Course: DD2325 - Exercise Set 1

Exercise 1: *Recursion I*

If the program below executed with `cheers(3)` what will it print out

```
function cheers(n)
if n == 1
  disp('Hurray')
else
  disp('Hip ')
  cheers(n-1);
end
```

Note! The general outline of a successful recursive function defintion is:

- One or more cases in which the function accomplishes its task by using recursive call(s) to accomplish one or more smaller versions of the task.
- One or more cases in which the function accomplishes its task without the use of any recursive calls. These cases without any recursive calls are called **base cases** or **stopping cases**.

What is the **stopping case** in the program `cheers`?

Exercise 2: *Recursion II*

Write pseudo-code to compute $\mathtt{x}^n$ where `n` is a positive integer using recursion and also iteration.

Exercise 3: *Recursion III*

Write pseudo-code to compute $\mathtt{x}^n$ where `n` may be either a positive or negative integer using recursion and also iteration.

Remember ! If you are writing a recursive function that returns a value, you need to check the following:

1. There is no infinite recursion.
2. Each stopping case returns the correct value for that case.

3. For the cases that involve recursion: if all recursive calls return the correct value, then the final value returned by the function is the correct value.

Exercise 4: *Stacks I*

Describe a real world "stack".

Exercise 5: *Stacks II*

Imagine a very simple text editor. Each character you type on the keyboard is pushed onto the stack `S`. If you type the erase character, this character is recognised and not pushed onto `S` but causes the editor to pop `S`. Then when you finish your sentence you type the "return" key. When this happens the elements of `S` are printed on the screen in *reverse* order.

How can the stack be written in reverse order?

Exercise 6: *Binary Search*

Let `b` be the array containing the numbers `1` to `1000` inclusive and in order, that is `b = [1, 2, 3, ..., 1000]`. I create a new array `c` by copying `b` and removing `n` of its elements. Note that `c` is still sorted so that `c(i) > c(i-1)` for `i>1`. I now give you a number `x` such that `1` $\leq$ `x` $\leq$ `1000` and ask the question - Does the array `c` contain `x`? How can you answer this question efficiently and correctly ?

What are the pros and cons in terms of efficiency and correctness of:

1. Just answering yes?
2. Searching sequentially?
3. Performing a binary search?

How does the value of `n` effect your choice of algorithm?

If `n=1` what is the most number of comparisons your algorithm would have to perform during a binary search?

Exercise 7: *Binary Search + Recursion*

In the lecture on Tuesday you were shown how to implement a binary search algorithm iteratively. In pseudo-code write down how you could implement binary search recursively.

Exercise 8: *Reverse Polish Notation I*

Write the reverse Polish notation for the following *infix* expressions

very basic:	`4+5`	`4*5`	`4/5`
basic:	`4+(5+6)`	`(4+5)+6`	
moderate:	`(4+5)*6`	`4+5*6`	
less moderate:	`(4+5*6)/2`	`4/2 + 5*6`	

Exercise 9: *Reverse Polish Notation II*

By hand run through the algorithm (given in the lab notes) that produces a reverse Polish notation string from an infix string for the expression `(4+5*6)/2` and `4/2 + 5*6`.

Exercise 10: *Reverse Polish Notation III*

By hand run through the algorithm (given in the lab notes) that evaluates a reverse Polish notation string. Use the strings found in the previous exercise.