Course C^{++}

Exercise List 3

Deadline: 17.03.2015

Topic of this task are the *essential methods*.

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

```
class stack
{
   size_t current_size;
   size_t current_capacity;
      // size_t is an integer number >= 0. It should be used for
      // sizes of objects, for indexing (because an index lies
      // between 0 and the size of the object) and for hash values
      // (because a hash value will be used for indexing.)
      // size_t is guaranteed to be big enough for the memory
      // of every computer, now and in the future.
      // size_t is an alias. Hence you need to include something
      // from standard library in order to have it.

   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( size_t 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 the essential methods.
      // Later we will also encounter
      // 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 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.

   void reset( size_t s );
      // Pops element until stack has size s.
      // PRECONDITION: s <= current_size.

   double& top( );
   double top( ) const;
      // The second one is used when stack was declared const.
      // The first one allows assignment.
      // Both have precondition that the stack is non-empty.

   size_t size( ) const { return current_size; }
   bool empty( ) const { return current_size == 0; }

};
```

Below is a definition of `ensure_capacity()`. Write the other methods by yourself. (in a file with name `stack.cpp`) Small methods (up to three lines) can be written in `stack.h`. Be sure to use initializers wherever possible.

```
void stack::ensure_capacity( size_t 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( size_t 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(45); s1. push(45); s1. push(46);

      stack s2 = s1;

      // j is not size_t, because multiplying size_t with itself is
      // unnatural:

      for( unsigned int j = 0; j < 20; ++ j )
         s2. push( j * j );

      s1 = s2;
         // Assignment.
      s1 = s1;
         // Self assignemnt should always be checked.
#if 0
      // Won't compile. In order to get it compiled, remove const:

      const stack& sconst = s1;
      sconst. top( ) = 20;
      sconst. push(15);
#endif
   }
```

   Use the `top` command in another terminal, to ensure that the memory use of your program is not increasing. You may also use `valgrind`, which is installed in lab 07 and 107.

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 );
```