# Stacks & Queues

# Stacks & Queues

*Introduction*

## Background

We are going to create our own stack library from scratch. The result is a library that can be used to create multiple stacks and hide the implementation details of the stacks from the users. Below, the requirements of the stack library are enumerated.

## Requirements

Non-functional

1. The stacks are implemented as a single library
2. The stacks are non-reentrant i.e. thread-safety is not a requirement
3. The stacks are built as a separate single library and used as such
4. A stack shall be designated by its handle

Functional

1. The stacklib shall be able to create multiple stacks
2. The stacklib shall push data to a designated stack
3. The stacklib shall pop data from a designated stack
4. The stacklib shall be able to destroy a designated stack
5. The stacklib shall report the number of elements residing in a designated stack

# *Part 1*

Guideline: ± ?? hours of work

## Preparatory study activities

Read the sheets about the operation of stacks and queues carefully

## Challenge details

- Implement the stack in directory mystacklib according to the requirements in this document and the API description that is given in the header file mystack.h
- Note that only the source files are given. You are expected to test the stack yourself by employing unit tests.
- You are expected to investigate what is needed to compile your own programs in directory mystack using your stack library by dynamically linking to it. There exist plenty of examples on the net of Makefiles that do this. Hint: if you have trouble running your code (error: libmystack.so => not found), please look at what the commandline program ldd can do for you and look into the LD_LIBRARY_PATH environment variable.
- You *must* dynamically link to your library: it is explicitly forbidden to directly compile your stack code into your application.
- Discuss what needs to change in the stack code to transform the stack into a queue. Write this down in your documentation.

# *Part 2*

Guideline: ± ?? hours of work

1. Write a program that reverses a linked list by using the mystacklib. The linked list must contain at least 1500 elements. Use a memory checker to verify the absence of memory leaks. Show proof in your documentation.

2. Write a different program: a calculator that uses reverse polish notation (postfix) to calculate the results **by using your stack library**. The calculator shall support the following operations: +, -, *, /, $x^2$, √. The calculator must be precise up to 3 decimals. +, -, * and / are binary operations i.e. they operate on two operands that are located on the stack. $x^2$ and √ are unary operators i.e. they operate on one operand (the top of the stack) only. The result of each operation is always pushed on top of the stack. If you have trouble with the $^2$ and √ symbols: just use any other symbol (e.g S for square and R for root).

*If you need an extra challenge here: change your program so it accepts normal (infix) calculations. Your program should then convert the infix notation to postfix notation (following the correct priorities) and finish by solving it. Example: "1 + 2 * 3" will then be converted to: "1 2 3 * +", which will then be solved to 7.*

Example calculations:

| Result | Calculation |
|---|---|
| **3** | 1 2 + |
| **7** | 1 2 3 * + |
| **8** | 5 6 9 √ - + |
| **19** | 5 3 * 4 + |

Please keep your user interface code and calculator code separated!

**Demonstrate:**

1. All code of all challenge parts
2. Your documentation that has:
   - 2.1. Title page with date and name
   - 2.2. Short introduction
   - 2.3. Your research: Show all relevant research steps (DOT) and elaborate on the choices made
   - 2.4. Your design: all diagrams with accompanying texts and research
   - 2.5. How you tested your implementation including proof (how you validated the results and so on)
   - 2.6. Reflection on what you have learned
   - 2.7. Bibliography (sources)