General Instructions

1. Download `Practical03.zip` from the course website.

   Extract the file under the `[CppCourse]\[Practicals]` folder. Make sure that the file structure looks like:

```
[CppCourse]
   -> [boostRoot]
   ...
   -> [Practicals]
      -> [Practical01]
      -> ...
      -> [Practical03]
         -> ...
         -> LinkedLists.hpp
         -> ...
         -> [Src]
            -> LinkedLists.cpp
            -> ...
```

2. Open the text file `[CppCourse]\CMakeLists.txt`, uncomment the following line by removing the `#`:

   ```
   #add_subdirectory(Practicals/Practical03)
   ```

   and save the file. This registers the project with `cmake`.

3. Run `cmake` in order ot generate the project.

4. The declaration of the functions to be implemented are in `Practical03Exercises.hpp`.

5. Create one or more `.cpp` files under the `[Src]` folder for the implementation.

6. Implement the functions into the newly added `cpp` file(s). Do not modify any of the other files.

7. Compile and run your code. If the minimum requirements are met, an output text file will be created. Hand in your `cpp` files and the output file

   `Practical03_output.txt`

   via Moodle.

The exercises are based on the `LinkedList03` example from the lecture. For convenience, the declarations and definitions of linked list are reproduced in the practical project (although duplicating code is not particularly good practice).

Exercise 1

```
unsigned int length(const LinkedList03 & l);
```

```
unsigned int lengthRecursive(const LinkedList03 & l);
```

Both of these functions take a linked list, and return its length. The first function should not call itself recursively, whereas the second one is expected to be the recursive version.

Exercise 2

```
bool hasElement(const LinkedList03 & l, int i);
```

```
bool hasElementRecursive(const LinkedList03 & l, int i);
```

Both of these functions take a linked list together with an integer, and return `true` exactly when the integer is the data element in at least one node of the list. The first function should not call itself recursively, whereas the second one is expected to be the recursive version.

Exercise 3

```
int lastElement(const LinkedList03 & l);
```

```
int lastElementRecursive(const LinkedList03 & l);
```

Both of these functions take a linked list, and return the data in the last node of the input list. Note: each list has a head-node, hence the function is always well defined. The first function should not call itself recursively, whereas the second one is expected to be the recursive version.

Exercise 4

```
int nextToLast(const LinkedList03 & l);
```

```
int nextToLastRecursive(const LinkedList03 & l);
```

Both of these functions take a linked list, and return the data in the next-to-last node of the input list. Note: there is no guarantee that the input list has a next-to-last node, hence the function is not well defined. For simplicity, you can assume that the input list has at least two nodes. Discussion question: what should the function do if we don't make the assumption?
The first function should not call itself recursively, whereas the second one is expected to be the recursive version.

Exercise 4

```
1  LinkedList03 clone(const LinkedList03 & l);
```

The function takes a linked list as input, and returns a linked list, which is a clone of the input argument. Note: the clone is a linked list that has the same number of nodes as the original list, and the nth node in the clone has the same data as the nth node in the original list. Moreover, the clone does not share any node instances with the original list.