- Solution

In this lesson, we will go over the solution to replacing the auto keyword with explicit data types.

WE'LL COVER THE FOLLOWING

Solution

```
// C++ 14
#include <chrono>
#include <functional>

#include <future>
#include <initializer_list>
#include <map>
#include <string>
#include <thread>
#include <tuple>

int main(){

  std::initializer_list<int> myInts = {1, 2, 3};
  std::initializer_list<int>::iterator myIntBegin = myInts.begin();

  std::map<int, std::string> myMap = {{1, std::string("one")}, {2, std::string("two")}};
  std::map<int, std::string>::iterator myMapBegin = myMap.begin();

  std::function<std::string(const std::string&)> func = [](const std::string& a){ return a; }

  std::future<std::string> futureLambda = std::async([]{ return std::string("Hello"); });

  std::chrono::time_point<std::chrono::system_clock> begin = std::chrono::system_clock::now()

  std::pair<int, std::string> pa = std::make_pair(1, std::string("second"));

  std::tuple<std::string, int, double, bool, char> tup = std::make_tuple(std::string("second"

}
```

Explanation

We have changed the code in the exercise and replaced the `auto` keywords with their explicit deduction types.

- In line 14, we have defined an `std::initializer_list<int>` and in line 15, we have defined its iterator `std::initializer_list<int>::iterator`.
- In line 17, we have defined a map `std::map<int, std::string>` and in line 18, we have defined its iterator `std::map<int, std::string>::iterator`
- In line 20, we have defined a `std::function<std::string(const std::string&)>` for the lambda expression.
- In line 22, we have defined a `std::future<std::string>`.
- In line 24, we have defined `std::chrono::time_point<std::chrono::system_clock>`.
- In line 26, we have defined `std::pair<int, std::string>`.
- In line 28, we have defined a tuple with `std::tuple<std::string, int, double, bool, char>`.

Replacing `auto` with explicit types can be a difficult task since the user must remember the relevant details for the libraries in use. As an embedded systems programmer, you must work with features from multiple libraries, and the `auto` helps you handle those features.

For further information, read here auto.

In the next lesson, we will learn about scoped enumerations.