Bern University
of Applied Sciences

394661-FS2018-0 - C++ Programming I

EXERCISE-05

TABLE OF CONTENTS

1 Introduction

This exercise of 394661-FS2018-0 will focus on **Constructors** and **Destructor** in C++. With your knowledge about classes and memory management you're now able to design and write classes which are safe and efficient to use. Therefore, you write a class `vector` which stores the data internally in a dynamic array using raw memory and hence, is responsible for memory management. You'll implement some of the well known functionality of the STL-Vector class and test it's functionality. Gaining these insights, you'll hopefully use the provided STL-vector class for future implementations.

You will learn the following topics when completing this exercise:

- Class design for dynamic memory management
- Copy Constructors
- Move Constructors
- Destructor
- Implementation of your own vector class

2 Excercises

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. Separate class declaration and implementation in header and source file, respectively.

2.1 Vector

- **Write a class `Vector`** for a the data type `int` providing the following functionality. The code snippet below to test your vector class outlines the detailed requirements of the vector.

  1. The class `vector` stores the integer values in a dynamic allocated array, *i.e.* a private data member `int* m_data`. Use constructors and destructor to ensure proper memory management
  2. Provide three ways of Initialization:
     - Empty vector: *i.e.* `vector v;`
     - Vector of given size initialized with zeros: *i.e.* `vector v(100);`
     - Vector of given size initialized with value of choice: *i.e.* `vector v(100,42);`
  3. Your vector must always know its size. Provide a public getter function `size()`
  4. Since you're using a pointer to raw memory you need to implement a copy- & move constructor ensuring a deep copy of the data.
  5. Element access with `at(index)`. Make sure that out-of-range access is not possible and display a warning in case!
  6. Add element to back of vector with `push_back()`. This one is a little trickier!
     - Allocate new array with required size
     - Copy data
     - Delete old data
  7. Remove element with `pop_back()` (see `push_back()`)
  8. Clear all elements with `clear`

- **Test your classes** with the following test program (ex05.cpp):

```cpp
#include <iostream>
#include "vector" // Your implementation of vector


int main()
{
    std::cout << "*****************************" << std::endl;
    std::cout << "******** VECTOR TEST ********" << std::endl;
    std::cout << "*****************************\n" << std::endl;

    // 1) Initialisation
    vector v1; // empty vector
    vector v2(100); // vector with 100 elements initialised to 0!
    vector v3(100,42); // vector with 100 elements initialised to 42!

    std::cout << "v1 has size " << v1.size() << std::endl;
    std::cout << "v2 has size " << v2.size() << std::endl;
    std::cout << "v3 has size " << v2.size() << "\n" << std::endl;

    // 2) Element access
    std::cout << "v1 contains value: " << v1.at(0) << std::endl; // -> warning!
    std::cout << "v2 contains value: " << v2.at(0) << std::endl;
    std::cout << "v3 contains value: " << v3.at(0) << std::endl;
    std::cout << "v3 contains value: " << v3.at(142) << std::endl; // -> warning!

    // 3) Add Element
    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);
    v1.push_back(4);
    v1.push_back(5);

    std::cout << "\n" << "v1 has size " << v1.size() <<  " and containes: " <<
        std::endl;
    for (int i = 0; i < v1.size(); ++i)
    {
        std::cout << i << ": " << v1.at(i) << std::endl;
    }

    // 4) Remove Element
    v1.pop_back();
    v1.pop_back();

    std::cout << "\n" << "v1 has size " << v1.size() <<  " and containes: " <<
        std::endl;
    for (int i = 0; i < v1.size(); ++i)
    {
        std::cout << i << ": " << v1.at(i) << std::endl;
    }

    // 4) Clear Elements
    v1.clear();
    std::cout << "\n" << "\n" << "v1 has size " << v1.size() <<  std::endl;

    // 5) Chech Copy Constructor
    vector vCopy(v3);
    std::cout << "vCopy has size " << vCopy.size() << "\n" << std::endl;
    std::cout << "vCopy contains value: " << vCopy.at(0) << "\n" << std::endl;

    // 6) Check Move Constructor
    vector vMove = std::move(v3);
    std::cout << "vMove has size " << vCopy.size() << "\n" << std::endl;
    std::cout << "vMove contains value: " << vCopy.at(0) << "\n" << std::endl;

    std::cout << "v3 has size " << v3.size() << "\n" << std::endl;

    std::cout << "*****************************" << std::endl;
    std::cout << "**** VECTOR TEST PASSED *****" << std::endl;
    std::cout << "*****************************" << std::endl;
    return 0;
}
```

3 Submission

Submit your source code (as a zip-file) to Ilias EXERCISE-05 **before the deadline** specified in Ilias.