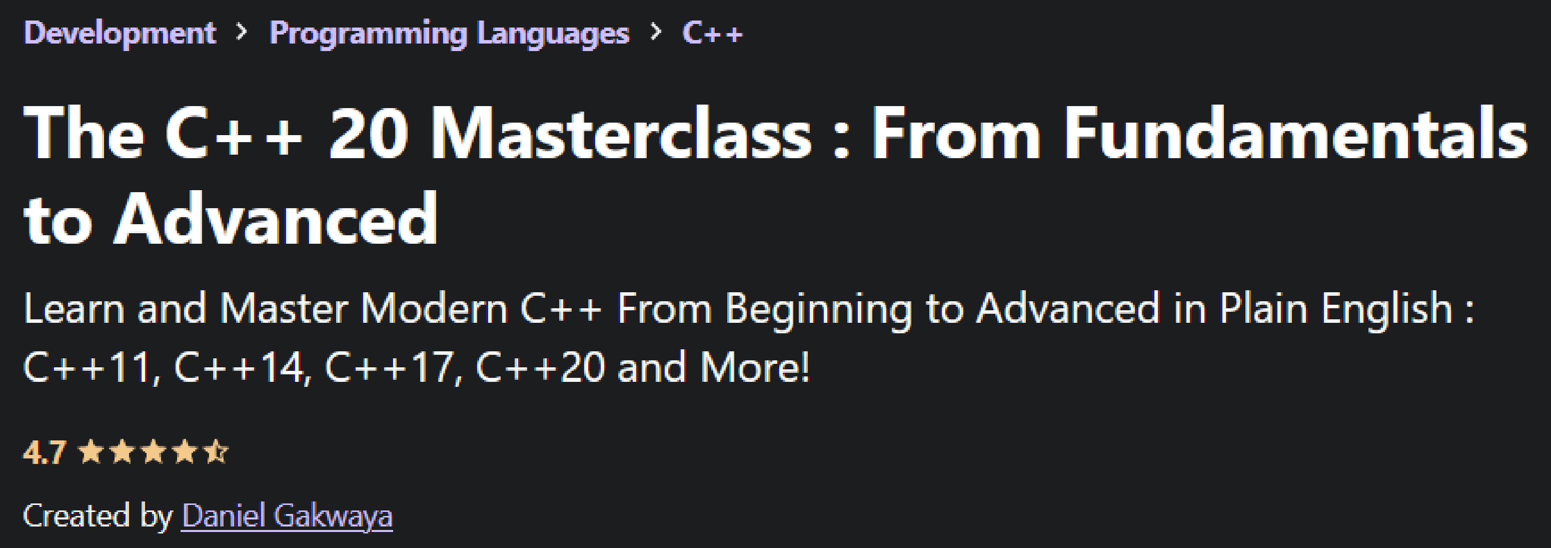
Slides


Development › Programming Languages › C++

The C++ 20 Masterclass : From Fundamentals to Advanced

Learn and Master Modern C++ From Beginning to Advanced in Plain English : C++11, C++14, C++17, C++20 and More!

4.7 ★★★★☆

Created by Daniel Gakwaya

# Section : C++20 Modules

Slide intentionally left empty

# C++20 Modules

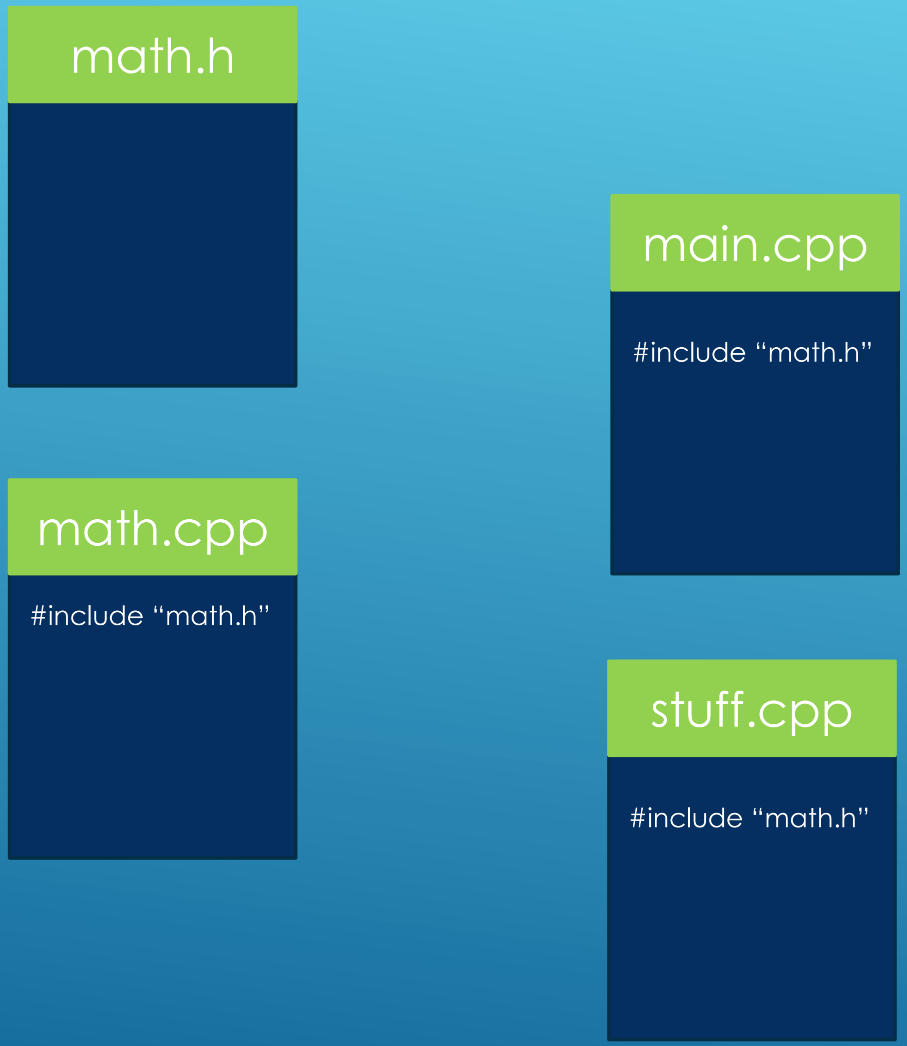
Concepts

Ranges

Coroutines

Modules

- One of the big C++20 features
- Changes the compilation model of C++ programs



math.cpp:

```
#include “math.h”
```

main.cpp:

```
#include “math.h”
```

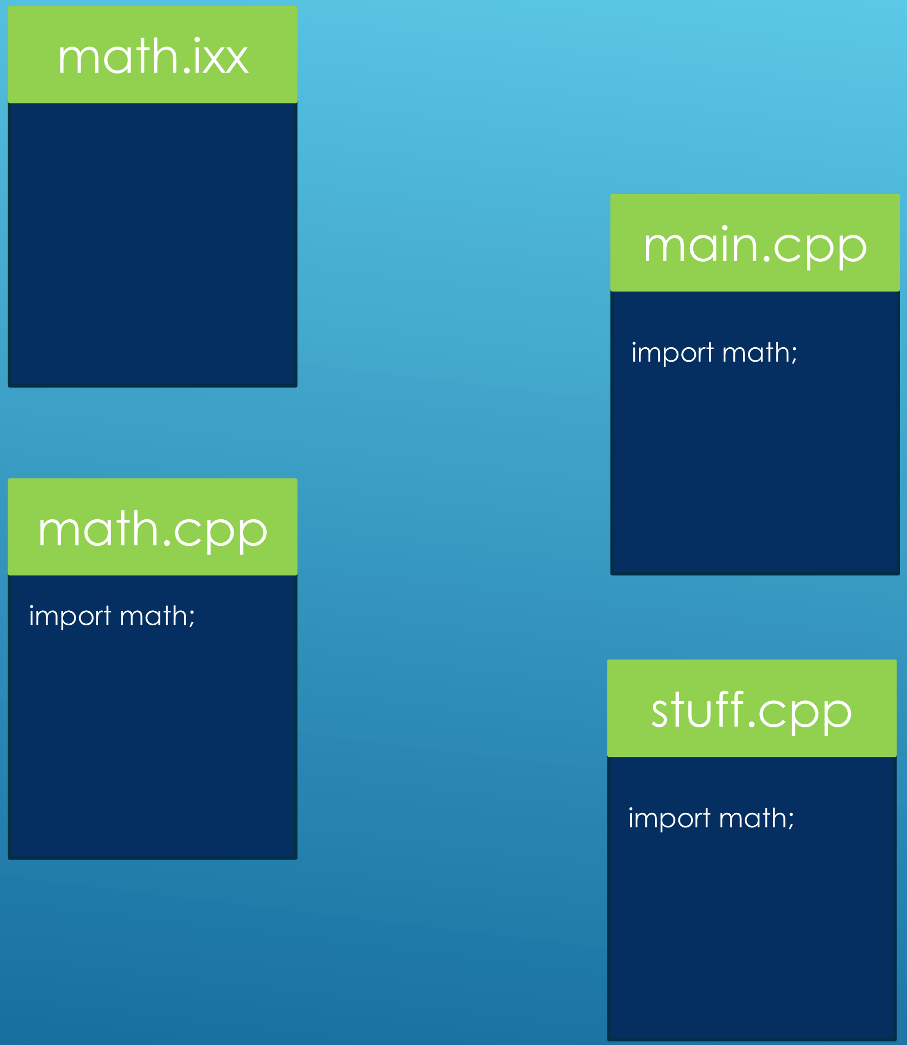
stuff.cpp:

```
#include “math.h”
```

## Problems with headers

- Compilation speed
- ODR violations
- Include order
- …

```cpp
//include "file1.h"
void f1(){  // f1 brought in by #include "file1.h"
    f2();  // f2  brought in by #include "file2.h"
}

//#include "file2.h"

int main()
{
   f1()
}
```


math.ixx

math.cpp

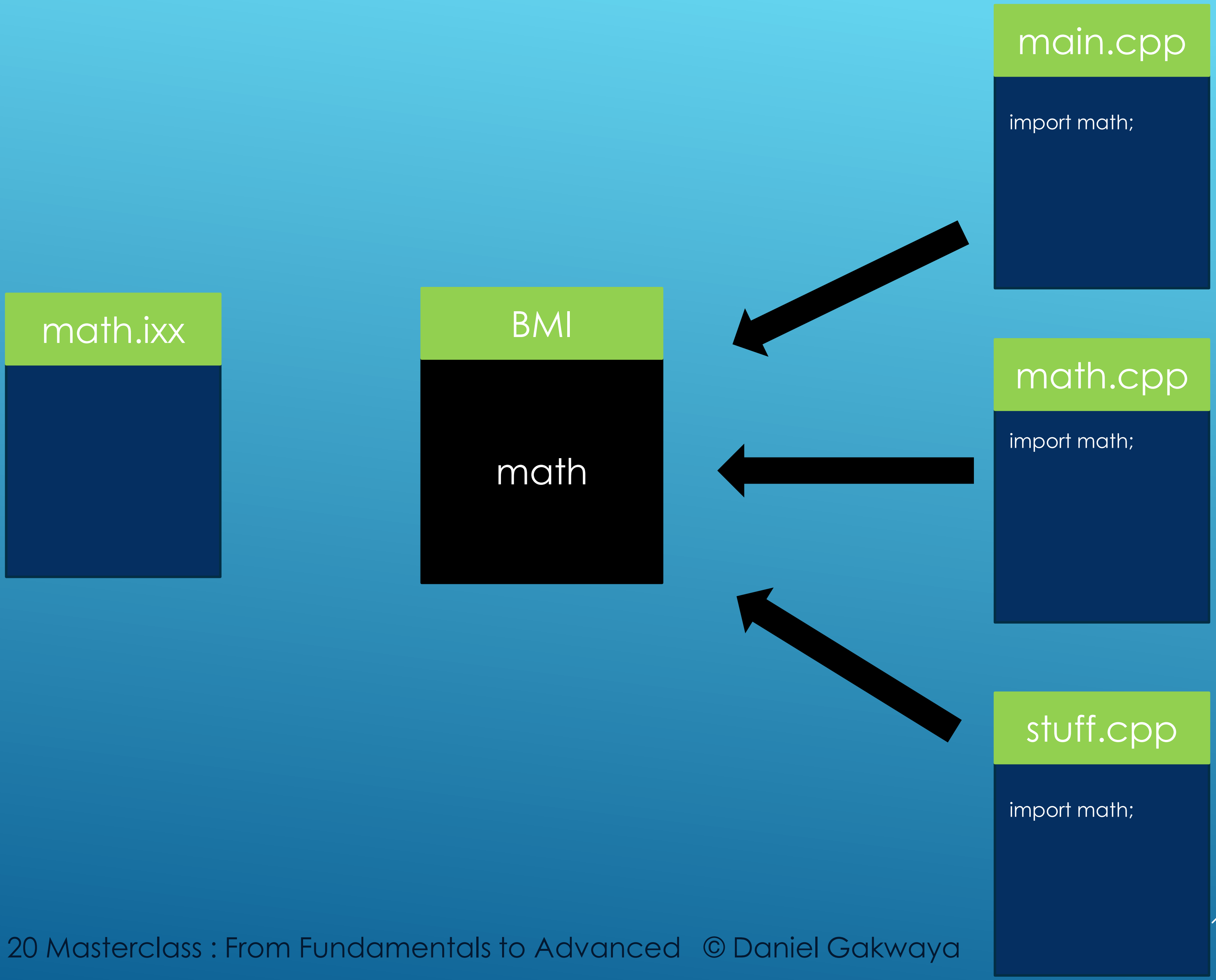
```
import math;
```

main.cpp

```
import math;
```

stuff.cpp

```
import math;
```



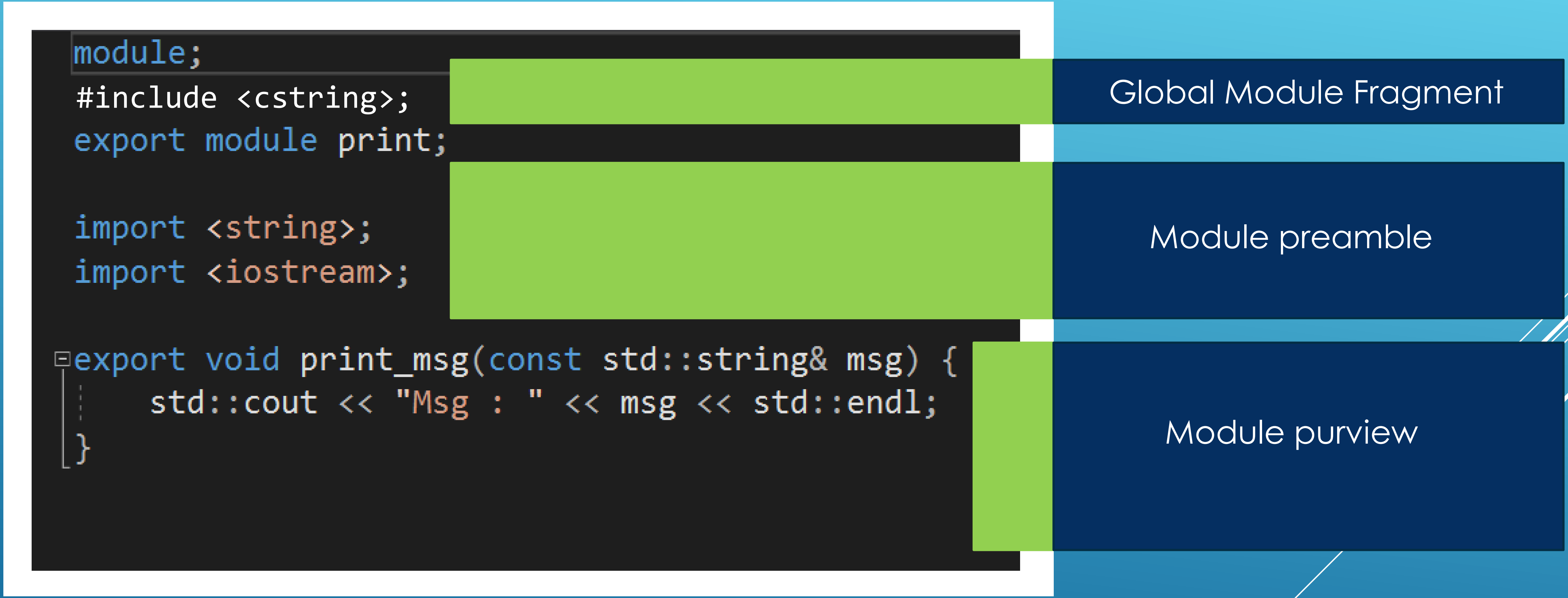
```
import math;
```

```
import math;
```

```
import math;
```

## With modules

- Compilation speed improved by BMI
- ODR violations. None. BMI lives in one object file
- Include order. Doesn't matter. Modules available in some pre-compiled form in the BMI file
- …



```
module;
#include <cstring>;
export module print;

import <string>;
import <iostream>;

export void print_msg(const std::string& msg) {
    std::cout << "Msg : " << msg << std::endl;
}
```

## Global Module fragment

- Can only contain preprocessor directives
- Allows old style #include'ing of headers in a module
- #include statements can only show up in the global module fragment, if you put them elsewhere, you'll get weird errors

## Module preamble

- Should only contain import statements. May be importing proper module files, or just plain old non C-library headers

## Module purview

- Contains the meat of the module; functions, types ,… that make up the interface of the module
- Some entities are exported, making them visible outside the module
- Those not exported are only usable inside the module itself

## Facts

- Each module is compiled independently into an object file
- Modules must be compiled in an order in such a way that all the dependencies are compiled first.
- If you are compiling manually on the command line yourself, you have to take care of this order yourself
- In practice however, IDEs and build systems will figure out this order automatically.

