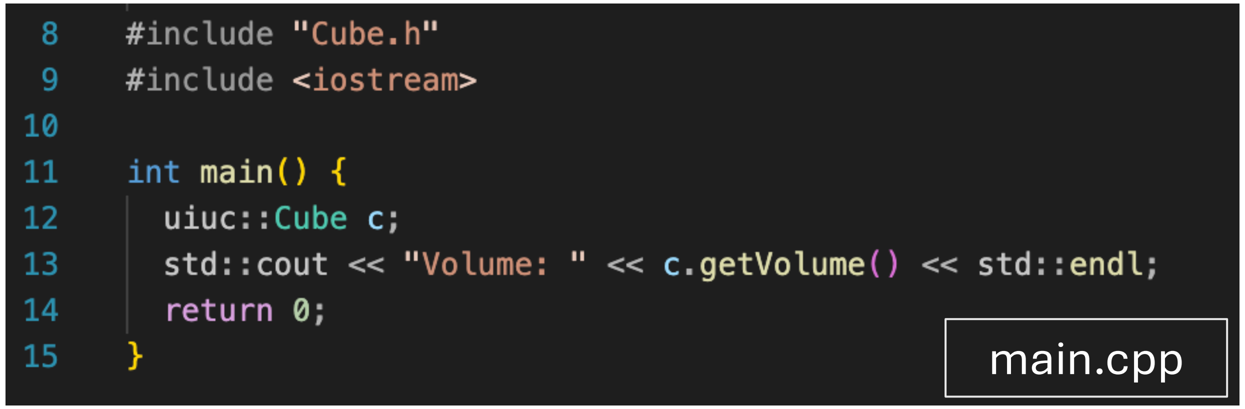
cube.h

```cpp
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}
```



cube.cpp

```cpp
#include "Cube.h"

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Cube::Cube() {
    length_ = 1;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

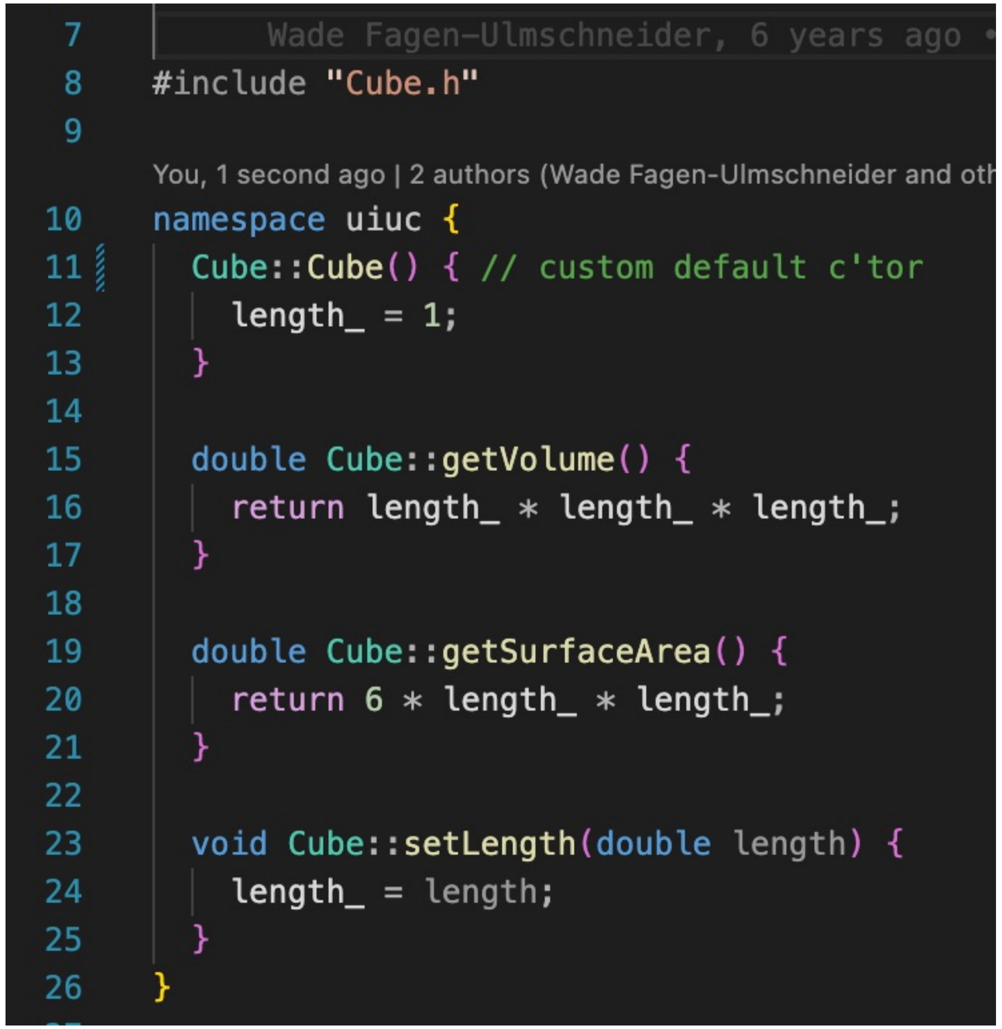


main.cpp

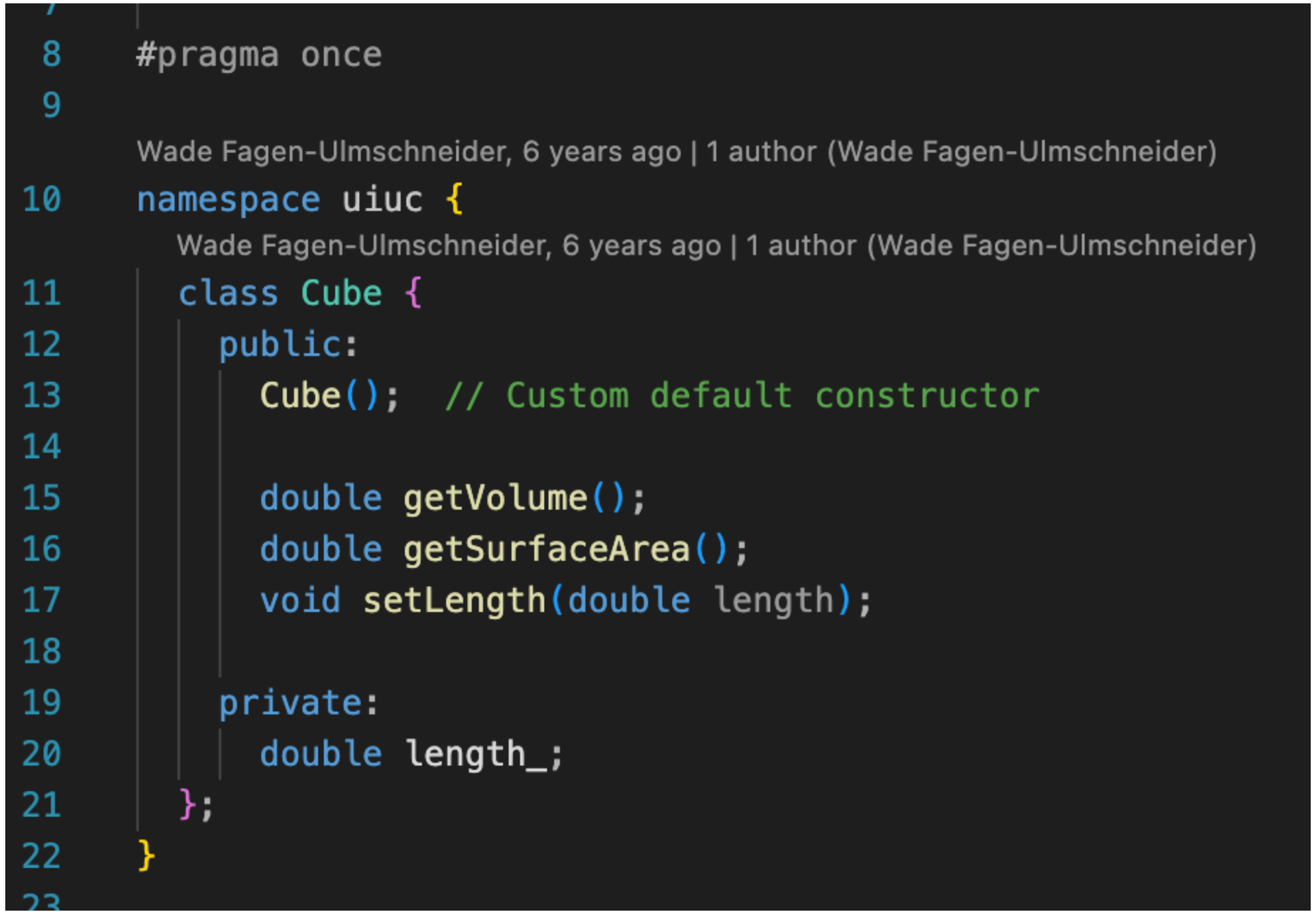
```cpp
#include "Cube.h"
#include <iostream>

