Course C^{++}, Exercise List 7

Deadline: 21.04.2015

1. We first try to understand how often a vector copies or moves its contents. Consider the following class:

```
struct xx
{
   int val;
   static unsigned int copied;
   static unsigned int moved;

   explicit xx( int val ) : val{val}
   { }

   xx( xx&& x )
      : val{ x. val }
   {
      std::cout << "move constr " << val << "\n";
      ++ moved;
   }

   xx( const xx& x )
      : val{ x. val }
   {
      std::cout << "copy constr " << val << "\n";
      ++ copied;
   }
};

unsigned int xx::copied = 0;
unsigned int xx::moved = 0;
```

   The static variables are variables that exist independent of the class objects. They are created when the program starts, and destroyed when the program terminates.

   In general, one should be careful with static variables in C^{++} because they obscure information flow, and there are problems with initialization order, when static variables are initialized in different files.

(a) Write code that creates an empty `std::vector<xx>`, and that pushes a lot (≈ 20) of elements. How often are objects copied? Is the move constructor used? Is the move constructor used when reallocating the vector?

(b) Add **noexcept** to the move constructor, and try again.

(c) Verify that the vector is never reallocated by `pop_back( )`. Verify that the vector reallocates when `shrink_to_fit( )` is called.

(d) If you know in advance that the vector will be big, you can use `reserve( size_t )` to reserve the required space at once. This will avoid reallocations. Check that it works.

2. Do the same with a list, (creating a list, and pushing approximately twenty elements. Verify that no objects are moved or copied. In order to avoid any use of copy constructors, you can use `emplace_back( )`.

3. Write a function `double average( const std::vector<double> & )` that computes the average value of the contents of a vector of **doubles**. Average is defined as $\frac{\sum_{i=1}^{n} v_i}{n}$.

   This can be done in two different ways, either using iterators or using **range**-for. Implement both ways.

4. Averages can be computed for every number field that has reasonable definitions of addition and division. In particular, it should be possible for the rational class of List 2.

   We could write functions

```
std::ostream& operator << ( std::ostream& out,
                            const std::vector< rational > & vect );
rational average( const std::vector< rational > & r )
```

   but it is a bit silly to write very similar functions for different number types. Therefore, we will use a technique called *templates*:

```
template< typename X >
std::ostream& operator << ( std::ostream& out,
                            const std::vector<X> & vect );
template< typename X >
X average( const std::vector<X> & vect );
```

   Apply the template functions on vectors of different number types, at least on **rational**, **double** and **float**.

5. If one assumes that a vector $a = (a_0, a_1, a_2, \ldots, a_{n-1})$ represents the polynomial

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1},$$

   then one can define a function

```
std::vector<double >
  multiply( const std::vector<double> & p1,
            const std::vector<double> & p2 )
```

that multiplies polynomials. Polynomial multiplication is defined as

$$a.b = \lambda i \sum_{j \leq i} a_j . b_{i-j}.$$

(This expression identifies a_i and $a(i)$.)

Write such a multiplication function. How much is $(x + 0.1)(x + 0.2)(x + 0.3)...(x + 0.9)$?

6. Make the previous multiplication function into a **template**. How much is $(x + \frac{1}{2})(x + \frac{1}{3})(x + \frac{1}{4})$? How much is $(x + \frac{1}{2})^5$?

   We don't use `operator*` because it would be wrong to assume that every `std::vector<X>` is a polynomial. There are other uses of vector as well. If one wants to use `operator*`, one must define a class `template<typename X> polynomial<X>`.