Lab 10

Problem 1

* And

let **my\_and** conds = match conds with

    | (true, x) -> x

    | \_ -> false

;;

* Or

let **my\_or** conds = match conds with

    | (false, x) -> x

    | \_ -> true

;;

* Test Cases:

Printf.printf "and(T, T): %b\n" (my\_and (true, true));;

Printf.printf "and(T, F): %b\n" (my\_and (true, false));;

Printf.printf "and(F, T): %b\n" (my\_and (false, true));;

Printf.printf "and(F, F): %b\n" (my\_and (false, false));;

*(\* and(T, T) = true \*)*

*(\* and(T, F) = false \*)*

*(\* and(F, T) = false \*)*

*(\* and(F, F) = false \*)*

Printf.printf "or(T, T): %b\n" (my\_or (true, true));;

Printf.printf "or(T, F): %b\n" (my\_or (true, false));;

Printf.printf "or(F, T): %b\n" (my\_or (false, true));;

Printf.printf "or(F, F): %b\n" (my\_or (false, false));;

*(\* or(T, T) = true \*)*

*(\* or(T, F) = true \*)*

*(\* or(F, T) = true \*)*

*(\* or(F, F) = false \*)*

Problem 2

* Head

let **get\_head** list = match list with

    | head::\_ -> head

    | [] -> raise (Failure "get\_head: empty list")

;;

* Tail

let **get\_tail** list = match list with

    | \_::tail -> tail

    | [] -> raise (Failure "get\_tail: empty list")

;;

* Test Cases:

get\_head [1; 2; 3];;

get\_tail [1; 2; 3];;

*(\* int = 1 \*)*

*(\* int list = [2; 3] \*)*

get\_head [];;

get\_tail [];;

*(\* Exception: (Failure "get\_head: empty list") \*)*

*(\* Exception: (Failure "get\_tail: empty list") \*)*

Problem 3

* Reverse

let rec **reverse** list = match list with

    | [] -> []

    | head::tail -> reverse tail @ [head]

;;

* Test Cases:

reverse [1; 2; 3; 4; 5];;

*(\* int list = [5; 4; 3; 2; 1] \*)*

reverse [1];;

*(\* int list = [1] \*)*

reverse [];;

*(\* int list = [] \*)*

Problem 4

* Rotate Left

let **rotate\_left** list = match list with

    | head::tail -> tail @ [head]

    | [] -> []

;;

* Rotate Right

let **rotate\_right** = fun list ->

    List.rev (rotate\_left (List.rev list))

;;

* Test Cases:

rotate\_left [1; 2; 3; 4; 5];;

*(\* int list = [2; 3; 4; 5; 1] \*)*

rotate\_right [1; 2; 3; 4; 5];;

*(\* int list = [4; 5; 1; 2; 3] \*)*

Problem 5

* Maximum

let rec **maximum** list = match list with

    | [x] -> x

    | head::tail -> max head (maximum tail)

    | [] -> raise (Failure "maximum: empty list")

;;

* Test Cases:

maximum [1; 2; 3; 4; 5; 3; 4; 1];;

*(\* int = 5 \*)*

maximum [];;

*(\* Exception: (Failure "maximum: empty list") \*)*

Problem 6

* Apply

let rec **apply** func init list = match list with

    | [] -> init

    | head::tail -> func head (apply func init tail)

;;

* Test Cases:

let **sum** x y = x + y;;

let **mult** x y = x \* y;;

Printf.printf "%d\n" (apply sum 0 [1; 2; 3; 4; 5]);

*(\* int = 15 \*)*

Printf.printf "%d\n" (apply mult 1 [1; 2; 3; 4; 5]);

*(\* int = 120 \*)*

Printf.printf "%d\n" (apply sum 5 []);

*(\* int = 5 \*)*

Problem 7

* Complex

type **complex** = {re: float; im: float};;

* Complex Add

let **add** = fun num1 num2 ->

    {

        re = num1.re +. num2.re;

        im = num1.im +. num2.im

    }

;;

* Complex Multiply

let **mult** = fun num1 num2 ->

    {

        re = num1.re \*. num2.re -. num1.im \*. num2.im;

        im = num1.re \*. num2.im +. num1.im \*. num2.re;

    }

;;

* Test Cases:

let **num1** = {re = 1.2; im = 2.3};;

let **num2** = {re = 3.5; im = 4.25};;

add num1 num2;;

*(\* complex = {re = 4.7; im = 6.55};; \*)*

mult num1 num2;;

*(\* complex = {re = -5.575; im = 13.15};; \*)*