CS F301 Principles of Programming Languages [1st Semester 2024-2025]
Comprehensive Exam [December 14th, 2024] [Max Marks: 109.5] [Duration: 180 mins]

Part A: Lambda Calculus [Estimated Time: 40 minutes]

For each of the below mentioned Lambda Calculus expressions, (1) **Identify the free variables** (Write the original expression in the answer sheet and **underline** or **encircle** the free variables in the expression itself) (2) **Reduce each expression to its Beta Normal Form.** (Clearly show each step of the reduction process. Write the intermediate expressions explicitly in each step. If the expression is not reducible to Beta Normal Form, mention the same along with a one line reason.)

1. (λx.λy.λz.x (λw.wz) (y z)) (λz.zz) (λx.xy) y **[9 marks]**
2. (λa.λb.a (b (λa.aa))) (λx.xz) (λy.yy) (λz.z) **[7.5 marks]**
3. (λx.λy.y (x (λz.zw))) (λz.zz) (λw.ww) w **[9 marks]**
4. (λx.λy.x ((λz.zx) (λx.xz))) (λa.ab) (λb.bb) b **[9 marks]**

PART B: OCaml Programming Language [Estimated time: 140 minutes]

Implement a train reservation system using a custom recursive list data structure (**my_list**) instead of OCaml's built-in lists. The system should support common operations such as:
- Basic list operations (map, filter, fold, sort) for the custom list **(my_list)**
- Train sorting
- Seat availability checking and booking
- Tatkal pricing, Passenger management and searching

The required types and data structures are provided. Your task is to **implement the UNIMPLEMENTED functions according to their specifications.** Please note the following:
- Do not modify the already implemented functions.
- Do not change the function signatures (arguments and return types).
- Do not use any built-in list functions (like List.map, List.filter, etc.).
- Do not add any other functions in the global scope. You are free to add any helper functions within the body of the provided function definitions.
- Do not use for and while loops.
- You don't need to handle errors.
- Minor syntactical errors (; or ;;) will be overlooked. For any other syntactical errors, you may be penalised if it drastically affects the correctness of the code. Final decision will be taken by the instructor.
- For each function, we have provided a hint which contains the list of functions you can/should use to implement the function.
- You need not strictly follow the hint. But we advice you use the hint and not overthink.
- Please take 5-10 minutes to read and understand the full question paper carefully before you start.
- Most importantly, do not overthink! There are no tricky questions. Most of the questions are straightforward. All the best!

```
(* ===================== CODE STARTS HERE ===================== *)
(* Type definitions *)
type 'a option = None | Some of 'a

type 'a my_list = Empty | Node of 'a * 'a my_list

type station = {code: string; name: string; arrival_time: string;
          departure_time: string; distance_from_source: int}

type seat_class = Sleeper | AC3 | AC2 | AC1

type passenger = {name: string; age: int; gender: string;}

type seat_availability = {class_type: seat_class; price: float; available_seats: int}
```

type **train** = {train_number: string; train_name: string; classes: seat_availability my_list; schedule: station my_list; departure_time: string; arrival_time: string}

type **booking** = {user_name: string; train_number: string; class_booked: seat_class; passengers: passenger my_list; is_tatkal: bool}

(** **<u>Important Assumptions:</u>** For Simplicity,
1. Our train reservation system does not have any notion of date. So, whenever we talk about any operation such as **booking a ticket** or **checking seat availability** in a specific train, we do not care about date. In other words, there is only a single instance of any train for which we are going to perform operations. Think of reservation system for just a single day.
2. Passengers are not alloted any specific seat numbers. We will just check the seat availability in a specific class in the specific train to book tickets. While cancelling tickets, we will accordingly update the seat availability. ***)**

