

COMP1811 – Scheme Project Report

Name	Sanjida khan	SID	001276650 001339277
Partner:	Israt risma	SID	001339277

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.

F1(sim-utils):

1. Module Programming: It starts with defining the module itself. The function provides ('sim-secs2hour and sim-log'). This lists its dependencies which deals with the simulation of events and lanes. sim-sec2hour converts input to hours, minutes and seconds. On the other hand, 'sim-log' formats the event details, times, event descriptions and lanes involvements.

2. Strategy of solving: sim-utils is task to convert the inputs into hours, minutes and seconds. By using the 'sim-log' I confirmed that it includes the event details, event descriptions. This is mainly responsible for logging simulation events. Whereas sim-sec2hours is used for handling all the time related operations. It's useful when 'sim-log' presents the logging information accurately.

3. Function: I used high-order programming instead of recursion. It helps to have more flexible code. For example, I used 'event-time', 'event-user' as a parameter to other function make this code easier. Hence, if the implementation changes in the future, we only will need to change that function rather than everything. We also can reuse 'sim-log' in different events or functions. Moreover, by passing arguments it also can compose complex behaviour to simpler function. Higher order programming also can reduce dependencies between the functions. Although recursion would have made it more complex to edit since it's a group project.

(F2.Sim-Lane):

1. Module programming: The programme is organised in a modular fashion ,with several modules handling various aspects of the simulation. These may include handling, lane management, and consumer behaviour modelling, among other.

2. Strategy: The system is divided into more manageable modules or parts according to different functional domains, like event processing, lane management, and user behaviour modelling. The simulation is easy to comprehend and manage because each module focuses on a different part of it.

3.Function: I used higher-order functions that accept parameters from other functions and return functions when results are utilised, such as map and apply. Applying a function to every element of a list in lane->string, for instance, is accomplished with map .This is used to encourage reusability, improve code clarity and allow for more succinct implementations. The map function is a higher-order function which i used in lane->string: (map (lambda (x) (format "~a:" x)) (lane-queue ln)) It takes a function (the lambda) and applies it to each element of the list (lane-queue ln).

F3(sim-event):

1.Module Programming: In ‘sim-event’ the module encapsulates functionality related to simulation events. It represents the event as pairs where the first represents the time and second represents the nested pair containing data ,id and params. The ‘event?’ function shows if the a given value represents a valid object.it returns #t if it exists, or else #f.

2.Strategy: The constants LEAVE_EVENT, CHECKOUT_EVENT, and ENTER_EVENT are defined to represent various kinds of simulation events.

Constructor of Events: This function takes in the timestamp, ID, and parameters and builds a simulation event tuple with them.

Function of Predicate (event?):To determine whether a given value is a valid simulation event tuple, the event? function is used as the predicate.

Access Parameters (event-params):The parameters connected to a simulation event are obtained using this function.

3.Functions: The software is made more robust by the module's checks that make sure functions run on legitimate event tuples.

Through the provision of construction, access, and conversion functions, the event module efficiently oversees simulation events Complying with data abstraction principles and having a modular design make the coursework software easier to understand and maintain. Using Racket libraries also minimises implementation overhead and improves functionality.

F4(sim-lane-queue):

1.Module programming: The code supplied creates a priority queue system for simulation events using the sim-event-queue Racket module. Adding events to the queue is made possible by its main feature, sim-add-event.

The module is dependent upon an additional file, called "sim-event.rkt," which is most likely a file with definitions related to simulation events.

A sorted order of events is present in the queue, with the time attribute of each event serving as the main sorting factor. If events with the same time property are sorted again, this time according to the user attribute. It is the insert-event function that makes this sorting mechanism possible by making sure that freshly added events are placed correctly in the queue. When considering the size of the queue, the sorting process efficiency ensures that the time complexity is O(n).

2.Strategy: The code that is given creates a queue of simulation events that are ranked mostly by time in order to guarantee processing in the correct order. A more effective insertion approach is used, instead of sorting the entire queue upon addition. The insert-event function finds the proper location in the queue by linearly scanning it; new events are put in sorted order using this method. Events that have the same time attributes are ordered further based on other factors, such user attributes. This modular strategy improves readability and maintainability by separating queue code into the sim-event-queue module.

