Modules in C++20

Why?

Disadvantages of Headers

- Compilation speed:
  Same header files included in multiple translation units ⟶ Compiler has to reparse & reprocess all over again.

- Split header & code files:
  we repeat ourselves and have to pay more attention.

- Order dependency and cycles:
  Order in which you include the header files may matter (i.e in case of header that need predeclared headers).

- Interfacing with C++:
  When you need to call something in C++ from another language, and the interface is a header, the language has to understand C++.

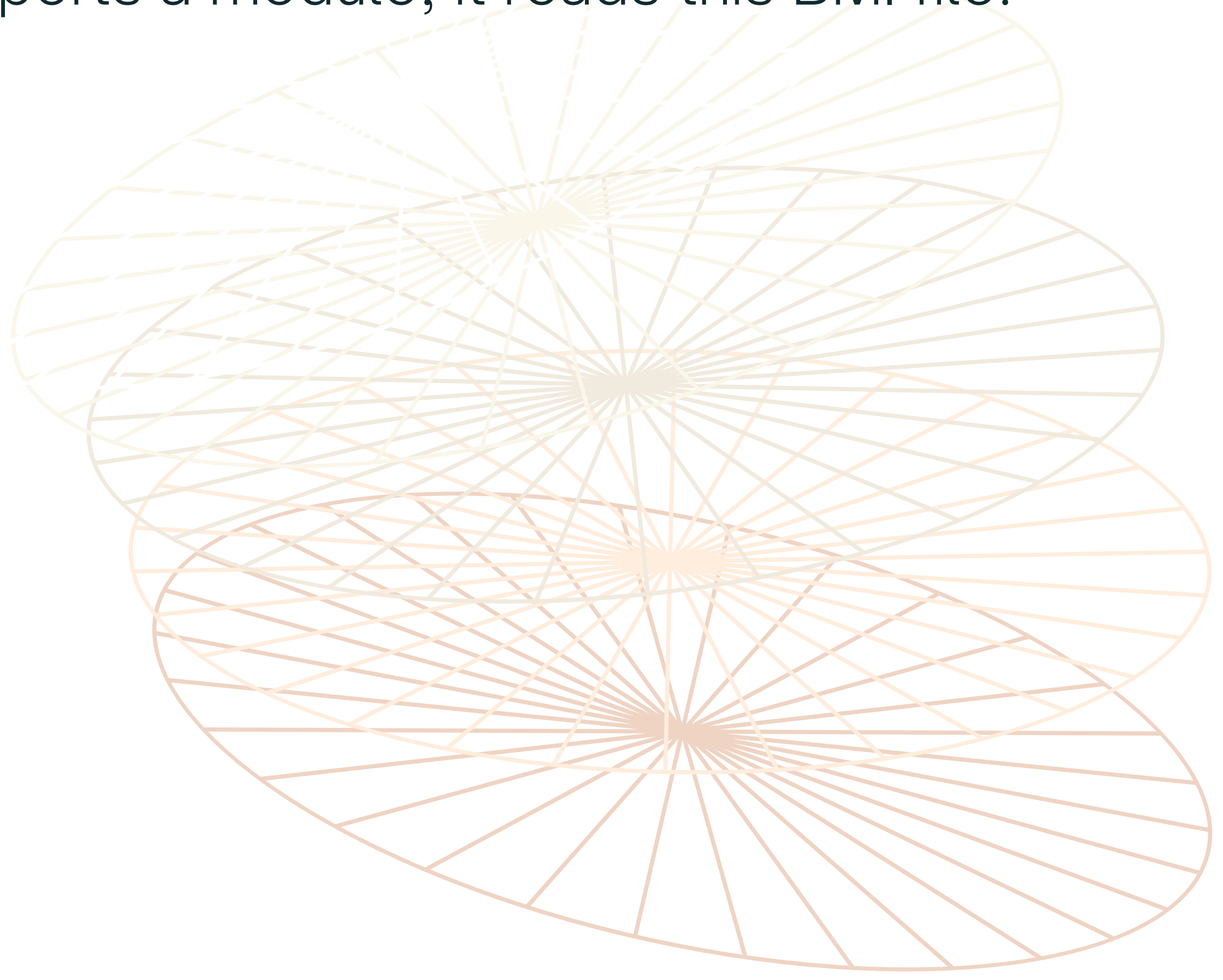
- Lack of Isolation:
  Headers and source codes can interchangeably change each other. (through macros for example).

- One Definition Rule (ODR) violations:
  Non-inline need to have a single def. in the entire program.
  Inline should have identical def. in each translation unit.

How?

Modules Build mechanisms

1. The module is compiled into Binary Model Interface (BMI). This happens **before** compiling the source code.
2. When a translation unit actually imports a module, it reads this BMI file.

