1

0. (basic ocaml expressions) 12 points total
Below are a series of ocaml expressions entered to the ocaml interpreter (EVAL). Directly below each, show the response from the ocaml interpreter (**as it would be displayed by ocaml**, including type). If you believe an error results, simply write 'ERROR'.

```
# [1;2;3;4]::[5;6;7;8];;
```
'ERROR'

```
# List.tl (List.tl [1;2;3;4;5]);;
```
- : int list = [3;4;5]

```
# List.tl (List.hd [1;2;3;4;5]);;
```
'ERROR'

```
# "6" @ [5;4;3;2;1];;
```
'ERROR'

```
# 6 :: [5;4;3;2;1];;
```
- : int list = [6;5;4;3;2;1]

```
# if (6==9) then print_string "unit" else print_string "I don't mind";;
```
I don't mind - : unit = ()

-1

2

1. (λ-calculus) 16 points total.
Recall the 4 productions which define the lambda calculus are:

<expression> ::=
<variable>
| <constant>
| (<expression> <expression>)
| (λ <variable> . <expression>)

Which of the following are syntactically correct λ-calculus expressions (using only the 4 productions in the lambda calculus)? Note: `lambda` = λ.

- **CIRCLE THOSE WHICH ARE SYNTACTICALLY CORRECT.**
- **IF AN EXPRESSION IS SYNTACTICALLY CORRECT, ALSO INDICATE THE OVERALL (TOP-LEVEL) TYPE (i.e., variable, constant, abstraction or combination).**

3/4

(lambda x.x)

variable ABSTRACTION

((lambda x.x) 4)

combination

((lambda x).x) 4))

((lambda x.x).(lambda x.x))

2. (ocaml functions, signatures and profiling) 32 points total.
Consider the following set of ocaml functions. They should look familiar.

```
(* do_something.ml -- for profiling
*)

let rec recursiveFn = function n->
if n==0 then []
        else
        "3520" :: recursiveFn (n-1) ;;

let rec do_something_aux = function (i,alist) ->
  if (alist == []) then []
                   else
                      (i, (List.hd alist)) ::
                      do_something_aux(i+1,(List.tl alist));;

let do_something = function (i,j) ->
    do_something_aux (i, (recursiveFn j));;

(* something to profile *)

let ans = do_something(10,20);;
```

(a) Show the ocaml function signatures (as displayed by EVAL) below each of the following functions (2 points each):

(i) recursiveFn

-1 val recursiveFn : int -> ~~int~~ string list = <fun>

(ii) do_something_aux

-1 val do_something_aux : int * 'a list -> (int * 'a) list = <fun>

(iii) do_something

-1 val do_something : int * int -> ~~'a~~ (int * string) list = <fun>

(b) Show the result (as ocaml would report it) for the following function evaluation (6 points):

-1 do_something(10,20);;

-: int list = [(10, 3520); (11, 3520); (12, 3520); ... ; (29, 3520); ...]

(c) These functions are to be profiled using `ocamlprof` **when evaluating** the above function application, i.e., `do_something(10,20)`. I've broken the profiling down into 2 cases:

1. Profiling function use (only); and
2. Profiling IF-THEN-ELSE use (only).

Fill in the values of ans #1 - ans #7, below, i.e., the profiler values where shown in each function for each case. (3 points each)

Case 1: Profiling Functions

```
$ ocamlcp -o do_something_prof do_something.ml
(* note defaults to -P fm; -p = -P for backwards compatibility *)

(* do_something.ml -- for profiling
*)

let rec recursiveFn = function n->
(* ans1= 21*) if n==0 then []
        else
        "3520" :: recursiveFn (n-1) ;;

let rec do_something_aux = function (i,alist) ->
  (* ans2= 21 *) if (alist == []) then []
                 else
                    (i, (List.hd alist)) ::
                    do_something_aux(i+1,(List.tl alist));;

let do_something = function (i,j) ->
    (* ans3=  1 *) do_something_aux (i, (recursiveFn j));;

(* something to profile *)

let ans = do_something(10,20);;
```

Case 2: Profiling Branches

```
$ ocamlcp -P i -o do_something_prof do_something.ml

(* do_something.ml -- for profiling
*)

let rec recursiveFn = function n->
if n==0 then (* ans4= 1 *) []
        else
        (* ans5= 20*) "3520" :: recursiveFn (n-1) ;;

let rec do_something_aux = function (i,alist) ->
  if (alist == []) then (* ans6= 1 *) []
                   else
                      (* ans7= 20 *) (i, (List.hd alist)) ::
                      do_something_aux(i+1,(List.tl alist));;

let do_something = function (i,j) ->
    do_something_aux (i, (recursiveFn j));;

(* something to profile *)

let ans = do_something(10,20);;
```

3. (lists, pattern matching, exceptions) 27 points total.
Here's some challenges with lists and pattern matching. Consider the following ocaml functions:

```
(* struct.caml
   for quiz 2
   hope you like this problem *)

let nav_list_of_list = function
  | hd::tl -> ([], hd, tl)
  | [] -> failwith "invalid_arg -- empty list";;

let current = function
  | (_, item, _) -> item;;

let next = function
  | (prev, item, next::next_tl) ->
      (item::prev, next, next_tl)
  | _ ->
      failwith "end of nav_list reached";;

let prev = function
  | prev::prev_tl, item, next ->
      prev_tl, prev, item::next
  | _ ->
      failwith "begin of nav_list reached"
```

Below each of the following ocaml evaluations, show the response as ocaml would report it:

```
# let mylist = nav_list_of_list [5;6;7;8;9];;
```

val mylist : 'a list * int * int list = ([], 5, [6;7;8;9])

```
# current mylist;;
```

- : int = 5

```
# let nextlist = next mylist;;
```

val nextlist : int list * int * int list = ([5], 6, [7;8;9])

```
# let nextnext = next nextlist;;
```
val nextnext: int list * int * int list = ([6;5], 7, [8;9])

```
# let nextnextnext = next nextnext;;
```
val nextnextnext: int list * int * int list = ([7;6;5], 8, [9])

```
# prev nextnextnext;;
```
- : int list * int * int list = ([6;5], 7, [8;9])

```
# prev nextnext;;
```
- : int list * int * int list = ([5], 6, [7;8;9])

```
# prev nextlist;;
```
- : int list * int * int list = ([], 5, [6;7;8;9])

```
# prev (prev nextlist);;
```
failure begin of nav-list reached

4. (`Software licenses and Copyrights`) 12 points total.
Each of the following statements are either TRUE or FALSE. Circle each TRUE statement.

- Copyright is a form of protection grounded in the U.S. Constitution and granted by law for original works of authorship fixed in a tangible medium of expression.
- Copyright protects original works of authorship including literary, dramatic, musical, and artistic works, such as poetry, novels, movies, songs, computer software, and architecture.
- Copyright does not protect facts, ideas, systems, or methods of operation, although it may protect the way these things are expressed.
- Copyright protects original works of authorship, while a patent protects inventions or discoveries.
- Your work is under copyright protection the moment it is created and fixed in a tangible form that it is perceptible either directly or with the aid of a machine or device.
- It is necessary to register with the U.S. Copyright office to be protected.