int main() {
  uiuc::Cube c;
  std::cout << "Volume: " << c.getVolume() << std::endl;
  return 0;
}
```

Constructors

- Compiler provides default ctor if the class doesn't define one
- Ctor's can be overloaded



```
   Wade Fagen-Ulmschneider, 6 years ago •
#include "Cube.h"

You, 1 second ago | 2 authors (Wade Fagen-Ulmschneider and oth
namespace uiuc {
  Cube::Cube() { // custom default c'tor
    length_ = 1;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

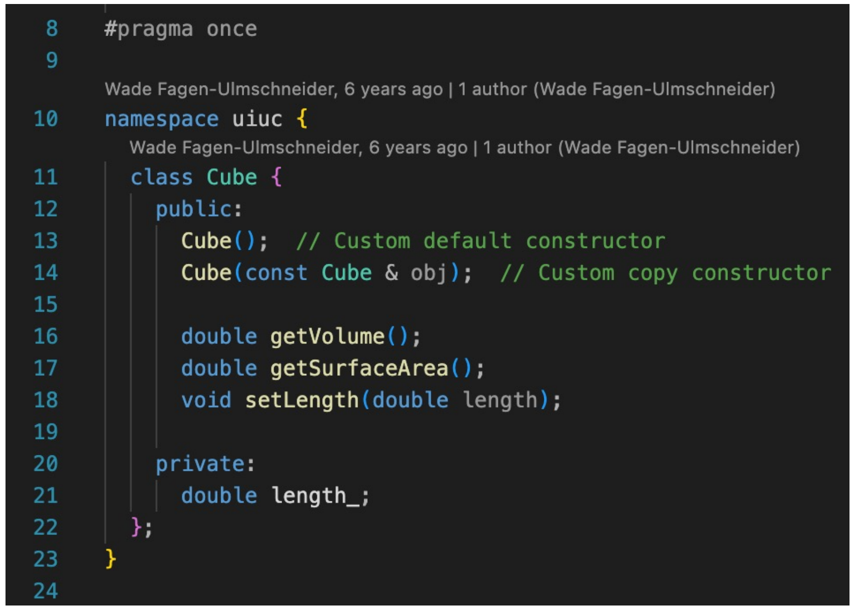


```
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}
```

