- Solution

Here is the solution to the exercise in the previous lesson.

WE'LL COVER THE FOLLOWING

- Explanation

The solution to the previous exercise can be found below:

```
#include <chrono>
#include <functional>

#include <future>
#include <initializer_list>
#include <map>
#include <string>
#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([](const std::string& s ) {return std::st

  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

- The solution is pretty straightforward. All we have to do is figure out the

- The solution is pretty straightforward. All we have to do is figure out the correct type for each entity.
- The `initializer_list` header has been imported as it is used to create `myInts` in line 12 and the iterator `myInts.begin()` in line 13.
- The lambda expression has the type `std::function<std::string(const std::string&)>`. For that purpose, we have included the `functional` header.

After this exercise, it is clear how helpful `auto` is in making our program clean and safe.

There is a high potential for error if we keep defining types explicitly. Hence, it's better to leave it to the compiler. It'll save us time as well.

For further information, see the official documentation for auto.

Next, we'll study another technique for automatic type deduction.