Lab 2: Job Scheduling System in OCaml

CSI 3120 A - Programming Language Concepts

Fall 2024
School of Electrical Engineering and Computer Science
University of Ottawa

Anthony Le - 300287511
Iaroslav Volkov - 300235591

Experiment Date: Sep 20, 2024
Submission Date: Oct 4th, 2024

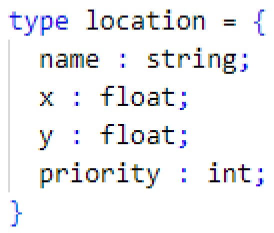
Lab tasks & their desired outputs

For this lab, we had been tasked with implementing a delivery and vehicle optimization program in OCaml. We continued our learning process on gathering data from input terminal inputs, and processing that input while creating various functions and approaches to maximize scheduling priority and minimize functional overlaps. Below, you will find the different functions & a small description & screenshot of their implementation, as well as the desired inputs & outputs for each of those functions.

Step 1: Define Location Representation

Define a `location` type that includes three fields: `name`, `coordinates(X, Y), and `priority` for each delivery location.
This structure is used to store information regarding the location of the shipment and its priority.
This 'job' type was implemented with the following steps:

- Defined **string** name
- Defined **float** x
- Defined **float** y
- Defined **int** priority



```
type location = {
  name : string;
  x : float;
  y : float;
  priority : int;
}
```

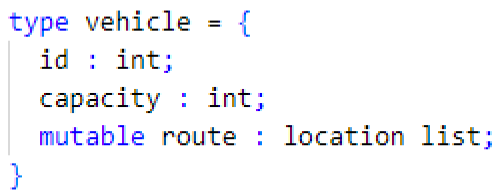
Figure 1. Code Snippet of Location type in OCaml.

There is no desired output, as this is a type definition.

Step 1: Define Vehicle Representation

Define a `vehicle` type that includes three fields: `id`, `capacity' , and `route` for each shipment vehicle
.
This structure is used to store information regarding the vehicle that will deliver the shipments and their capacity and route to the shipment locations.
This 'job' type was implemented with the following steps:

- Defined **int** id
- Defined **int** capacity
- Defined **a list of locations** route



```
type vehicle = {
  id : int;
  capacity : int;
  mutable route : location list;
}
```

Figure 2. Code Snippet of Vehicle Type in OCaml.

There is no desired output, as this is a type definition.

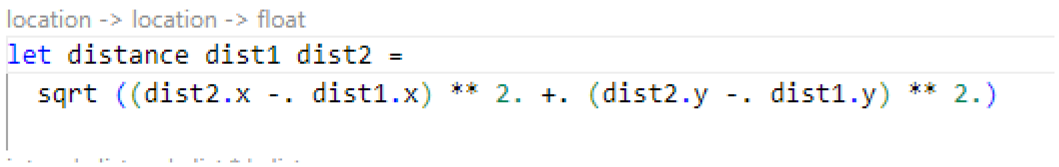
Step 2: Calculate the distance between two locations

Write a helper function that determines the distance between two locations to determine its total distance

Function that takes 2 locations and determines the total distance between them using the distance formula.

This function was implemented using the following implementation:

- Grabbing two locations and their x and y values and subbing them into the distance formula.



```
location -> location -> float
let distance dist1 dist2 =
  sqrt ((dist2.x -. dist1.x) ** 2. +. (dist2.y -. dist1.y) ** 2.)
```

Figure 3. Code Snippet distance Function in OCaml.

This function does not have a direct command line output, but is used as a helper function.

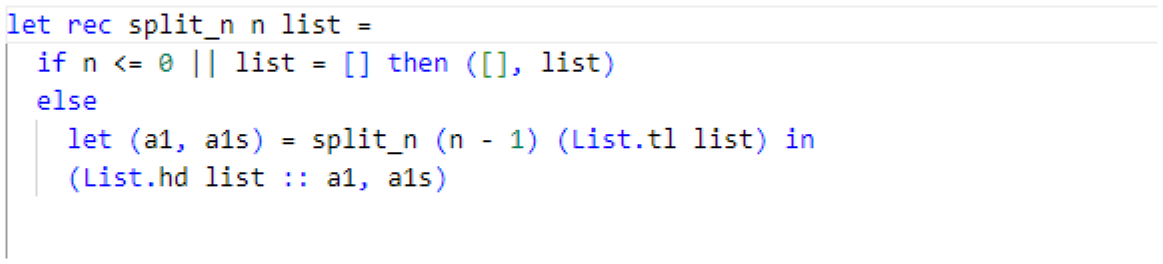
Step 3: Split_n helper function

Write a helper function that helps split the list of vehicles and locations to assign to different vehicles.