## Environment

- Windows 10
- Miscrosoft Visual Studio 2019 16.9.0 Preview 2

Compiler

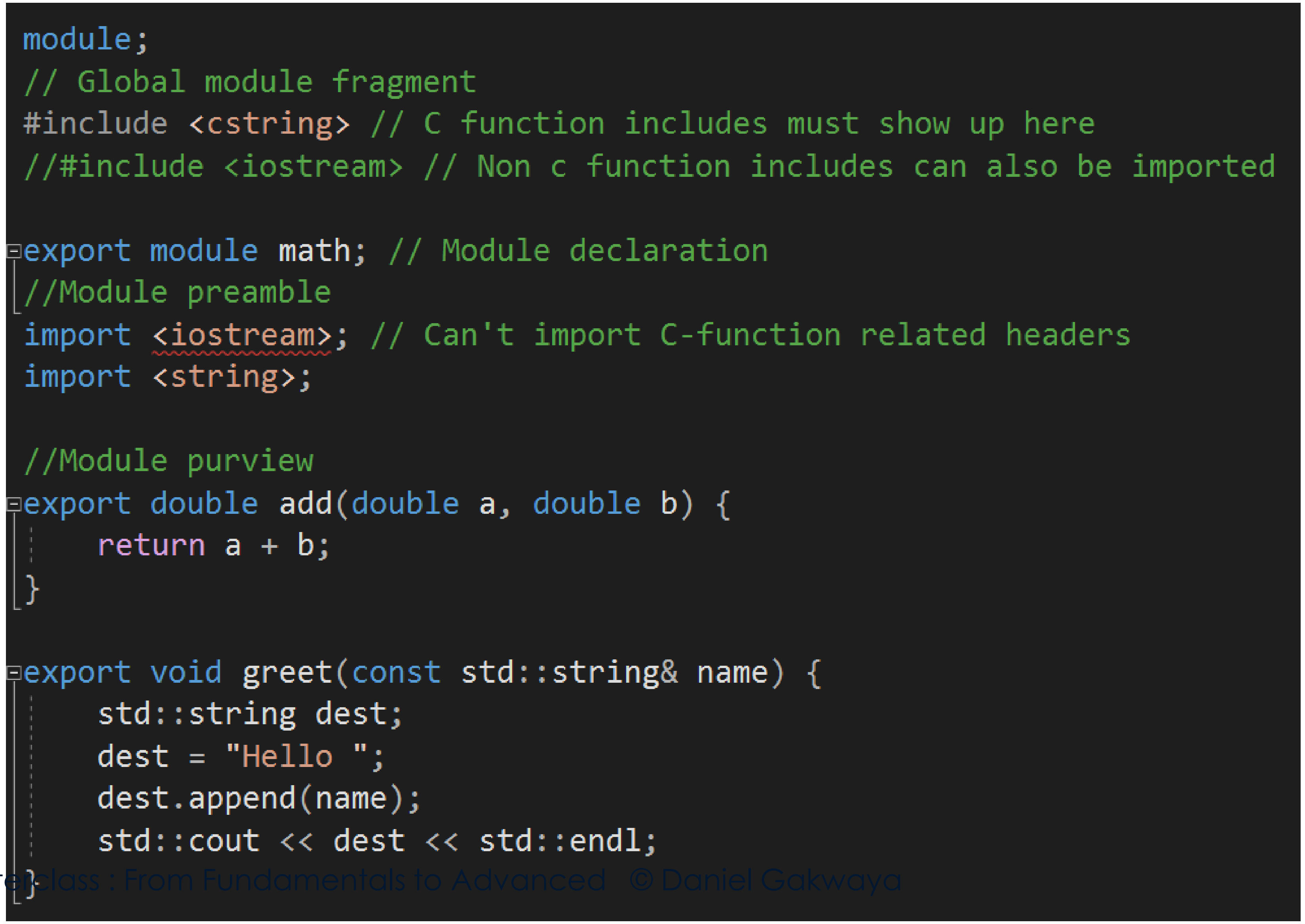
Build system

IDE

Slide intentionally left empty

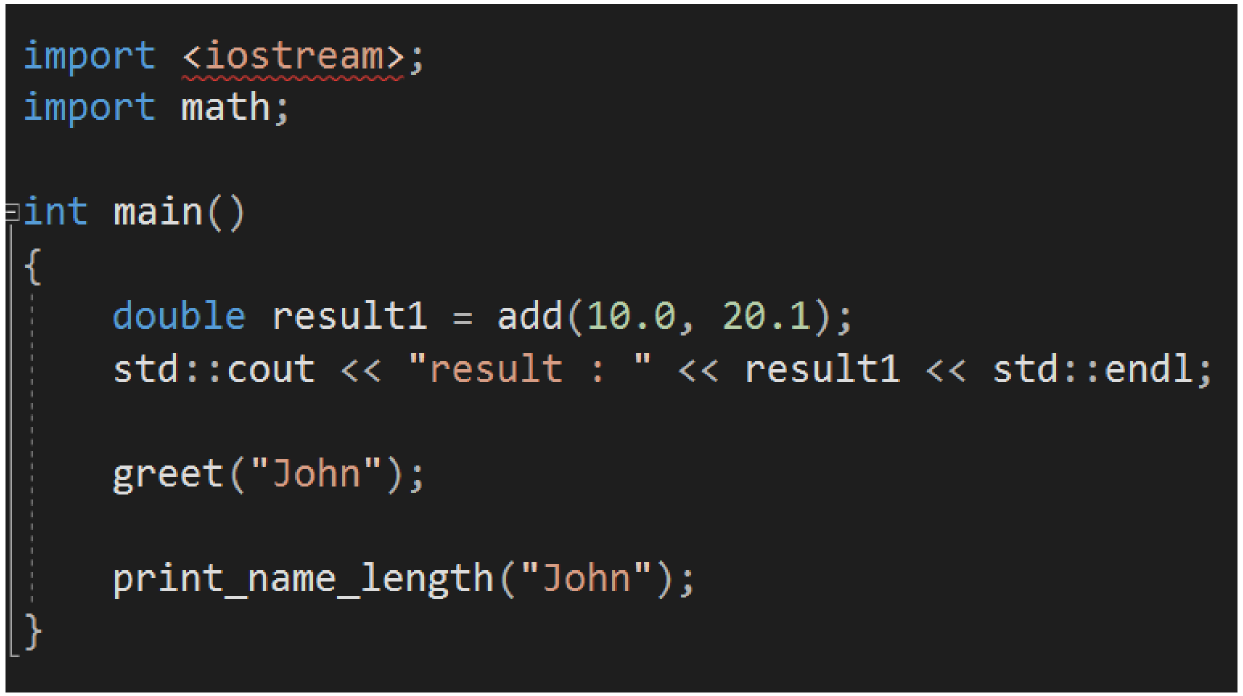
## Your First Module

## math.ixx



```cpp
module;
// Global module fragment
#include <cstring> // C function includes must show up here
//#include <iostream> // Non c function includes can also be imported

export module math; // Module declaration
//Module preamble
import <iostream>; // Can't import C-function related headers
import <string>;

//Module purview
export double add(double a, double b) {
    return a + b;
}

export void greet(const std::string& name) {
    std::string dest;
    dest = "Hello ";
    dest.append(name);
    std::cout << dest << std::endl;
}
```

main.cpp



```cpp
import <iostream>;
import math;

int main()
{
    double result1 = add(10.0, 20.1);
    std::cout << "result : " << result1 << std::endl;

    greet("John");

    print_name_length("John");
}
```

## Module interface file

- If the module declaration has export in front of it, the file containing the module becomes a module interface file
- The collection of entities export'ed from a module file make up the module interface

## Module interface file extension

msvc : .ixx
Gcc/Clang : .cc/.cppm

* gcc : https://gcc.gnu.org/wiki/cxx-modules

## Three options for working with modules

- Include translation
- Header importation
- Module importation

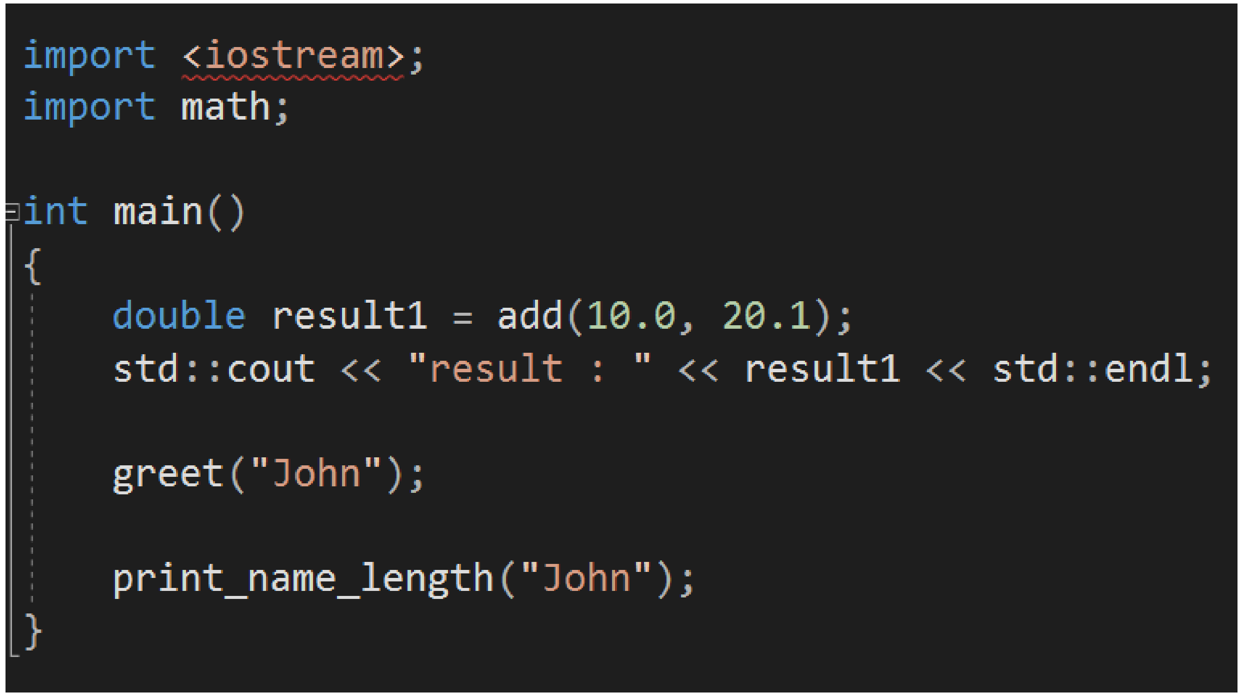
## Include translation

- Plain old raw #include statements in the global module fragment
- Other preprocessing directives should show up in the global module fragment too
- Headers that originate from the C Standard Library like <cmath>, <cstring>, … can only shop up  in the global module fragment. You'll regognize them because their names are prefixed with a "c"
- The magic by which non modular code in the global module fragment is converted and consumed in our module is out of scope for this course

## Header Importation

- Imports in the module preamble like import <iostream>;
- They are ended by a semicolon ";"
- Headers imported with the syntax "import <iostream>;" are called header units
- The compiler actually goes in and transforms the header into a legit module interface file by inserting proper module declarations and export statements where it makes sense

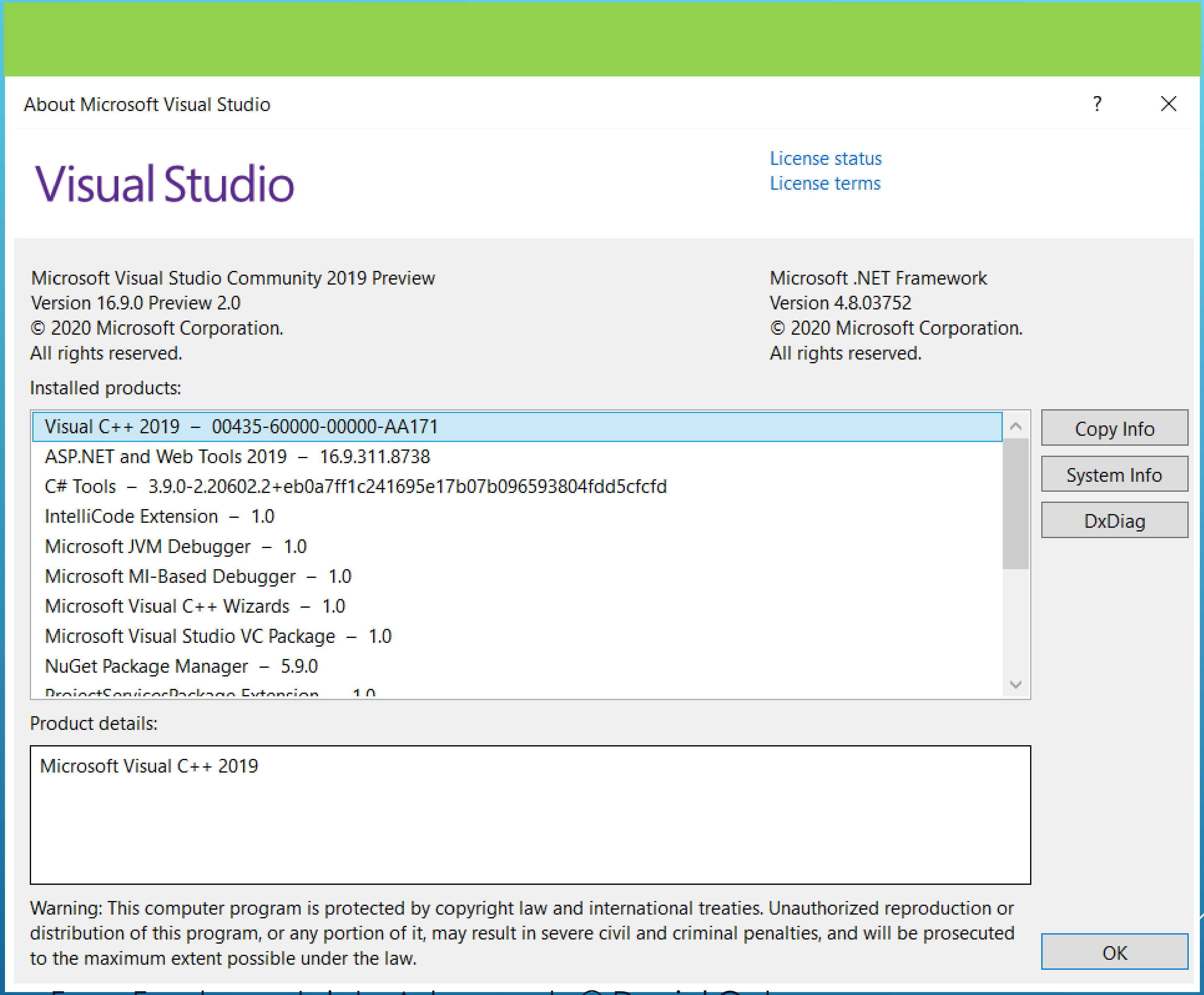
## Module importation



```
import <iostream>;
import math;

int main()
{
    double result1 = add(10.0, 20.1);
    std::cout << "result : " << result1 << std::endl;

    greet("John");

    print_name_length("John");
}
```

## Module importation

The name you use in your import statement is not based on the name of the file containing the module, it’s based on the name in your module declaration statement



About Microsoft Visual Studio

Visual Studio

License status
License terms

Microsoft Visual Studio Community 2019 Preview
Version 16.9.0 Preview 2.0
© 2020 Microsoft Corporation.
All rights reserved.

Microsoft .NET Framework
Version 4.8.03752
© 2020 Microsoft Corporation.
All rights reserved.

Installed products:

```
Visual C++ 2019  –  00435-60000-00000-AA171
ASP.NET and Web Tools 2019  –  16.9.311.8738
C# Tools  –  3.9.0-2.20602.2+eb0a7ff1c241695e17b07b096593804fdd5cfcfd
IntelliCode Extension  –  1.0
Microsoft JVM Debugger  –  1.0
Microsoft MI-Based Debugger  –  1.0
Microsoft Visual C++ Wizards  –  1.0
Microsoft Visual Studio VC Package  –  1.0
NuGet Package Manager  –  5.9.0
ProjectServicesPackage Extension  –  1.0
```

Product details:

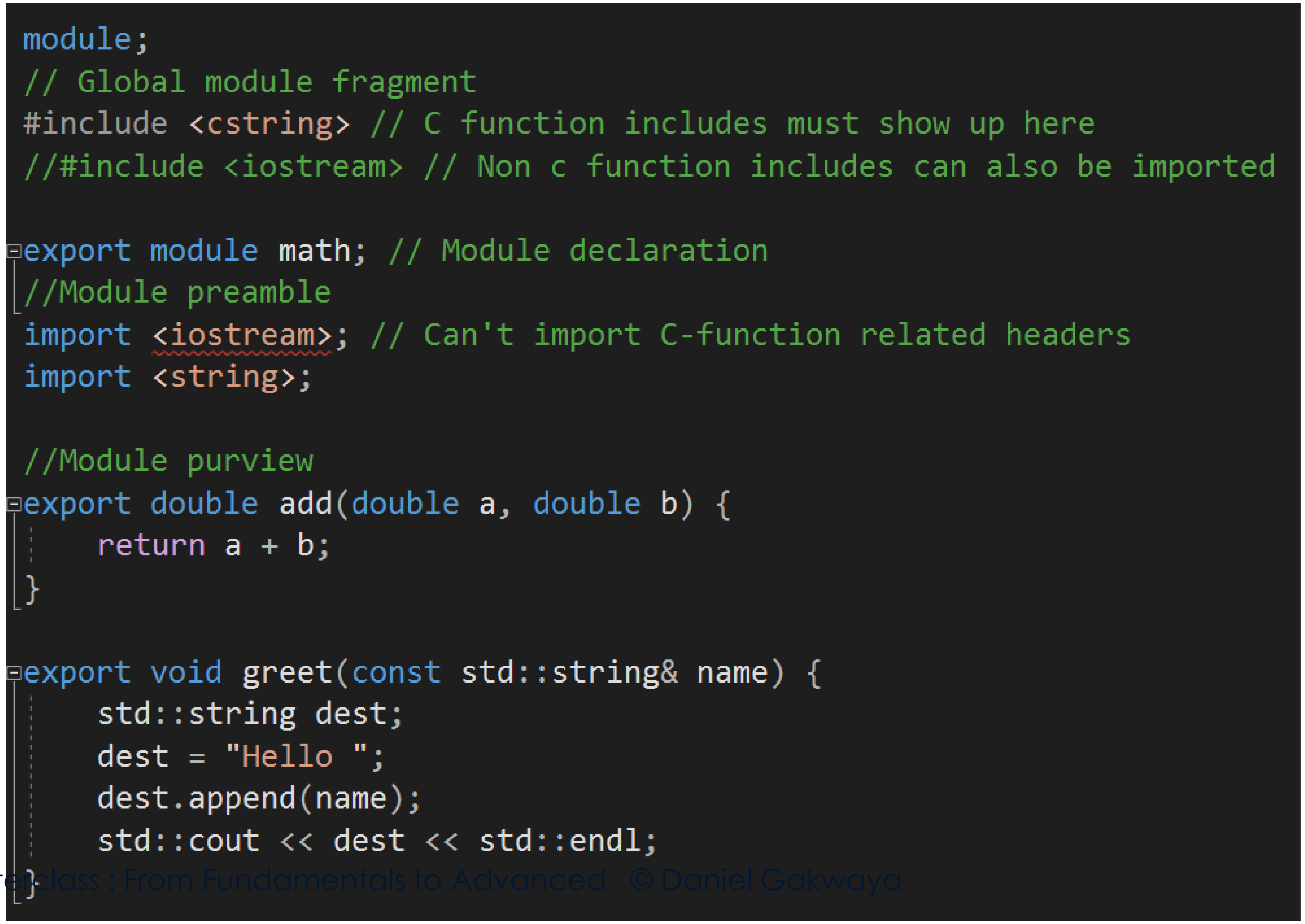
Microsoft Visual C++ 2019

Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Slide intentionally left empty

