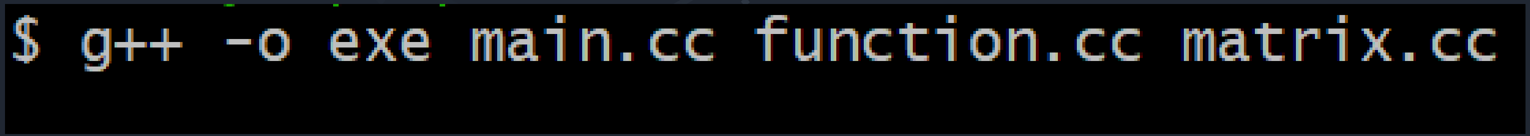
Session 8 / Part 1

Access Control

Lab Exercise:

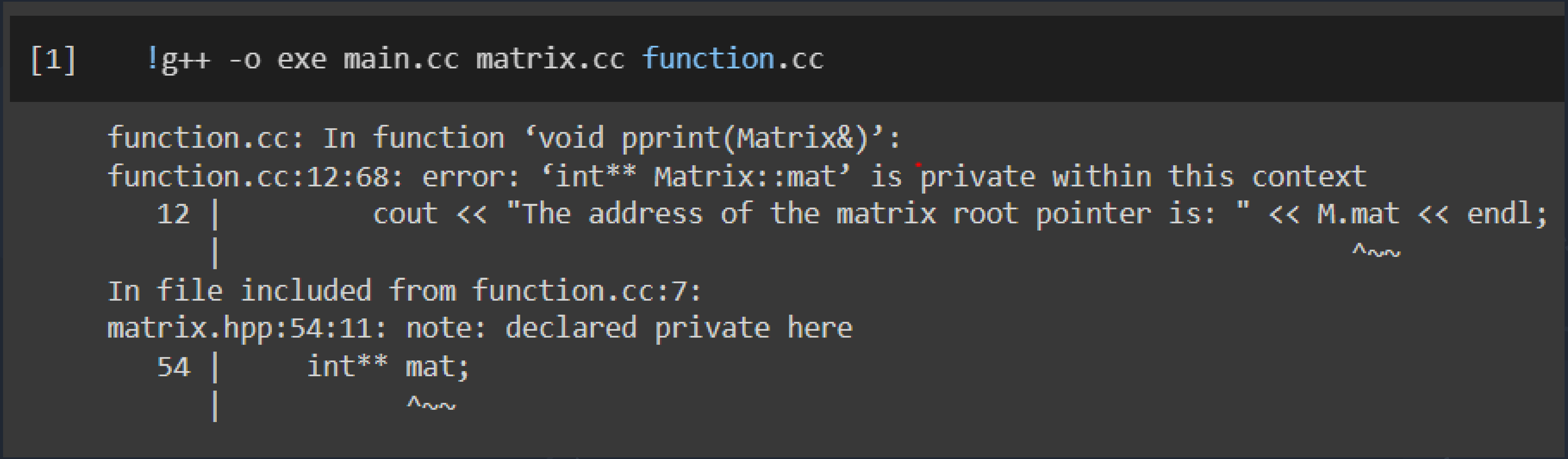
- You are given Matrix.cc, Matrix.hpp, and main.cc codes from our previous sessions. Two other files, function.hpp and function.cc, are provided as well.
- Implement "ppointer()" function in function.cc file which prints the address stored in **mat pointer.
- "pprint()" function will be later called inside main().
- **DO NOT** change any other file.

Reminder: You can generate the final executable files by:



```
$ g++ -o exe main.cc function.cc matrix.cc
```

Lab Exercise: You will get the following error for the lab exercise if you try to get access to **mat, which is a private attribute of M matrix.



```
[1]    !g++ -o exe main.cc matrix.cc function.cc

     function.cc: In function ‘void pprint(Matrix&)’:
     function.cc:12:68: error: ‘int** Matrix::mat’ is private within this context
        12 |         cout << "The address of the matrix root pointer is: " << M.mat << endl;
           |                                                                   ^~~
     In file included from function.cc:7:
     matrix.hpp:54:11: note: declared private here
        54 |     int** mat;
           |           ^~~
```

Matrix.cc & Matrix.hpp

```
-row
-col
-name
-mat
______________
-   allocate()
-   deallocate()
+   set name()
+   get row
+   get col()
+   get entry()
+   set entry()
+   multiply()
```

Developed by Reza
Anyone can use public
Methods or public
attributes

Main.cc

```
int main(){
Matrix A(2,2, "A);
Cout << A.get_row();
A.set_name("Anew");
A.mat = Null; // Error
A.deallocate() // Error
...
}
```

Developed by main
programmer.

x.cc &
x.hpp

y.cc &
y.hpp

Other files developed
By other programmer.

