Robert Werthman

CIS 579 Section 102

Professor Bruce Maxim

Table of Contents

[Lisp Project 1 1](#_Toc514224845)

[General approach to the problem 1](#_Toc514224846)

[Strengths of my approach to the problem 1](#_Toc514224847)

[Weaknesses of my approach to the problem 2](#_Toc514224848)

[Code (werthman\_assignment1.lisp) 2](#_Toc514224849)

[Test Code (werthman\_assignment1\_test.lisp) 5](#_Toc514224850)

[Test Output 7](#_Toc514224851)

# Lisp Project 1

## General approach to the problem

When I can I try use test driven development. My approach was to create tests for each of the functions based on the expected input and output from the assignment description. I would then run the tests, they would fail, then I would add code to the functions to get them to pass. You can’t always do this but for this assignment it worked.

I wanted to reuse as much of the code as possible. I used previously written functions in later functions. For example, in the MOVE function I used the WHERE function which in turn used the LOOK function. I did this to keep the code and functions shorter but also to test each function more thoroughly. To me testing the individual function is unit testing while testing multiple functions together is integration testing.

The information in the write up was very good. It gave useful hints and test cases. I had everything I needed to understand what each function should take as input and what each function should output. Lisp did work well for this assignment. It reminds me of Python because it is dynamically typed and interpreted. In Python you can use lists as well but I don’t think it has the built-in functions to handle nested lists like Lisp does.

To help understand the functions I would picture myself as the robot in the map trying to move from place to place with limited moves. When it came to the final function which was about finding a path for Robbie, I used the branch and bound algorithm given in class. I made some modifications to work with the ROOMS map. In this case Robbie had to go to a waypoint (the Library) so as part of the ending conditions I made sure that not only did Robbie’s path have to have the destination in it, it had to have the waypoint as well.

## Strengths of my approach to the problem

The functions were simple with few lines of code. This made it easier to debug, understand what was going, and write test cases for them. Lisp has good built-in functions to work on nested lists. The map appeared to be full of nested lists so Lisp worked well with it. I was able to create multiple test cases for each function to better verify that each function worked properly. This allowed me to find errors later on when working on other functions. The tests cases acted like regression tests and they showed me errors in the functions when I added new code. With the test cases I was able to ensure that the requirements from the assignment were met. The branch and bound algorithm did work successfully.

## Weaknesses of my approach to the problem

The code will only work with the given type of map. If you wanted to use another map with a different format you would have to rewrite the code to work with that map. The functions only take as input one type of Lisp variable, a literal atom. They do not take as input or produce output with different types. These things limit the codes extensibility to new problems with new inputs and outputs.

I find that the Lisp compiler is not very helpful. I lost a lot of time trying to understand what was wrong with my Lisp syntax. If there is a compiler error in C, it usually gives you the line number and reason why. The Lisp compiler does not give you a line number and it gives errors I had a hard time understanding. Perhaps using a different programming language would have allowed me to complete the assignment faster.

The robot has a very limited number of moves and locations. Not at all like a real person moving around a house. If the map was larger there would need to be modifications to the code to handle more locations and movements. Although the branch and bound algorithm was successful I think depth first search would have arrived at the solution sooner because the first path depth first search would have taken would have been the right path.

## Code (werthman\_assignment1.lisp)

; Here is the map Robbie will be using

(DEFVAR ROOMS)

(SETQ ROOMS

'((LIVING-ROOM (NORTH FRONT-STAIRS)

(SOUTH DINING-ROOM)

(EAST KITCHEN))

(UPSTAIRS-BEDROOM (WEST LIBRARY)

(SOUTH FRONT-STAIRS))

(DINING-ROOM (NORTH LIVING-ROOM)

(EAST PANTRY)

(WEST DOWNSTAIRS-BEDROOM))

(KITCHEN (WEST LIVING-ROOM)

(SOUTH PANTRY))

(PANTRY (NORTH KITCHEN)

(WEST DINING-ROOM))

(DOWNSTAIRS-BEDROOM (NORTH BACK-STAIRS)

(EAST DINING-ROOM))

(BACK-STAIRS (SOUTH DOWNSTAIRS-BEDROOM)

(NORTH LIBRARY))

(FRONT-STAIRS (NORTH UPSTAIRS-BEDROOM)

(SOUTH LIVING-ROOM))

(LIBRARY (EAST UPSTAIRS-BEDROOM)

(SOUTH BACK-STAIRS)))

)

;A function CHOICES which takes the name of the room and returns

;the table of permissible choices for Robbie's next destination.

