- Examples

Examples for using the auto keyword in different cases.

WE'LL COVER THE FOLLOWING

Example 1

```cpp
// auto.cpp

#include <iostream>
#include <vector>

int func(int){return 2011;}

int main(){

  auto i = 5;                  // int
  auto& intRef = i;            // int&
  auto* intPoint = &i;         // int*
  const auto constInt = i;     // const int
  static auto staticInt = 10;  // static int

  std::vector<int> myVec;
  auto vec = myVec;           // std::vector<int>
  auto& vecRef = vec;         // std::vector<int>&

  int myData[10];
  auto v1 = myData;           // int*
  auto& v2 = myData;          // int (&)[10]

  auto myFunc = func;          // (int)(*)(int)
  auto& myFuncRef = func;      // (int)(&)(int)

  // define a function pointer
  int (*myAdd1)(int, int) = [] (int a, int b){return a + b;};

  // use type inference of the C++11 compiler
  auto myAdd2 = [](int a, int b){return a + b;};
  std::cout << "\n";

  // use the function pointer
  std::cout << "myAdd1(1, 2) = " << myAdd1(1, 2) << std::endl;

  // use the auto variable
  std::cout << "myAdd2(1, 2) = " << myAdd2(1, 2) << std::endl;

  std::cout << "\n";

}
```

Explanation

In the example above, the complier automatically deduces the types depending on the value stored in the variable. The corresponding types of variables are mentioned in the in-line comments.

- In line 10, we have defined a variable, `i` and its type is deduced to be `int` because of the value 5 stored in it.
- In lines 11 - 14, we have copied the values into different variables and their type is deduced `auto`-matically depending on the value stored in it.
- Similarly, in lines 17 - 18, we copy a vector and its reference by using the assignment operator `=`. `auto` keyword takes care of `vec` and `vecRef` types.
- In lines 24 - 25, `auto` determines the type of `myFunc` as function pointer and `myFuncRef` as a reference to that function.
- In line 31, we have defined a lambda expression whose return type is inferred by the C++ compiler since we have used the `auto` keyword.

Example 2

```
// autoExplicit.cpp
#include <iostream>
#include <chrono>
#include <future>
#include <map>
#include <string>
#include <thread>
```

```
#include <tuple>

int main(){


  auto myInts = {1, 2, 3};
  auto myIntBegin = myInts.begin();

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

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

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

  auto begin = std::chrono::system_clock::now();

  auto pa = std::make_pair(1, std::string("second"));

  auto tup = std::make_tuple(std::string("second"), 4, 1.1, true, 'a');

}
```

Explanation

In the example above, we used the `auto` keyword in the highlighted lines and left it for the compiler to infer the type during the run time. Since we are handling different C++ libraries when writing extensive codes, it becomes difficult to keep track of each type. `auto` helps by bypassing this problem.

Let’s test your understanding of this concept with an exercise in the next lesson.