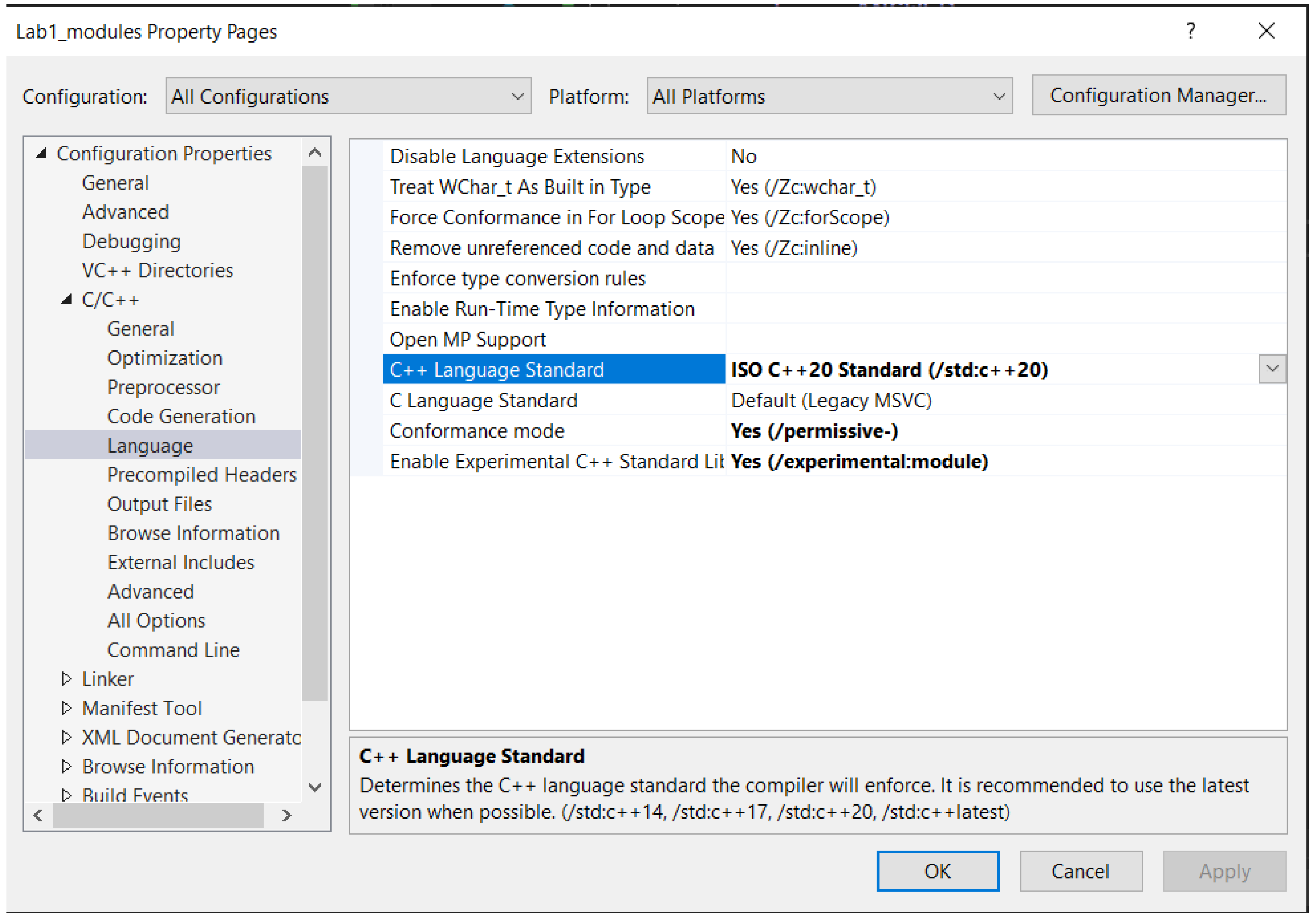


How to use C++ Modules

IDE: Visual studio

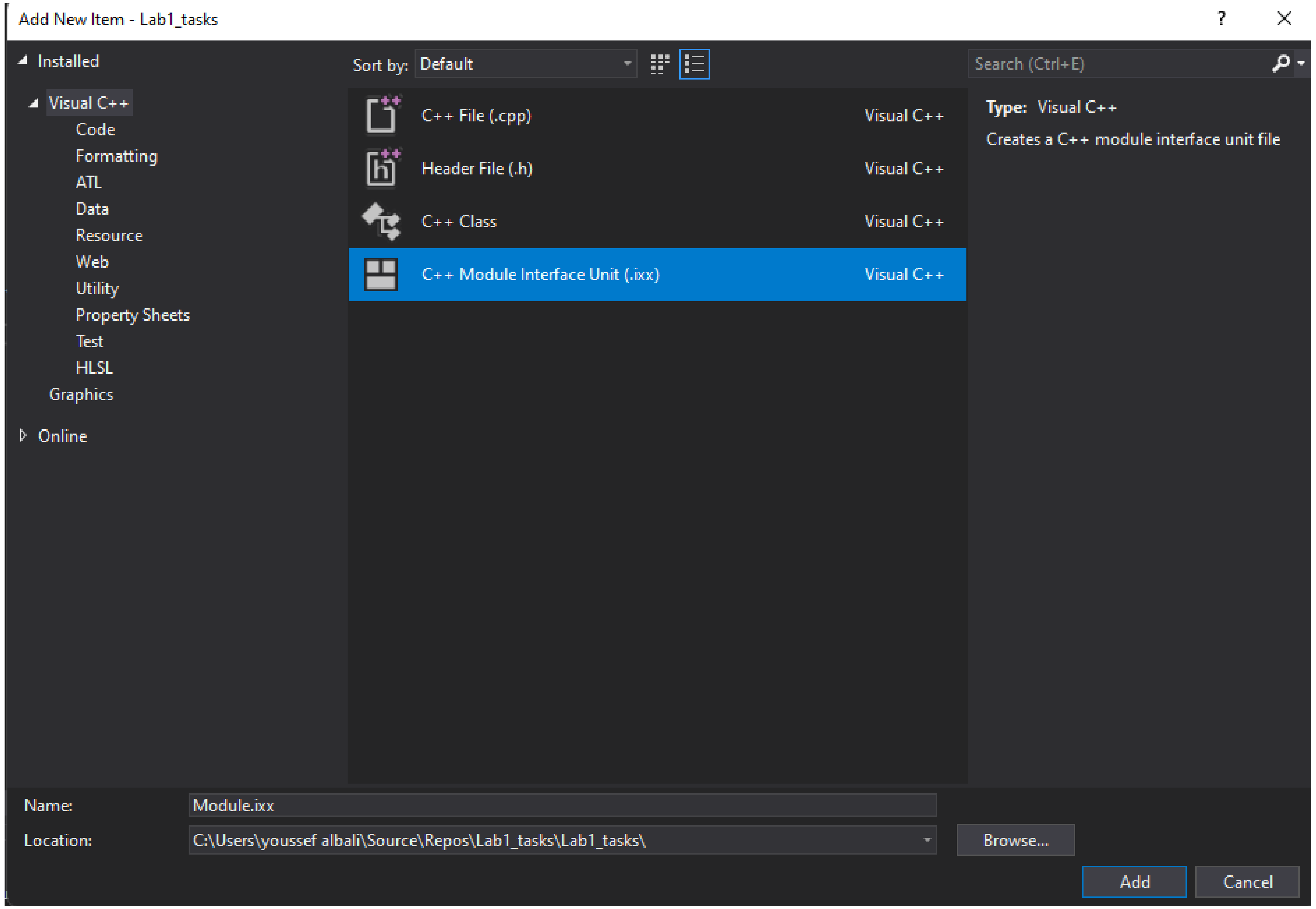
Setup:

1. To enable support for Standard Library modules, compile with /experimental:module and /std:c++latest. In a Visual Studio project, right-click the project node in **Solution Explorer** and choose **Properties**. Set the **Configuration** drop-down to **All Configurations**, then choose **Configuration Properties** > **C/C++** > **Language** > **Enable C++ Modules (experimental)**. A module and the code that consumes it must be compiled with the same compiler options.



Disable Language Extensions	No
Treat WChar_t As Built in Type	Yes (/Zc:wchar_t)
Force Conformance in For Loop Scope	Yes (/Zc:forScope)
Remove unreferenced code and data	Yes (/Zc:inline)
Enforce type conversion rules	
Enable Run-Time Type Information	
Open MP Support	
C++ Language Standard	**ISO C++20 Standard (/std:c++20)**
C Language Standard	Default (Legacy MSVC)
Conformance mode	**Yes (/permissive-)**
Enable Experimental C++ Standard Lit	**Yes (/experimental:module)**

C++ Language Standard
Determines the C++ language standard the compiler will enforce. It is recommended to use the latest version when possible. (/std:c++14, /std:c++17, /std:c++20, /std:c++latest)



Add New Item - Lab1_tasks

Installed

- Visual C++
  - Code
  - Formatting
  - ATL
  - Data
  - Resource
  - Web
  - Utility
  - Property Sheets
  - Test
  - HLSL
  - Graphics
- Online

Sort by: Default

Search (Ctrl+E)

C++ File (.cpp)	Visual C++
Header File (.h)	Visual C++
C++ Class	Visual C++
C++ Module Interface Unit (.ixx)	Visual C++