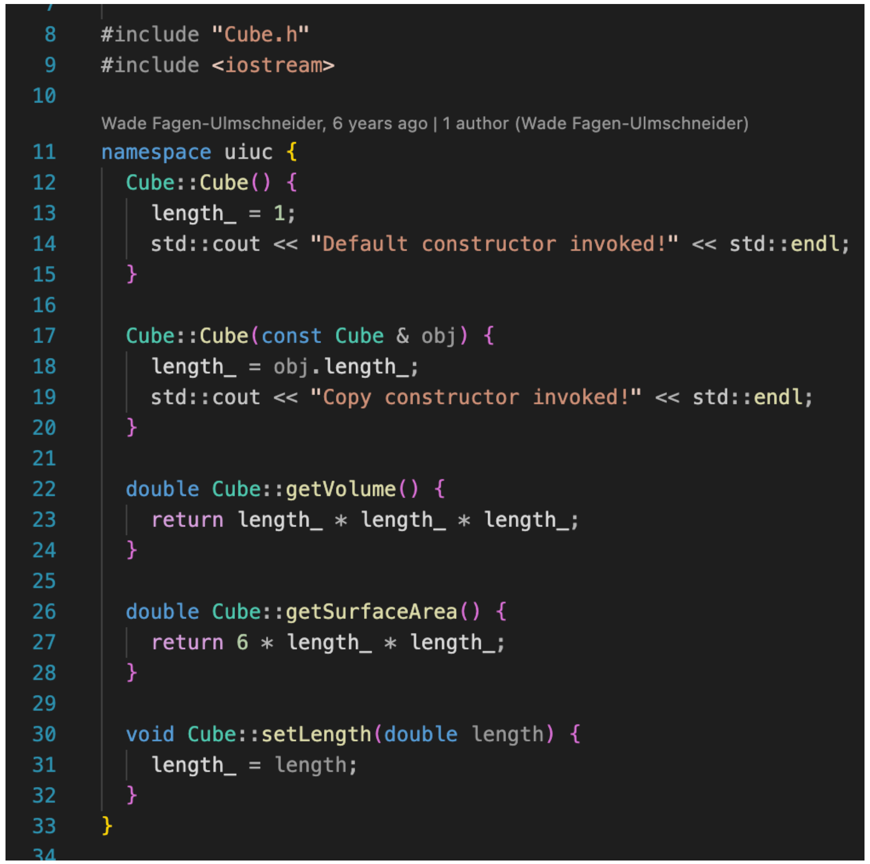
Copy Constructor

- The copy constructor is invoked whenever a **new class instance** is created from an **existing class instance**
- If you don’t provide one, the C++ compiler give you one for free.

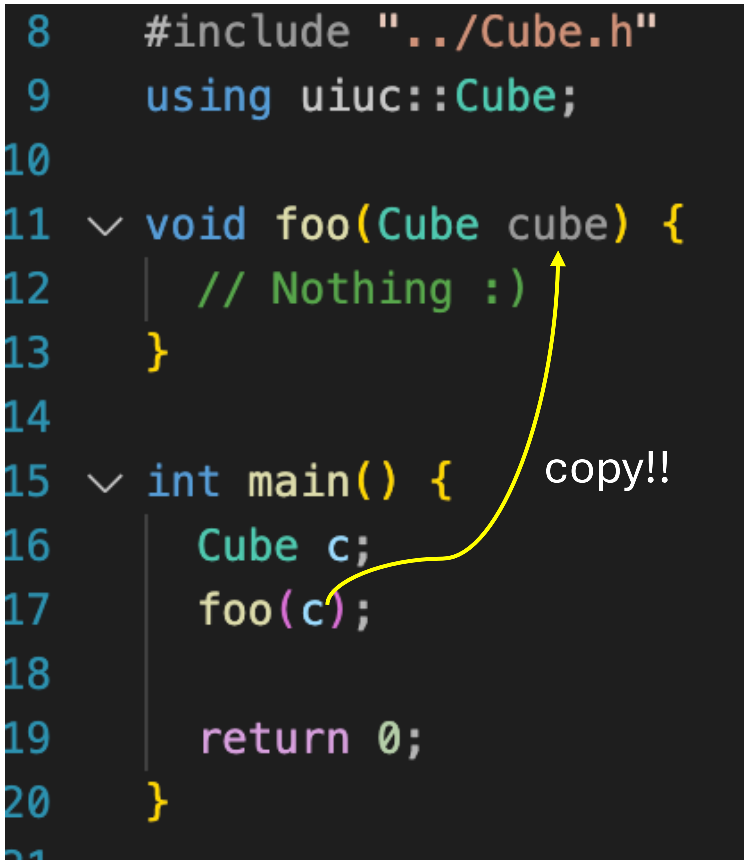
Copy Constructor: Class Object pass by value



```
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor
      Cube(const Cube & obj);  // Custom copy constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}
```

