C++ course – Exercises Set 6

Wouter Verkerke (Jan 2021)

Exercise 6.1 – A generic container class

The goal of this exercise is to rewrite an class Array that mimics the behavior of a C++ array, into a template class Array that works for any data type

- Start with the input class `ex6.2/Array.hh` that implements a simple implementation of an array of double values.

  Convince yourself that the class correctly implements the constructor, destructor, copy constructor and assignment operator.

- Change the `class Array` into a `template class Array`.

- Use `class Array` in a small test program to store an array of `int` and to store an array of `const char*` strings.

- Does the code in `operator[]` look safe to you? What happens if you try to access element 1000 of an array of length 10?

- Change the `operator[]` such that when an element beyond the range of array is accessed, the array is automatically extended to include that element using the `resize()` function.

Exercise 6.2 – Revisiting the class Stack

The goal of this exercise is to revisit the `Stack` *class from module 3 and revisit its storage strategy from a 'raw' C++ array into the use of the 'smart' class* `Array`

- Copy the provided solution for the `Stack` class from module 3, its main program as well as the `Array` class from exercise 6.1. Compile the code and verify that works OK.

- We will now rewrite `class Stack` to use `class Array` for internal storage. Start with the data members of class Stack: In the solution of Ex. 3.3, the data is stored using an array `double *s` with an associated length `int len`. Replace these two data members with an `Array<double> s`.
  (Don't forget to include the `Array.h` header file in `Stack.h`)

- There are several places in the code of `Stack` that use the length of the internal memory buffer that used to be stored in `len`. This information is now available from `Array<double> s`, so replace each occurrence of `len` in the code with a call to `s.size()`, which reports the size of the buffer in `s`.

- Do you still need an explicit copy constructor and destructor for `class Stack`? Hint: do you still have pointers to 'owned' memory as data members? If not, remove them.

- Adapt the constructor to initialize all data members: initialize `s` by calling its constructor with the size that is passed to the `Stack` constructor, and set the initial value of `count` to 0, as usual. You can eliminate the function `init()`, since it is now superfluous.

- Remove the `grow()` function entirely since its functionality is now mostly absorbed in `Array`. In `push()` replace `grow()` with a call to `s.resize()` that has the same effect

- Test the modified `Stack` class with the main program.

- As a final step, turn `class Stack` into a `template class Stack`. To do so, you need to change any reference to type `double` to a template type `T` in the code, and add a `template<class T>` line to the class declaration.

- Test the template class `Stack<T>` with the main program. First try this with a `Stack<double>`, then with a `Stack<const char*>`