;For example (CHOICES 'PANTRY) returns the list ((NORTH KITCHEN)

;(WEST DINING-ROOM)).

(defun CHOICES (x)

(

; look for a sublist in ROOMS containing x

let ((Y (assoc x ROOMS)))

; return the rest of the sublist containing x

; while not return x

(CDR Y)

)

)

;The function LOOK that takes two inputs, a direction and a room, and tells

; where he will end up if he moved in that direction from that room. For example if

;Robbie were to (LOOK 'NORTH 'PANTRY) he would get (KITCHEN), (LOOK 'WEST 'PANTRY)

; gives him (DINING-ROOM), (LOOK 'SOUTH 'PANTRY)gives NIL. (Hint: Use CHOICES somewhere.)

(defun LOOK (direction roomofhouse)

(

; Get the sublist of the room of the house robbie wants to go

; Get the room in the direction he wants to go

let\* ((x (CHOICES roomofhouse)) (y (assoc direction x)))

(CDR y)

)

)

; An expression which sets a global variable LOC to hold Robbie's current

; position in the PANTRY

(DEFVAR LOC)

(setq LOC 'PANTRY)

; A function HOW-MANY-CHOICES that tells how many choices Robbie has

; for where he may move given the current value of the variable LOC.

; If Robbie is in the PANTRY the function should return 2.

(defun HOW-MANY-CHOICES ()

(

let ((x (CHOICES LOC)))

(length x)

)

)

; A predicate function UPSTAIRSP that returns T if the input is an

; upstairs locations The UPSTAIRS-BEDROOM and the LIBRARY are the only rooms

; which qualify

(defun UPSTAIRSP (location)

; Check if the give location is one of the upstairs rooms

(cond ((equal location 'UPSTAIRS-BEDROOM) t)

((equal location 'LIBRARY) t)

(nil)

)

)

;A predicate function ONSTAIRSP which returns T if it’s input is either

;FRONT-STAIRS or BACK-STAIRS

(defun ONSTAIRSP (location)

; Check if the give location is on the stairs

(cond ((equal location 'BACK-STAIRS) t)

((equal location 'FRONT-STAIRS) t)

(nil)

)

)

;A function WHERE that requires no inputs and tells Robbie where he is.

;If he is in the LIBRARY where should return(ROBBIE IS UPSTAIRS IN THE LIBRARY),

; if he is in the KITCHEN it should return (ROBBIE IS DOWNSTAIRS IN THE KITCHEN),

;and if he is on the front stairs it should return (ROBBIE IS ON THE

; FRONT-STAIRS). (Hint: Use your predicate functions.)

(defun WHERE ()

; Check if Robbie is upstairs, downstairs, or on the stairs and return

; related string

(cond ((equal (ONSTAIRSP LOC) t) (append '(ROBBIE IS ON THE) (list LOC)))

((equal (UPSTAIRSP LOC) t) (append '(ROBBIE IS UPSTAIRS IN THE) (list LOC)))

((append '(ROBBIE IS DOWNSTAIRS IN THE) (list LOC)))

)

)

;A function MOVE that takes one input, a direction, and moves Robbie in that

;direction. MOVE should make use of your LOOK function. If he can't move in

;the requested direction, an appropriate message should be returned.

;For example, if Robbie is in the PANTRY (MOVE 'SOUTH) should return

;(OUCH! ROBBIE HIT A WALL), while (MOVE 'NORTH) should return (ROBBIE IS

;DOWNSTAIRS IN THE KITCHEN.

(defun MOVE (direction)

; If robbie can't move to a place then he hits wall

(cond ((equal (LOOK direction LOC) nil) '(OUCH! ROBBIE HIT A WALL))

(

; Otherwise, Robbie moves to the new location

(setq LOC (car (LOOK direction LOC)))

; Return the description of the new location

(WHERE)

)

)

)

;Adapt one of the search functions presented in class to help Robbie

;find a path from PANTRY to the KITCHEN thatpasses through the LIBRARY

; I'm going to adapt branch and bound to this function

(defun ROBBIESEARCH (source destination waypoint)

(branch source destination waypoint)

)

(defun branch (start finish waypoint) ;similar to best, but queue sorted

(branch1 (list (list start)) finish waypoint)) ;based on path length

(defun branch1 (queue finish waypoint) ;helper fcn for branch

(cond

((null queue) nil)

