# Stacks and Queues

You’re going to learn:

* working of stack and queue abstract data structures
* difference between static and dynamic library
* and of course, a bit of algorithms and extra experience with programming

## Part 1: Stacks

We are going to create our own stack library from scratch. The result is a library that can be used to create stacks with generic values (values of generic types). The library will hide the implementation details of the stacks from the users. The created stack library will be used later on as a dynamic library to create a queue application of the second part of the assignment

Below, the requirements of the stack library are described.

## Requirements

Non-functional

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

Functional

1. The *mystack* shall be able to create a stack
2. The *mystack* shall be able to push data
3. The *mystack* shall be able to pop data
4. The *mystack* shall be able to destroy earlier created stack
5. The *mystack* shall report the number of elements residing in a stack
6. The *mystack* allocates and frees memory necessary to store the stack elements which are passed in void \* pointers
7. The *mystack* allocates and frees memory necessary to create all stack data structures

Implement the stack according to the requirements in this document and the API description that is given in the header file mystack.h. Note that only the library files are given. You are expected to test the mystack yourself by employing unit tests. Don’t forget to check the memory use of your unit tests.

*TIP: think well how to store data passed in the void \*obj pointer. We’re not storing only pointers but the data which they point to. What is the size of this data? Memcpy function could be useful.*

## Part 2: Queues

You're going to create a simple *myqueue* library. This library will make use of dynamic *mystack* library created in the first part of the assignment.

All necessary Makefiles are provided. Please, do your own research how to use them (you don't have to change them only understand them in big lines) to build *myqueue* unit test.

Your queue library will be made based on algorithm that uses 2 stacks to create a queue. There are many references to this algorithm on Internet, one of them is here:

<https://www.youtube.com/watch?v=AN0axYeLue0>

The basic queue structure for this algorithm is provided in myqueue.h, you have to reuse it and of course you are not allowed to change the API (the same is valid for *mystack* library).

## Part 3 (Optional): Stack / Queue Algorithm

The optional assignment is only students willing to exercise even more and earn better grades.

Solve the following stack/queue assignment:

You are given a stack of **N** integers such that the first element represents the top of the stack and the last element represents the bottom of the stack.

You need to pop at least one element from the stack. At any one moment, you can convert stack into a queue. The bottom of the stack represents the front of the queue. You cannot convert the queue back into a stack. Your task is to remove exactly **K** elements such that the sum of the **K**removed elements is maximised.

Input format :

The first line consists of two space-separated integers **N** and **K**.

The second line consists of **N** space-separated integers denoting the elements of the stack.

Output format :

Print the maximum possible sum of the **K**removed elements

Constraints :

1≤N≤100000

1≤K≤N

1≤Ai≤100000

**SAMPLE INPUT**

10 5 10 9 1 2 3 4 5 6 7 8

**SAMPLE OUTPUT**

40

Explanation

Pop two elements from the stack. i.e {10,9}

Then convert the stack into queue and remove first three elements from the queue. i.e {8,7,6}.

The maximum possible sum is 10+9+8+7+6 = 40

You can test your program similarly to assignment 1, by inputting provided test files.

program.exe < input\_test\_file.

The following test files are provided:

1. in1: expected result 12595
2. in2: expected result 7510
3. in3: expected result 2942
4. in4: expected result 1455267608604

or by running ‘make test’.

### Delivery:

Please deliver a zip file of ONLY! ass3 directory and not some other folders.

This is the last assignment of the C part of SD and during grading the quality of the code (good unit tests) and the code cleanness/readability will be a very important part. If you feel like you need to explain any programming decisions, don’t hesitate to put it in a document (separately delivered, so not in the same zip). Document is not compulsory.