COMP1811 – Scheme Project Report

Name	Jake Cunningham	SID	001211278
Partner:	Ethan Moss	SID	001275239

1. Software Design

Briefly describe the design of your coursework software and the strategy you took for solving it. – e.g. did you choose either recursion or high-order programming and why…

Complete individually. Describe the design of the parts you implemented.

Both High-order programming and Recursion was used throughout to complete the required code for this project. Firstly, high order is used in functions like less-crowded, this is because in order to sort a comparison function is used enabling effective organisation of the inputted lists.

Additionally, recursion is used in functions such as lane-list?, It is used to check the properties of the list reputedly so that all of the inputs can be checked to ensure that they meet the requirements.

2. Code Listing

Provide a complete listing of the entire Scheme code you developed. Make sure your code is well commented, especially stressing informally the contracts for parameters on every symbol you may define. The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! Make it easy for the tutor to read. Marks WILL BE DEDUCTED if screenshots are included. Add explanatory narrative if necessary – though your in-code comments should be clear enough.

2.1 F1.i sim-secs2hour

```
(define (sim-secs2hour seconds) ; Define function
    (let* ((hours (quotient seconds 3600)) ; Calculate number of hours in seconds
           (remaining-seconds (modulo seconds 3600)) ; Calculate remaining seconds
           (minutes (quotient remaining-seconds 60)) ; Calculate number of minutes
           (secs (modulo remaining-seconds 60))) ; Calculate the remaining seconds
      (list hours minutes secs))) ; create a list containing hours, minutes, and seconds


  (sim-secs2hour 3661)
```

2.2 F2.I LANE?

```
(define (lane? val)
    (and (pair? val)           ; Check if val is a pair
         (number? (car val))   ; Check if the car of val is a number
         (pair? (cdr val))     ; Check if the cdr of val is a pair
         (or (null? (cadr val)) ; Check if the second element of val is null
             (number? (cadr val))) ; or a number
         (list? (cddr val)))) ; Check if the rest of the elements of val are a list
```

2.3 F2.II LANE-LENGTH

```
(define (lane-length ln)
    (if (lane? ln)            ; Check if input is a lane
        (length (cddr ln))    ; calculate the length of the queue portion
        #f)                   ; If not a lane, return false
    )
```

2.4 F2.III LANE-BUSSY

```
(define (lane-bussy? lane)
    (if (lane? lane) ; Check if the input is a lane
        (not (null? (cadr lane))) ; check if the cadr is not empty
        #f) ; Return false if not a lane
    )
```

2.5 F2. IV LANE-ID

```
(define (lane-id lane)
    (if (lane? lane)   ; Check if the input is a lane
        (car lane)     ; return the car of the lane object
        #f)            ; Return false if not a lane)
    )
```

2.6 F2.V LANE-USER

```
(define (lane-user val)
    (if (lane? val); Check if the input is a lane
        (cadr val) ; return the cadr of the lane object
        #f); Return false if not a lane)
    )
```

2.7 F2.VI LANE-QUEUE

```
(define (lane-queue val)
    (if (lane? val); Check if the input is a lane
        (cddr val); return the cddr of the lane object
        #f); Return false if not a lane)
    )
```

2.8 F3.I EVENT

```
(define (event time id params)  ; Define a function named event with three parameters: time, id, and params
    (cons time (cons id params)))  ; Create a list containing the time and another list containing id and params, then return this list
```

2.9 F3.II EVENT?

```
(define (event? val)

    (and (pair? val) ; Check if val is a pair

         (number? (car val)) ; Check if the car is a number

         (pair? (cdr val)) ; Check if the cdr of val is a pair

         (or (null? (cadr val)) (number? (cadr val))) ; Check if the second element of val is null or a number

))
```

2.10 F3.III EVENT-TIME

```
(define (event-time ev)

    (if (event? ev)      ; Check if ev is an event

        (car ev)         ; return its car

        #f)              ; If ev is not an event, return #f

    )
```

2.11 F3.IV EVENT-TYPE

```
(define (event-type ev)

    (if (event? ev)      ; Check if ev is an event

        (cadr ev)        ; returns the cadr

        #f)              ; If ev is not an event, returns #f

    )
```