(<u>QUESTION 1</u>: Implement my_map**
This function applies a function to each element of a list to create a new list
Input:
- f: function of type ('a -> 'b) to apply to each element
- lst: input list of type 'a my_list
Returns:
- a new list of type 'b my_list containing f applied to each element
Examples:
my_map (fun x -> x * 2) (Node(1, Node(2, Node(3, Empty))))
(* returns Node(2, Node(4, Node(6, Empty))) because each number is multiplied by 2 *)
Hint: Use pattern matching and recursion ***)**

let rec my_map (f: 'a -> 'b) (lst: 'a my_list) : 'b my_list =
  (* QUESTION 1: YOUR CODE HERE [3 marks] *)

(<u>QUESTION 2</u>: Implement my_filter**
This function returns only those elements in a my_list that satisfy a given condition
Input:
- f: function of type ('a -> bool) that tests each element
- lst: input list of type 'a my_list
Returns:
- a new list of type ‘a my_list containing only elements for which f returns true
Examples:
my_filter (fun x -> x > 2) (Node(1, Node(3, Node(2, Empty))))
(* returns Node(3, Empty) because only 3 is greater than 2 *)
Hint: Use pattern matching and recursion ***)**

let rec my_filter (f: 'a -> bool) (lst: 'a my_list) : 'a my_list =
  (* QUESTION 2: YOUR CODE HERE [4 marks] *)

(<u>QUESTION 3</u>: Implement my_length**
  This function counts the number of elements in a list
  Input:
  - lst: input list of type 'a my_list
  Returns:
  - an integer representing the number of elements in the list
  Examples:
  my_length (Node(1, Node(2, Node(3, Empty))))
  (* returns 3 because there are three elements: 1, 2, and 3 *)
  Hint: Use pattern matching and recursion ***)**
let rec my_length (lst: 'a my_list) : int =
  (* QUESTION 3: YOUR CODE HERE [3 marks] *)

(** **QUESTION 4: Implement my_fold_left**
This function combines all elements of a list using an accumulator function
Input:
- f: function of type ('a -> 'b -> 'a) that combines accumulator with each element
- acc: initial accumulator value of type 'a
- lst: input list of type 'b my_list

Returns:
- final accumulator value of type 'a after processing all elements (starting from left of the list)

Examples:
my_fold_left (fun acc x -> acc ^ x) "" (Node("a", Node("b", Node("c", Empty))))
(* returns "abc" because: "" ^ "a" = "a", then "a" ^ "b" = "ab", then "ab" ^ "c" = "abc" *)

Hint: Use pattern matching and recursion ***)**

let rec my_fold_left (f: 'a -> 'b -> 'a) (acc: 'a) (lst: 'b my_list) : 'a =
(* QUESTION 4: YOUR CODE HERE [4 marks] *)

(** **QUESTION 5: Implement insert_sorted**
This function inserts an element into a sorted list (ascending order) using a comparison function
Input:
- cmp: comparison function of type ('a -> 'a -> int) that returns:
    negative if first arg < second arg
    zero if args are equal
    positive if first arg > second arg
- x: element to insert
- lst: sorted input list

Returns:
- a new sorted list with x inserted in the correct position

Examples:
insert_sorted compare 2 (Node(1, Node(3, Empty)))
(* returns Node(1, Node(2, Node(3, Empty))) because 2 belongs between 1 and 3 *)

Hint: Use pattern matching, recursion, and cmp ***)**

let rec insert_sorted (cmp: 'a -> 'a -> int) (x: 'a) (lst: 'a my_list) : 'a my_list =
(* QUESTION 5: YOUR CODE HERE [4 marks] *)

(** **QUESTION 6: Implement my_sort**
This function sorts a list in ascending order using a comparison function
Input:
- cmp: comparison function of type ('a -> 'a -> int) that returns:
    negative if first arg < second arg
    zero if args are equal
    positive if first arg > second arg
- lst: input list to sort
Returns:
- a new list with all elements sorted according to the comparison function
Examples:
my_sort compare (Node(3, Node(1, Node(2, Empty))))
(* returns Node(1, Node(2, Node(3, Empty))) because 1 < 2 < 3 *)
Hint: Use pattern matching, insert_sorted, cmp and recursion ***)**