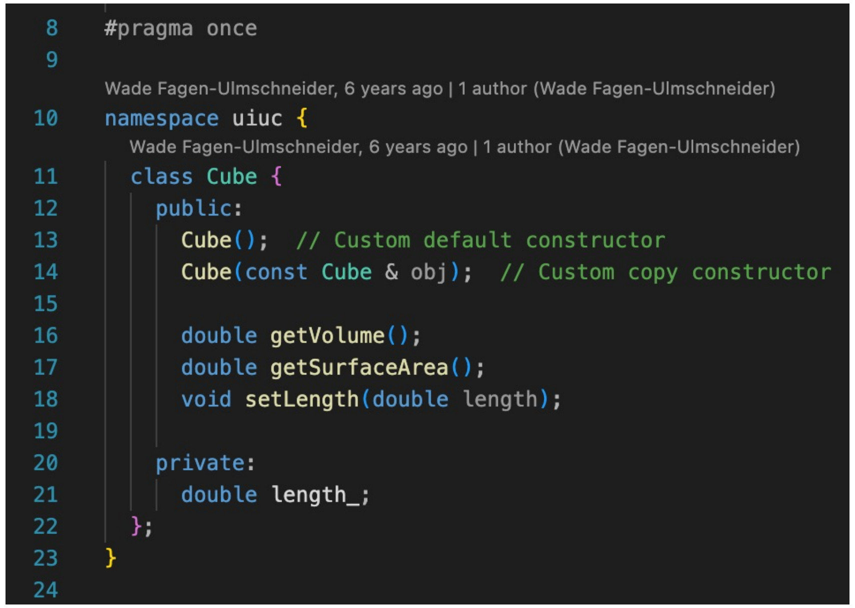


```
#include "Cube.h"
#include <iostream>

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Cube::Cube() {
    length_ = 1;
    std::cout << "Default constructor invoked!" << std::endl;
  }

  Cube::Cube(const Cube & obj) {
    length_ = obj.length_;
    std::cout << "Copy constructor invoked!" << std::endl;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

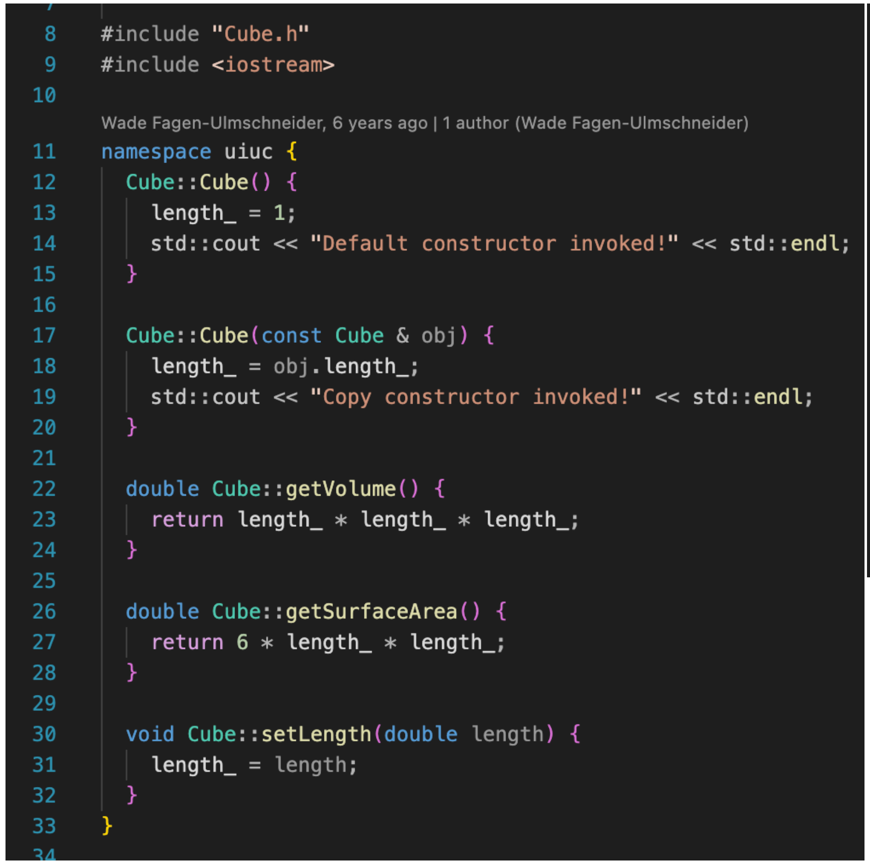


```
#include "../Cube.h"
using uiuc::Cube;

void foo(Cube cube) {
  // Nothing :)
}

int main() {
  Cube c;
  foo(c);

  return 0;
}
```

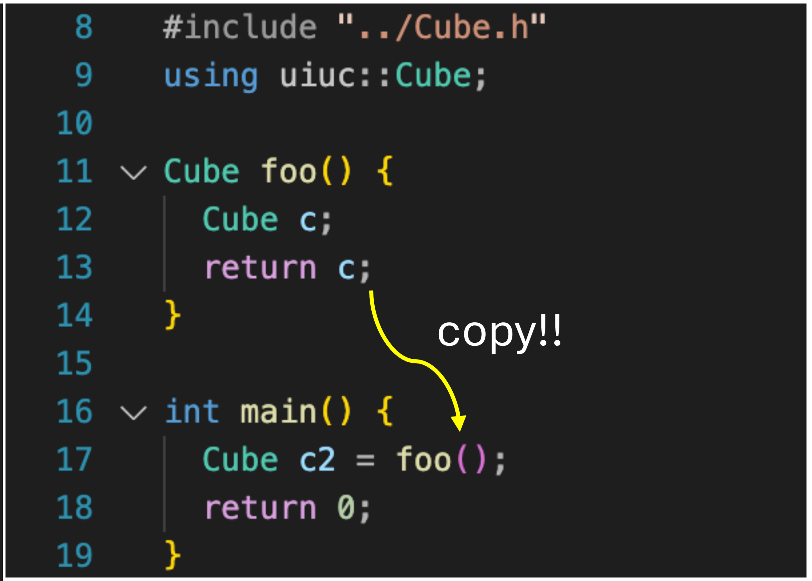
Copy Constructor: Function returns class object by value



