Exercise 6 - Simple Stack

Based on our struct `Element` *from last weeks exercise sheet, we are going to implement a stack-like structure.*

If you have any questions or problems, please write in the Moodle!

Exercise 1 — Groundwork

For this Exercise write your methods into `Stack.cpp`*. Declare your methods in* `Stack.h`*.*
Try out your code as usual in `main.cpp`*.*

To get started, have a look at `Stack.h`*, in which you will find the* `class Stack` *. Examine the private fields of the given* `class` *and the struct declaration of* `Element`*.*

a) Write the method `void push(int value)`, which creates a new `Element` holding the given `value`. That `Element` is than pushed to the beginning of the stack. i.E. `head` points to the new `Element` and the `next` pointer of the new `Element` points to where the head was previously pointing to. Remember to also change the length of your stack.
Hint: You can use `new Element{value}` *to create a new Element with the passed value.*

b) Complete the method `void pop()` you can find in `Stack.cpp`. The method should properly remove the current `head Element` from the `Stack`. Therefore the previously `second Element` of the `Stack` shall now be stored in `head`. Remember to also change the length of your stack.

Hint: `delete <object-pointer>` *will* `free` *the object from the* `heap`*. Leaving you with just a* `Pointer` *to an invalid object.*

c) Write the method `int size()` returning the `length` of the `stack`.

Exercise 2 — Useful functionalities

For this Exercise write your methods into `Stack.cpp`*. Declare your functions in* `Stack.h`*.*
Try out your code as usual in `main.cpp`*.*

a) Write the method `void print()` iterating over the entire `Stack` and printing each `value` stored.
Format your output as seen below:

```
Stack my_Stack;
my_Stack.push(3);
my_Stack.push(2);
my_Stack.push(1);
my_Stack.print();
```

Output: [1,2,3]

b) In `main.cpp` write a function `Stack primeStack(int upper_bound)` returning a Stack filled with all `prime numbers` less than the `upper_bound`.

   You can use your `isPrime function` *from previous exercises or our implementation provided in* `main.cpp`*.*

c) Finally write the `deconstructor: ~Stack()` using the `delete <object-pointer>` again. The method shall iterate over the `Stack` freeing all `Elements` from the `heap`.

   You might want to print a message inside of the `deconstructor`*, so you can see when it is automatically invoked when running your code.*