; We can only finish when our destination and waypoint are in the path

((and (equal finish (caar queue)) (member waypoint (car queue)))

(print queue)

(reverse (car queue)))

(t ;(print queue)

(branch1

(sort ;new predicate used

(append

(expand (car queue)) (cdr queue)

)

'shorterp

)

finish waypoint

)

)

)

)

(defun shorterp (p1 p2) ;check path lengths

(< (length p1) (length p2)))

(defun expand (path)

(remove-if #'

(lambda (path) (member (car path) (cdr path))) ;kill cycles

(mapcar #'(lambda (child) (cons child path))

; Get the children of each room from the ROOMS map

(apply 'append (mapcar #'(lambda (children) (car children) (cdr children)) (CHOICES (car path))))

)

)

)

## Test Code (werthman\_assignment1\_test.lisp)

(change-directory "/Users/bobby/Desktop/cisc579/assignment1")

(compile-file "werthman\_assignment1.lisp" :load t)

; Test cases for the function LOOK

; LOOK 'NORTH 'PANTRY) he would get (KITCHEN), (LOOK 'WEST 'PANTRY)

; gives him (DINING-ROOM), (LOOK 'SOUTH 'PANTRY)gives NIL

(print "TESTING FOR LOOK FUNCTION")

(print "Test case 1 for LOOK LOOK 'NORTH 'PANTRY")

(if (equal (LOOK 'NORTH 'PANTRY) '(KITCHEN))

(print (format nil "PASS: LOOK returned ~a." (LOOK 'NORTH 'PANTRY)))

(print (format nil "FAIL: LOOK returned ~a." (LOOK 'NORTH 'PANTRY)))

)

(print "")

(print "Test case 2 for LOOK LOOK 'WEST 'PANTRY")

(if (equal (LOOK 'WEST 'PANTRY) '(DINING-ROOM))

(print (format nil "PASS: LOOK returned ~a." (LOOK 'WEST 'PANTRY)))

(print (format nil "FAIL: LOOK returned ~a." (LOOK 'WEST 'PANTRY)))

)

(print "")

(print "Test case 3 for LOOK LOOK 'SOUTH 'PANTRY")

(if (equal (LOOK 'SOUTH 'PANTRY) nil)

(print (format nil "PASS: LOOK returned ~a." (LOOK 'SOUTH 'PANTRY)))

(print (format nil "FAIL: LOOK returned ~a." (LOOK 'SOUTH 'PANTRY)))

)

(print "")

; Test case for create a global variable LOC

; that stores Robbies current location

; and set it to pantry

(print "TESTING FOR SET GLOBAL VARIABLE LOC")

(print "Test case 1 for LOC set to Pantry")

(setq LOC 'PANTRY)

(if (equal LOC 'PANTRY)

(print (format nil "PASS: LOC set to ~a." LOC))

(print (format nil "FAIL: LOC set to ~a." LOC))

)

(print "")

; Test case for HOW-MANY-CHOICES

; If Robbie is in the PANTRY the function should return 2.

(print "TESTING FOR HOW-MANY-CHOICES")

(print "Test case 1 for HOW-MANY-CHOICES")

(if (equal (HOW-MANY-CHOICES) 2)

(print (format nil "PASS: HOW-MANY-CHOICES returns ~a." (HOW-MANY-CHOICES)))

(print (format nil "FAIL: HOW-MANY-CHOICES returns ~a." (HOW-MANY-CHOICES)))

)

(print "")

; Test case for UPSTAIRSP

; The UPSTAIRS-BEDROOM and the LIBRARY will return t

; All others will return nil

(print "TESTING FOR UPSTAIRSP")

(print "Test case 1 for UPSTAIRSP LIBRARY")

(if (equal (UPSTAIRSP 'LIBRARY) t)

(print (format nil "PASS: UPSTAIRSP returns ~a." (UPSTAIRSP 'LIBRARY)))

(print (format nil "FAIL: UPSTAIRSP returns ~a." (UPSTAIRSP 'LIBRARY)))

)

(print "")

(print "Test case 2 for UPSTAIRSP UPSTAIRS-BEDROOM")

(if (equal (UPSTAIRSP 'UPSTAIRS-BEDROOM) t)

(print (format nil "PASS: UPSTAIRSP returns ~a." (UPSTAIRSP 'UPSTAIRS-BEDROOM)))