3.Function: Recursion is used by the insert-event function in the supplied code to add a new event to the sorted queue. This is a succinct explanation:

Recursion: Recursion is a programming approach where a problem is solved by a function calling itself. Recursion is utilised in the insert-event function to search through the queue and locate the ideal spot to insert the new event. The function repeatedly calls itself using the queue's tail until it finds the ideal spot for insertion.

High-Level Code Development: Functions are used as first-class citizens in high-order programming, which allows them to be returned as results or supplied as arguments to other functions.

Although the given code includes transferring an argument from one function (insert-event) to another (sim-add-event), this is not high-order programming.

F5(sim-event): My coursework software appears to design to simulate and manage a list of lanes, possibly representing queues in various scenarios such as checkout lines or service centres.

1.Modular programming: My module is structured as a module (sim-lane-list.rkt) that provides functionalities related to managing a list of lanes. The main module of the software is called sim-lane-list, and it is arranged into modules. For better organisation, understanding and functionality encapsulation are made possible by this modular approach.

2.Strategy : I have opted for a method in my coursework software that makes more use of higher-order programming than recursion. Programming at a higher level entails treating functions like first-class citizens and enabling them to be returned as values or supplied as arguments to other functions. Functions that accept functions as arguments are found in my code, such as sort, all-lanes?, and and map. The sort function, for example, allows the sorting behaviour to be customised by passing in a comparison function (compare-lanes) as an argument.

Function: To handle and alter data, I have used higher-order functions called map and sort, which are examples of functional abstractions. These functions encourage code reuse and readability by encapsulating popular computational patterns.

My code does not employ recursion directly, even though it could have been used for operations on lanes or traversing lists. Instead, I make use of the Racket language's built-in higher-order functions and abstractions, which frequently result in code that is more succinct and expressive. Ultimately, my decision to highlight higher-order programming is in line with Racket's functional design and helps me to write code for my coursework software that is understandable, succinct, and expressive.

2. CODE LISTING

Provide a complete listing of the entire Scheme code you developed. Make sure your code is well commented, specially 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.

Complete as a pair. Highlight which features each partner implemented.

2.1 F1.... ISRAT RISMA¹

```
(module sim-utils racket
  (provide sim-secs2hour
            sim-log!)

  (require "sim-event.rkt")
  (require "sim-lane.rkt")

  (define (sim-secs2hour seconds)
    (let* ((hours (quotient seconds 3600))
           (remaining-seconds (- seconds (* hours 3600)))
           (minutes (quotient remaining-seconds 60))
           (seconds (modulo remaining-seconds 60)))
      (list hours minutes seconds)))

  ; IO()
  (define (sim-log! time event lanes) ;; can omit time?
    (let*
        ([aux (sim-secs2hour (event-time event))]

         [hours (first aux)]
         [minutes (second aux)]
         [secs (third aux)]
         [ev (event->string event)]
         [user (event-user event)])
```

```

[lanes-str (apply
    string-append
    (map
        (lambda (x)(format "~n\t~a" (lane->string x)))
    lanes))]
)

(printf "~a:~a:~a|User: ~a|~a~a~n"
    (~r hours #:min-width 2 #:pad-string "0")
    (~r minutes #:min-width 2 #:pad-string "0")
    (~r secs #:min-width 2 #:pad-string "0")
    (~r user #:min-width 2 )
    ev
    lanes-str
)
))

)

```

2.2 F2. ... SANJIDA KHAN

```

;;done

(module sim-lane racket

(provide lane

lane?
lane-id
lane-user
lane-busy?)

```

```

lane->string

lane-length

lane-pop

lane-queue

; These routines fire events

lane-append!

lane-pop!

lane-bussy-set!

)

;; Import required module

(require racket/match)

(define sim-fire-event! null) ; hack to passed procedure

;; Define lane data structure

(define (lane id sym)

  (set! sim-fire-event! sym) ;; dirty hack

  (cons id (cons null null)))

;; Get length of a lane queue

(define (lane-length ln)

  ;; Contract: lane-length: lane -> integer

  ;; Returns the length of the queue in the given lane