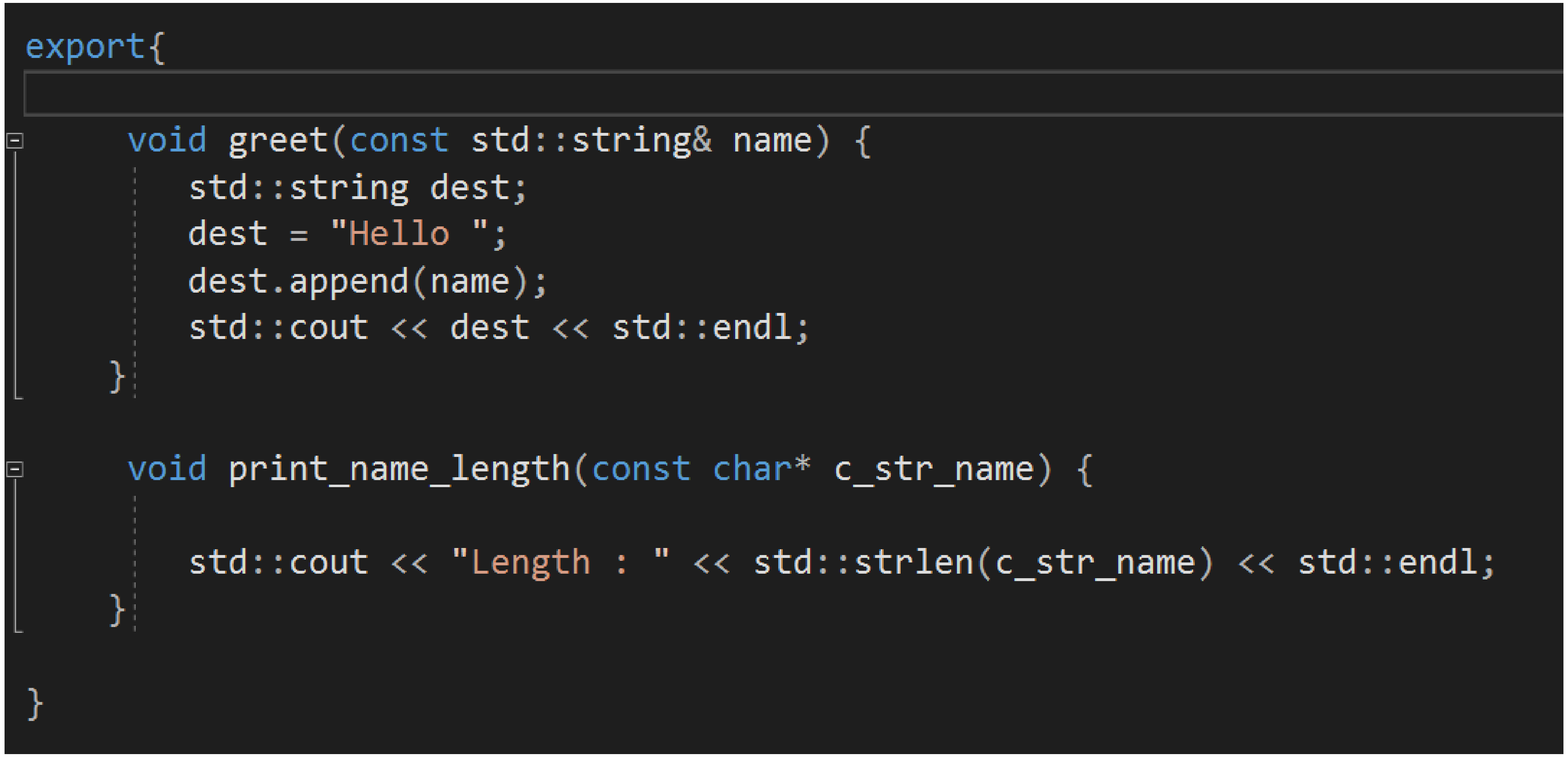
## Block Export

## math.ixx



```cpp
module;
// Global module fragment
#include <cstring> // C function includes must show up here
//#include <iostream> // Non c function includes can also be imported

export module math; // Module declaration
//Module preamble
import <iostream>; // Can't import C-function related headers
import <string>;

//Module purview
export double add(double a, double b) {
    return a + b;
}

export void greet(const std::string& name) {
    std::string dest;
    dest = "Hello ";
    dest.append(name);
    std::cout << dest << std::endl;
}
```

## math.ixx

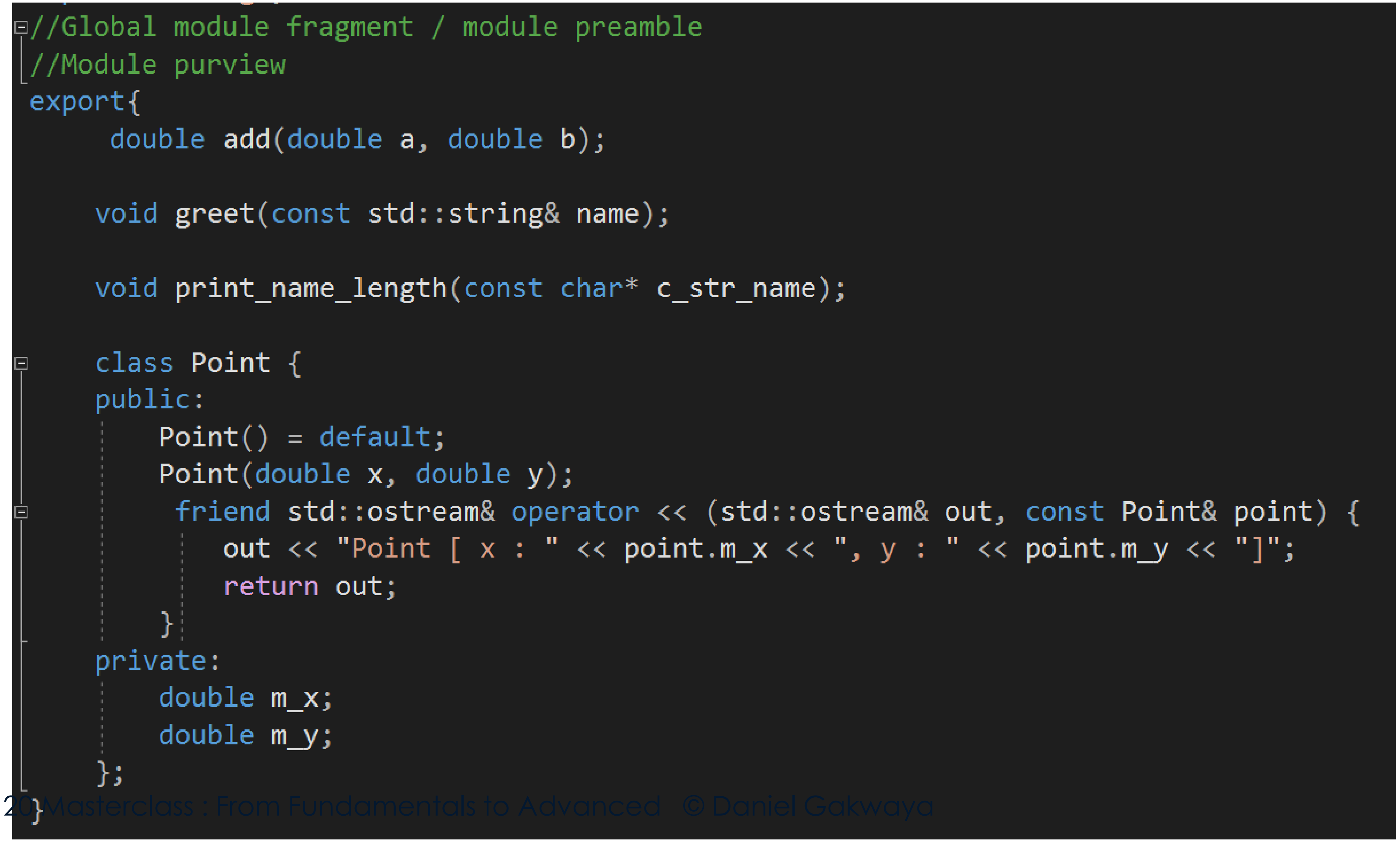


```
export{

    void greet(const std::string& name) {
        std::string dest;
        dest = "Hello ";
        dest.append(name);
        std::cout << dest << std::endl;
    }

    void print_name_length(const char* c_str_name) {

        std::cout << "Length : " << std::strlen(c_str_name) << std::endl;
    }

}
```

Slide intentionally left empty

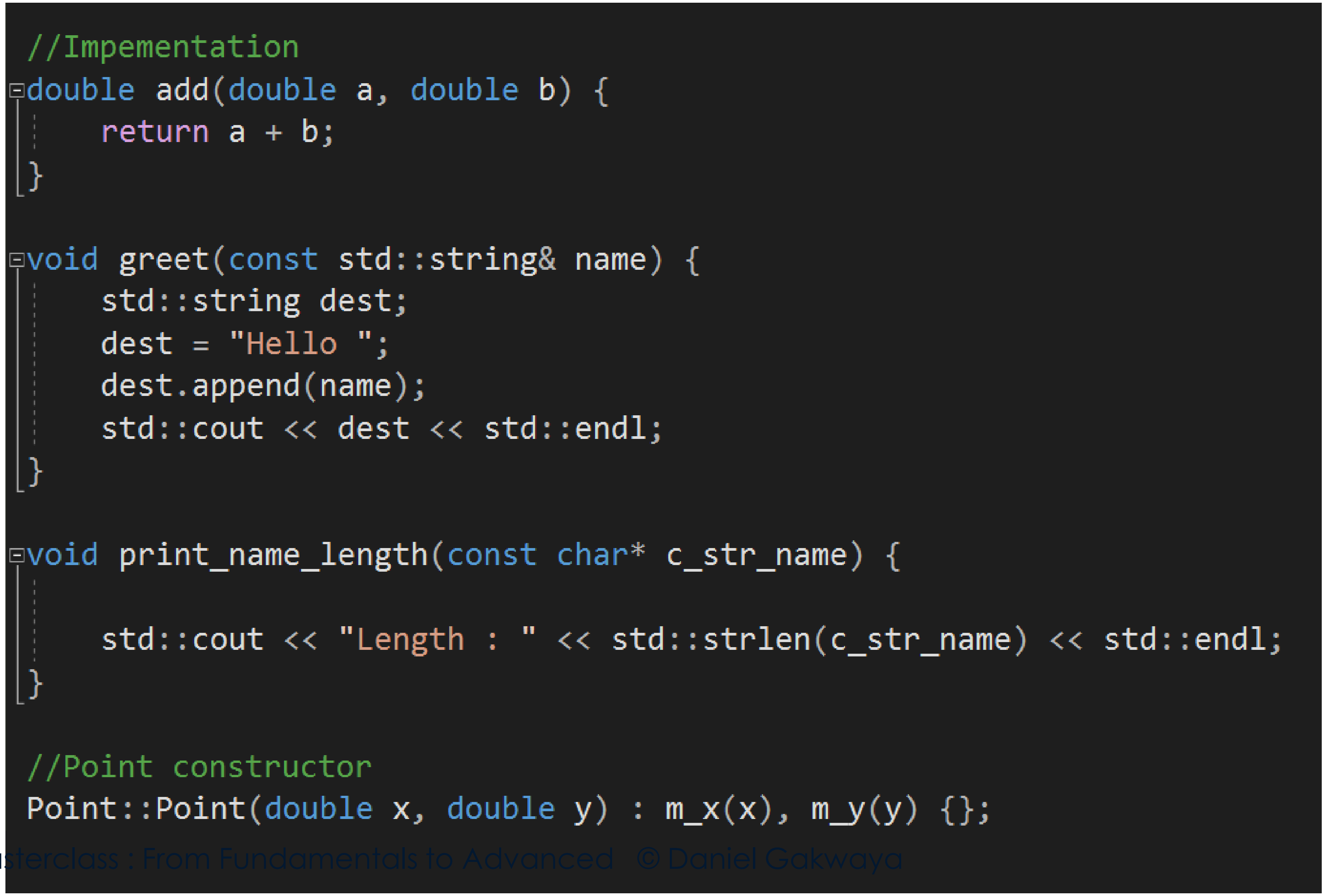
## Separating the interface from the implementation

## Interface



```
//Global module fragment / module preamble
//Module purview
export{
     double add(double a, double b);

    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);

    class Point {
    public:
        Point() = default;
        Point(double x, double y);
         friend std::ostream& operator << (std::ostream& out, const Point& point) {
            out << "Point [ x : " << point.m_x << ", y : " << point.m_y << "]";
            return out;
        }
    private:
        double m_x;
        double m_y;
    };
}
```

## Implementation

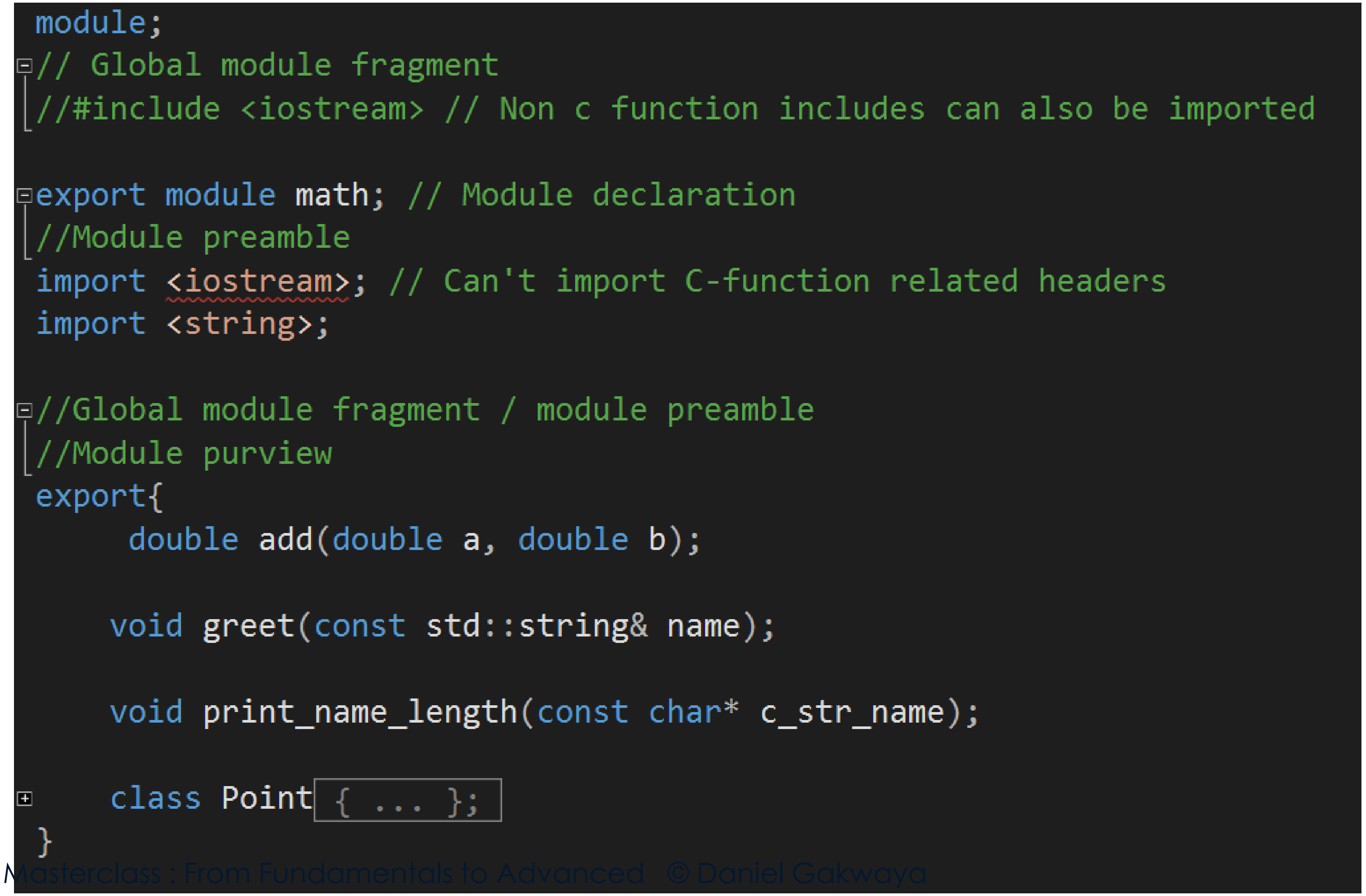


```
//Impementation
double add(double a, double b) {
    return a + b;
}

void greet(const std::string& name) {
    std::string dest;
    dest = "Hello ";
    dest.append(name);
    std::cout << dest << std::endl;
}

void print_name_length(const char* c_str_name) {

    std::cout << "Length : " << std::strlen(c_str_name) << std::endl;
}

//Point constructor
Point::Point(double x, double y) : m_x(x), m_y(y) {};
```