(print (format nil "FAIL: UPSTAIRSP returns ~a." (UPSTAIRSP 'UPSTAIRS-BEDROOM)))

)

(print "")

(print "Test case 3 for UPSTAIRSP PANTRY")

(if (equal (UPSTAIRSP 'PANTRY) nil)

(print (format nil "PASS: UPSTAIRSP returns ~a." (UPSTAIRSP 'PANTRY)))

(print (format nil "FAIL: UPSTAIRSP returns ~a." (UPSTAIRSP 'PANTRY)))

)

(print "")

; Test case for ONSTAIRSP

; The FRONT-STAIRS and the BACK-STAIRS will return t

; All others will return nil

(print "TESTING FOR ONSTAIRSP")

(print "Test case 1 for ONSTAIRSP FRONT-STAIRS")

(if (equal (ONSTAIRSP 'FRONT-STAIRS) t)

(print (format nil "PASS: ONSTAIRSP returns ~a." (ONSTAIRSP 'FRONT-STAIRS)))

(print (format nil "FAIL: ONSTAIRSP returns ~a." (ONSTAIRSP 'FRONT-STAIRS)))

)

(print "")

(print "Test case 2 for ONSTAIRSP BACK-STAIRS")

(if (equal (ONSTAIRSP 'BACK-STAIRS) t)

(print (format nil "PASS: ONSTAIRSP returns ~a." (ONSTAIRSP 'BACK-STAIRS)))

(print (format nil "FAIL: ONSTAIRSP returns ~a." (ONSTAIRSP 'BACK-STAIRS)))

)

(print "")

(print "Test case 3 for ONSTAIRSP PANTRY")

(if (equal (ONSTAIRSP 'PANTRY) nil)

(print (format nil "PASS: ONSTAIRSP returns ~a." (ONSTAIRSP 'PANTRY)))

(print (format nil "FAIL: ONSTAIRSP returns ~a." (ONSTAIRSP 'PANTRY)))

)

(print "")

; Test case for WHERE

; If he is in the LIBRARY where should return(ROBBIE IS UPSTAIRS IN THE LIBRARY),

; if he is in the KITCHEN it should return (ROBBIE IS DOWNSTAIRS IN THE KITCHEN),

; and if he is on the front stairs it should return (ROBBIE IS ON THE

; FRONT-STAIRS)

(print "TESTING FOR WHERE")

(print "Test case 1 for WHERE LIBRARY")

(setq LOC 'LIBRARY)

(if (equal (WHERE) '(ROBBIE IS UPSTAIRS IN THE LIBRARY))

(print (format nil "PASS: WHERE returns ~a." (WHERE)))

(print (format nil "FAIL: WHERE returns ~a." (WHERE)))

)

(print "")

(print "Test case 2 for WHERE KITCHEN")

(setq LOC 'KITCHEN)

(if (equal (WHERE) '(ROBBIE IS DOWNSTAIRS IN THE KITCHEN))

(print (format nil "PASS: WHERE returns ~a." (WHERE)))

(print (format nil "FAIL: WHERE returns ~a." (WHERE)))

)

(print "")

(print "Test case 3 for WHERE FRONT-STAIRS")

(setq LOC 'FRONT-STAIRS)

(if (equal (WHERE) '(ROBBIE IS ON THE FRONT-STAIRS))

(print (format nil "PASS: WHERE returns ~a." (WHERE)))

(print (format nil "FAIL: WHERE returns ~a." (WHERE)))

)

(setq LOC 'PANTRY)

(print "")

; Test case for MOVE

;For example, if Robbie is in the PANTRY (MOVE 'SOUTH) should return

;(OUCH! ROBBIE HIT A WALL), while (MOVE 'NORTH) should return (ROBBIE IS

;DOWNSTAIRS IN THE KITCHEN.

