Bern University
of Applied Sciences

394661-FS2020-0 - C++ Programming I

EXERCISE-10

TABLE OF CONTENTS

1 Introduction

This exercise of 394661-FS2020-0will focus on smart pointers. In particular, you'll write your own version of `unique_ptr` and `shared_ptr`.

You will learn the following topics when completing this exercise:

- Understanding smart pointers
- Repeat templates and template specialization
- Repeat operator overloading (`*`, `->`)
- Repeat copy constructor and copy assignment operator
- Repeat move constructor and move assingment operator
- Writing a class `unique_ptr`
- Writing a class `shared_ptr`

In order to successfully solve this exercise, read **Lesson 26** *Understanding Smart Pointers* in the book. Further research on the web is recommended.

2 Exercises

Create CMake-Projects with C++ 11 compiler support and Debug/Release build options for the exercise. Add additional files manually to the project to gain full control over the included project files. Implement a **header only** version of the smart pointer classes.

2.1 Implementation of simplified class `unique_ptr`

A `unique_ptr` is a smart pointer that owns and manages another object through a pointer and disposes of that object when the `unique_ptr` goes out of scope. As the the name suggest, a `unique_ptr` guaranties exclusive ownership and thus, is **not** copy-constructible or copy-assignable but is move-constructible or move-assignable. Implement your own class `UniquePtr` with the following functionality:

- Implement constructor and destructor for managing a single object allocated with `new` (template class)
- Implement operator overloading for dereference operator `*` and member selection operator `->`
- Disable the copy constructor and copy assignment operator to guarantee unique ownership
- Implement move constructor and move assignment operator
- Provide a template specialization for dynamically-allocated arrays of objects (template specialization)

2.2 Implementation of simplified class `shared_ptr`

In contrast to the `unique_ptr`, a `shared_ptr` is allowed to copy, *i.e.* intended to share. That means several `shared_ptr` objects may own the same object. The object is destroyed and its memory deallocated when ether the last remaining `shared_ptr` owning the object is destroyed. To keep track of the number of owners a reference counting mechanism is implemented. Implement your own class `SharedPtr` with the functionality of `UniquePtr` but with:

- a copy constructor and copy-assignment operator
- a reference counting mechanism
- a function `useCount` returning the number of `SharedPtr` objects referring to the same managed object

Test your smart pointers on the example code provided by `testscript.cpp`

3 Submission

Submit your source code (as a zip-file) to Ilias EXERCISE-10 **before the deadline** specified in Ilias.