```

```
(car(cdr(cdr ln)))  
)  
  
;; Check if a value represents a lane
```

```
(define (lane? val)  
(and (pair? val)  
(pair? (cdr val))  
(null? (cddr val)))  
)
```

```
;;Check if a lane is busy
```

```
(define (lane-busy? lane)  
(not (null? (cadr lane))))
```

```
;; Set the user of a lane
```

```
(define (lane-bussy-set ln user)  
(match ln  
[(cons id (cons _ q))  
(cons id (cons user q))]))
```

```
;; Get the ID of a lane
```

```
(define (lane-id lane)  
(car lane))
```

```

;; Get the user of a lane

(define (lane-user lane)
  (cadr lane))

;; Get the queue of a lane

(define (lane-queue lane)
  (cddr lane))

;; Converts lane data structure to string

(define (lane->string ln)
  (let
    ([lane-qu-str (apply
                    string-append
                    (map
                      (lambda (x)(format "~a:" x))
                      (lane-queue ln)))])

    )
  (format "L(~a)[~a]|<=~a"
         (lane-id ln)
         (~a
          (if (lane-busy? ln) (lane-user ln) "None" ) #:min-width 4)
         lane-qu-str)))

;; Append a user to a lane

(define (lane-append ln user)

```

```

(match ln
  [(cons id (cons u q))
   (cons id (cons u (append q (list user))))])))

;; Remove a user from a lane

(define (lane-pop ln)
  (match ln
    [(cons id (cons u (cons q qs)))
     (values q
            (cons id (cons u qs))))])))

; IO() fires an event

; Sets the user of a lane and fire a event

(define (lane-bussy-set! ln user)
  (let
    ([new-lane (lane-bussy-set ln user)])
    (begin
      (if (and
            (> (lane-length new-lane) 0)
            (not (lane-busy? new-lane))) ;; till is free
          (begin
            (sim-fire-event! 2(first (lane-queue new-lane)) (lane-id new-lane))) ;; CHECKOUT
            (void)))
      )
    new-lane)))

```

```

;; IO () fires an event

;; Append a user to a lane an dfire an event

(define (lane-append! In user)

(let

([new-lane (lane-append In user)])

(begin

(if (and

(= (lane-length new-lane) 1) ;; i'm the only

(not (lane-busy? new-lane))) ;; till is free

(sim-fire-event! 2 user (lane-id new-lane)) ;; CHECKOUT

(void))

)

new-lane))

```

```

;; IO() fires an event

;; Remove a user from a lane and fire an event

(define (lane-pop! In )

(let*-values

([(user new-lane) (lane-pop In )]

[ (new-la2) (lane-bussy-set new-lane user)]


)

(begin

(sim-fire-event! 3 user (lane-id new-la2)) ;; CHECKOUT

new-la2)))

```

```
)
```

```
)
```

2.3 F3... ISRAT RISMA

```
(module event racket
```

```
  (provide event
```

```
    event?
```

```
    event-time
```

```
    event-type
```

```
    event-user
```

```
    event-lane
```

```
    event-params
```

```
    ENTER_EVENT
```

```
    CHECKOUT_EVENT
```

```
    LEAVE_EVENT
```

```
    event->string
```

```
)
```

```
(define ENTER_EVENT 1)
```

```
(define CHECKOUT_EVENT 2)
```

```
(define LEAVE_EVENT 3)
```

```
(define event-dict  
  (list (cons ENTER_EVENT "ENTER_EVENT")  
        (cons CHECKOUT_EVENT "CHECKOUT_EVENT")  
        (cons LEAVE_EVENT "LEAVE_EVENT")))
```

;; constructor.

```
;; ( time . ( id . (user . id-lane)))  
(define (event time id params)  
  (cons time (cons id params)))  
)
```

;; P: True

```
(define (event? val)  
  
  (pair? (car val))  
  (pair? (cdr val))  
  (pair? (cdr (cdr val)))))
```