This recursive function was used in order to split the locations between the vehicles for a specific value of n.

This function was implemented using the following implementation:

- If the value of n was empty or if the list was empty, then we simply return an empty list
- If not, we would continue a recursive function and de-increment the value of n until it reached the base case.



```
let rec split_n n list =
  if n <= 0 || list = [] then ([], list)
  else
    let (a1, a1s) = split_n (n - 1) (List.tl list) in
    (List.hd list :: a1, a1s)
```

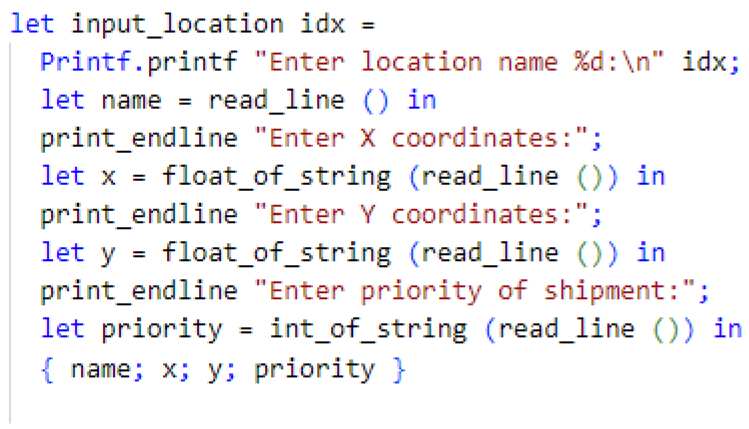
Figure 4. Code Snippet distance Function in OCaml.

Step 4: input in locations and vehicle

This function is responsible for inputting in the number of shipment locations that required delivery, and was the function that gathered the input from the terminal.

This function was used to gather the user's input in order to figure out the most optimal path to each location with the given number of locations and its information.

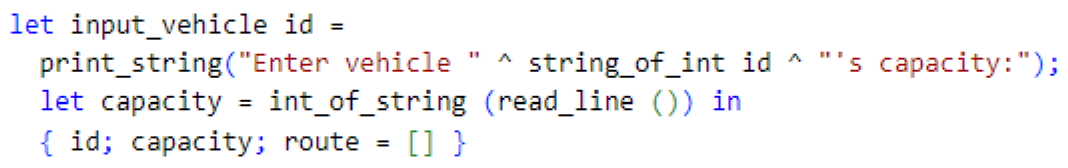
- This function simply asks for the location name, as well as its X and Y coordinates and the priority of the shipment.



```
let input_location idx =
  Printf.printf "Enter location name %d:\n" idx;
  let name = read_line () in
  print_endline "Enter X coordinates:";
  let x = float_of_string (read_line ()) in
  print_endline "Enter Y coordinates:";
  let y = float_of_string (read_line ()) in
  print_endline "Enter priority of shipment:";
  let priority = int_of_string (read_line ()) in
  { name; x; y; priority }
```

Figure 5.1. Code Snippet of the input_location function in OCaml.

Similarly, the input_vehicle function was used to gather input from the terminal regarding the number of vehicles used and the capacity of each vehicle while also initializing an empty list for the route in order to properly instantiate a vehicle object.

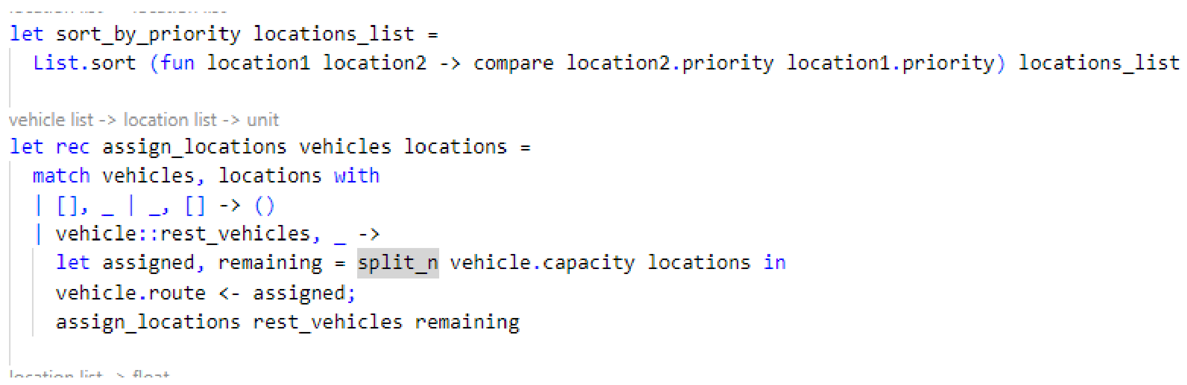


```
let input_vehicle id =
  print_string("Enter vehicle " ^ string_of_int id ^ "'s capacity:");
  let capacity = int_of_string (read_line ()) in
  { id; capacity; route = [] }
```

