Course C^{++}

Exercise List 3

Deadline: 18.03.2014

Topic of this task are the *essential methods*.

1. Define (in a file `stack.h`) a class

```
class stack
{
   unsigned int current_size;
   unsigned int current_capacity;
   double* tab;
      // class invariant is that tab is always
      // allocated with a block with current_capacity.
      // We ignore the fact that normally,
      // elements between current_size and current_capacity
      // are not initialized.
   void ensure_capacity( unsigned int c );
      // Ensure that stack has capacity of at least c.

public:
   stack( );              // Constructs empty stack.
   stack( const stack& s );
   ~stack( );
   void operator = ( const stack& s );
      // These are three essential methods.
      // Later we will also use
      // void operator = ( stack&& s ) and
      // stack( stack&& s ).

   void push( double d );  // Use ensure_capacity, so that
                           // pushing is always possible, as
                           // long as memory is not full.

   void reset( unsigned int s );
                           // Pops element until stack has size s.

   void pop( );
```

```
         // Remove one element from the stack. It’s OK to write
         // code that crashes, as long as you write clearly what are
         // your preconditions, so:
         // PRECONDITION:  The stack is not empty.

      double& top( );
      double top( ) const;
         // The second one is used when stack is const.
         // The first one allows assignment.
         // Both have precondition that the stack is non-empty.
      unsigned int size( ) const { return current_size; }
      bool nonempty( ) const { return current_size; }

   };
```

Below is the definition of `ensure_capacity()`. Write the other methods by yourself (in a file with name `stack.cpp`)

```
   stack::ensure_capacity( unsigned int c )
   {
      if( current_capacity < c )
      {
         // New capacity will be the greater of c and
         // 2 * current_capacity.

         if( c < 2 * current_capacity )
            c = 2 * current_capacity;

         double* newtab = new double[ c ];
         for( unsigned int i = 0; i < current_size; ++ i )
            newtab[i] = tab[i];

         current_capacity = c;
         delete[] tab;
         tab = newtab;
      }
   }
```

2. If you wrote the copy constructor, the assignment operator, and the destructor correctly, then your class now has *value semantics.*

   It is time to check that your implementation of stack has no memory leaks. The easiest way to test this, is by implementing the following program, which contains initialization, assignment, self assignment, and destruction.

```
for( unsigned int i = 0; i < 1000000; ++ i )
{
   stack s1;
   s1. push_back(45); s1. push_back(45); s1. push_back(46);

   stack s2 = s1;
   for( unsigned int j = 0; j < 20; ++ j )
      s2. push_back( j * j );

   s1 = s2;
   s1 = s1;
}
```

Use the `top` command in another terminal, to ensure that the memory use of your program is not increasing.

3. Write

```
std::ostream& operator << ( std::ostream& , const stack& s );
```

Make it a friend of `class stack`, by adding

```
friend std::ostream& operator << ( std::ostream& stream, const stack& s );
```