```
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor
      Cube(const Cube & obj);  // Custom copy constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}
```

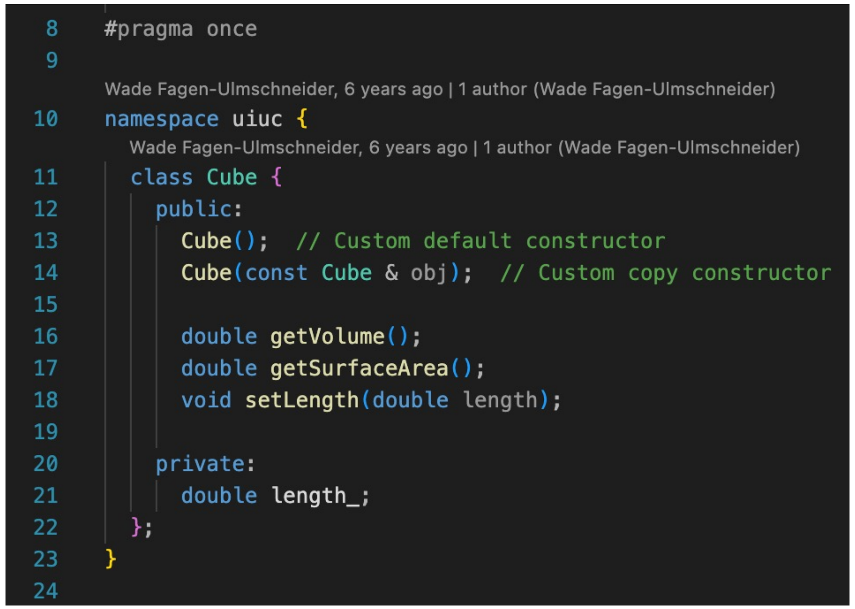


```
#include "Cube.h"
#include <iostream>

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Cube::Cube() {
    length_ = 1;
    std::cout << "Default constructor invoked!" << std::endl;
  }

  Cube::Cube(const Cube & obj) {
    length_ = obj.length_;
    std::cout << "Copy constructor invoked!" << std::endl;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

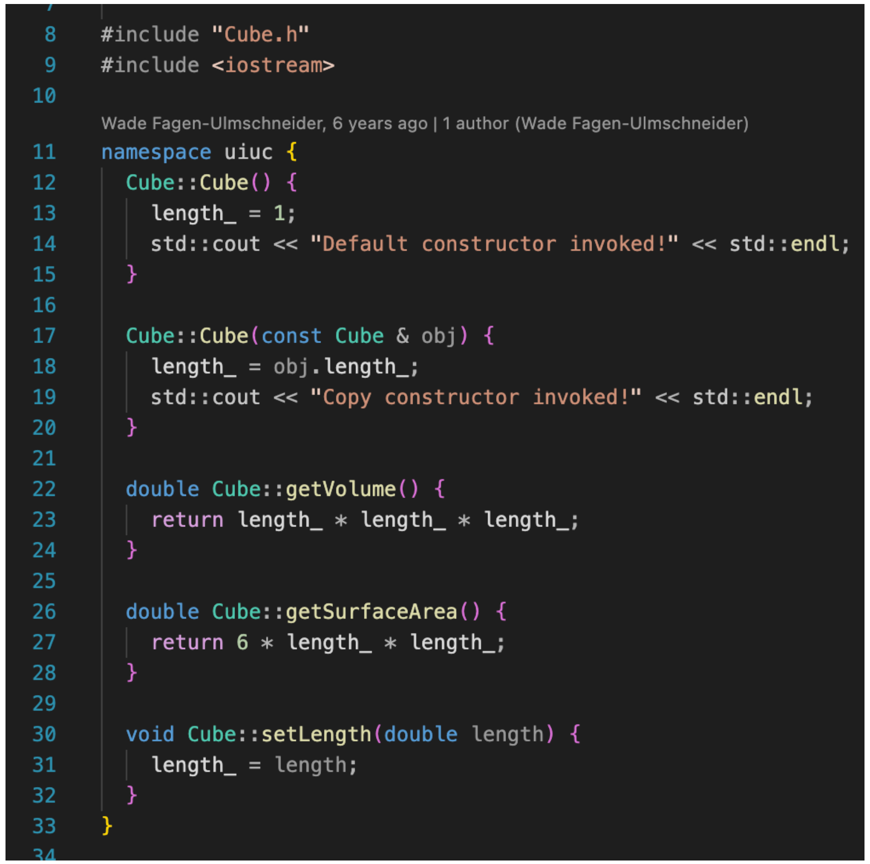


```
#include "../Cube.h"
using uiuc::Cube;

Cube foo() {
  Cube c;
  return c;
}

int main() {
  Cube c2 = foo();
  return 0;
}
```

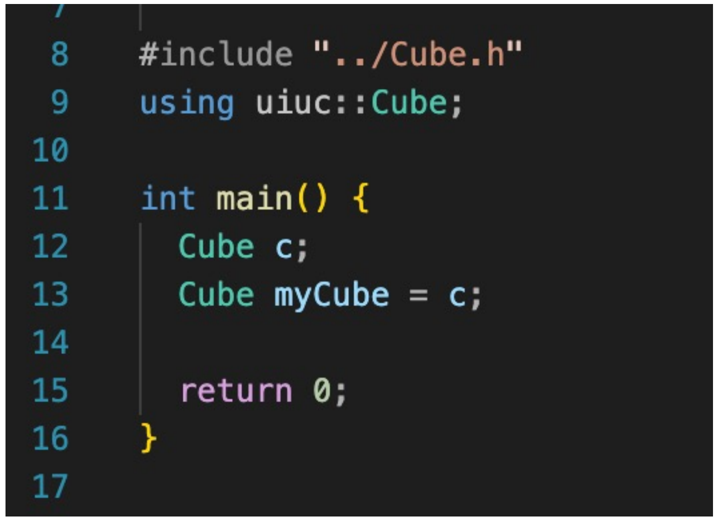
Copy Constructor:



```cpp
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor
      Cube(const Cube & obj);  // Custom copy constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}