Figure 5.2. Code Snippet of the input_vehicle function in OCaml.

Step 5: prioritizing delivery locations and assigning

These two functions were responsible for prioritizing the locations based off of a comparison of the location's priorities, while also assigning the shipment locations to the appropriate vehicles. These two functions were used in tandem with the main function in order to deliver the final result of each vehicle's route.

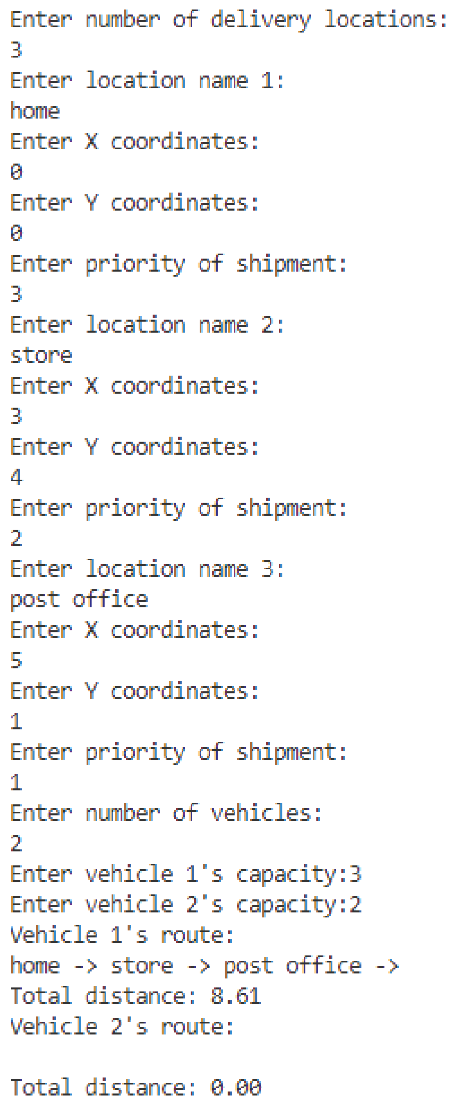


```
let sort_by_priority locations_list =
  List.sort (fun location1 location2 -> compare location2.priority location1.priority) locations_list

vehicle list -> location list -> unit
let rec assign_locations vehicles locations =
  match vehicles, locations with
  | [], _ | _, [] -> ()
  | vehicle::rest_vehicles, _ ->
    let assigned, remaining = split_n vehicle.capacity locations in
    vehicle.route <- assigned;
    assign_locations rest_vehicles remaining
```

Figure 6. Code snippet of the assign_locations and sort_by_priority functions

Desired inputs and outputs.



```
Enter number of delivery locations:
3
Enter location name 1:
home
Enter X coordinates:
0
Enter Y coordinates:
0
Enter priority of shipment:
3
Enter location name 2:
store
Enter X coordinates:
3
Enter Y coordinates:
4
Enter priority of shipment:
2
Enter location name 3:
post office
Enter X coordinates:
5
Enter Y coordinates:
1
Enter priority of shipment:
1
Enter number of vehicles:
2
Enter vehicle 1's capacity:3
Enter vehicle 2's capacity:2
Vehicle 1's route:
home -> store -> post office ->
Total distance: 8.61
Vehicle 2's route:

Total distance: 0.00
```

Figure 7. Inputs and outputs of final code.

Side note:
Apologies for the not-so detailed report, it has been a busy week and this was the best I can do.

References used for the lab.

- OCaml Documentation (https://ocaml.org/docs)
- ChatGpt [used for syntax help, & a general idea of how to properly code the priority of each shipment] (https://openai.com/chatgpt/)
- CSI3110 OCaml Programming (https://cs3110.github.io/textbook/chapters/basics/intro.html)