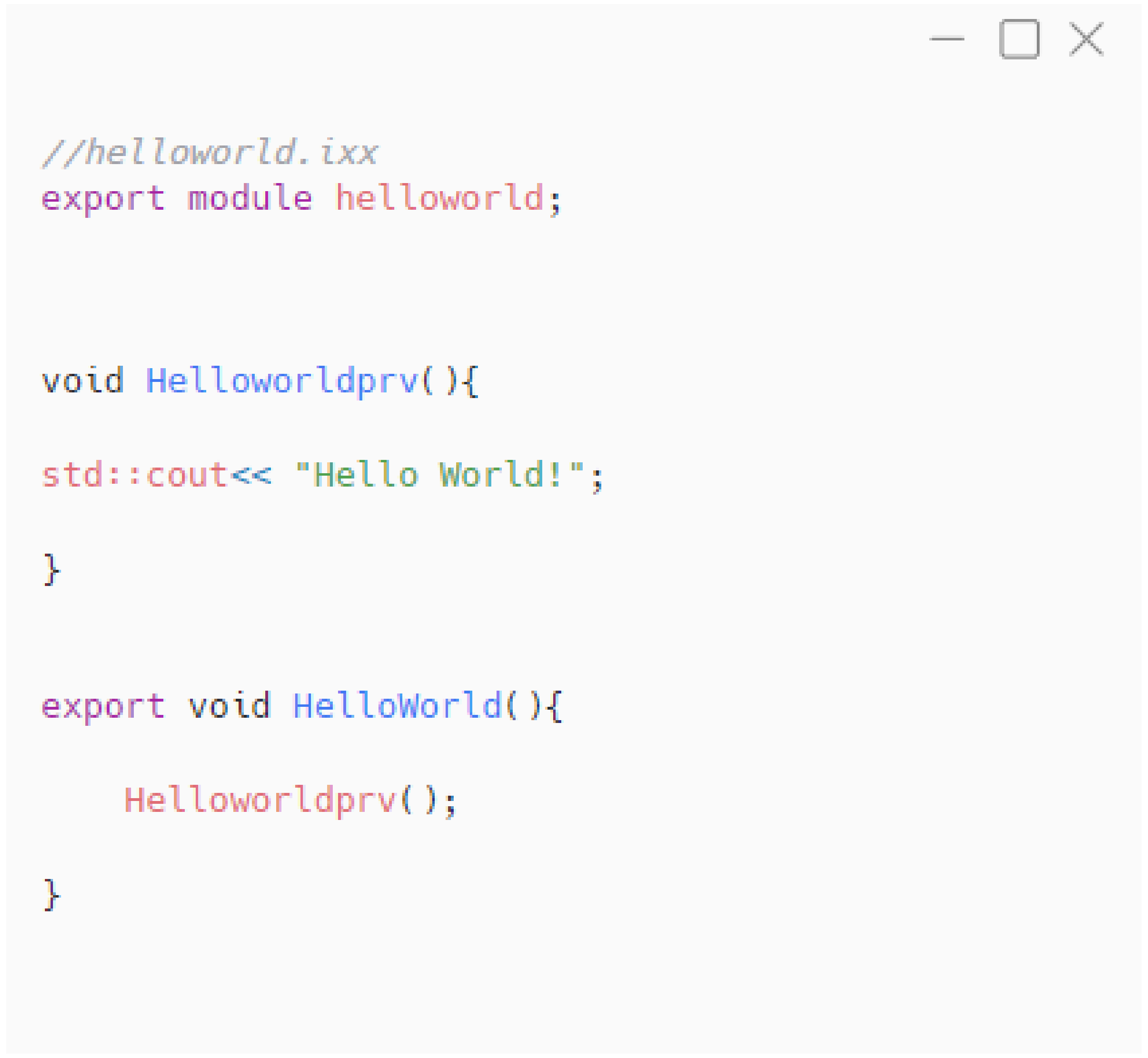
Type: Visual C++

Creates a C++ module interface unit file

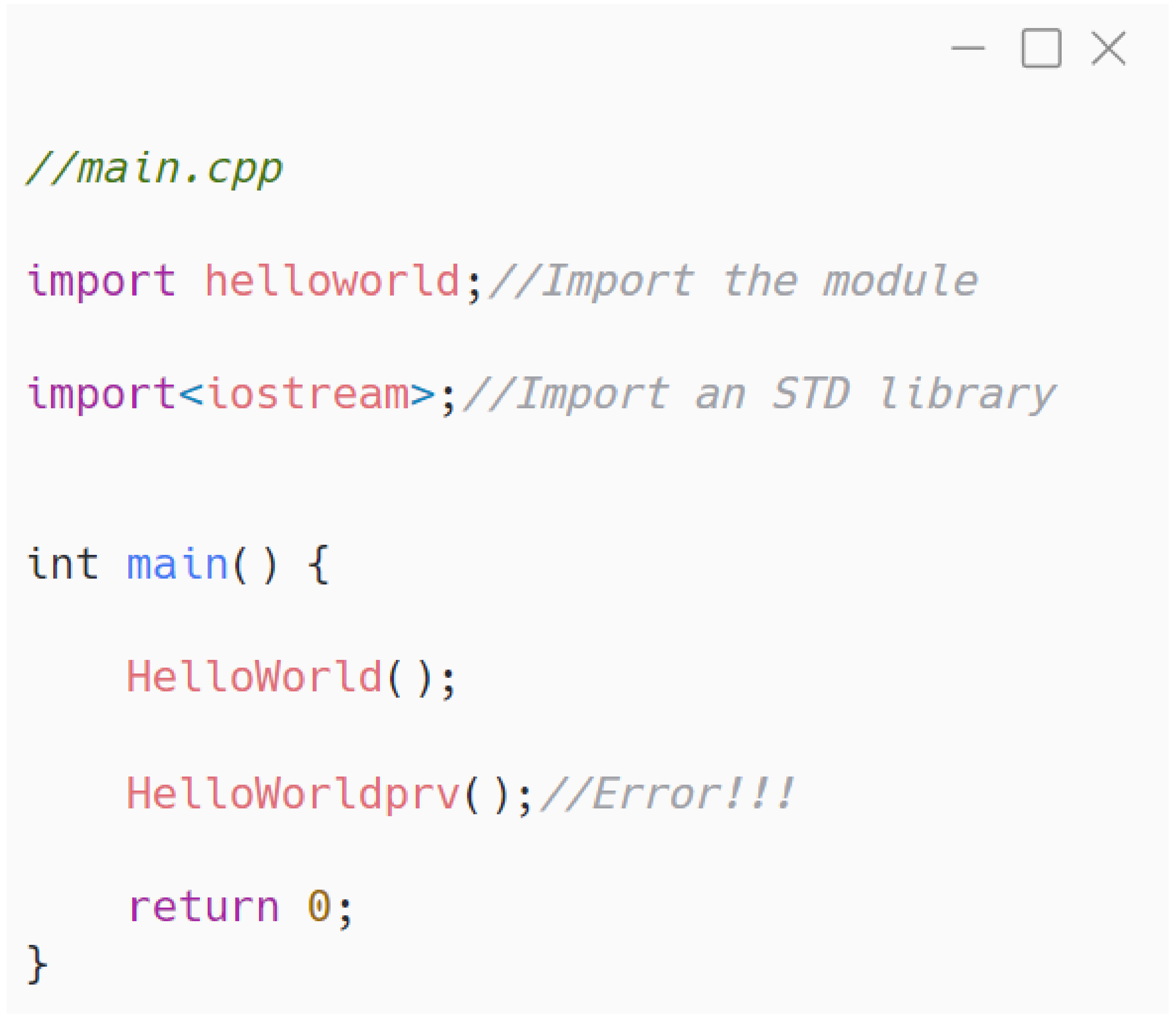
Name: Module.ixx