2.12 F3.V EVENT-PARAMS

```
(define (event-params ev)

    (if (event? ev) ; Check if ev is an event

        (cons (caddr ev) (cdddr ev)) ;create a pair with the caddr and cdddr

        #f)) ; If ev is not an event, returns #f
```

2.13 F3.VI EVENT-USER

```
(define (event-user ev)

    (if (event? ev) ; Check if ev is an event

        (caddr ev) ; returns the caddr

        #f) ;If ev is not an event, returns #f

    )
```

2.14 F3.VII EVENT-LANE

```
(define (event-lane ev)

    (if (event? ev)  ; Check if ev is an event

        (cdddr ev)   ; return the cddr

        #f)          ; If ev is not an event, returns #f

    )
```

2.15 F4.I SIM-ADD-EVENT

2.16 F5.I LANE-LIST?

```
(define (lane-list? lst)  ; Define a function called lane-list?

    (cond

      ((null? lst) #t)    ; If the list is empty, return true

      ((not (list? (car lst))) #f)  ; If the first element of the list is not a list,
return false

      (else (lane-list? (cdr lst))))) ; Otherwise, call lane-list? on the rest of the list
```

2.17 F5.II LESS-CROWDED

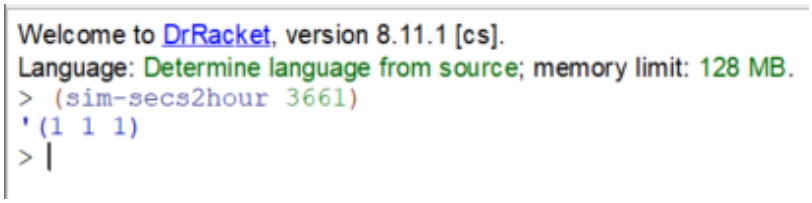
```
(define (less-crowded sim-lanes ) ; Define a function called less-crowded

    (car (sort sim-lanes ; take the car of the sorted the lanes based on the following

          (λ (a b) ; Define an anonymous function taking two lanes a and b

            (< ;less then to find smallest

             (+ (if (lane-bussy? a) 1 0) (lane-length a)) ; Add 1 to the length of lane a
if it's busy, otherwise add 0

             (+ (if (lane-bussy? b) 1 0) (lane-length b)) ; Add 1 to the length of lane b
if it's busy, otherwise add 0

          ))

    )))
```

RESULTS – OUTPUT OBTAINED

Provide screenshots that demonstrate the results generated by running your code. That is show the output obtained in the REPL when calling your functions. Alternatively, you may simply cut and paste from the REPL.

3.1 F1

Screenshot showing results of sim-secs2hour.

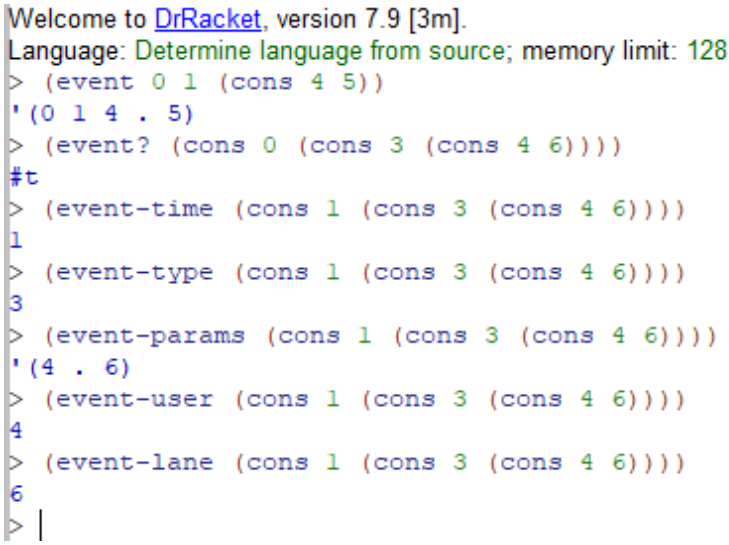


3.2 F2

Screenshot showing results of lane?, lane-length, lane-bussy?, lane-id, lane-user and lane-queue.

3.3 F3

Screenshot showing results of event, event?, event-time, event-type, event-params, event-user and event-lane.