Slide intentionally left empty

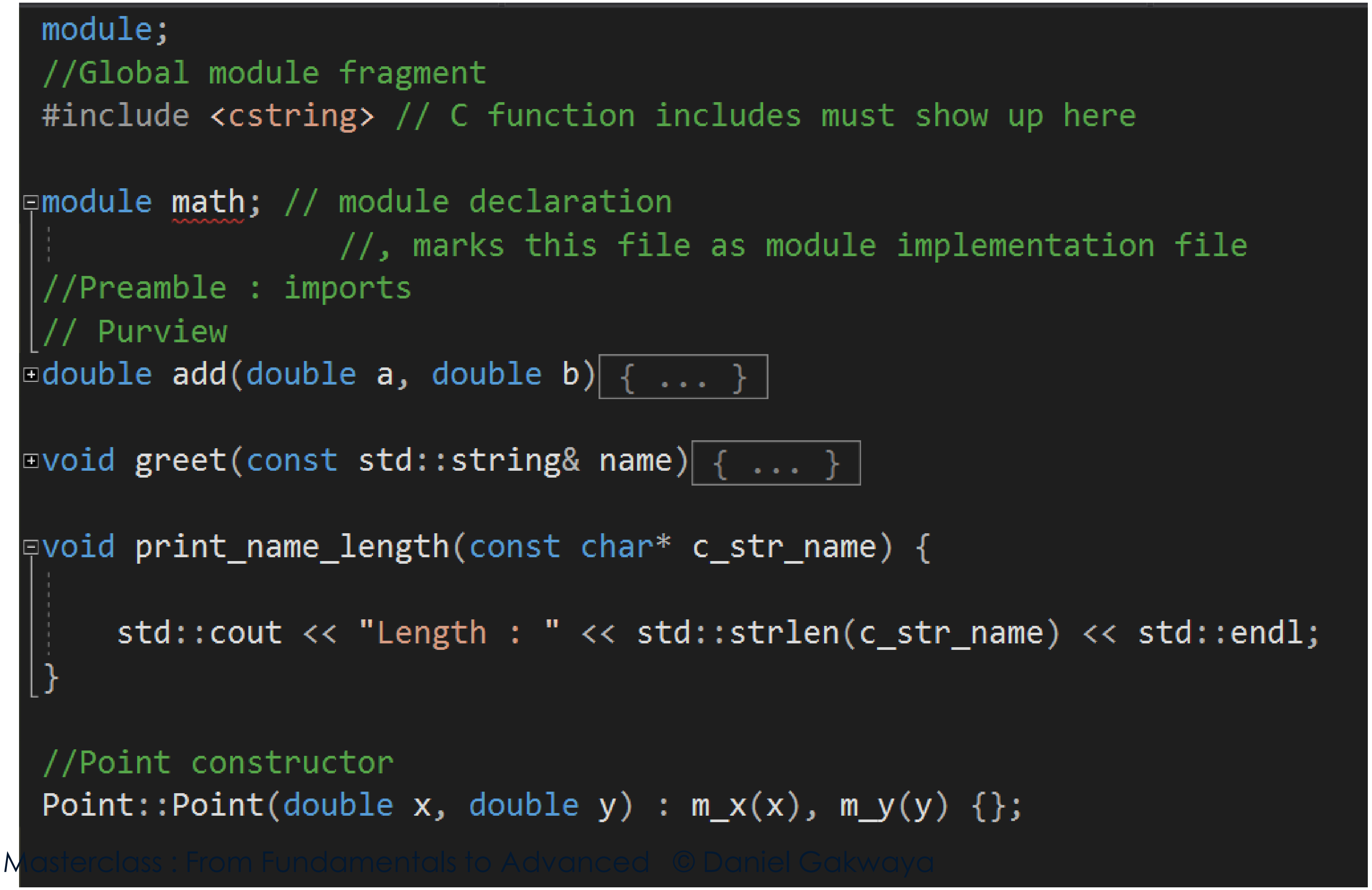
## Separating the interface from the implementation : Different files

## Interface : lives in module interface file (.ixx)



```
module;
// Global module fragment
//#include <iostream> // Non c function includes can also be imported

export module math; // Module declaration
//Module preamble
import <iostream>; // Can't import C-function related headers
import <string>;

//Global module fragment / module preamble
//Module purview
export{
     double add(double a, double b);

    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);

    class Point { ... };
}
```

## Module implementation file

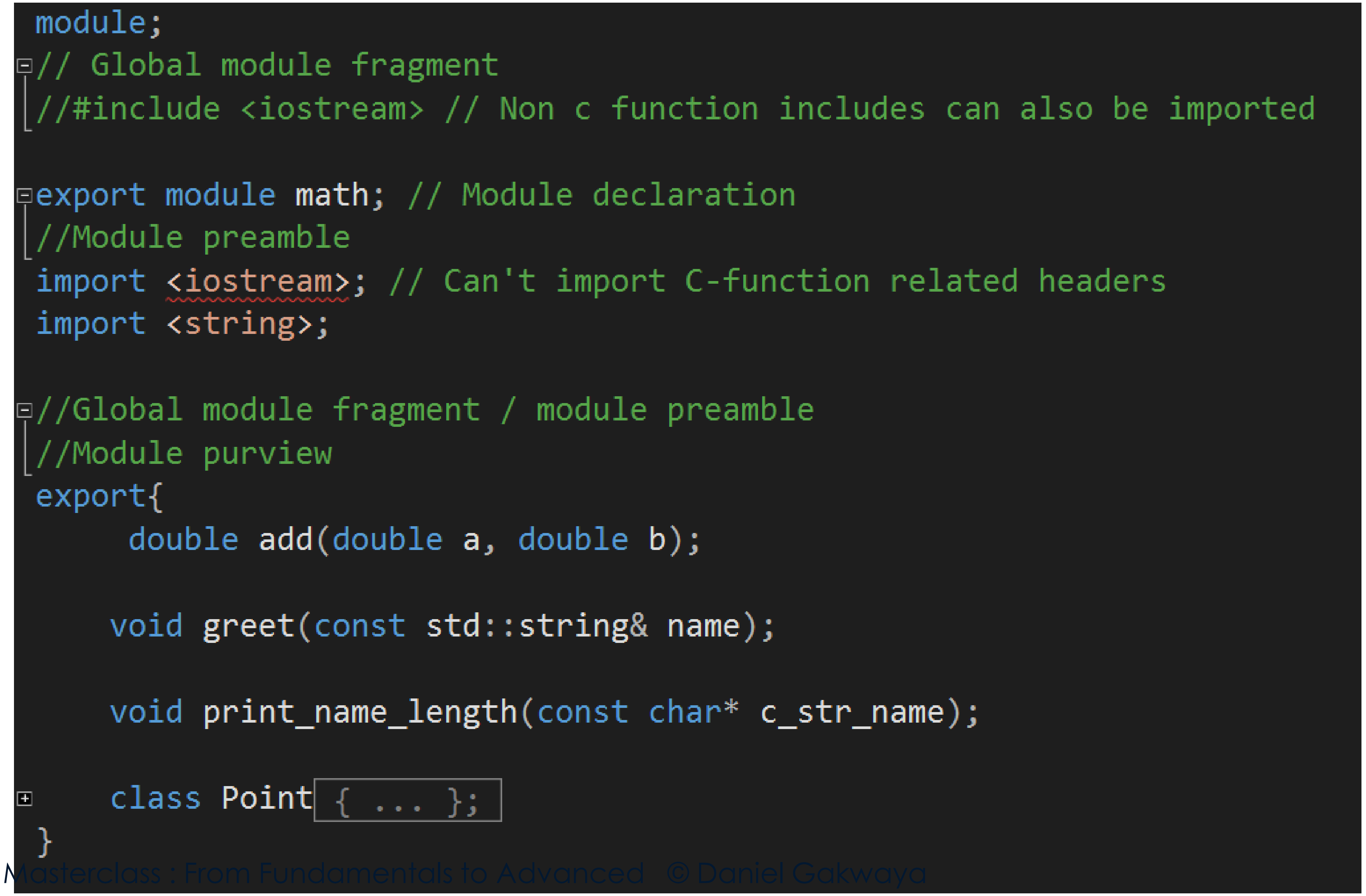


```
module;
//Global module fragment
#include <cstring> // C function includes must show up here

module math; // module declaration
                //, marks this file as module implementation file
//Preamble : imports
// Purview
double add(double a, double b) { ... }

void greet(const std::string& name) { ... }

void print_name_length(const char* c_str_name) {

    std::cout << "Length : " << std::strlen(c_str_name) << std::endl;
}

//Point constructor
Point::Point(double x, double y) : m_x(x), m_y(y) {};
```

Slide intentionally left empty

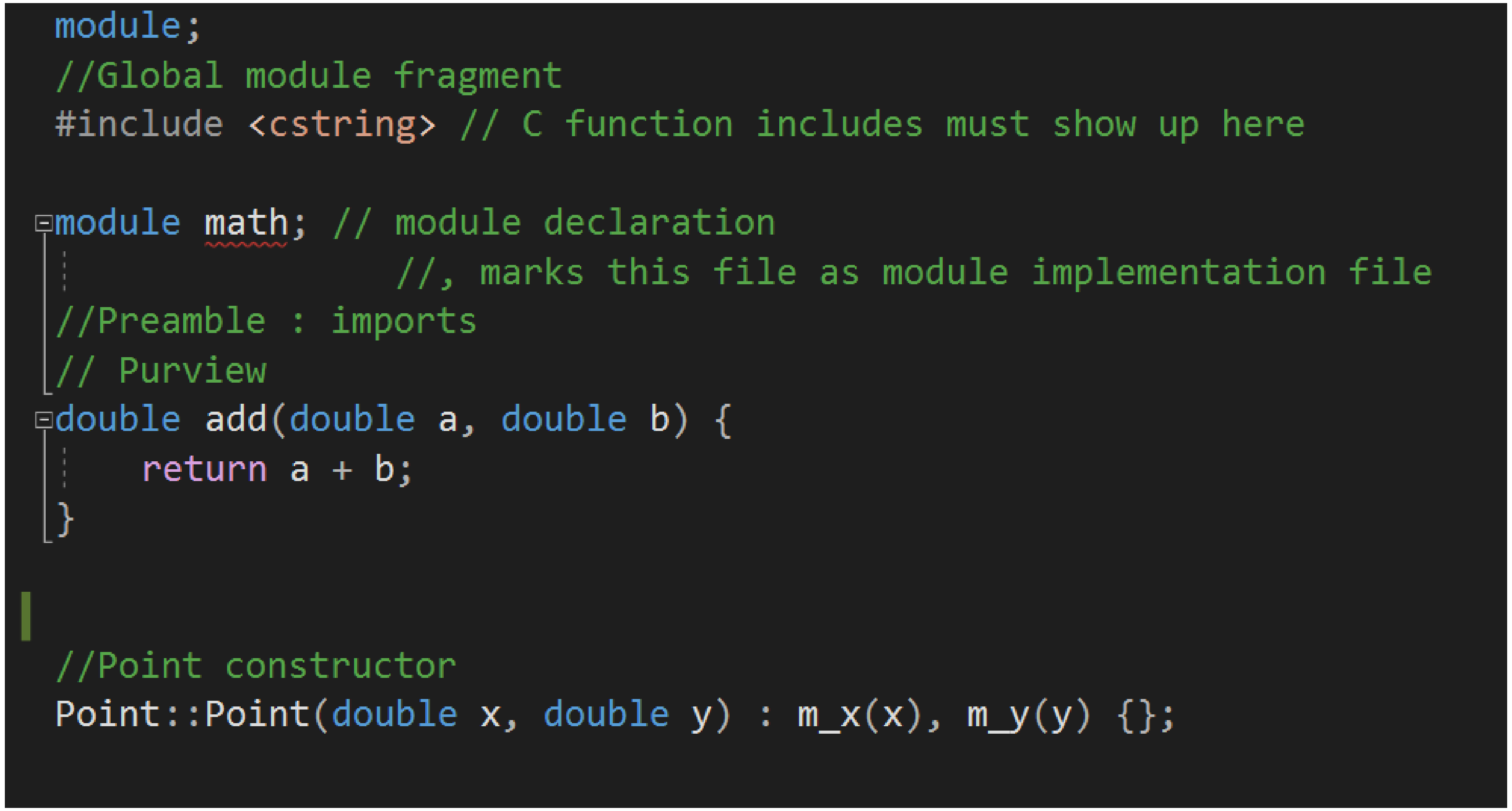
## Multiple Module Implementation Files

## Interface : lives in module interface file (.ixx)



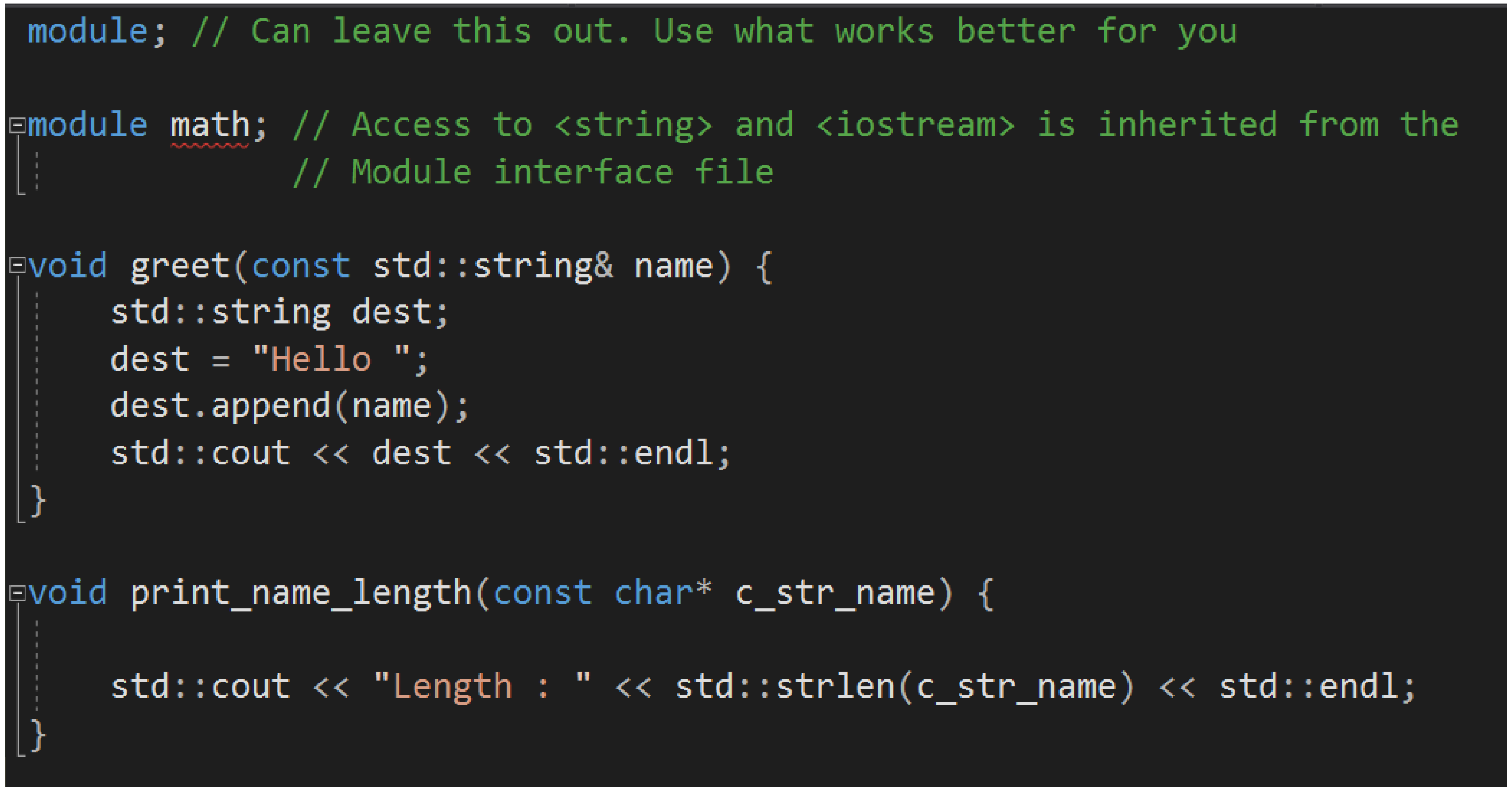
```
module;
// Global module fragment
//#include <iostream> // Non c function includes can also be imported

export module math; // Module declaration
//Module preamble
import <iostream>; // Can't import C-function related headers
import <string>;

//Global module fragment / module preamble
//Module purview
export{
     double add(double a, double b);

    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);

    class Point { ... };
}
```

## math.cpp



```cpp
module;
//Global module fragment
#include <cstring> // C function includes must show up here

module math; // module declaration
                //, marks this file as module implementation file
//Preamble : imports
// Purview
double add(double a, double b) {
    return a + b;
}


//Point constructor
Point::Point(double x, double y) : m_x(x), m_y(y) {};
```