Location: C:\Users\youssef albali\Source\Repos\Lab1_tasks\Lab1_tasks\

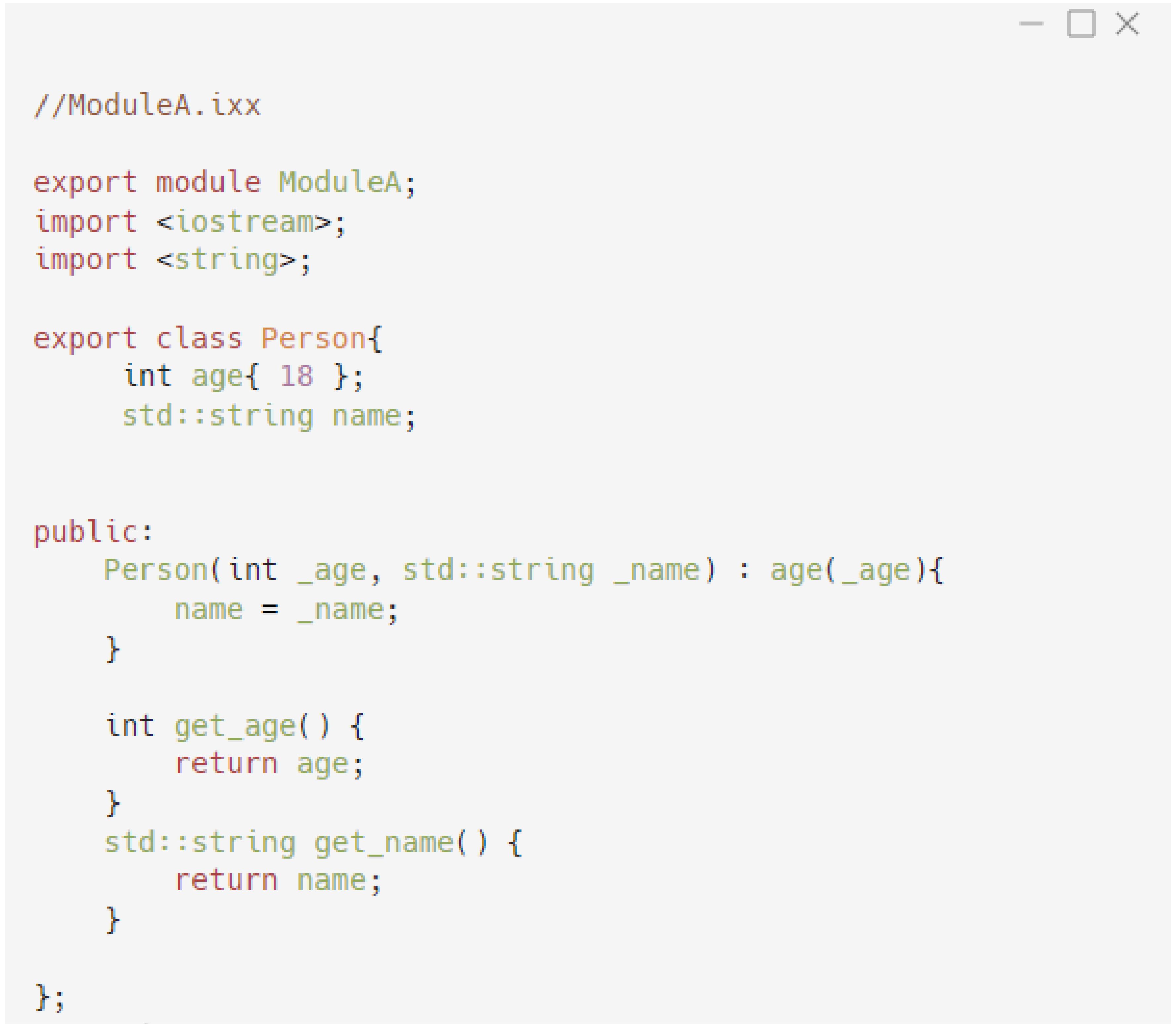
Basic Module



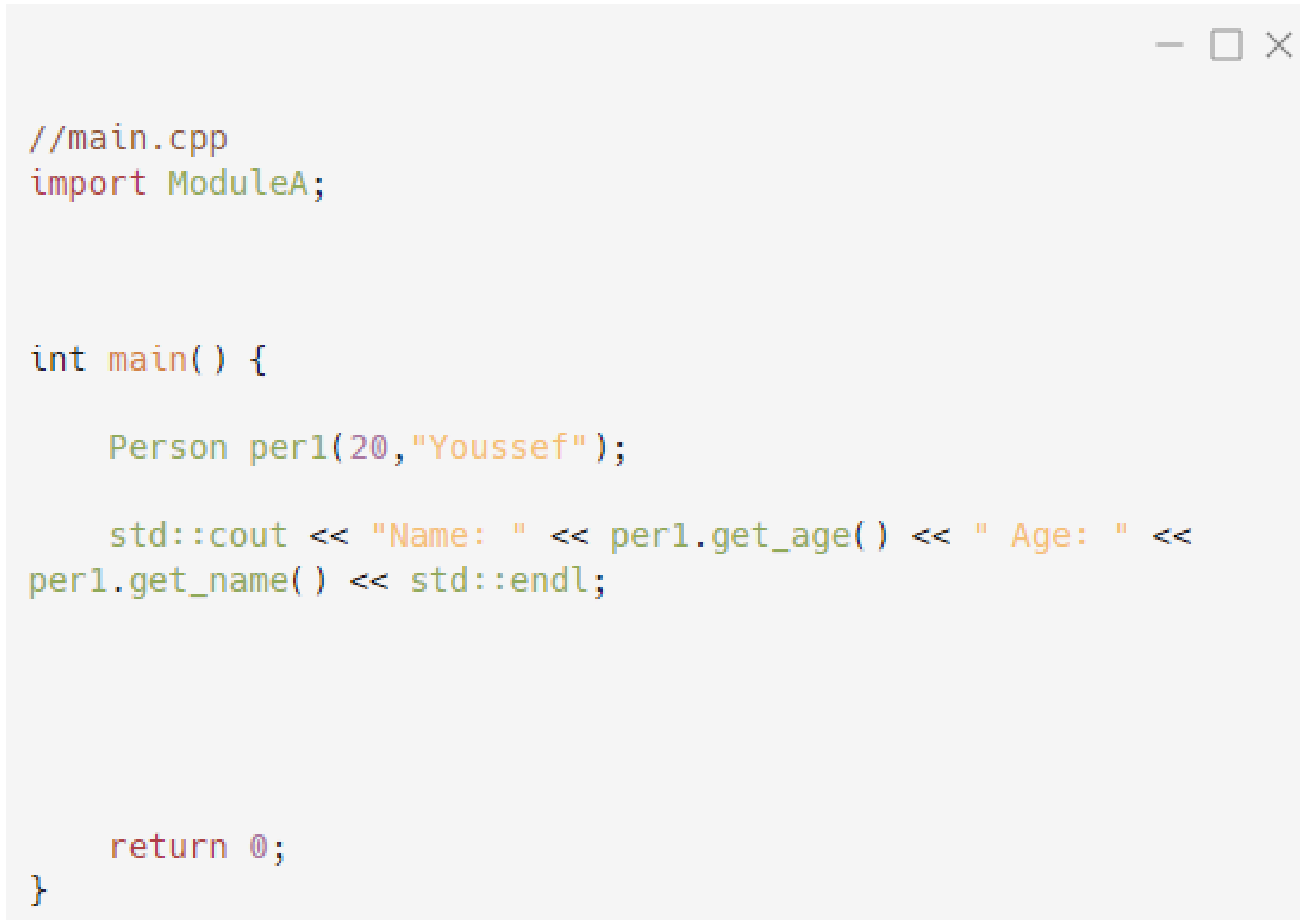
```
//helloworld.ixx
export module helloworld;


void Helloworldprv(){

std::cout<< "Hello World!";

}


export void HelloWorld(){

    Helloworldprv();

}
```