3.4 F4

3.5 F5

Screenshot showing results of lane-list and less-crowded.

```
Welcome to DrRacket, version 7.9 [3m].
Language: Determine language from source; memory limit: 128 MB.
> (lane-list?
   (list
    (cons 1 (cons 2 empty))
    (cons 2 (cons null empty))
    (cons 3 (cons 4 (list 4 5 6)))))
#t
> (less-crowded
(list
(cons 1 (cons 2 empty))
(cons 2 (cons null empty))
(cons 3 (cons 4 (list 4 5 6)))))
'(2 ())
>
```

3. TESTING

Provide a test plan covering all of your functions and the results of applying the tests.

Test	Test Description	Expected Results	Actual Results	Pass/Fail
1	Sim-utils: Sec2hour input of 3661 seconds	Program show how manty hours, minutes, and seconds in a list (1 1 1)	'(1 1 1)	Pass
2	Sim-lane: Lane? Input (lane? (cons 0 (cons 3 (list 2))))	Results should return #t showing that the input is a list	#t	Pass
3	Sim-lane: Lane? Input (lane? (list 2))	Results should return #f showing that the input is not a list	#f	Pass
4	Sim-lane: lane-length Input (lane-length (cons 0 (cons 3 (list 1))))	Results should return 1 for the length of the list	1	Pass
5	Sim-lane: lane-length Input (lane-length (cons 0 (cons 3 (list 5 4 9))))	Results should return 3 for the length of the list	3	Pass
6	Sim-lane: lane-bussy Input (lane-bussy? (cons 0 (cons null (list 2 3))))	Lane is null so #f should be returned showing it is not busy	#f	Pass
7	Sim-lane: lane-bussy Input (lane-bussy? (cons 0 (cons 5 (list 2 3))))	Lane is 5 so #t should be returned showing the lane is busy	#t	Pass
8	Sim-lane: lane-id Input (lane-id (cons 10 (cons 3 (list 2 3))))	Lane ID is 10 so program should return 10	10	Pass

9	Sim-lane: lane-id Input (lane-id (cons 7 (cons 3 (list 2 3))))	Lane ID is 7 so program should return 7	7	Pass
10	Sim-lane: lane-user Input (lane-user (cons 0 (cons 7 (list 2 3))))	User ID is entered as 7 so program should return 7	7	Pass
11	Sim-lane: lane-user Input (lane-user (cons 0 (cons 3 (list 2 3))))	User ID is entered as 3 so program should return 3	3	Pass
12	Sim-lane: lane-queue Input (lane-queue (cons 0 (cons 3 (list 2 3))))	Queue contains 2 and so should return (2 3)	'(2 3)	Pass
13	Sim-lane: lane-queue Input (lane-queue (cons 0 (cons 3 (list 10 7 3))))	Queue contains 2 and so should return (10 7 3)	'(10 7 3)	Pass
14	Sim-event: event Input (event 0 1 (cons 4 5))	(0 1 4.5)	'(0 1 4	Pass
15	Sim-event: event? Input (event? (cons 0 (cons 3 (cons 4 6))))	Format is correct so #t should return	#t	Pass
16	Sim-event: event-time Input (event-time (cons 1 (cons 3 (cons 4 6))))	Time is 1, output should be 1	1	Pass
17	Sim-event: event-type Input (event-type (cons 1 (cons 3 (cons 4 6))))	Type is 3, output should be 3	3 >	Pass
18	Sim-event: event-params Input (event-params (cons 1 (cons 3 (cons 4 6))))	Last list is (4 6), output should be (4.6)	'(4 . 6)	Pass
19	Sim-event: event-user Input (event-user (cons 1 (cons 3 (cons 4 6))))	User id is 4, output should be 4	4	Pass
20	Sim-event: event-lane Input (event-lane (cons 1 (cons 3 (cons 4 6))))	Lane id is 6, output should be 6	6	Pass
21	Sim-lane-list: lane-list Input (lane-list? (list (cons 1 (cons 2 empty)) (cons 2 (cons null empty)) (cons 3 (cons 4 (list 4 5 6)))))	Format is correct so should return #t	#t	Pass
22	Sim-lane-list: Less crowdedInput (less-crowded (list (cons 1 (cons 2 empty)) (cons 2 (cons null empty)) (cons 3 (cons 4 (list 4 5 6)))))	Second list is the smallest so output should be 2	'(2 ())	Pass