let rec my_sort (cmp: 'a -> 'a -> int) (lst: 'a my_list) : 'a my_list =
(* QUESTION 6: YOUR CODE HERE [4 marks] *)

```
(** QUESTION 7: Implement my_mem
  This function checks if an element exists in a list
  Input:
  - x: element to search for
  - lst: list to search in

  Returns:
  - true if element exists in list, false otherwise

  Examples:
  my_mem 2 (Node(1, Node(2, Node(3, Empty))))
  (* returns true because 2 is in the list *)

  Hint: Use pattern matching and recursion *)

let rec my_mem (x: 'a) (lst: 'a my_list) : bool =
  (* QUESTION 7: YOUR CODE HERE  [4 marks] *)

(** QUESTION 8: Implement sort_trains_by_class
    This function sorts trains based on price or seat availability in a specific seat_class
    Input:
    - trains: list of trains to sort
    - class_type: seat class to compare (Sleeper, AC3, etc.)
    - sort_by: string indicating sort criterion ("price" or "available_seats")
    Returns:
    - list of trains sorted in ascending order by the specified criterion

    Examples:
    sort_trains_by_class trains Sleeper "price"
    (* returns trains sorted by price of Sleeper class *)
    sort_trains_by_class trains AC3 "available_seats"
    (* returns trains sorted by available seats in AC3 class *)

    Hint: Use pattern matching, my_sort, my_filter *)

let sort_trains_by_class (trains: train my_list) (class_type: seat_class)
    (sort_by: string) : train my_list =
  (* QUESTION 8: YOUR CODE HERE  [6 marks] *)

(** QUESTION 9: Implement check_seat_availability
    This function verifies if requested number of seats are available in a specific seat_class
    Input:
    - train: train to check
    - class_type: seat class to check (Sleeper, AC3, etc.)
    - num_passengers: number of seats needed
    Returns:
    - true if enough seats are available, false otherwise
    Examples:
    check_seat_availability train Sleeper 2
    (* returns true if Sleeper class has at least 2 seats available *)
    check_seat_availability train AC1 5
    (* returns false if AC1 class has less than 5 seats available *)
    Hint: Use pattern matching and my_filter
    [Before attempting this question, look at important assumptions mentioned just before question 1] *)

let check_seat_availability (train: train) (class_type: seat_class)
    (num_passengers: int) : bool =
  (* QUESTION 9: YOUR CODE HERE  [6 marks] *)
```

(<u>QUESTION 10</u>: Implement tatkal_pricing**
This function implements dynamic pricing for tatkal tickets
Input:
- surcharge: float (e.g., 1.5 for 50% extra charge)

Returns:
- a function of type (float -> float) that takes base price as input and calculates tatkal price

Examples:
let apply_tatkal = tatkal_pricing 1.5 in
apply_tatkal 1000.0;; (* returns 1500.0 because base_price * surcharge = 1000.0 * 1.5 *)

Hint: No hints for this question ***)**

let tatkal_pricing (surcharge: float) : (float -> float) =
(* QUESTION 10: YOUR CODE HERE [3 marks] *)

(<u>QUESTION 11</u>: Implement combine_passenger_lists**
This function merges two passenger lists into one
Input:
- acc: first passenger list (accumulator)
- p: second passenger list to add

Returns:
- a new list containing all passengers from both lists

Examples:
let p1 = {name="Alice"; age=25; gender="F";} in
let p2 = {name="Bob"; age=30; gender="M";} in
combine_passenger_lists (Node(p1, Empty)) (Node(p2, Empty))
(* returns Node(p2, Node(p1, Empty)) because it combines both lists *)

Hint: Use pattern matching and recursion *)

let rec combine_passenger_lists (acc: passenger my_list) (p: passenger my_list)
: passenger my_list =
(* QUESTION 11: YOUR CODE HERE [4 marks] *)

(** **<u>QUESTION 12</u>: Implement get_passengers_for_class**
This function finds all matching passengers in a specific class across all the bookings (need not necessarily belong to the same train). Passengers should satisfy the matching criteria.
Input:
- class_type: seat class to search in
- bookings: list of all bookings to search through
- filter_fn: function of type (passenger -> bool) that defines matching criteria for the passenger
Returns:
- matching_passenger_list