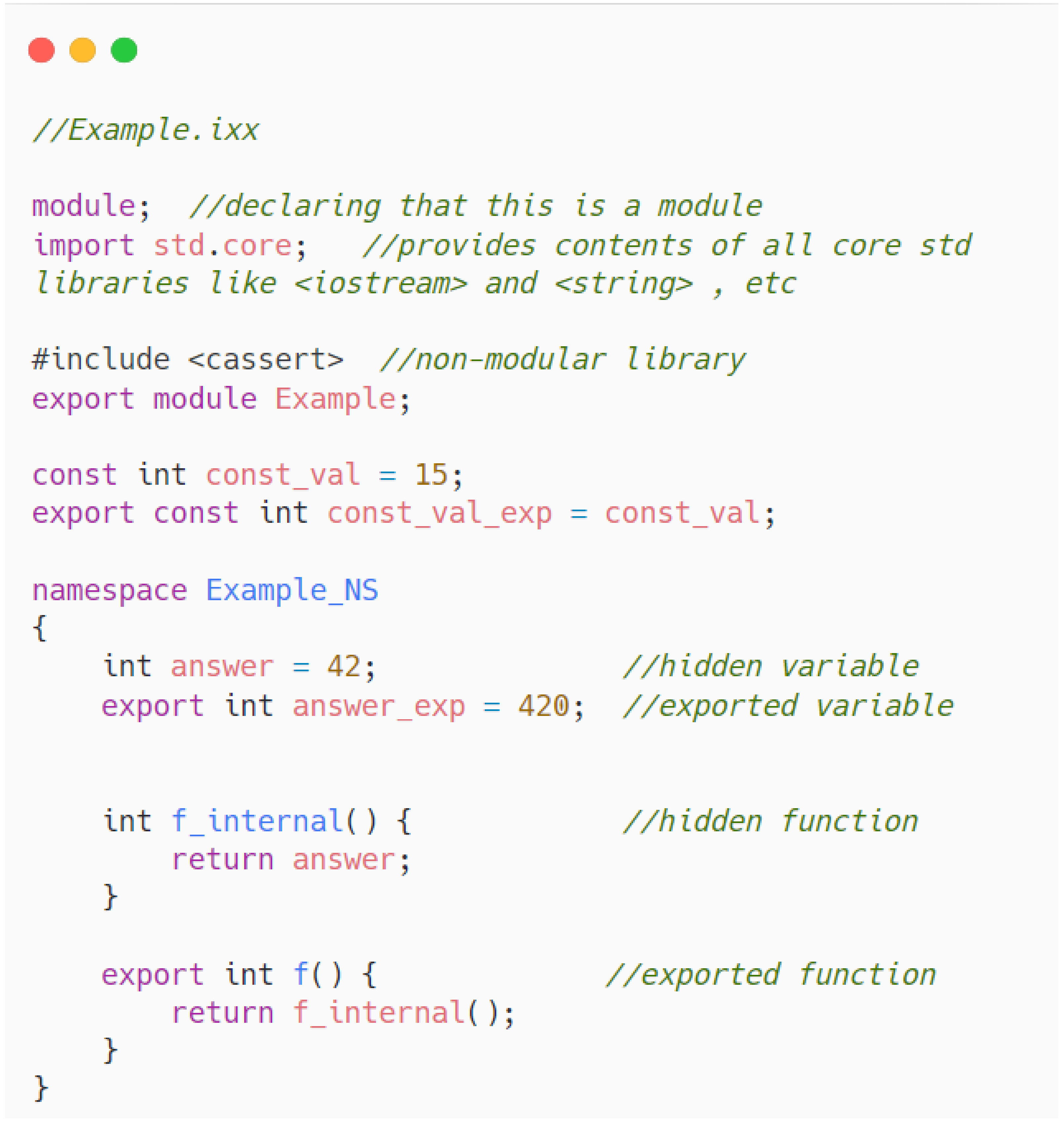
```
//main.cpp

import helloworld;//Import the module

import<iostream>;//Import an STD library


int main() {

    HelloWorld();

    HelloWorldprv();//Error!!!

    return 0;
}
```



