1 BindSecond

1.1 Aufgabe

In dieser Aufgabe lernen Sie im Detail, wie Lambdas und `std::bind` funktionieren.

Sie sollen die Template-Funktion `BindSecond` schreiben, welche es erlaubt, aus einem Funktionsobjekt, welches zwei Parameter vom gleichen Typ entgegennimmt, ein neues Funktionsobjekt zu erstellen, welches nur noch einen Parameter benötigt. Der zweite Parameter soll auf einen fixen Wert gesetzt werden.

Damit dies möglich wird, benötigen wir eine Klasse `BindSecondHelper`, welche innerhalb der Funktion `BindSecond` instanziiert und zurückgegeben wird. Dementsprechend ist `BindSecondHelper` das neue Funktionsobjekt, welches einen fixen Wert für den zweiten Parameter der gekapselten Funktion speichern muss. Zudem muss es auch eine Kopie der ursprünglichen binären Funktion speichern. Beide können von beliebigen Typ sein.

> Tipp: An vielen Stellen in dieser Aufgabe benötigen Sie die Information für entsprechende Typen. Z.B. der Typ, welcher von den beiden Funktionsobjekten zurückgegeben wird. Da dies teilweise nicht so einfach ist, diesen Typ zu evaluieren, können wir es dem Compiler überlassen, indem wir `auto` verwenden.

Schreiben Sie auch eine entsprechende Test-Applikation, welche z.B. aus einem Lambda, welches evaluiert ob `lhs` kleiner als `rhs`, ein neues Funktionsobjekt macht, welches testet, ob `lhs` kleiner als 2 ist.

1.2 Lösung

```
1  template<class BinaryFunction, typename T>
2  class BindSecondHelper {
3   public:
4    BindSecondHelper(const BinaryFunction& func, const T& arg)
5      : function_(func)
6      , second_arg_(arg)
7    {}
8
9    auto operator()(const T& first_arg) const {
10     return function_(first_arg, second_arg_);
11   }
12
13   private:
14    const BinaryFunction function_;
15    const T second_arg_;
```

```
16 };
17
18 template<class BinaryFunction, typename T>
19 auto BindSecond(BinaryFunction func, const T& y) {
20   return BindSecondHelper<BinaryFunction, T>(func, y);
21 }
22
23 void TestBindSecond() {
24   std::cout << "TestBindSecond" << std::endl;
25   std::cout << "--------------" << std::endl;
26
27   const auto less = [](int lhs, int rhs) {
28     return lhs < rhs;
29   };
30
31   auto less2 = BindSecond(less, 2);
32
33   std::cout << "less2(1): " << std::boolalpha << less2(1) << std::endl;
34   std::cout << "less2(3): " << std::boolalpha << less2(3) << std::endl;
35
36   std::cout << std::endl;
37 }
```