Examples:
get_passengers_for_class AC3 bookings (fun p -> p.gender = "F")
(* returns Node({name="Alice"; ...}, Node({name="Mary"; ...}, Empty))
if Alice and Mary are the female passengers in AC3 *)

Hint: Use my_fold_left, my_filter, and combine_passenger_lists ***)**

let get_passengers_for_class (class_type: seat_class) (bookings: booking my_list)
(filter_fn: passenger -> bool) : passenger my_list =
(* QUESTION 12: YOUR CODE HERE [6 marks] *)

(** **<u>QUESTION 13:</u> Implement search_passengers**
   This function finds matching passengers across all seat classes and all bookings. Passengers should satisfy the matching criteria.
   Input:
   - classes: list of seat classes to search through
   - bookings: list of all bookings to search through
   - filter_fn: function of type (passenger -> bool) that defines matching criteria for the passenger
   Returns:
   - list of tuples, each tuple of the type (seat_class * matching_passenger_list)
   Examples:
   search_passengers classes bookings (fun p -> p.age > 60)
   (* returns Node((Sleeper, Node({name="John"; age=65; ...}, Empty)),
           Node((AC2, Node({name="Mary"; age=70; ...}, Empty)), Empty))
     if John and Mary are senior citizens in their respective classes *)
   Hint: Use my_fold_left and get_passengers_for_class ***)**

let search_passengers (classes: seat_class my_list) (bookings: booking my_list)
   (filter_fn: passenger -> bool) : (seat_class * passenger my_list) my_list =
  (* QUESTION 13: YOUR CODE HERE [6 marks] *)

(<u>QUESTION 14:</u> Implement update_seats**
   This function modifies the available seats count for a specific class in a specific train
   Input:
   - train: train to update
   - class_type: seat class to modify
   - num_seats: number of seats to add (positive) or remove (negative)
   Returns:
   - updated (new) train record with modified seat count
   Examples:
   update_seats train Sleeper (-2) (* returns updated train with 2 fewer Sleeper seats available ***)**
   Hint: Use my_map *)

let update_seats (train: train) (class_type: seat_class) (num_seats: int) : train =
  (* QUESTION 14: YOUR CODE HERE [6 marks] *)

(QUESTION 15: Implement book_ticket**
This function tries to create booking for passengers and accordingly updates the train record
   Input:
   - user_name: name of person making the booking
   - train: train to book tickets on
   - passengers: list of passengers to book for
   - class_booked: desired seat class
   - is_tatkal: whether this is a tatkal booking
   Returns:
   - Some (booking_record, updated_train_record) if successful, None if seats not available

   Examples:
   book_ticket "Alice" train passengers Sleeper false
   (* returns Some (booking_record, updated_train_record) if booking succeeds *)
   book_ticket "Bob" train passengers AC1 true
   (* returns None if required seats are not available *)

   Hint: Use check_seat_availability, update_seats and my_length
   [Before attempting this question, look at important assumptions mentioned just before question 1] ***)**

let book_ticket (user_name: string) (train: train) (passengers: passenger my_list)
class_booked: seat_class) (is_tatkal: bool) : (booking * train) option =
  (* QUESTION 15: YOUR CODE HERE [6 marks] *)

(** **QUESTION 16: Implement cancel_tickets**
   This function removes specified seats from a specific booking in a specific train
   Input:
   - booking: booking to modify
   - train: train whose seat availability needs to be updated
   - passengers_to_cancel: list of passengers whose tickets need to be cancelled

   Returns:
   - tuple containing (updated booking * updated train)

   Examples:
   cancel_tickets booking train (Node({name = "Alice"; age = 25; gender = "F"}, Empty))
   (* returns (booking without Alice, train with 1 more available seat) *)