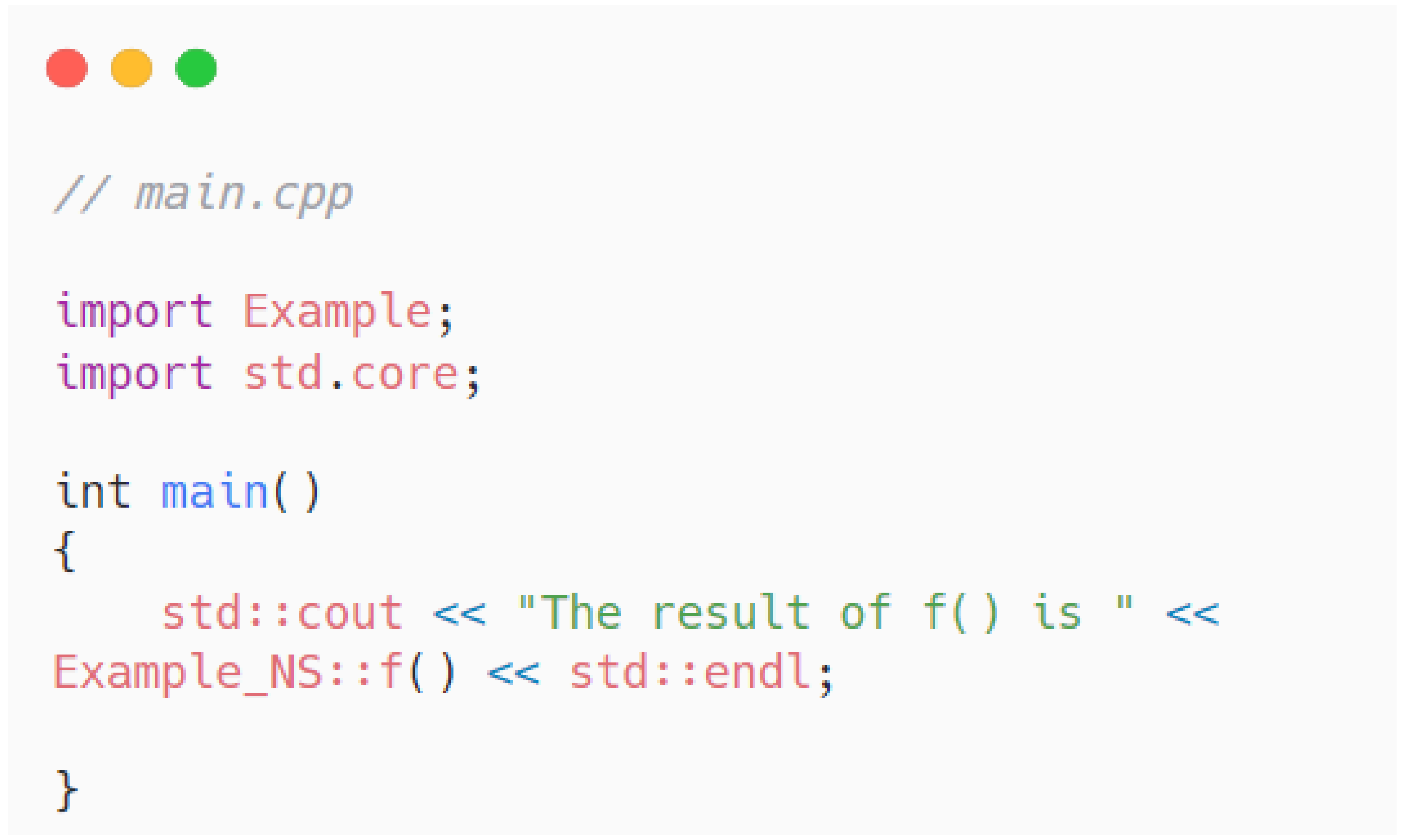
```cpp
//ModuleA.ixx

export module ModuleA;
import <iostream>;
import <string>;

export class Person{
     int age{ 18 };
     std::string name;


public:
    Person(int _age, std::string _name) : age(_age){
        name = _name;
    }

    int get_age() {
        return age;
    }
    std::string get_name() {
        return name;
    }

};
```



```cpp
//main.cpp
import ModuleA;



int main() {

    Person per1(20,"Youssef");

    std::cout << "Name: " << per1.get_age() << " Age: " <<
per1.get_name() << std::endl;





    return 0;
}
```



```
//Example.ixx

module;  //declaring that this is a module
import std.core;   //provides contents of all core std
libraries like <iostream> and <string> , etc

#include <cassert>  //non-modular library
export module Example;

const int const_val = 15;
export const int const_val_exp = const_val;

namespace Example_NS
{
    int answer = 42;               //hidden variable
    export int answer_exp = 420;   //exported variable


    int f_internal() {             //hidden function
        return answer;
    }

    export int f() {              //exported function
        return f_internal();
    }
}
```



```
// main.cpp

import Example;
import std.core;

int main()
{
    std::cout << "The result of f() is " <<
Example_NS::f() << std::endl;

}
```

Tasks:

1. Setup your IDE (MSVC) to be compatible with c++20
2. remember to import needed std libraries not #include them (import std.core;)
3. Create a basic module (Module_A) and import it in main.cpp.
4. in Module_A create a class player with 3 types of member variables and a 3-arg constructor that sets the 3 member variable to the 3 passed arguments.
5. Create another module (Module_B). Create and assign different values for 5 (of each type you used in your class Player) inside your module.
6. import Module_B in main.cpp and use the variables to instantiate 5 objects of class Player. (hint: you can use a namespace containing all variables and export it).
7. import Module_A in Module_B
8. Create Module_C that imports Module_B only. and import it in Main.cpp.
9. In Module_C : Declare and define 3 functions that change the values of variables in Module_B
10. in main(): use functions from task 9 to change values of variables in Module_B. create Player objects with the newly changed variables.
11. Declare some variable in Module_A (just for testing) and export it.
12. Test, can you access Module_A exported variables\functions\classes in Module_C (without directly importing Module_A inside Module_C) (try creating a Player object)? can you think of a solution?
13. Display contents of your class objects to the console
14. Try to optimize your code to minimize the number of imports (hint: only 1 import can be used in main.cpp)