```

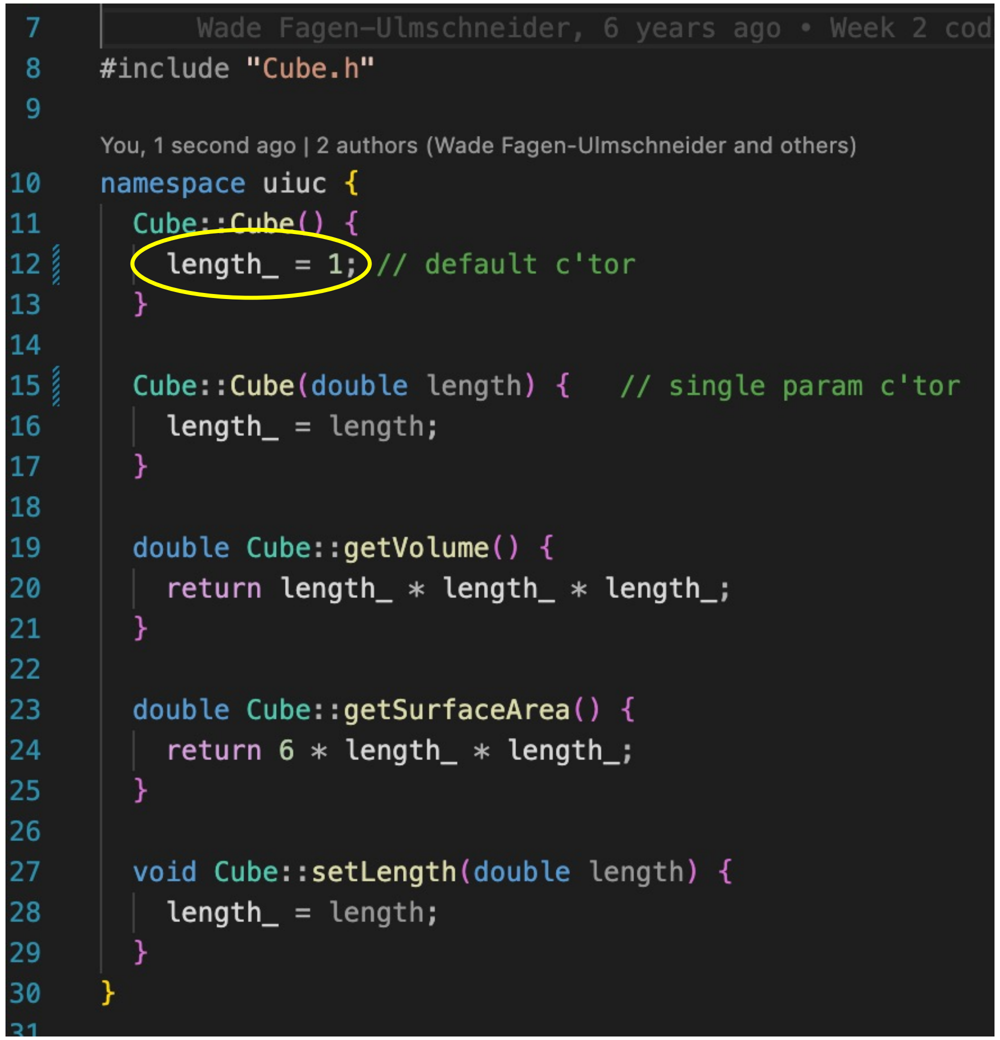


```cpp
#include "Cube.h"
#include <iostream>

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Cube::Cube() {
    length_ = 1;
    std::cout << "Default constructor invoked!" << std::endl;
  }

  Cube::Cube(const Cube & obj) {
    length_ = obj.length_;
    std::cout << "Copy constructor invoked!" << std::endl;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

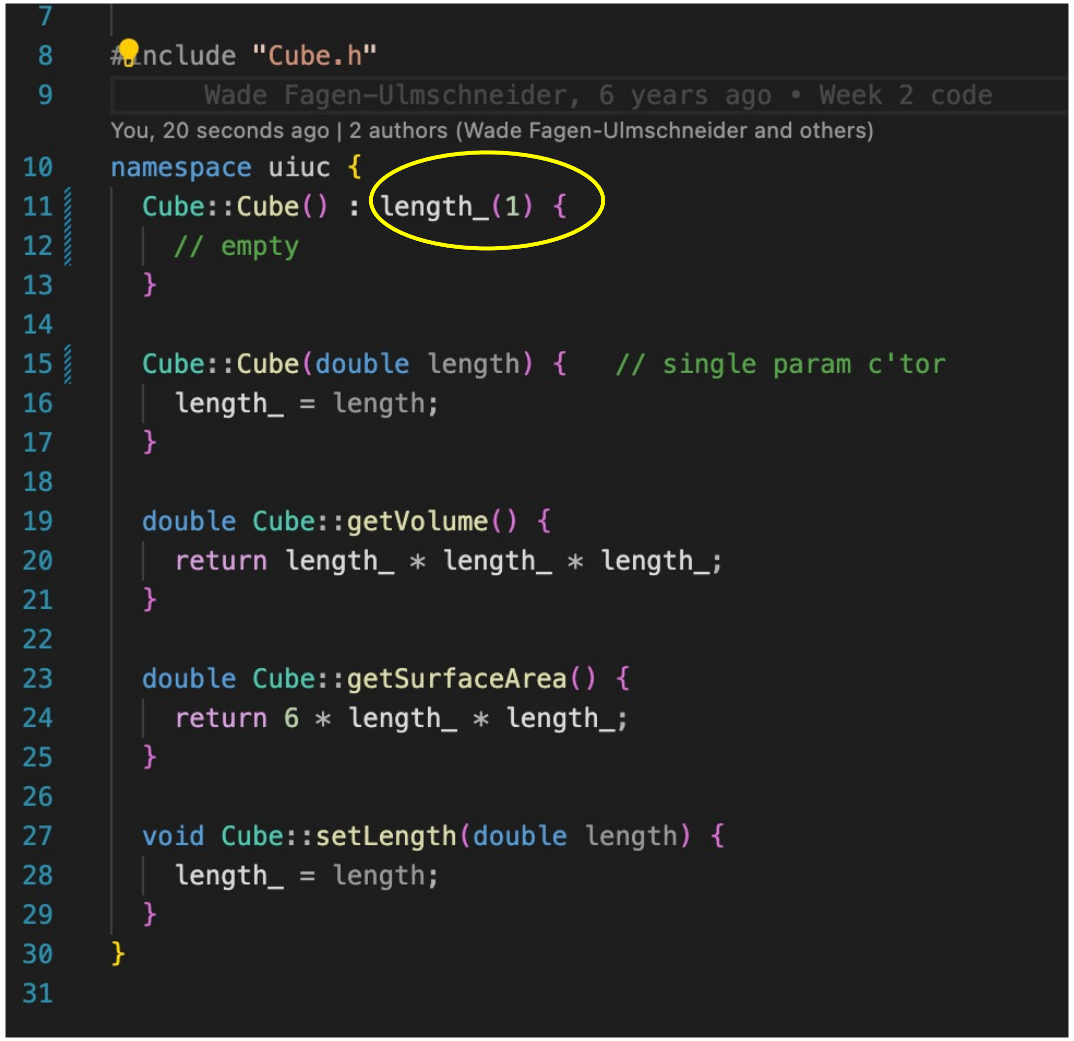