## print.cpp

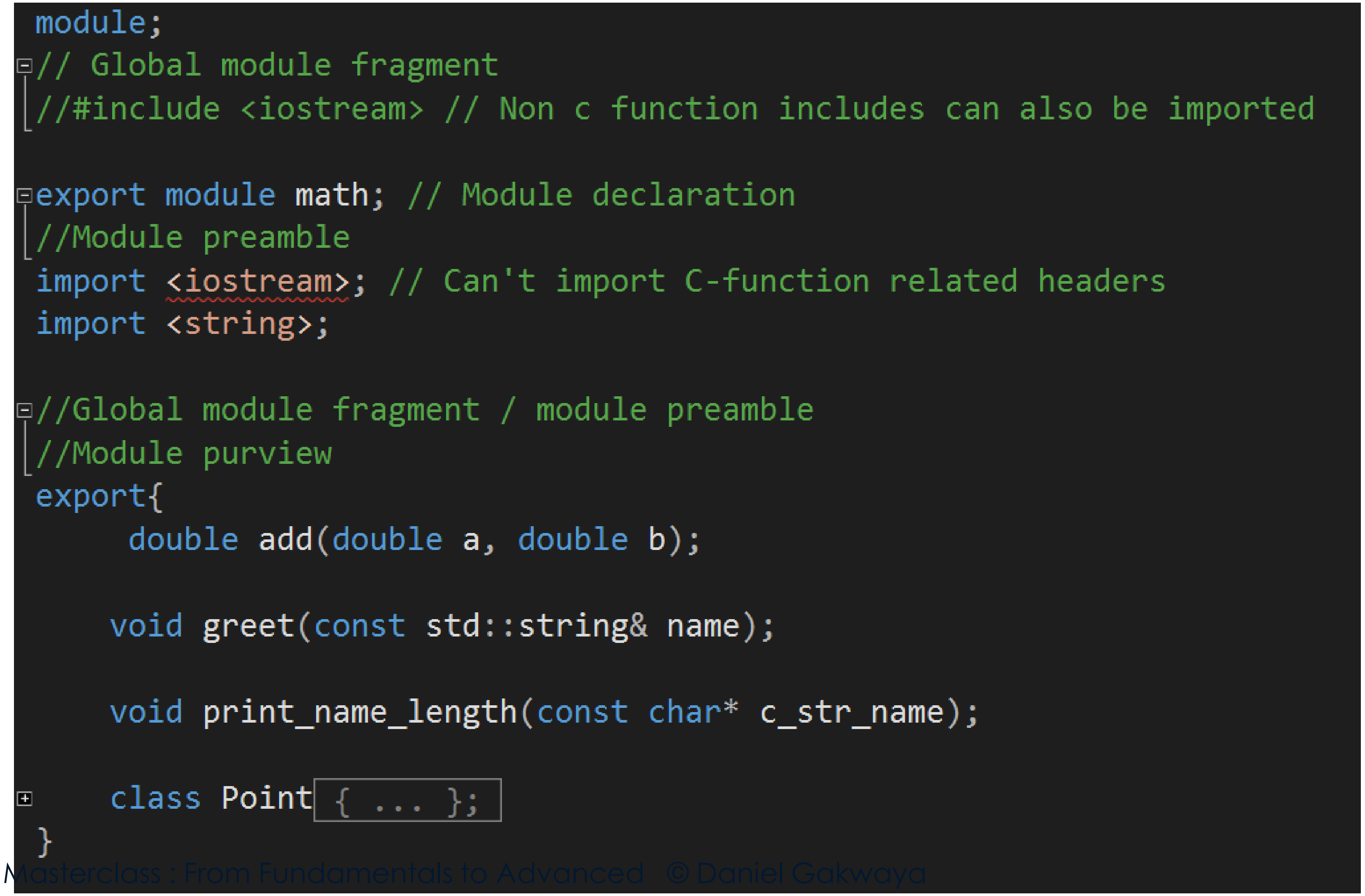


```cpp
module; // Can leave this out. Use what works better for you

module math; // Access to <string> and <iostream> is inherited from the
             // Module interface file

void greet(const std::string& name) {
    std::string dest;
    dest = "Hello ";
    dest.append(name);
    std::cout << dest << std::endl;
}

void print_name_length(const char* c_str_name) {

    std::cout << "Length : " << std::strlen(c_str_name) << std::endl;
}
```

Slide intentionally left empty

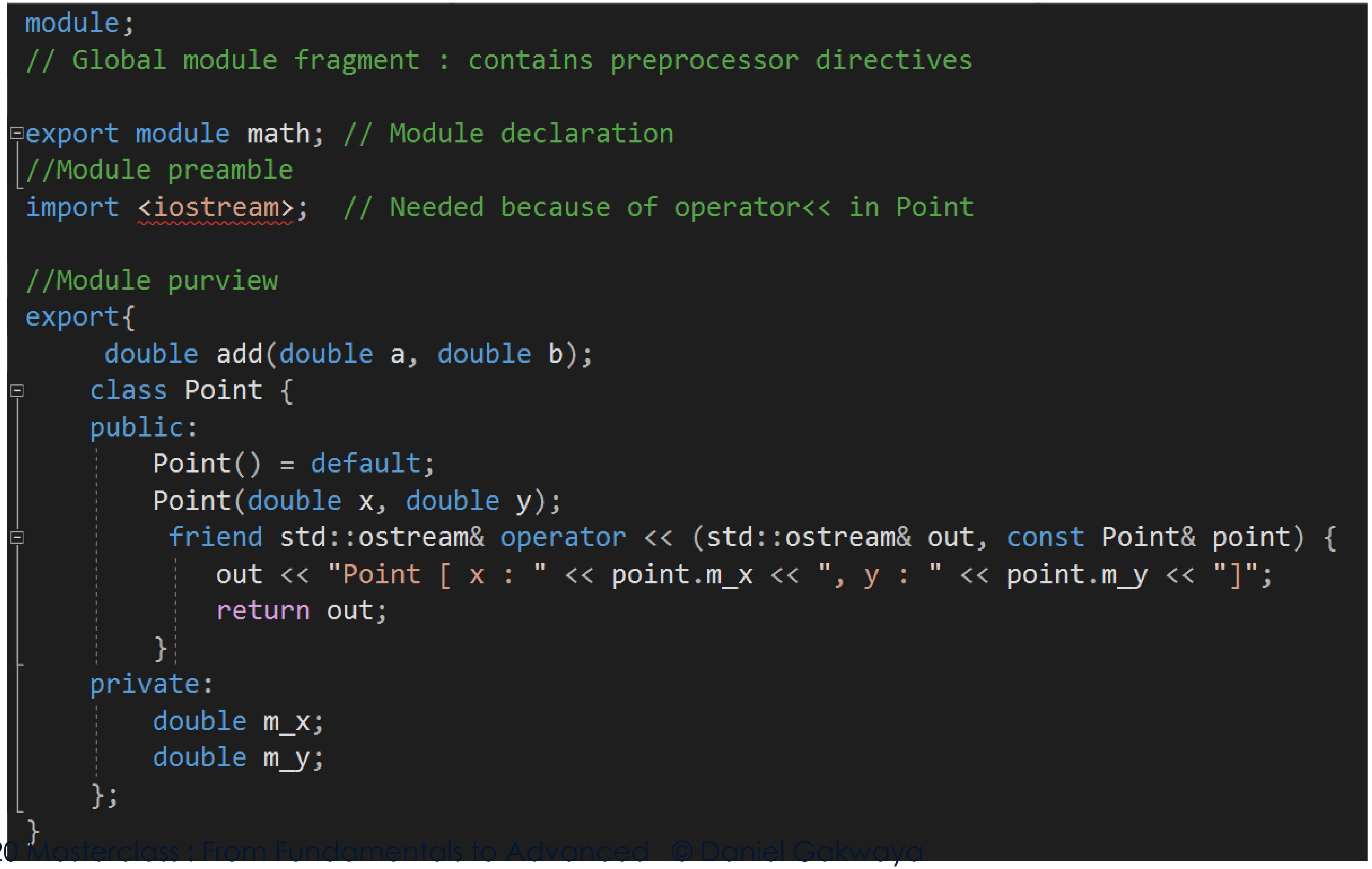
## Multiple Interface Files

## math.ixx (OLD)



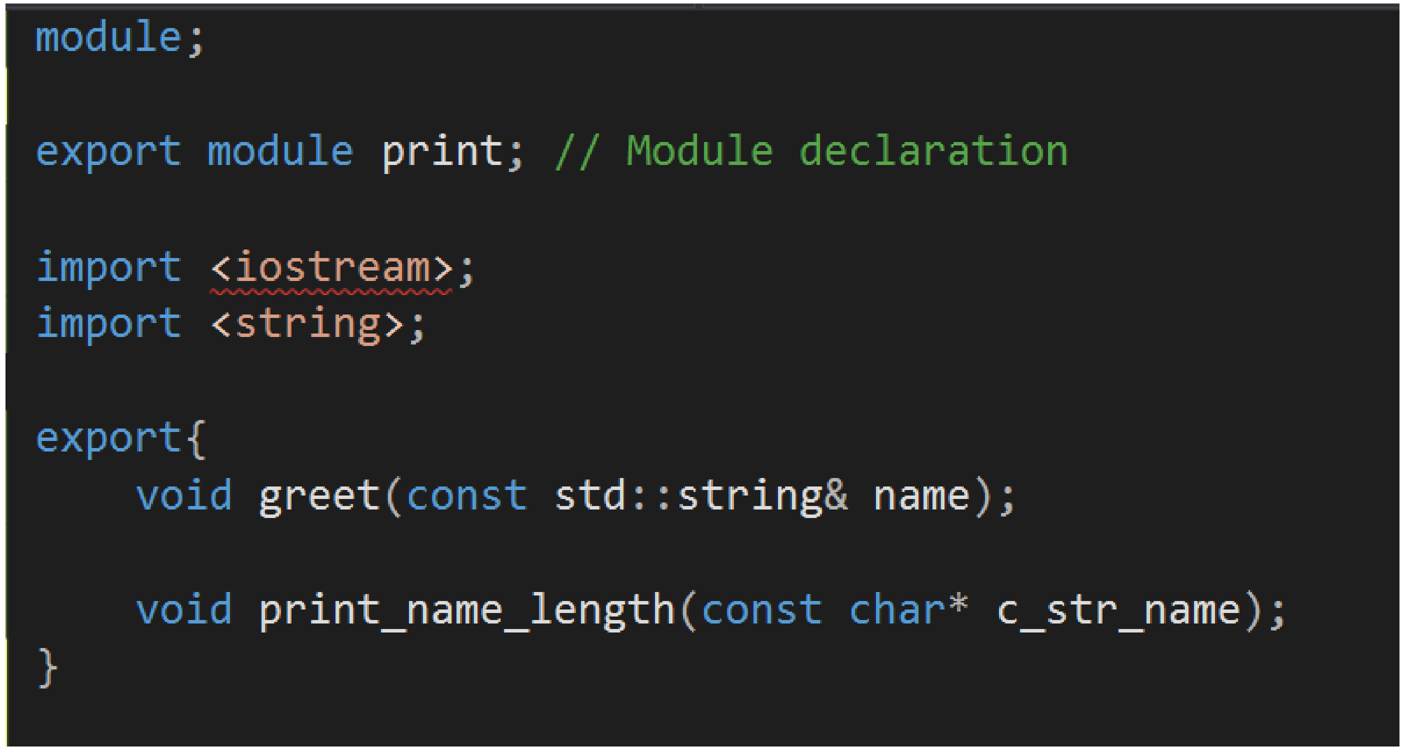
```
module;
// Global module fragment
//#include <iostream> // Non c function includes can also be imported

export module math; // Module declaration
//Module preamble
import <iostream>; // Can't import C-function related headers
import <string>;

//Global module fragment / module preamble
//Module purview
export{
     double add(double a, double b);

    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);

    class Point { ... };
}
```

## math.ixx



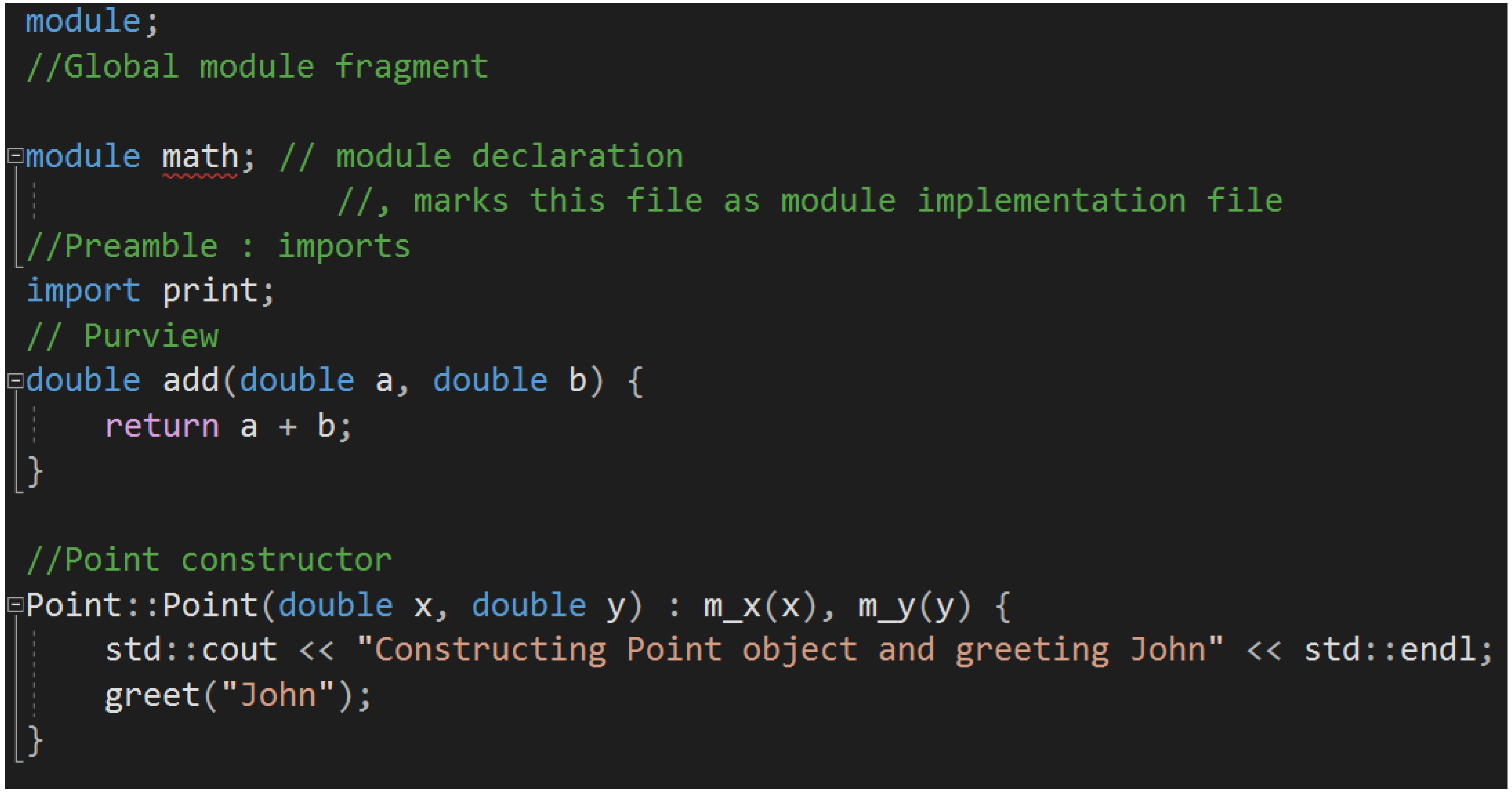
```cpp
module;
// Global module fragment : contains preprocessor directives

export module math; // Module declaration
//Module preamble
import <iostream>;  // Needed because of operator<< in Point

//Module purview
export{
     double add(double a, double b);
    class Point {
    public:
        Point() = default;
        Point(double x, double y);
         friend std::ostream& operator << (std::ostream& out, const Point& point) {
            out << "Point [ x : " << point.m_x << ", y : " << point.m_y << "]";
            return out;
        }
    private:
        double m_x;
        double m_y;
    };
}
```

## print.ixx



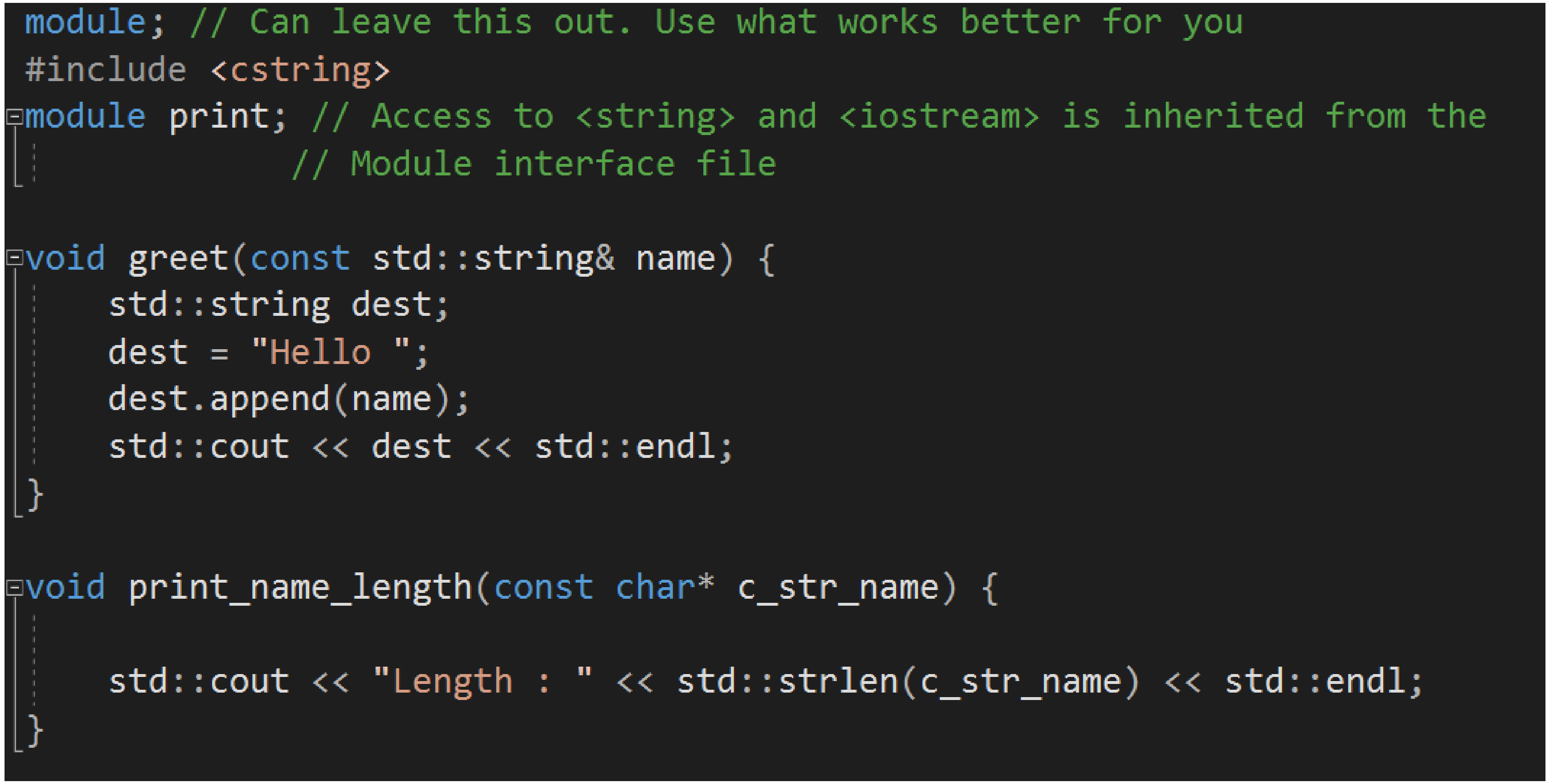
```
module;

export module print; // Module declaration

import <iostream>;
import <string>;

export{
    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);
}
```