;; P: valid event

```
(define (event-time ev)  
  (car ev))
```

;; P: valid event

```
(define (event-type ev)  
  (car(cdr ev)))
```

;; P: event-params

```

(define (event-params ev)
  (cdr(cdr ev)))

(define (event-user ev)
  (car(cdr(cdr ev)))))

(define (event-lane ev)
  (cdr(cdr(cdr ev)))))

(require racket/dict)

(define (event->string ev)
  (format "~a"
         (dict-ref event-dict (event-type ev)))
  ))
```

2.4 F4..... ISRAT RISMA

```

(module event racket

(provide event
  event?
  event-time
  event-type
  event-user
  event-lane
  event-params
  ENTER_EVENT
  CHECKOUT_EVENT
  LEAVE_EVENT
  event->string
  )

(define ENTER_EVENT 1)
(define CHECKOUT_EVENT 2)
(define LEAVE_EVENT 3)

(define event-dict
  (list (cons ENTER_EVENT "ENTER_EVENT")
        (cons CHECKOUT_EVENT "CHECKOUT_EVENT")
        (cons LEAVE_EVENT "LEAVE_EVENT")))

;; constructor.
;; ( time . ( id . (user . id-lane)))
(define (event time id params)
  (cons time (cons id params)))
  )

;; P: True
(define (event? val)
  (raise 'event?))

;; P: valid event
(define (event-time ev)
  (car ev))

;; P: valid event
(define (event-type ev)
  (car(cdr ev)))

;; P: event-params
(define (event-params ev)
  (cdr(cdr ev)))

(define (event-user ev)
  (car(cdr(cdr ev)))))

(define (event-lane ev)
  (cdr(cdr(cdr ev))))
```

```
(require racket/dict)
(define (event->string ev)
  (format "~a"
         (dict-ref event-dict (event-type ev))
         )))
)
```

2.4 F5.... SANJIDA KHAN

```
;; done
```

```
(module sim-lane-list racket
```

```
(provide lane-list?
```

```
less-crowded
```

```
)
```

```
(require "sim-lane.rkt")
```

```
;; Check if a list consists fo lanes
```

```
(define (lane-list? lst)
```

```
;; Checks if the given value is a list consisting of lanes
```

```
(and (list? lst)
```

```
(for/list ([lane lst]
```

```
(and (pair? lane)
```

```
(lane? lane))))))
```

```
;; Find the least crowded lane in a list of lanes
```

```
;; This is a sort of "priority queue"
```

```

;; in O(nlog(n))

(define (less-crowded sim-lanes)
  ;; Finds and returns the least crowded lane in the given list of lanes
  (cond [(null? sim-lanes) '()]
        ;If the list is empty, return empty list
        [else
         (let ((sorted-lenes (sort sim-lanes
                                     (lambda (x y)
                                       (< (lane-length x) (lane-length y))))))
           (car sorted-lenes))))]) ; Return the first (least crowded) lane

```

3. 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.i

```

Welcome to DrRacket, version 8.11.1 [cs].
Language: Determine language from source; memory limit: 128 MB.
> (sim-secs2hour 3661)
'(1 1 1)
>

```

3.2 F2.i

F2.i. Sanjida khan

```

Welcome to DrRacket, version 8.11.1 [cs].
Language: Determine language from source; memory limit: 128 MB.
> (lane? (cons 0 (cons 3 (list 2))))
#t
>

```

F2.ii.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (lane-length (cons 0 (cons 3 (list 2 3))))  
2  
> |
```

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (lane-length (cons 0 (cons 3 (list 2 3))))  
2  
> |
```

F2.iii.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (lane-busy? (cons 0 (cons null (list 2 3))))  
#f  
> |
```

F2.iv.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (lane-id (cons 0 (cons 3 (list 2 3))))  
0  
> |
```

F2.v.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (lane-user (cons 0 (cons 3 (list 2 3))))  
3  
> |
```

F2.vi.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (lane-queue (cons 0 (cons 3 (list 2 3))))  
'(2 3)  
> |
```

