Übung 3: *Standard-Bibliothek, Funktoren und Smart-Pointers*

Aufgabe 1: *Standard-Bibliothek und Funktoren*

Folgendes Code-Segement soll funktionsfähig gemacht werden
(Vorlage auf Moodle: *StdLibFunctor.cpp*).

```
vector<Person> persons;
persons.push_back(Person("Marianne", 71));
persons.push_back(Person("Oliver",    8));
persons.push_back(Person("Anna",     44));
persons.push_back(Person("Dorian",   10));

cout << endl << "Unsorted:" << endl;
printPersons(persons);

cout << endl << "Sorted by Name:" << endl;
sort(persons.begin(), persons.end());
printPersons(persons);

cout << endl << "AgeMinMaxFunctor:" << endl;
AgeMinMaxFunctor ageMinMax;
ageMinMax = for_each(persons.begin(), persons.end(), ageMinMax);
cout << "Minimal Age = " << ageMinMax.getMin() << endl;
cout << "Maximal Age = " << ageMinMax.getMax() << endl;


/* Session-Log:

Unsorted:
Marianne  : 71
Oliver    :  8
Anna      : 44
Dorian    : 10

Sorted by Name:
Anna      : 44
Dorian    : 10
Marianne  : 71
Oliver    :  8

AgeMinMaxFunctor:
Minimal Age = Oliver    :  8
Maximal Age = Marianne  : 71

*/
```

Hinweis:

sort() benötigt den Vergleichs-Operator **<** :

```
bool operator<(const Person& pOtherPerson) const
```


Aufgabe 2: Smart-Pointer

Es sollen die Eigenschaften der ***Smart-Pointer*** der *Standard-Bibliothek* demonstriert werden (gem. Skript Folie 119 u. 120).

Aufgabe 2.a: *Test-Klasse*

Es soll eine Test-Klasse erstellt werden welche sich so verhält wie im folgenden Code-Segment dargestellt.
Die Klasse enthält nur ein *string*-Attribut welches mit dem Konstruktor gesetzt wird.
Im Weiteren soll der Konstruktor sowie der Destruktur jeweils eine *Trace*-Ausgabe auf die Konsole generieren wie im *Session-Log* ersichtlich ist.

Code-Segment:

```
{
  TestClass myTestObj("Bla bla.");
  cout << "Inhalt von Test-Objekt  : " << myTestObj.getString() << endl;
}
```

Session-Log:

```
TestClass::TestClass()  : Bla bla.
Inhalt von Test-Objekt  : Bla bla.
TestClass::~TestClass() : Bla bla.
```

Aufgabe 2.b: *unique_ptr*

In der Theorie haben wir die Problematik betreffend *Resource Acquisition is Initialization (RAII)* (Folie 99 u. 100) diskutiert und dabei gesehen, dass man beschaffte Ressourcen in ein Wrapper-Objekt *einpackt*. In unserem Fall war dies die Klasse *MyClassWrapper*.
Statt jener Klasse soll nun jenes Szenario unter Benutzung der Smart-Pointer Klasse ***uniqe_ptr*** programmiert werden (unter Einsatz obiger Klasse *TestClass*).

Hinweise:

- Die Smart-Pointer sind im Namespace ***std*** und im Header-File ***memory*** definiert (`#include <memory>`).
- Die Objekte, welche gewrapped werden sollen, sollten immer mit ***new*** direkt beim Konstruktor-Aufruf des Smart-Pointer erzeugt werden.
  Beispiel:
  `unique_ptr<TestClass>` **`uniquePtr(new TestClass("Test mit unique_ptr"));`**
- Alle Smart-Pointer-Klassen haben eine ***get()-***Methode um den '*gewrappten'* Pointer explizit zu erhalten.
- Je nach Kompiler muss man mit Optionen explizit festlegen, dass die neuen Features des C++11-Standards benutzt werden können.
  - Bei *Eclipse/CDT*:
    Projekt-Properties>C/C++ General>Paths and Symbols># Symbols>GNU C++:
    Mit “Add...” ein neues Symbol hinzufügen:
    - Name: ***__GXX_EXPERIMENTAL_CXX0X__***
    - Value: ***1***
  - Bei *Eclipse/CDT* mit *Cygwin* und einem *Executable*-Projekt:
    Projekt-Properties>C/C++ Build>Settings>Cygwin C++ Compiler>Miscellaneous:
    bei “Other flags” hinten hinzufügen: ***-std=c++11***
  - Bei einem *Makefile*-Projekt:
    Obige Option im Makefile bei den Kompiler-Flags hinzufügen.
    Somit: `CFLAGS = -g` ***`-std=c++11`***

Aufgabe 2.c: *shared_ptr*

Nun soll ein ***shared_ptr*** mit einem *TestClass*-Objekt erzeugt werden.
Im Weiteren in einem neuen Gültigkeitsbereich ein zweiter *shared_ptr*, welcher mit dem ersten *shared_ptr* initialisiert wird (*Initialisation*, nicht *Zuweisung*):

- Zeigen beide *shared_ptr* auf das *selbe Objekt* ?
- Stimmt der interne Zähler der *shared_ptr's* jeweils ?
  (verifizieren mit der Methode ***use_count()***)
- Passiert am Ende des Gültigkeitsbereich des zweiten *shared_ptr* das richtige ?

Es soll ein dritter *shared_ptr* erzeugt werden mit einem weiteren *TestClass*-Objekt.
Dieser dritte *shared_ptr* soll nun dem ersten *shared_ptr* zugewiesen werden (*Zuweisung*, nicht *Initialisation*).

- Passiert das richtige ?
- Passiert am Ende des Gültigkeitsbereich auch was erwartet wird ?