## math.cpp



```cpp
module;
//Global module fragment

module math; // module declaration
               //, marks this file as module implementation file
//Preamble : imports
import print;
// Purview
double add(double a, double b) {
    return a + b;
}

//Point constructor
Point::Point(double x, double y) : m_x(x), m_y(y) {
    std::cout << "Constructing Point object and greeting John" << std::endl;
    greet("John");
}
```

## print.cpp

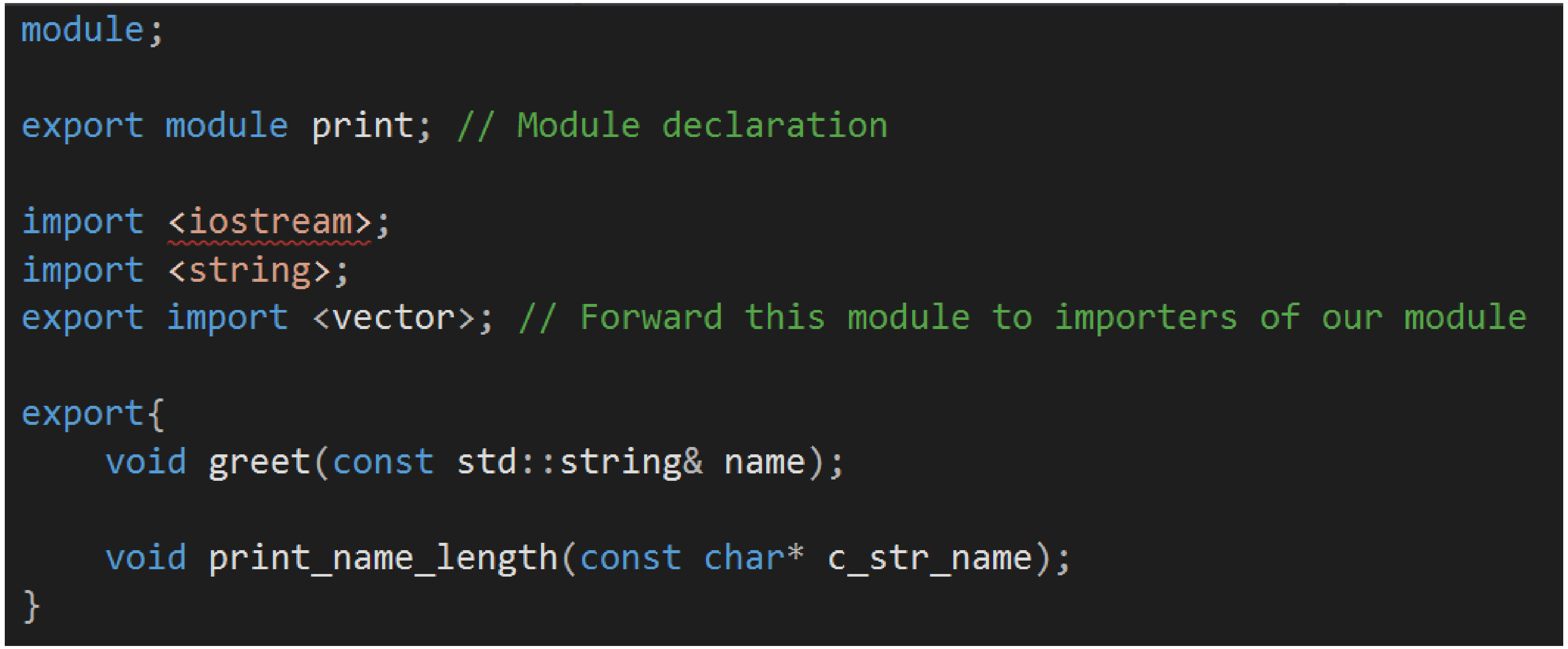


```cpp
module; // Can leave this out. Use what works better for you
#include <cstring>
module print; // Access to <string> and <iostream> is inherited from the
              // Module interface file

void greet(const std::string& name) {
    std::string dest;
    dest = "Hello ";
    dest.append(name);
    std::cout << dest << std::endl;
}

void print_name_length(const char* c_str_name) {

    std::cout << "Length : " << std::strlen(c_str_name) << std::endl;
}
```

Slide intentionally left empty

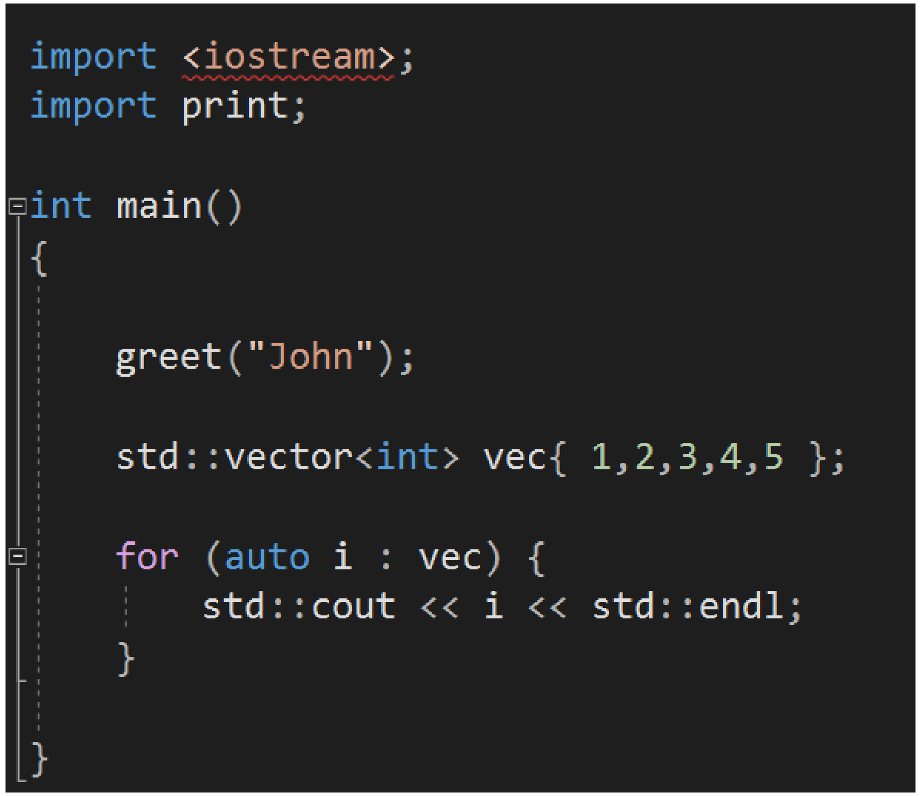
## Export Import

## print.ixx



```cpp
module;

export module print; // Module declaration

import <iostream>;
import <string>;
export import <vector>; // Forward this module to importers of our module

export{
    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);
}
```

## main.cpp



```
import <iostream>;
import print;

int main()
{

    greet("John");

    std::vector<int> vec{ 1,2,3,4,5 };

    for (auto i : vec) {
        std::cout << i << std::endl;
    }

}
```

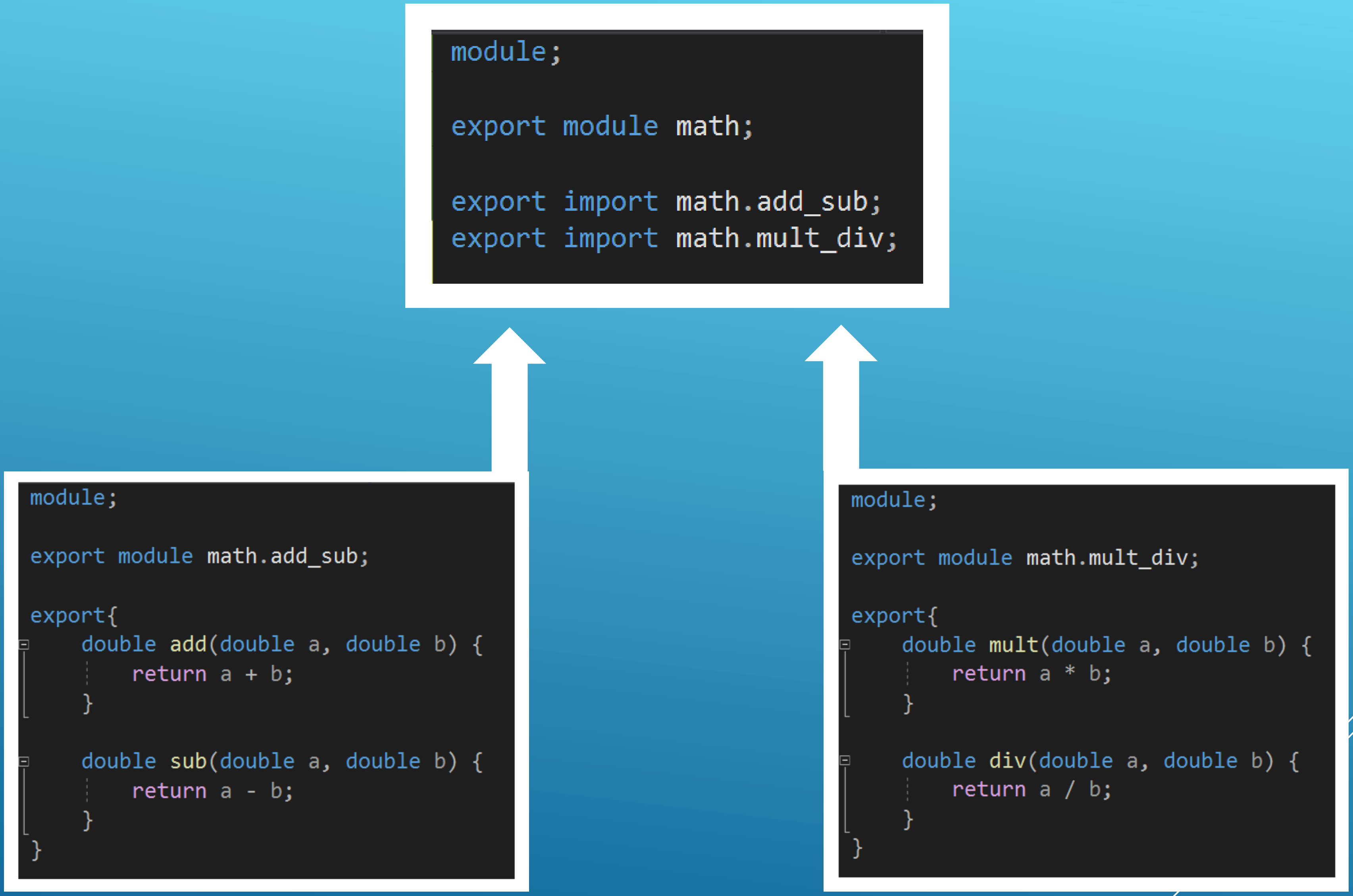
Slide intentionally left empty

## Sub-modules

math

math.add_sub

math.mult_div

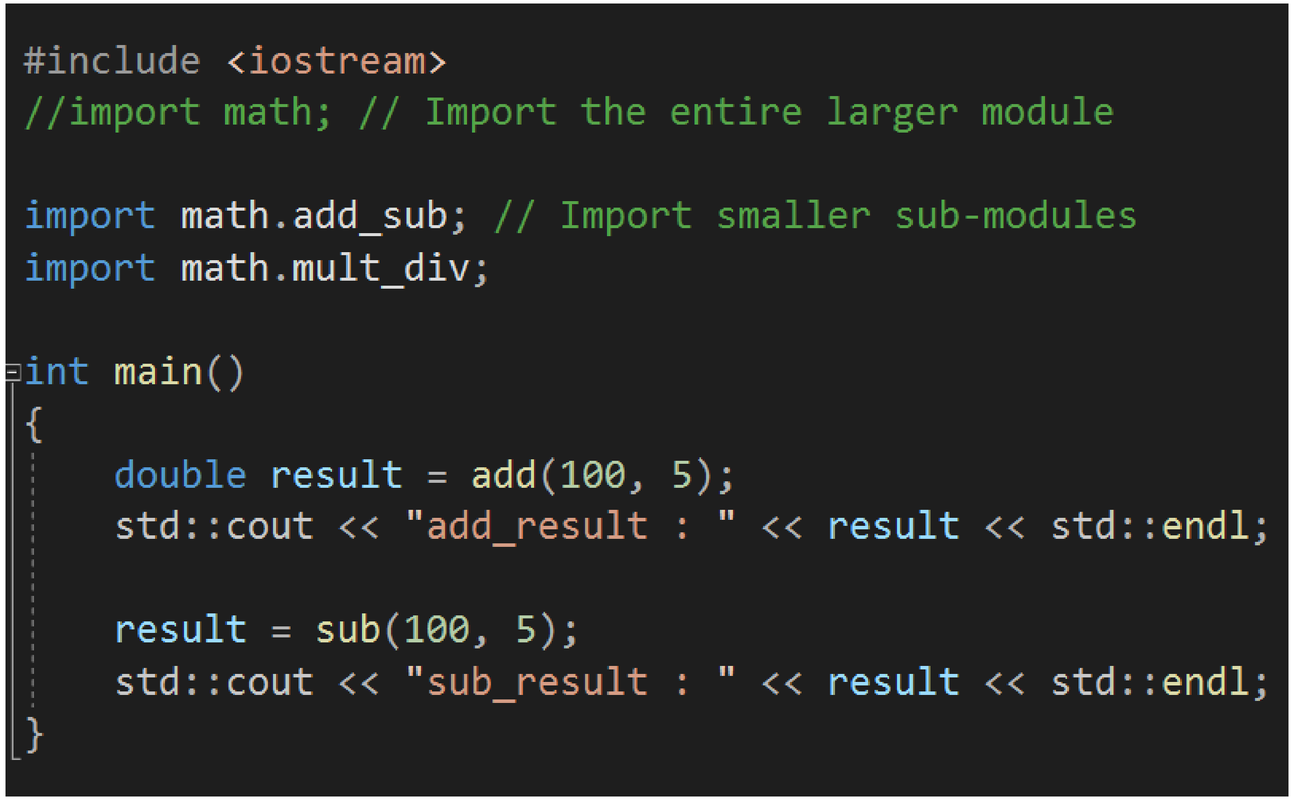


```cpp
module;

export module math;

export import math.add_sub;
export import math.mult_div;
```

```cpp
module;

export module math.add_sub;

export{
    double add(double a, double b) {
        return a + b;
    }

    double sub(double a, double b) {
        return a - b;
    }
}
```

```cpp
module;

export module math.mult_div;

export{
    double mult(double a, double b) {
        return a * b;
    }

    double div(double a, double b) {
        return a / b;
    }
}
```

