CSCI 235, Programming Languages, C^{++}
Exercise 3

Deadline: 10/12.09.2018 (the day of your lab)

Topic of this exercise are the use of defined operators, and the use of private fields.

1. Extend the **string** class, that can be downloaded with the exercise, with the following methods (in **class string**):

   ```
   char operator [] ( size_t i ) const;
   char& operator [] ( size_t i );
   size_t size( ) const;
   ```

   Show in **main** that you know how to use these methods.

   If you want the indexing operators to check bounds, throw a `std::runtime_error`.

   As a general rule, it is better not to check preconditions in C^{++}. Preconditions that are completely obvious, need not be announced (thou shalt not divide by zero, thou shalt not cast thy index outside the scope of thy container.) If there are non-obvious preconditions, they should be announced in a comment (in the **.h** file.)

   The very worst that you can do, when a precondition is broken, is return a nonsense value.

2. Complete the following operators (in file **string.cpp**):

   ```
   string& operator += ( char c ); // Append c at the end.
   string& operator += ( string& s1, const string& s2 );
   ```

   The first operator is defined as member function of **class string** in file **string.cpp**. It needs to reallocate **p**, so it is better when it is a member.

   The second operator is not a member function. It simply calls the first operator with the characters in `s2`. This implementation is very inefficient, but we are not going to worry about it.

   Make sure that the second operator handles self assignent correctly: `s += s;`

3. Extend the **string** class with
   `string operator + ( string s1, const string& s2 )`. Just use `operator +=`.
   We don't worry about efficiency in this exercise.

4. Add the following to **class string**:

```
using iterator = char* ;
using const_iterator = const char* ;
const_iterator begin( ) const { return p; }
const_iterator end( ) const { return p + len; }
iterator begin( ) { return p; }
iterator end( ) { return p + len; }
```

   Now you can rewrite `operator <<` in file **string.cpp** as follows:

```
std::ostream& operator << ( std::ostream& out, const string& s )
{
   for( char c : s )
      out << c;
   return out;
}
```

5. Implement the following operators in file **string.cpp**:

```
bool operator == ( const string& s1, const string& s2 );
bool operator < ( const string& s1, const string& s2 );
```

   Add the following operators (as **inlines**) in file **string.h**:

```
bool operator != ( const string& s1, const string& s2 );
bool operator > ( const string& s1, const string& s2 );
bool operator <= ( const string& s1, const string& s2 );
bool operator >= ( const string& s1, const string& s2 );
```