F3.i. Israt Risma

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event 0 1 (cons 4 5))  
'(0 1 4 . 5)  
>
```

F3.ii.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event? (cons 1 (cons 3 (cons 4 6))))  
#t  
> |
```

F3.iii.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event-time (cons 1 (cons 3 (cons 4 6))))  
1  
> |
```

F3.iv.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event-type (cons 1 (cons 3 (cons 4 6))))  
3  
> |
```

F3.v.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event-params (cons 1 (cons 3 (cons 4 6))))  
'(4 . 6)  
> |
```

F3.vi.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event-user (cons 1 (cons 3 (cons 4 6))))  
4  
> |
```

F3.vii.

```
Welcome to DrRacket, version 8.11.1 [cs].  
Language: Determine language from source; memory limit: 128 MB.  
> (event-lane (cons 1 (cons 3 (cons 4 6))))  
6  
> |
```

F4. Israt Risma

Welcome to DrRacket, version 8.11.1 [cs].
Language: Determine language from source; memory limit: 128 MB.

```
> (sim-add-event
  (list
    (event 0 1 (cons 2 0))
    (event 0 1 (cons 3 0)))
  (event 0 2 (cons 1 0)))
'((0 2 1 . 0) (0 1 2 . 0) (0 1 3 . 0))
> (sim-add-event
  (list
    (event 0 1 (cons 2 0))
    (event 10 1 (cons 3 0)))
  (event 5 3 (cons 1 0)))
'((0 1 2 . 0) (5 3 1 . 0) (10 1 3 . 0))
>
```

F5. Sanjida khan

(event 5 2 (cons 1 0)))
21:39 (0 1 2 . 0) (5 2 1 . 0) (10 1 3 . 0)

3.4

F5.I

OUTPUTS AGAINST THE SPECIFICATION

```
> (lane-list?
  (list
    (cons 1 (cons 2 empty))
    (cons 2 (cons null empty))
    (cons 3 (cons 4 (list 4 5 6)))))
```

#t

F5.II

OUTPUTS AGAINST THE SPECIFICATION

```
> (less-crowded
  (list
    (cons 1 (cons 2 empty))
    (cons 2 (cons null empty))
    (cons 3 (cons 4 (list 4 5 6)))))

'(2 ())
```

OUR OWN EXAMPLE

```
> (less-crowded
  (list
    (cons 1 (cons null empty))
    (cons 2 (cons 4 empty))
    (cons 3 (cons 3 (list 8 1 3)))))

'(1 ())
```

COMP1811 Scheme CW Error! Reference source not found.Error! Reference source not found.

4. TESTING

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

F1(sim-utils):

Function	Input	Clarification	Output	Analyzing
Sim-secs2hours	(sim-secs2hour 3661)	Check to ensure that the code is converting the input in hours, minutes and seconds	'(1 1 1)	Expected
Sim-secs2hours	(sim-secs2hour 7783)	Check to ensure that the code is converting the input in hours, minutes and seconds	'(2 9 43)	Expected
Sim-secs2hours	(sim-secs2hour 2593)	Check to ensure that the code is converting the input in hours, minutes and seconds	'(0 43 13)	Expected
Sim-secs2hours	(sim-secs2hour 8735)	Check to ensure that the code is converting the input in hours, minutes and seconds	'(2 25 35)	Expected

F2.sim-lane:

- i. Lane?:

Function	Input	Clarification	Output	Analyzing
lane?	(lane? (cons 0(cons 3(list 2))))	Checks that it returns #t if the value is in the same format	#t	Expected
lane?	(lane? (cons 0(cons 3(list 5))))	Checks that it returns #t if the value is in the same format	#t	Expected
lane?	(lane? (cons 1(cons 4(list 8))))	Checks that it returns #t if the value is in the same format	#t	Expected

ii.lane-length:

Function	Input	Clarification	Output	Analyzing
Lane-length	(lane-length (cons 0(cons 3(list 2 3))))	Checks that if it returns the amount of the customers waiting in the queue	2	Expected
Lane-length	(lane? (cons 0(cons 3(list 5 6))))	Checks that if it returns the amount of the customers waiting in the queue	5	Expected
Lane-length	(lane? (cons 1(cons 4(list 8 9))))	Checks that if it returns the amount of the customers waiting in the queue	8	Expected

iii. Lane-busy?