   Hint: Use my_filter, my_mem, my_length and update_seats
   [Before attempting this question, look at important assumptions mentioned just before question 1] ***)**

```
let cancel_tickets (booking: booking) (train: train) (passengers: passenger my_list)
   : booking * train =
  (* QUESTION 16: YOUR CODE HERE  [6 marks] *)
```

(MAIN FUNCTION - DO NOT MODIFY. YOU DON'T NEED THIS CODE. THIS IS JUST FOR SHOWING HOW THE ABOVE FUNCTIONS WORK. *)**

```
let main () =
  (* Create test data *)
  let train = {
    train_number = "12345";
    train_name = "Express A";
    classes = Node ({class_type = Sleeper; price = 500.0; available_seats = 100},
           Node ({class_type = AC3; price = 1200.0; available_seats = 50}, Empty));
    schedule = Node ({ code = "DEL"; name = "Delhi"; arrival_time = "--";
                   departure_time = "10:00"; distance_from_source = 0 },
                 Node ({ code = "BPL"; name = "Bhopal"; arrival_time = "15:00";
                     departure_time = "15:15"; distance_from_source = 700 }, Empty));
    departure_time = "10:00";
    arrival_time = "22:00"
  } in
  let trains = Node(train, Empty) in
  (* Test implemented functions *)
  assert (my_map (fun x -> x * 2) (Node(1, Node(2, Node(3, Empty)))) =
        Node(2, Node(4, Node(6, Empty))));

  assert (my_filter (fun x -> x > 2) (Node(1, Node(3, Node(2, Empty)))) =
        Node(3, Empty));

  assert (my_length (Node(1, Node(2, Node(3, Empty)))) = 3);

  assert (my_fold_left (+) 0 (Node(1, Node(2, Node(3, Empty)))) = 6);

  assert (my_sort compare (Node(3, Node(1, Node(2, Empty)))) =
        Node(1, Node(2, Node(3, Empty))));

  assert (my_mem 2 (Node(1, Node(2, Node(3, Empty)))) = true);
  assert (my_mem 4 (Node(1, Node(2, Node(3, Empty)))) = false);

  let sorted_trains = sort_trains_by_class trains Sleeper "price" in
  assert (my_length sorted_trains = 1);
  assert (check_seat_availability train Sleeper 2 = true);
  assert (check_seat_availability train Sleeper 101 = false);

  assert ((tatkal_pricing 1.5) 1000.0 = 1500.0);
  assert ((tatkal_pricing 2.0) 1000.0 = 2000.0);

  let booking = {
    user_name = "Alice";
    train_number = "12345";
    class_booked = Sleeper;
    passengers = Node({name = "Alice"; age = 25; gender = "F";},
              Node({name = "Bob"; age = 30; gender = "M";}, Empty));
    is_tatkal = false
  } in
  let bookings = Node(booking, Empty) in

  let senior_passengers = get_passengers_for_class Sleeper bookings (fun p -> p.age > 60) in
  assert (my_length senior_passengers = 0);

  let combined = combine_passenger_lists
    (Node({name = "Eve"; age = 40; gender = "F";}, Empty))
    (Node({name = "Frank"; age = 45; gender = "M";}, Empty)) in
  assert (my_length combined = 2);

  let classes = Node(Sleeper, Node(AC3, Empty)) in
  let adult_passengers = search_passengers classes bookings (fun p -> p.age >= 30) in
  assert (my_length adult_passengers = 2);

  let (updated_booking, updated_train) = cancel_tickets booking train (Node({name = "Alice"; age =
25; gender = "F"}, Empty)) in
  assert (my_length updated_booking.passengers = 1);

  let booking_result = book_ticket "Alice" train
    (Node({name = "Alice"; age = 30; gender = "F";}, Empty))
    Sleeper false in
  assert (booking_result <> None);

  sorted_trains

let () =
  try
    ignore (main ());
    print_endline "All tests passed!"
  with Assert_failure (file, line, position) ->
    Printf.printf "Test failed at %s, line %d, position %d\n" file line position
```
(* ===================== CODE ENDS HERE ====================== *)