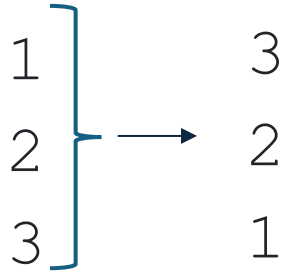
Lab Stack/Queue & Arrays/LinkedLists

(Exercise 1) Revert raws

- Implement a Stack (of integers) by using a LinekedList
  - You can reuse your implementation, if you have already implemented such a stack in the previous lab or in the previous exercise
- Write a program that takes as input a file with an integer per line and it copies in another file the raw of the input file but inverting their order
- The library <fstream> can be used

Variant: check the input from the file are only numbers, discard otherwise



(Exercise 2) Control the formulae

- Implement a Stack (of integers) by using a LinekedList
  - You can reuse your implementation, if you have already implemented such a stack in the previous lab or in the previous exercise
- Write a program that takes as input, from the user, a sequence of characters (max 100 characters) and determines if the parenthesis are balanced

- Implementation constraints:
  - Each node of the stack is defined as

```
struct node {
   int value;
   node* next;
};
```

  - The sequence of characters is provided by the user via standard input
  - The sequence of characters is stored in an array of chars (e.g., “char buffer[101];”)
  - The characters in the sequence can be any valid characters, i.e., not only parentheses

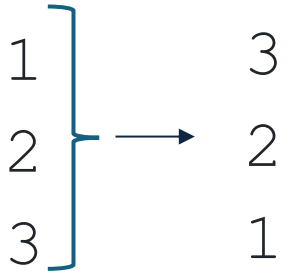
```
( ( ( ) ( ) ) )   =>   YES

( ) ( ) ( ( )     =>   NO
```

(Exercise 3) Revert stack

- Implement a Stack (of integers) by using a LinekedList
  - You can reuse your implementation, if you have already implemented such a stack in the previous lab or in the previous exercise
- Implement a Queue (of integers) by using a LinekedList
- Write a program that reads some input (integer numbers) from a user (i.e., from the standard input), stores it in a stack, and then inverts the elements of the stack by using a queue

- Implementation constraints:
  - Each node of the stack and in queue is defined as

```
struct ... {
   int value;
   ...* next;
};
```



(Exercise 4) Palindrome

- Implement a Stack (of integers/chars) by using a LinekedList
  - You can reuse your implementation, if you have already implemented such a stack in the previous lab or in the previous exercise
- Implement a Queue (of integers/chars) by using a LinekedList
  - You can reuse your implementation, if you have already implemented such a stack in the previous lab or in the previous exercise
- Write a program that, given an input word (maximum 100 valid characters representing a word), says if it is palindrome word or not using a stack and / or a queue.

Def. A palindrome is a word or a sequence of symbols that reads the same backwards as forwards, For instance, *madam* or *racecar, kayak, deified, rotator, repaper, deed, peep, wow, noon, civic, racecar, level, mom, bird rib, taco cat, …*

Implementation constraints

#1: Each node of the stack and in queue is defined as

```
struct … {
   int value;  //or alternatively: char value;
   …* next;
};
```

#2: The word (sequence of characters) is provided by the user via standard input, and it is stored in an array of chars (e.g., “char buffer[101];”)

#3: You can't take advantage of the knowledge of the length of the word (you can't even re-calculate it)

(Exercise 5) Recursive functions with List

Implement a program that create and use a linked list with two operations:

- *insert*: add a new node at the end of the linked list (it is used to create the list from inputs provided from the user or read from a file).
- *traverse*: traverse the linked list and print the data stored in each node (it is used to show in the standard output the built list).

Example #1:

Input: Values to be inserted = 1, 2, 3, 4, 5

Output: 1 -> 2 -> 3 -> 4-> 5 -> NULL

Example #2:

Input: Values to be inserted = 7, 8, 4, 11, 44, 6

Output: 7 -> 8 -> 4 -> 11 -> 44 -> 6 -> NULL

Implementation constraint #1: Use an an iterative function to implement the operation *insert*, and use a recursive function to implement the operation *traverse*

Variant #2: Use two recursive functions, one for implementing the operation *insert* and the other to implement the operation *traverse*

(Exercise 6) Search in a List

Implement a program that given a linked list of integers and a key (integer), checks if key value is present in the linked list or not.

Example #1

- Input: 14 -> 21 -> 11 -> 30 -> 10, key = 14
- Output: Yes

Example #2:

- Input: 6 -> 21 -> 17 -> 30 -> 10 -> 8, key = 13
- Output: No

Implementation constraint : The *key* is provided from the user via standard input while the list can be fixed, generated randomly, or generated with numbers read from a file

Variant: Use a recursive function to implement also the *search* operation

(Exercise 7) Ordering a List

Implement a program that:

- It reads some integer numbers (less than 100) from the user (standard input), from a file, or it generates them randomly
  - If needed, the program can ask to the user the number of integers she wants to insert
- It builds a linked list with these integers,
- It prints the value of the integers in the list
- It generates a new list of ordered items of the original list (ascending order)
- It prints the value of the integers of the new ordered list

Example #1

- Input nr. integers: 8
- Input list: 11 1 0 2 7 88 78 7
- Output: 0 -> 1 -> 2 -> 7 -> 7 -> 11 -> 78 -> 88

Implementation constraint : Use a recursive function to implement also the *ordering operation*

(Exercise 8) Josephus Problem

You need to solve the Josephus problem using a circular linked list.

- There are N people in a circle like 1->2->3...->N->1 and there is a token.
- The person who has the token eliminates the person next person, and she hands over the token, i.e., if the person 2 has the token, in a list like: 1->2->3->1, then 2 eliminates 3 and hands over the token to 1.
- This process continues until there is only one person left, i.e., there is no one left to eliminate.
- This last person is deemed as the winner.
- Initially the token is with person 1.

For a given integer N, you have to define a circular linked list containing elements from 1 to N with head initially at 1. As solution, you need to output a single integer denoting the winner.

Implementation constraint: use a circular linkedlist

Example #1

- For n=5, the moves are as followed (the person in bracket holds the token):
- (1)->2->3->4->5->1 (1 eliminates 2 and hands token to 3)
- 1->(3)->4->5->1 (3 eliminates 4 and hands token to 5)
- 1->3->(5)->1 (5 eliminates 1 and hands token to 3)
- (3)->5 (3 eliminates 5 and is the only person left)
- Therefore, 3 is the winner.