```cpp
#include "../Cube.h"
using uiuc::Cube;

int main() {
  Cube c;
  Cube myCube = c;

  return 0;
}

```

Class member Initialization

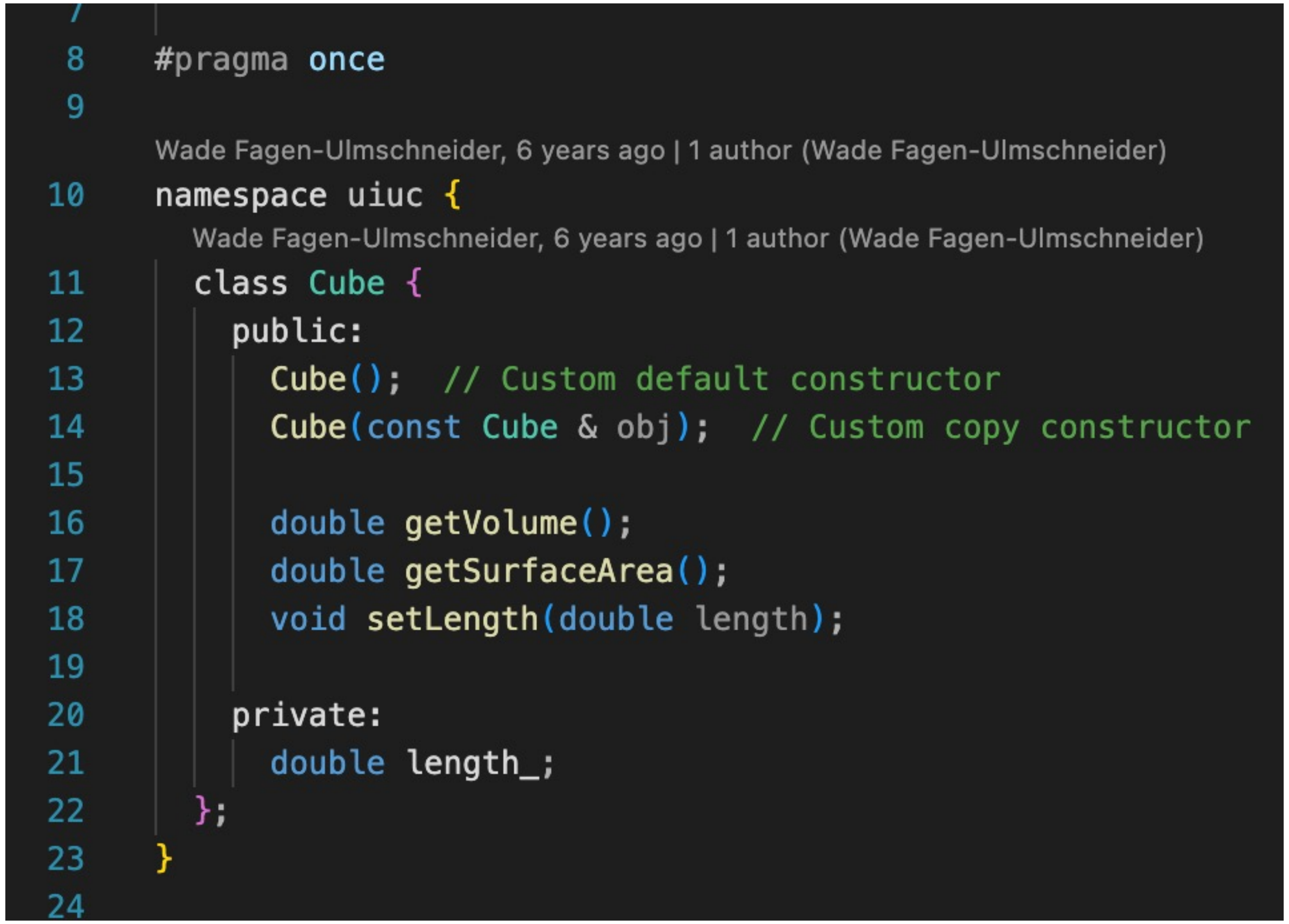