function.cc &
function.hpp

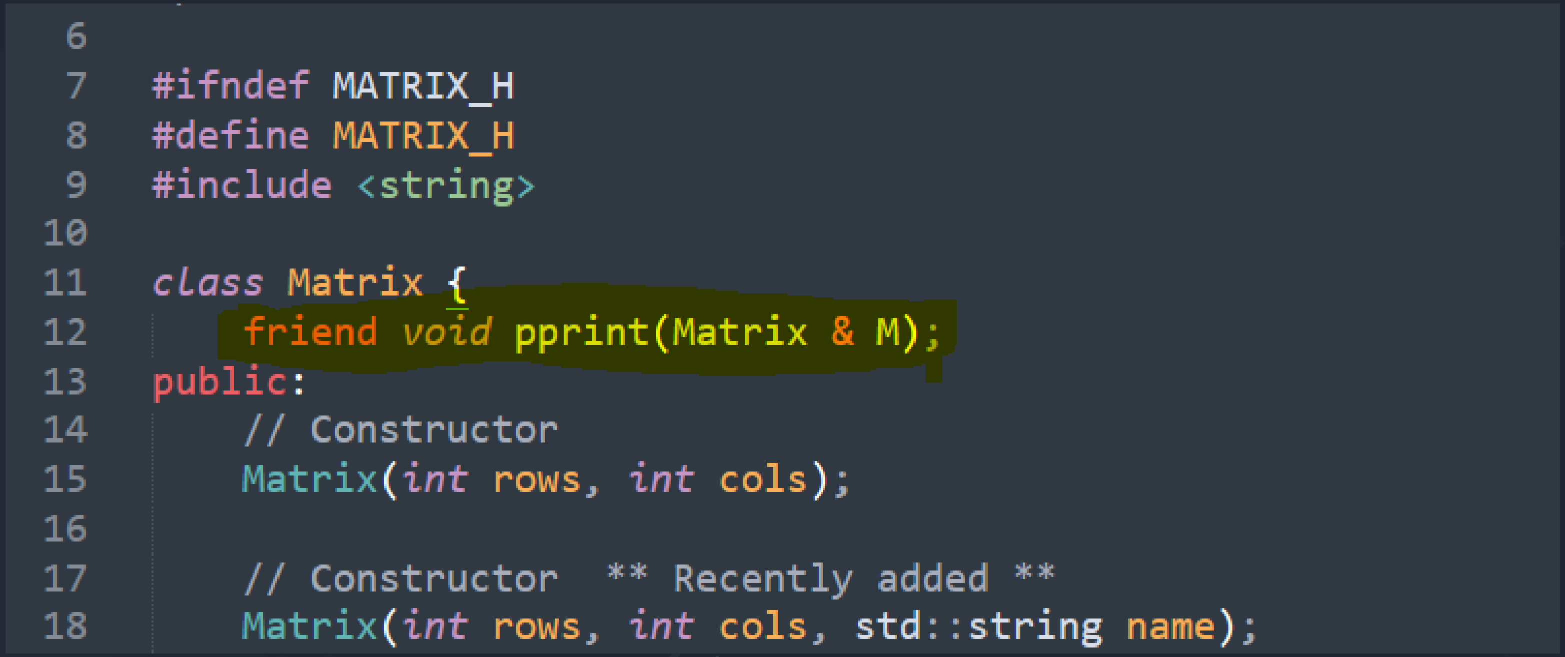
```
Void pprint(Matix & M){
Cout << M.get_row();
Cout << M.mat ; //error;

}
```

Developed by Bob.

I, developer of Matrix.cc, know bob. I know that he won't misuse my private attributes, like mat pointer. So, how can I grant access to him for using my non-public attributes and functions? --> Answer: Telling the compiler that he is my **friend**!

We can fix the issue by making telling compiler that "pprint" fucntion is a friend of Matrix class (i.e. Pprint is a friend of any Matrix object) in the following way:



```

#ifndef MATRIX_H
#define MATRIX_H
#include <string>

class Matrix {
    friend void pprint(Matrix & M);
public:
    // Constructor
    Matrix(int rows, int cols);

    // Constructor  ** Recently added **
    Matrix(int rows, int cols, std::string name);
```

Note: pprint() is a stand-alone function. We can have a friend who is a member function of another class, or we can have an entire other class as a friend. These two other scenarios follow the same concept and roughly the same syntax.

Review:
Friend Keyword:

1) **Concept:** With OOP mindset we have grouped some data and functions, which also entails <u>Encapsulation</u> and <u>Information Hiding</u>. However, sometimes we need to give some people a full access to non-public attribute/functions of our class. We can grant this full access to them by making them our "Friend".
2) **Necessity:** In order to be concise, I only give you two examples to see when and how it is necessary to use "Friend":
   1) **Unit testing:** Image you have written Matrix class. You are not so sure about you implementation. So, you ask your friend to write a Class called TestM to unit test every part of your code for Matrix class.
      You can declare class TestM as a friend to you Matrix class in order to able him to test your code.
   2) **Operator Overloading:** Sometimes you can outsource some functionalities to other functions and classes to whom you have trust. In this case, it is easier that you make them friend and let them implement that functionality. One example of this case is Operator Overloading.
3) **Syntax:** The syntax is simple. Read lectures for details.