Function	Input	Clarification	Output	Analyzing
Lane-busy	(lane-busy?(cons 0(cons null(list 2 3))))	Checks that if the till busy or not	#f	Expected

Lane-busy	(lane-busy?(cons 0(cons null(list 3 4))))	Checks that if the till busy or not	#f	Expected
Lane-busy	(lane-busy?(cons 0(cons null(list 3 6))))	Checks that if the till busy or not	#f	Expected

Iv.lane-id:

Function	Input	Clarification	Output	Analyzing
Lane-id	(lane-id (cons 0(cons 3(list 2 3))))	Checks if it can return its ID or not.	0	Expected
Lane-id	(lane-id (cons 2(cons 5(list 2 3))))	Checks if it can return its ID or not.	2	Expected
Lane-id	(lane-id (cons 0(cons 7(list 2 8))))	Checks if it can return its ID or not. t	0	Expected

F2.v. lane-user:

Function	Input	Clarification	Output	Analyzing
Lane-user	(lane-user(cons 0(cons 3(list 2 3))))	Checks that if it returns the id of the user.	3	Expected
Lane-user	(lane-user(cons 0(cons 6(list 1 3))))	Checks that if it returns the id of the user.	6	Expected
Lane-user	(lane-user(cons 0(cons 8(list 2 3))))	Checks that if it returns the id of the user.	8	Expected

F2.vi. Lane-queue:

Function	Input	Clarification	Output	Analyzing

Lane-queue	(lane-queue(cons 0(cons 3(list 2 3))))	Checks that if it returns the id of the user.	'(2 3)	Expected
Lane-queue	(lane-queue(cons 0(cons 3(list 6 9))))	Checks that if it returns the id of the user.	'(6 9)	Expected
Lane-queue	(lane-queue(cons 0(cons 3(list 5 3))))	Checks that if it returns the id of the user.	'(5 3)	Expected

F3.event . Israt Risma

F3(i):event

Function	Input	Clarification	Output	Analyzing
event	(event 0 1(cons 4 5))	Check to see whether it returns both customer and lane id.	'(0 1 4 . 5)	Expected
event	(event 8 0(cons 4 5))	Check to see whether it returns both customer and lane id.	'(8 0 4 . 5)	Expected
event	(event 4 5(cons 9 0))	Check to see whether it returns both customer and lane id.	'(4 5 9 . 0)	Expected

F3 ii.event?

Function	Input	Clarification	Output	Analyzing
event?	(event? (cons 1 (cons 3 (cons 4 6))))	Check to see whether it calculates correctly and provides a reasonable outcome.	#t	Expected
event?	(event? (cons 3 (cons 1 (cons 8 9))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	#t	Expected
event?	(event? (cons 4 (cons 5 (cons 6 0))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	#t	Expected

F3iii.event-time

event-time	(event-time (cons 1 (cons 3 (cons 4 6))))	Check to see whether it calculates correctly and provides a reasonable outcome.	1	Expected		
event-time	(event-time (cons 3 (cons 4 (cons 6 8))))	to ensure that, in the instance of this code, it does not interfere with a rather	3	Expected		

		frequent occurrence				
event-time	(event-time(cons 9(cons 5(cons 4 7))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	9	Expected		

F3.iv.event-type

Function Name	Input	Clarification	Output	Evaluation
event-type	(event-type(cons 1(cons 3(cons 4 6))))	Check to see whether it calculates correctly and provides a reasonable outcome.	3	Expected
event-type	(event-type(cons 5(cons 6(cons 1 3))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	6	Expected
event-type	(event-type(cons 4(cons 7(cons 3 5))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	7	Expected