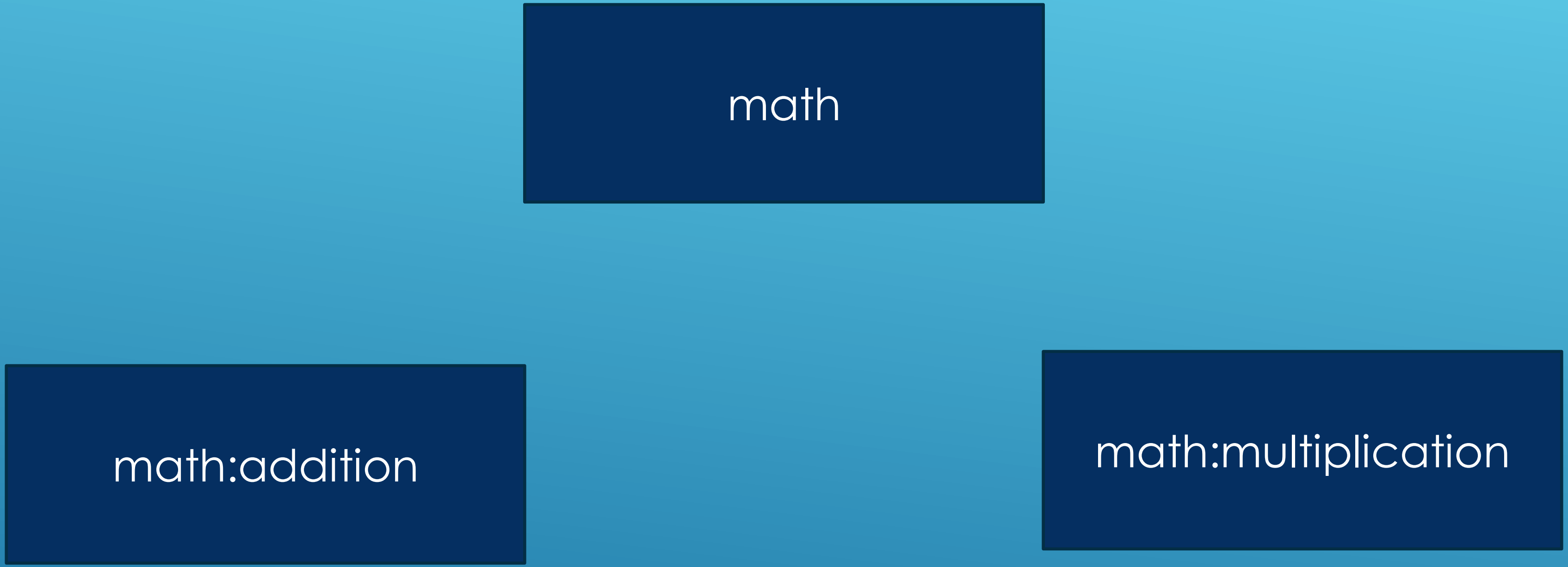
## main.cpp

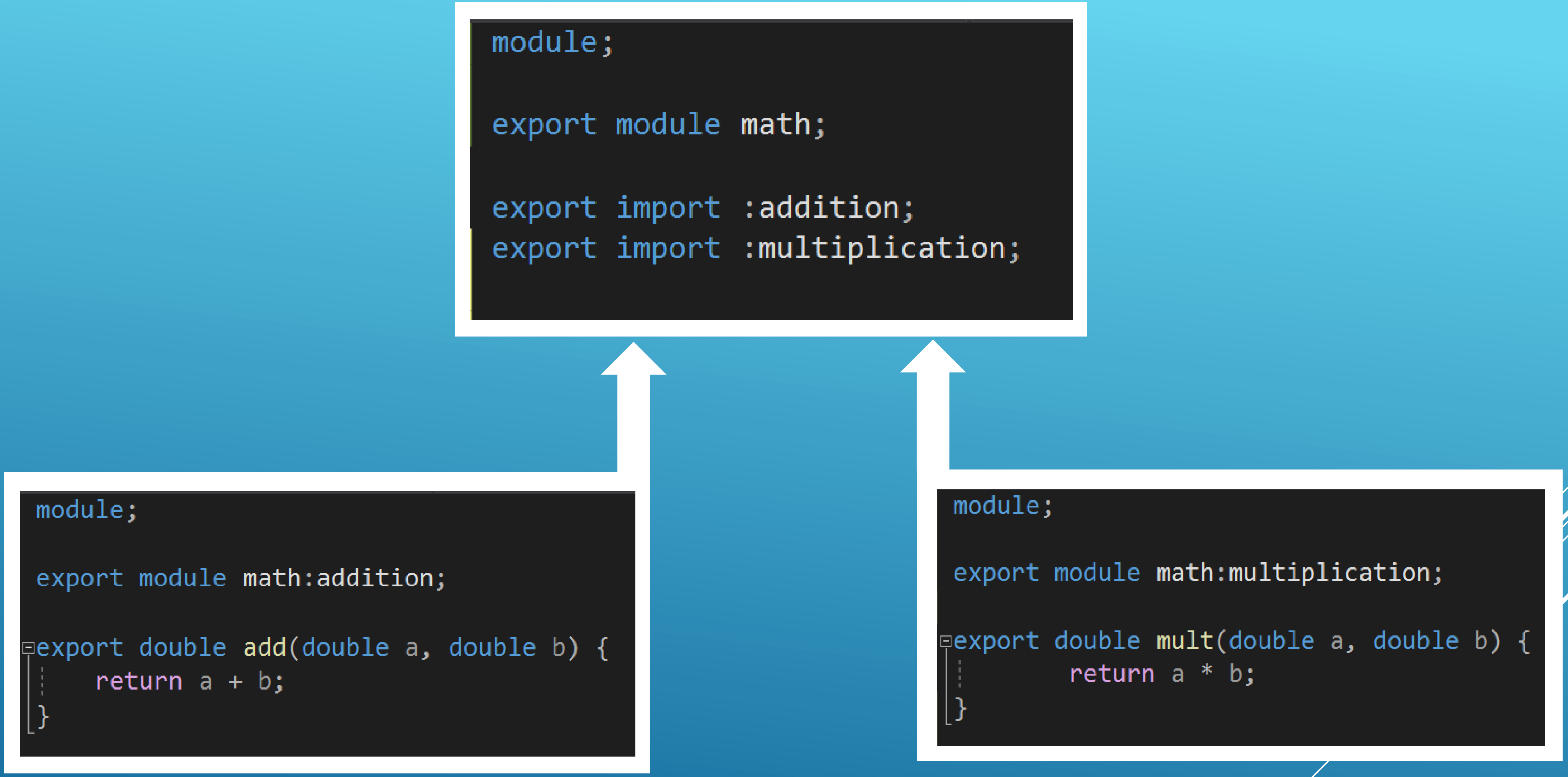


```cpp
#include <iostream>
//import math; // Import the entire larger module

import math.add_sub; // Import smaller sub-modules
import math.mult_div;

int main()
{
    double result = add(100, 5);
    std::cout << "add_result : " << result << std::endl;

    result = sub(100, 5);
    std::cout << "sub_result : " << result << std::endl;
}
```

Slide intentionally left empty

## Module interface partitions


math
math:addition
math:multiplication

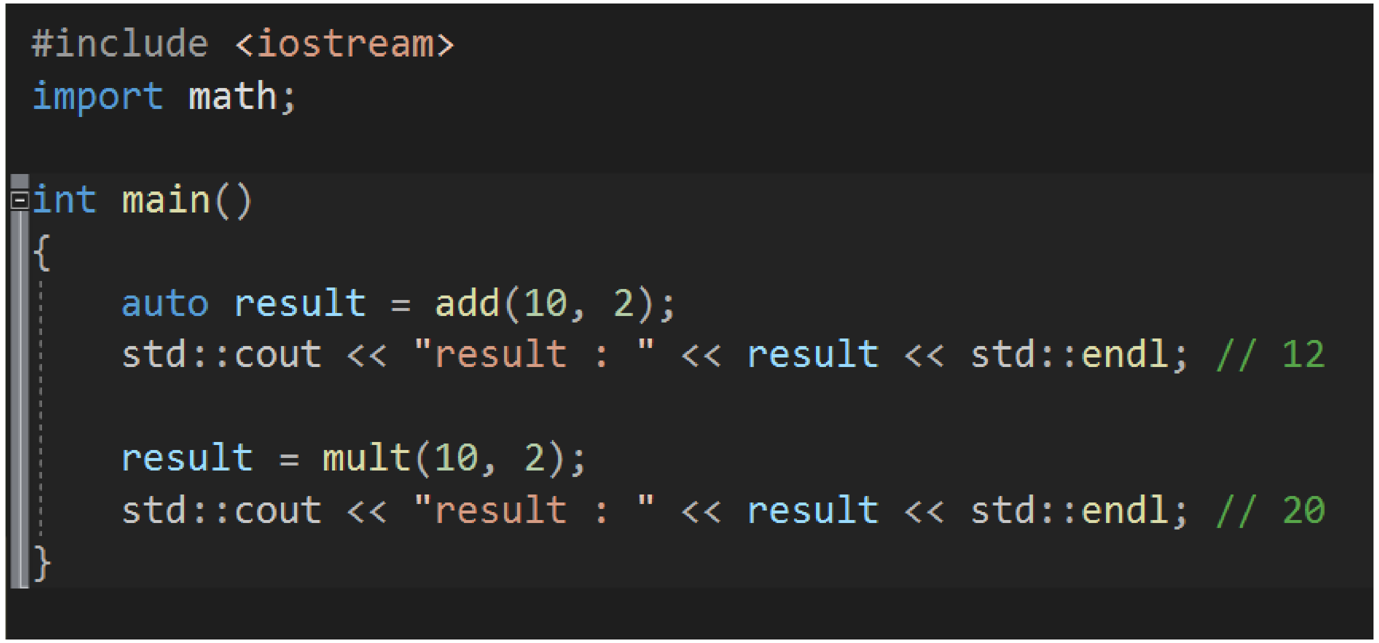


```cpp
module;

export module math;

export import :addition;
export import :multiplication;
```

```cpp
module;

export module math:addition;

export double add(double a, double b) {
    return a + b;
}
```

```cpp
module;

export module math:multiplication;

export double mult(double a, double b) {
        return a * b;
}
```

## main.cpp



```cpp
#include <iostream>
import math;

int main()
{
    auto result = add(10, 2);
    std::cout << "result : " << result << std::endl; // 12

    result = mult(10, 2);
    std::cout << "result : " << result << std::endl; // 20
}
```

## More on Modules

- Module Implementation Partitions
- Visiblility and Reachability
- …

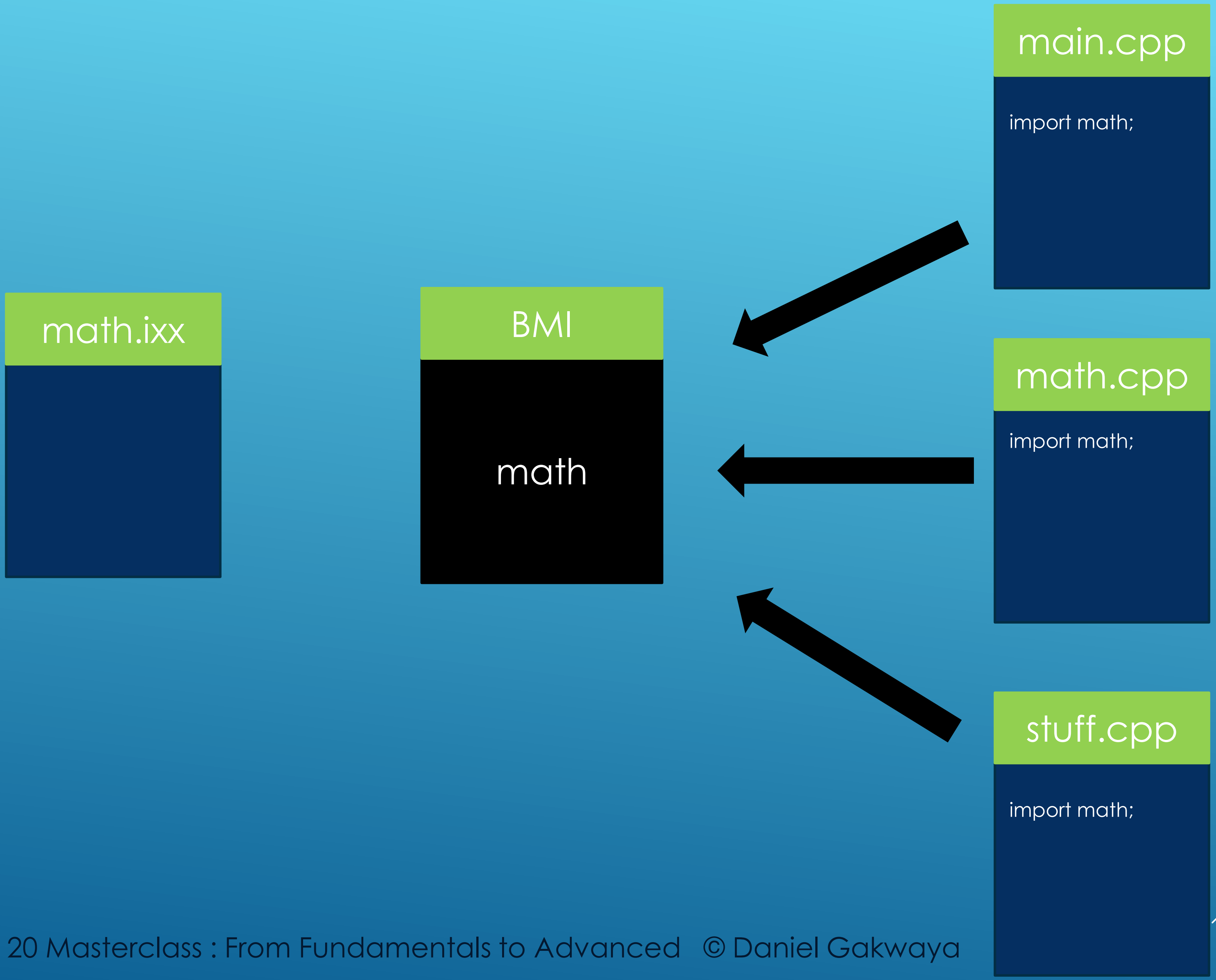
## References

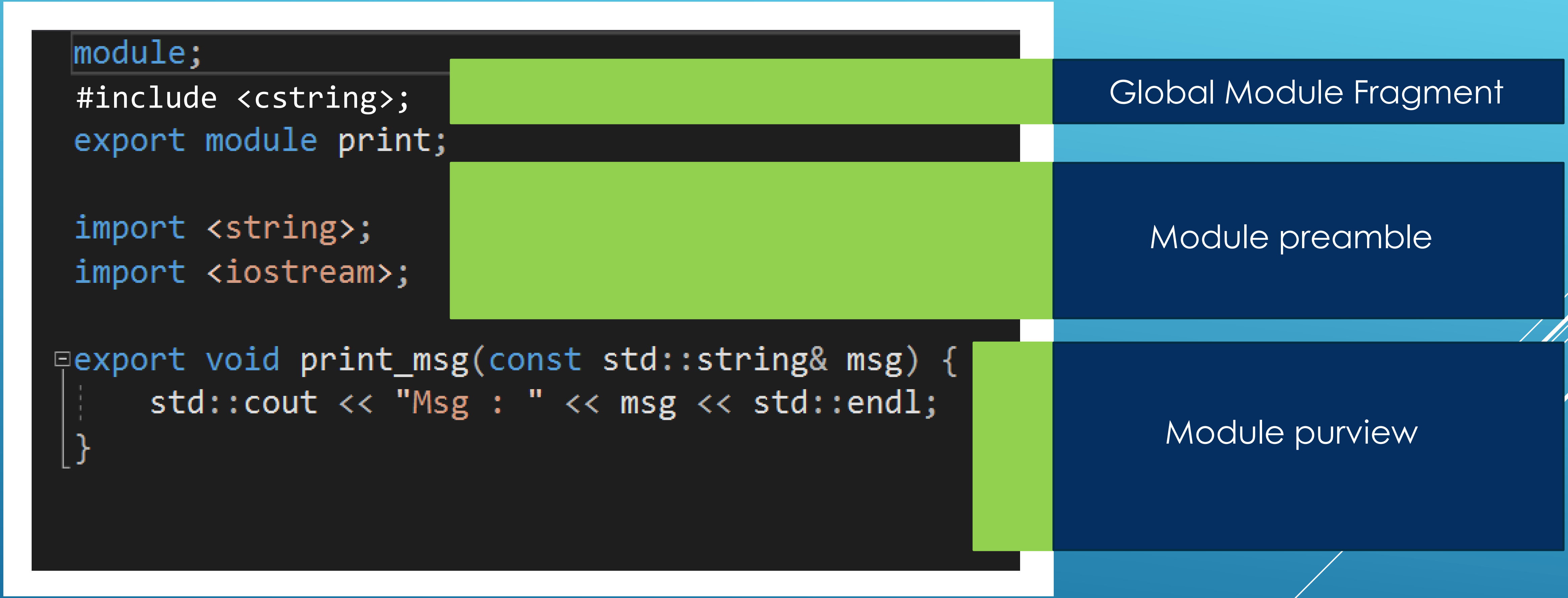
https://vector-of-bool.github.io/2019/03/10/modules-1.html

Slide intentionally left empty

# C++20 Modules : Summary

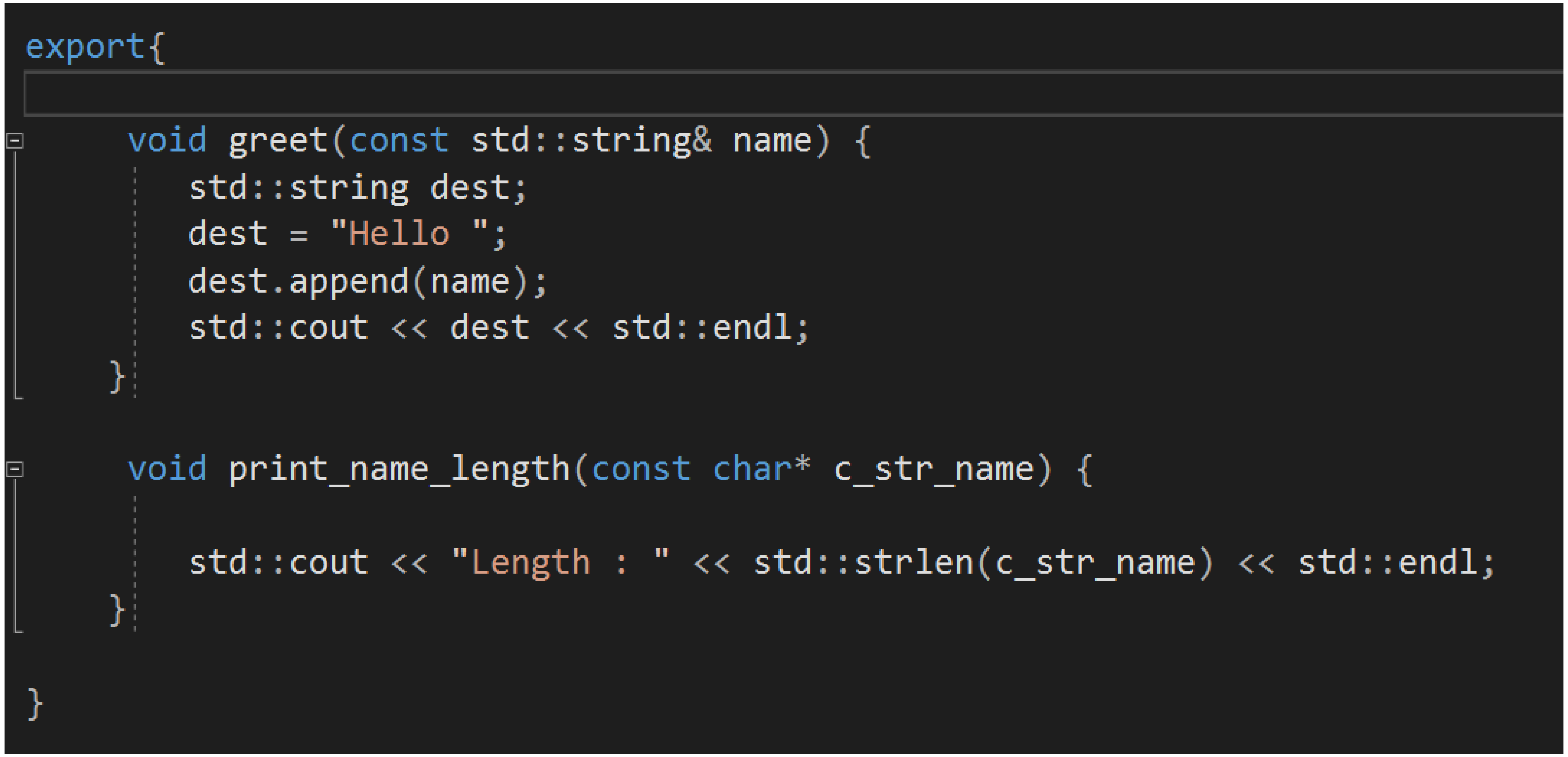
- One of the big C++20 features
- Changes the compilation model of C++ programs


math.ixx
BMI
math
main.cpp
import math;
math.cpp
import math;
stuff.cpp
import math;



