## Wooyoung Jung 114744214 - CSE216_07h

### Taylor Expansion Again (60)

1. Start by researching how to write comments in OCaml. Then explore the exponential function ** in OCaml. What is its type signature? Write your result of this question as a comment below.
   # (* float -> float -> float = <fun> *);;
2. Implement a function *factorial* with a type signature int -> int. For example, computing the factorial of 5 should yield a result of 120. Fill in the following:

```
# let rec factorial n =
    if n = 1 then 1
    else n * factorial (n-1);;
```

3. Design a Taylor expansion function *taylor* with the type float -> int -> float. This function should compute Taylor expansion of *e^x* around 0, of the first n terms. When you call your function with the arguments taylor 0.1 3, it should return exactly 1.105. Using taylor 0.1 10 should produce a result close to but different from 1.105. (1) Include your Taylor implementation below and (2) Record the result of taylor 0.1 10 as as a comment. Fill in the following:

(1)
```
# let rec taylor n =
    if n = 1 then 1.
    else (exp x (n-1) /. float_fact (n-1)) +.
taylor x (n-1)
    (* computes x^n and n! separately using the
helper functions exp and float_fact, which are listed
below.
    val taylor : float -> int -> float = <fun>
*);;

# let exp x n =
    x ** float_of_int n
    (* computes x^n by casting n as a float and
using the float exponentiation operator **
    val exp : float -> int -> float = <fun> *);;

# let rec factorial n =
    if n <= 1 then 1
    else n * factorial (n-1)
    (* computes n! recursively by computing n *
factorial (n-1) until it reaches its base case n <=1 ->
```

```
1
              val factorial : int -> int = <fun> *);;

       # let float_fact n = float_of_int (factorial n)
       (* computes n! and casts it into a float value so
that it can be used within the taylor function
       val float_fact : int -> float = <fun> *);;

   (2)(* taylor 0.1 10 = 1.10517091807564727 *);;
```

**Tower of Hanoi (4)**

First, play the game of Tower of Hanoi yourself to get an idea: https://www.mathisfun.com/games/towerofhanoi.html

After you understand the rule of the game, implement a function move of type

```
int -> string -> string -> string -> unit
```

so that

```
move n src dst aux
```

moves n disks from src to dst using aux as an auxiliary disk.

Hint for the implementation:

- If n is 1, print the movement from src to dst
- otherwise, move n-1 disks from src to aux, move 1 disk from src to dst, and move n-1 disks from aux to dst.
- use Printf.printf "Move from %s to %s\n"
- for a series of expressions use being ... end, e.g. begin move...; move...; move... end.
- You probably need to do some additional research and much try-and-error to get your ocaml code work and run.

Task: Fill in the following:

```
let rec move n src dst aux =
    if n = 1 then Printf.printf "Move from %s to %s" src
dst
    else begin
        move (n-1) src aux dst;
        move (1) src dst aux;
        move (n-1) aux dst src;
    end
```

```
(* for testing *)
let test() =
    move 3 "A" "C" "B"

let _ = test ()
```

How to test: Suppose the code above is in a file hanoi.ml, then running ocaml hanoi.ml will generate:

```
Move from A to C
Move from A to B
Move from C to B
Move from A to C
Move form B to A
Move from B to C
Move from A to C
```