```
    Wade Fagen-Ulmschneider, 6 years ago • Week 2 cod
#include "Cube.h"

You, 1 second ago | 2 authors (Wade Fagen-Ulmschneider and others)
namespace uiuc {
  Cube::Cube() {
    length_ = 1; // default c'tor
  }

  Cube::Cube(double length) {   // single param c'tor
    length_ = length;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

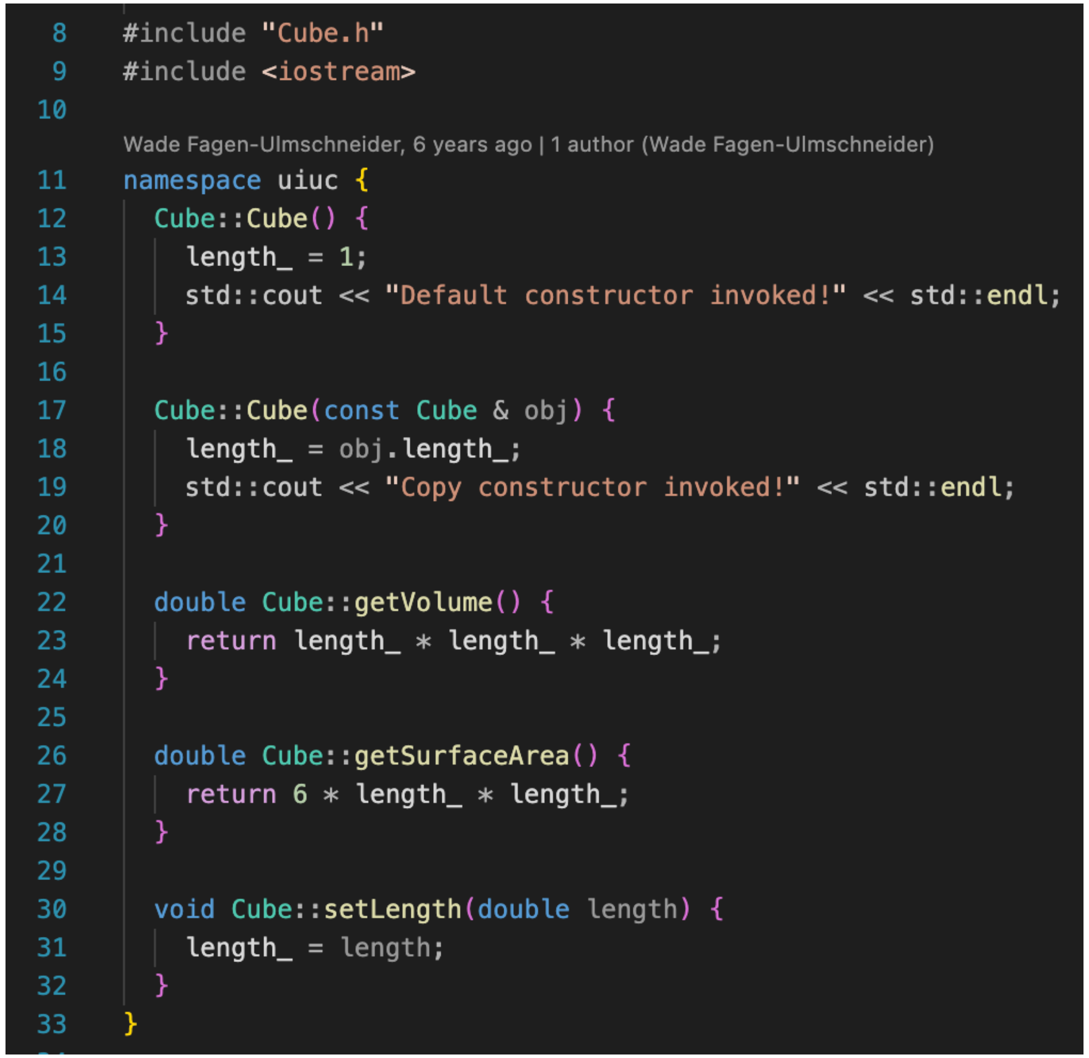


```
#include "Cube.h"
    Wade Fagen-Ulmschneider, 6 years ago • Week 2 code
You, 20 seconds ago | 2 authors (Wade Fagen-Ulmschneider and others)
namespace uiuc {
  Cube::Cube() : length_(1) {
    // empty
  }

  Cube::Cube(double length) {   // single param c'tor
    length_ = length;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```

Copy Constructor



```
#pragma once

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
  class Cube {
    public:
      Cube();  // Custom default constructor
      Cube(const Cube & obj);  // Custom copy constructor

      double getVolume();
      double getSurfaceArea();
      void setLength(double length);

    private:
      double length_;
  };
}
```



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

Wade Fagen-Ulmschneider, 6 years ago | 1 author (Wade Fagen-Ulmschneider)
namespace uiuc {
  Cube::Cube() {
    length_ = 1;
    std::cout << "Default constructor invoked!" << std::endl;
  }

  Cube::Cube(const Cube & obj) {
    length_ = obj.length_;
    std::cout << "Copy constructor invoked!" << std::endl;
  }

  double Cube::getVolume() {
    return length_ * length_ * length_;
  }

  double Cube::getSurfaceArea() {
    return 6 * length_ * length_;
  }

  void Cube::setLength(double length) {
    length_ = length;
  }
}
```