F3.v.event-params

Function Name	Input	Clarification	Output	Evaluation
---------------	-------	---------------	--------	------------

event-params	(event-type(cons 2(cons 4(cons 6 8))))	Check to see whether it calculates correctly and provides a reasonable outcome.	'(6 . 8)	Expected
event-params	> (event-params(cons 3(cons 6(cons 9 12))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(9.12)	Expected
event-params	(event-params(cons 6(cons 8(cons 11 13))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(11. 13)	Expected

F3.vi.event-user

Function Name	Input	Clarification	Output	Evaluation
event-user	(event-user(cons 4(cons 3(cons 7 5))))	Check to see whether it calculates correctly and provides a reasonable outcome.	'(7)	Expected
event-user	> (event-user(cons 3(cons 6(cons 2 5))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(2)	Expected

event-user	(event-user(cons 5(cons 6(cons 7 3))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(7)	Expected
------------	--	---	------	----------

F3.vii.event-lane

Function Name	Input	Clarification	Output	Evaluation
event-lane	(event-lane(cons 4(cons 3(cons 7 5))))	Check to see whether it calculates correctly and provides a reasonable outcome.	'(5)	Expected
event-lane	(event-lane(cons 3(cons 5(cons 9 11))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(11)	Expected
event-lane	(event-lane(cons 2(cons 3(cons 9 12))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(12)	Expected

F4i. Israt Risma

Function	Input	Clarification	Output	analyzing
Sim-add-event	sim-add-event (list (event 0 2(cons 2 0)) (event 0 2(cons 3 0))) ;current list	Check to see whether it calculates correctly and provides a reasonable outcome.	((0 2 2 . 0) (0 2 3 . 0)) '(0 3 1 . 0)	Expected

	(event 0 3(cons 1 0)) ;new event			
Sim-add-event	sim-add-event (list (event 0 3(cons 3 0)) (event 0 3(cons 4 0))) ;current list (event 0 4(cons 2 0)) ;new event	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'((0 3 3 . 0) (0 3 4 . 0)) '(0 4 2 . 0)	Expected
Sim-add-event	sim-add-event (list (event 0 4(cons 4 0)) (event 0 4(cons 5 0))) ;current list (event 0 5(cons 3 0)) ;new event	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'((0 4 4 . 0) (0 4 5 . 0)) '(0 5 3 . 0)	Expected
Function	Input	Clarification	Output	Analyzing
Sim-add-event	sim-add-event (list (event 0 2(cons 2 0)) (event 0 2(cons 3 0))) ;current list (event 0 3(cons 1 0)) ;new event	Check to see whether it calculates correctly and provides a reasonable outcome.	((0 2 2 . 0) (0 2 3 . 0)) '(0 3 1 . 0)	Expected
Sim-add-event	sim-add-event (list (event 0 3(cons 3 0)) (event 0 3(cons 4 0))) ;current list (event 0 4(cons 2 0)) ;new event	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'((0 3 3 . 0) (0 3 4 . 0)) '(0 4 2 . 0)	Expected
Sim-add-event	sim-add-event (list (event 0 4(cons 4 0)) (event 0 4(cons 5 0))) ;current list (event 0 5(cons 3 0)) ;new event	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'((0 4 4 . 0) (0 4 5 . 0)) '(0 5 3 . 0)	Expected

F5i. Sanjida Khan

Function Name	Input	Clarification	Output	Evaluation
lane-list?	(lane-list? (list (cons 1(cons 2 empty)) (cons 2(cons null empty)) (cons 3(cons 4(list 4 5 6))))	Check to see whether it calculates correctly and provides a reasonable outcome.	#t	Expected
Lane-list?	(lane-list? (list (cons 2(cons 3 empty)) (cons 3(cons null empty)) (cons 3(cons 4(list 7 8 9))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	#t	Expected
Less-crowded	(less-crowded (list (cons 1(cons 2 empty)) (cons 2(cons null empty)) (cons 3 (cons 4(list 4 5 6))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(2 ())	Expected
Less-crowded	(less-crowded (list (cons 2(cons 3 empty)) (cons 3(cons null empty)) (cons 5 (cons 4(list 5 6 7))))	to ensure that, in the instance of this code, it does not interfere with a rather frequent occurrence	'(3 ())	Expected