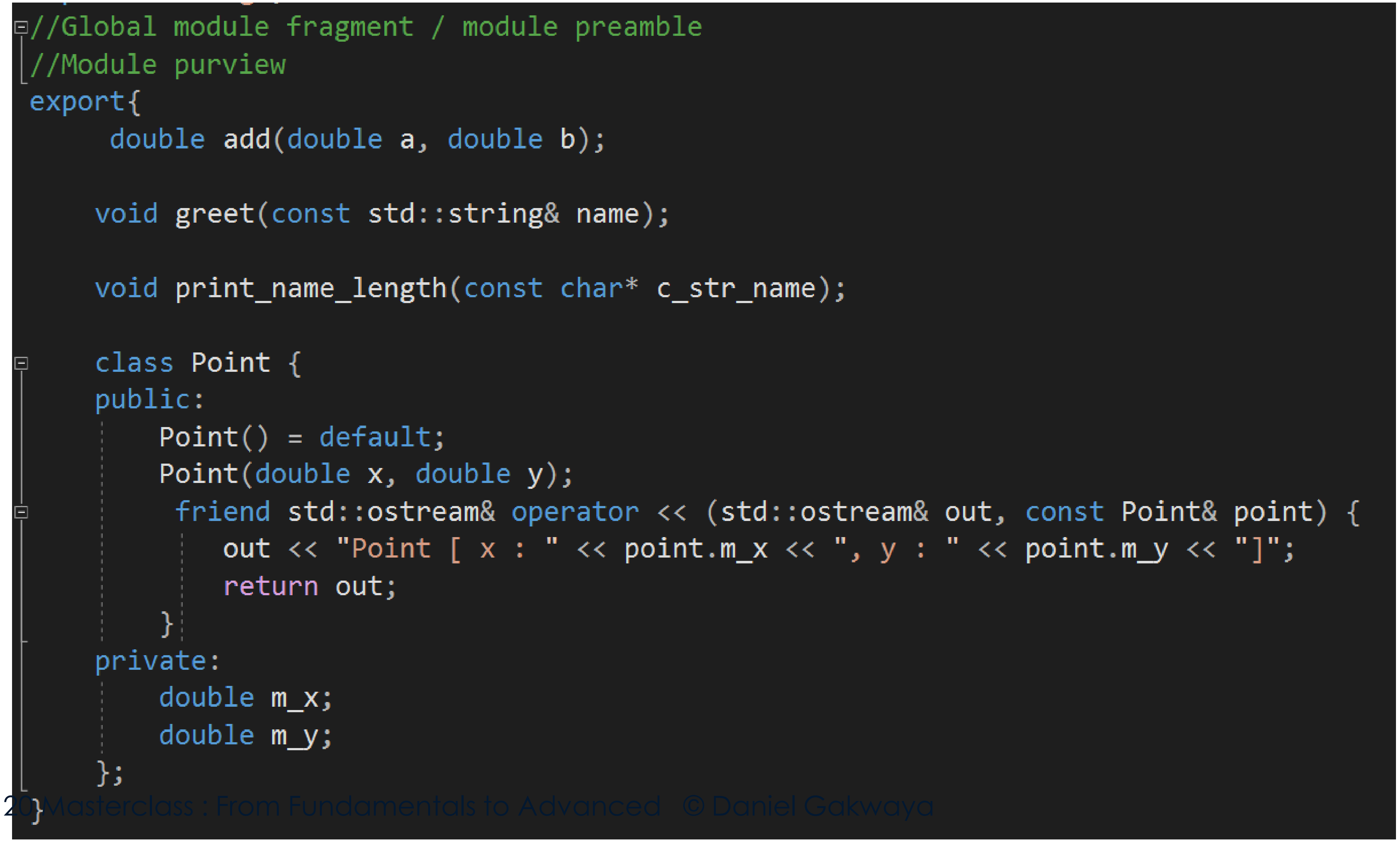
```cpp
module;
#include <cstring>;
export module print;

import <string>;
import <iostream>;

export void print_msg(const std::string& msg) {
    std::cout << "Msg : " << msg << std::endl;
}
```

## Block export



```
export{

    void greet(const std::string& name) {
        std::string dest;
        dest = "Hello ";
        dest.append(name);
        std::cout << dest << std::endl;
    }

    void print_name_length(const char* c_str_name) {

        std::cout << "Length : " << std::strlen(c_str_name) << std::endl;
    }

}
```

## Separate Interface from Implementation – Same File/ Different files

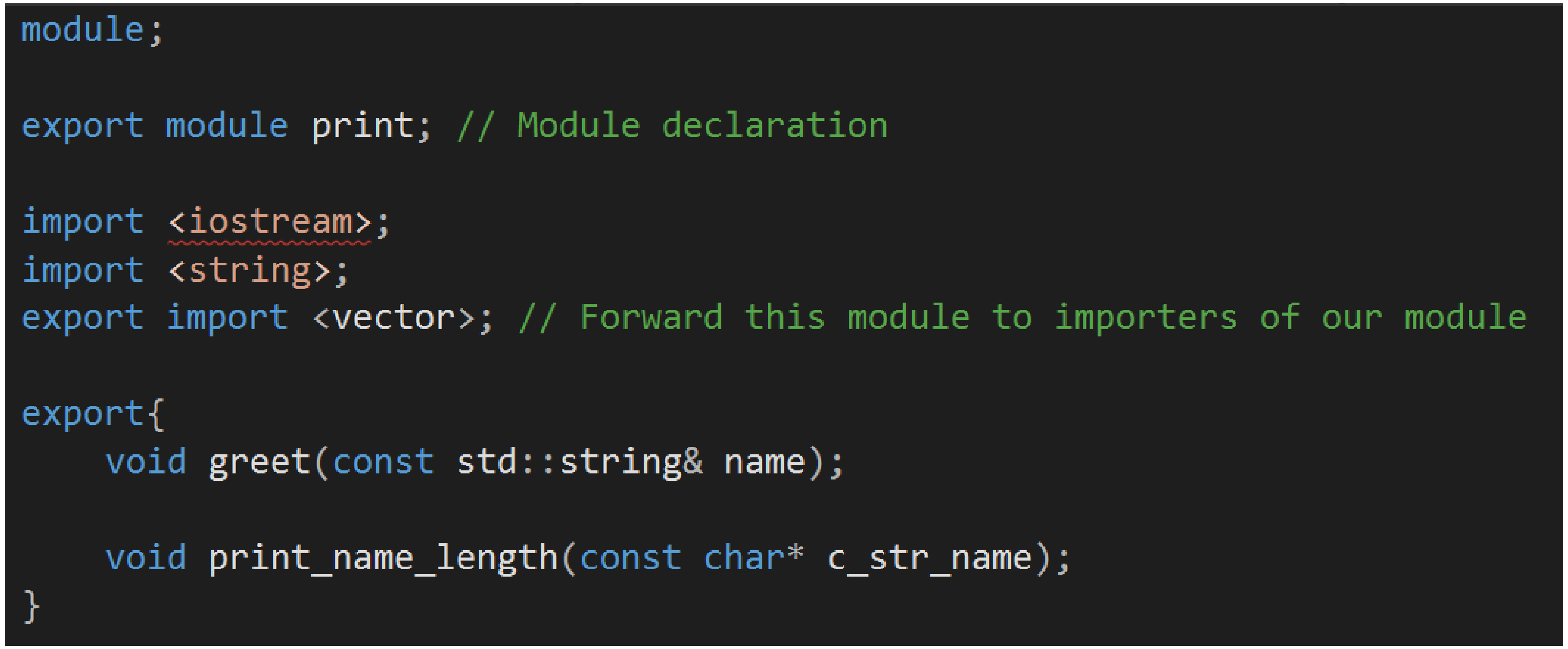


```
//Global module fragment / module preamble
//Module purview
export{
     double add(double a, double b);

    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);

    class Point {
    public:
        Point() = default;
        Point(double x, double y);
         friend std::ostream& operator << (std::ostream& out, const Point& point) {
            out << "Point [ x : " << point.m_x << ", y : " << point.m_y << "]";
            return out;
        }
    private:
        double m_x;
        double m_y;
    };
}
```

## Multiple implementation files for an interface

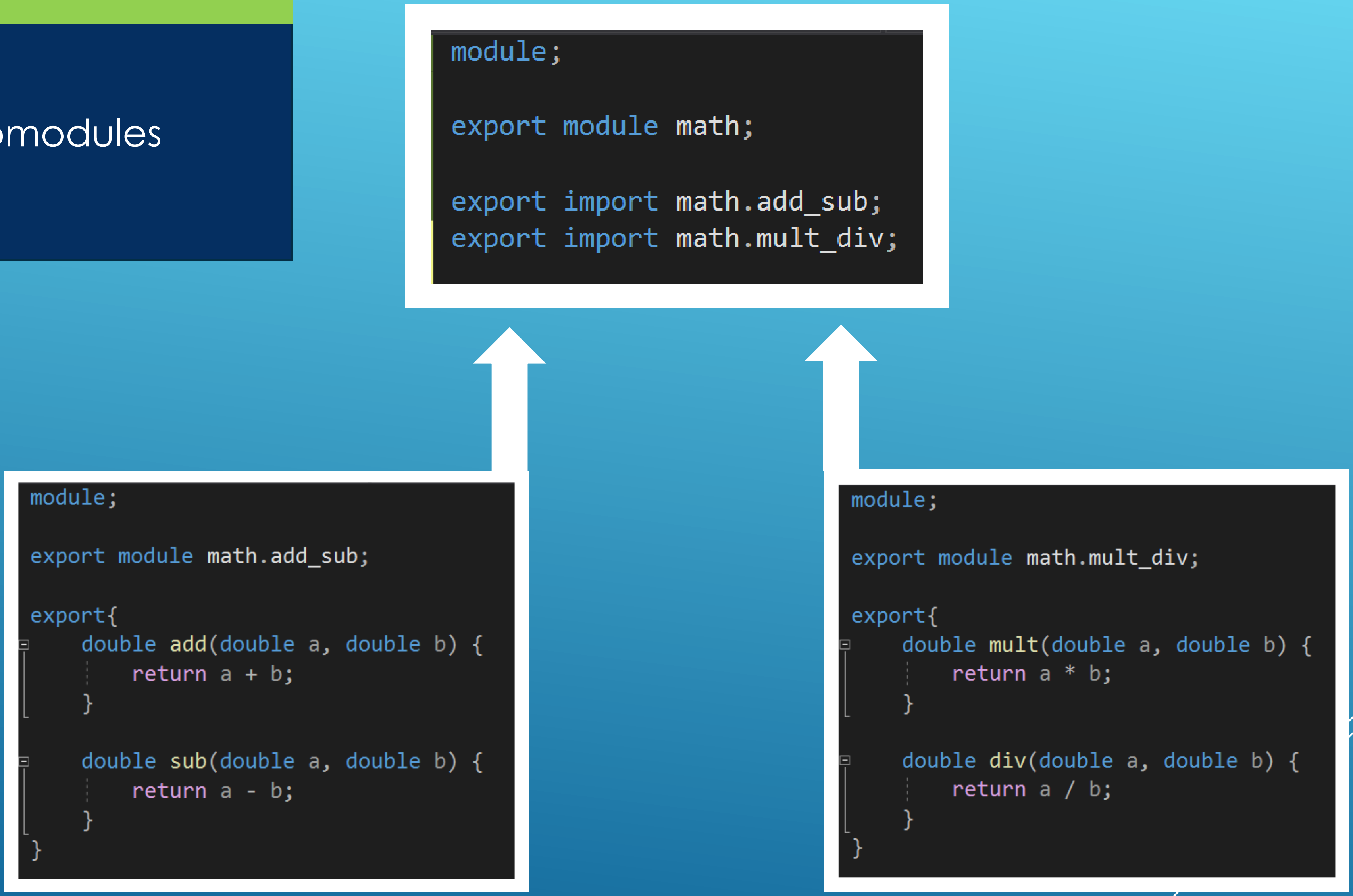
## Multiple Interffaces

## Export Import



```cpp
module;

export module print; // Module declaration

import <iostream>;
import <string>;
export import <vector>; // Forward this module to importers of our module

export{
    void greet(const std::string& name);

    void print_name_length(const char* c_str_name);
}
```

## Submodules

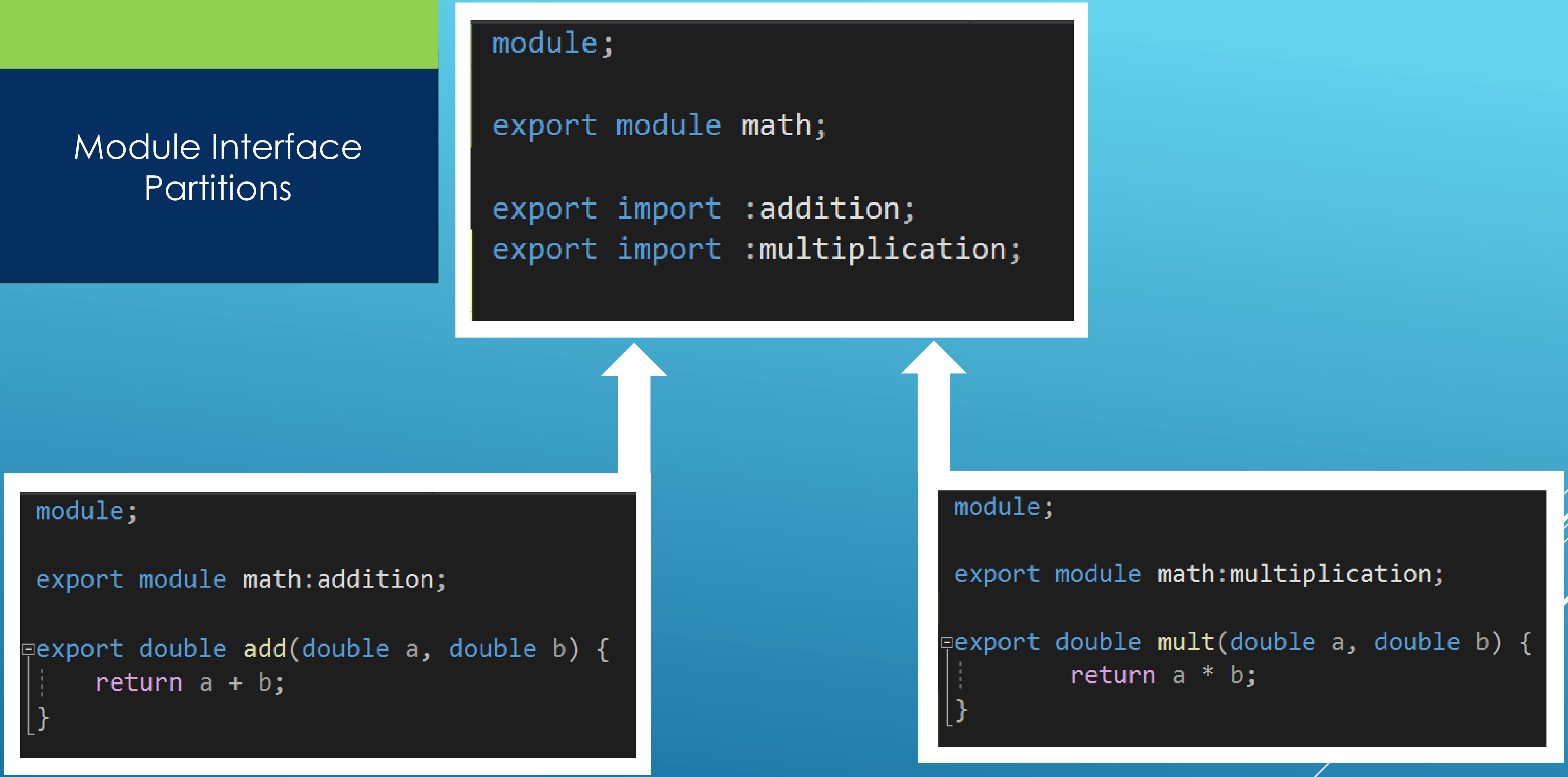


```
module;

export module math;

export import math.add_sub;
export import math.mult_div;
```

```
module;

export module math.add_sub;

export{
    double add(double a, double b) {
        return a + b;
    }

    double sub(double a, double b) {
        return a - b;
    }
}
```

```
module;

export module math.mult_div;

export{
    double mult(double a, double b) {
        return a * b;
    }

    double div(double a, double b) {
        return a / b;
    }
}
```

## Module Interface Partitions



```cpp
module;

export module math;

export import :addition;
export import :multiplication;
```

```cpp
module;

export module math:addition;

export double add(double a, double b) {
    return a + b;
}
```

```cpp
module;

export module math:multiplication;

export double mult(double a, double b) {
        return a * b;
}
```

Slide intentionally left empty