(print "TESTING FOR MOVE")

(print "Test case 1 for MOVE SOUTH")

(setq LOC 'PANTRY)

(let ((x (MOVE 'SOUTH)))

(if (equal x '(OUCH! ROBBIE HIT A WALL))

(print (format nil "PASS: MOVE returns ~a." (MOVE 'SOUTH)))

(print (format nil "FAIL: MOVE returns ~a." (MOVE 'SOUTH)))

)

)

(print "")

(print "Test case 2 for MOVE NORTH")

(setq LOC 'PANTRY)

(let ((x (MOVE 'NORTH)))

(if (equal x '(ROBBIE IS DOWNSTAIRS IN THE KITCHEN))

(print (format nil "PASS: MOVE returns ~a." x))

(print (format nil "FAIL: MOVE returns ~a." x))

)

)

(print "")

; Test cases for SEARCH

; Help Robbie find a path from PANTRY to the KITCHEN thatpasses

; through the LIBRARY

(print "TESTING FOR SEARCH")

(print "Test case 1 for SEARCH")

(let ((x (ROBBIESEARCH 'PANTRY 'KITCHEN 'LIBRARY)))

(if (equal x '(PANTRY DINING-ROOM DOWNSTAIRS-BEDROOM BACK-STAIRS LIBRARY UPSTAIRS-BEDROOM FRONT-STAIRS LIVING-ROOM KITCHEN))

(print (format nil "PASS: SEARCH returns ~a." x))

(print (format nil "FAIL: SEARCH returns ~a." x))

)

)

(print "")

## Test Output

"TESTING FOR LOOK FUNCTION"

"Test case 1 for LOOK LOOK 'NORTH 'PANTRY"

"PASS: LOOK returned (KITCHEN)."

""

"Test case 2 for LOOK LOOK 'WEST 'PANTRY"

"PASS: LOOK returned (DINING-ROOM)."

""

"Test case 3 for LOOK LOOK 'SOUTH 'PANTRY"

"PASS: LOOK returned NIL."

""

"TESTING FOR SET GLOBAL VARIABLE LOC"

"Test case 1 for LOC set to Pantry"

"PASS: LOC set to PANTRY."

""

"TESTING FOR HOW-MANY-CHOICES"

"Test case 1 for HOW-MANY-CHOICES"

"PASS: HOW-MANY-CHOICES returns 2."

""

"TESTING FOR UPSTAIRSP"

"Test case 1 for UPSTAIRSP LIBRARY"

"PASS: UPSTAIRSP returns T."

""

"Test case 2 for UPSTAIRSP UPSTAIRS-BEDROOM"

"PASS: UPSTAIRSP returns T."

""

"Test case 3 for UPSTAIRSP PANTRY"

"PASS: UPSTAIRSP returns NIL."

""

"TESTING FOR ONSTAIRSP"

"Test case 1 for ONSTAIRSP FRONT-STAIRS"

"PASS: ONSTAIRSP returns T."

""

"Test case 2 for ONSTAIRSP BACK-STAIRS"

"PASS: ONSTAIRSP returns T."

""

"Test case 3 for ONSTAIRSP PANTRY"

"PASS: ONSTAIRSP returns NIL."

""

"TESTING FOR WHERE"

"Test case 1 for WHERE LIBRARY"

"PASS: WHERE returns (ROBBIE IS UPSTAIRS IN THE LIBRARY)."

""

"Test case 2 for WHERE KITCHEN"

"PASS: WHERE returns (ROBBIE IS DOWNSTAIRS IN THE KITCHEN)."

""

"Test case 3 for WHERE FRONT-STAIRS"

"PASS: WHERE returns (ROBBIE IS ON THE FRONT-STAIRS)."

""

"TESTING FOR MOVE"

"Test case 1 for MOVE SOUTH"

"PASS: MOVE returns (OUCH! ROBBIE HIT A WALL)."

""

"Test case 2 for MOVE NORTH"

"PASS: MOVE returns (ROBBIE IS DOWNSTAIRS IN THE KITCHEN)."

""

"TESTING FOR SEARCH"

"Test case 1 for SEARCH"

((KITCHEN LIVING-ROOM FRONT-STAIRS UPSTAIRS-BEDROOM LIBRARY BACK-STAIRS DOWNSTAIRS-BEDROOM DINING-ROOM PANTRY) (FRONT-STAIRS UPSTAIRS-BEDROOM LIBRARY BACK-STAIRS DOWNSTAIRS-BEDROOM DINING-ROOM LIVING-ROOM KITCHEN PANTRY) (DINING-ROOM DOWNSTAIRS-BEDROOM BACK-STAIRS LIBRARY UPSTAIRS-BEDROOM FRONT-STAIRS LIVING-ROOM KITCHEN PANTRY))

"PASS: SEARCH returns (PANTRY DINING-ROOM DOWNSTAIRS-BEDROOM BACK-STAIRS LIBRARY UPSTAIRS-BEDROOM FRONT-STAIRS LIVING-ROOM KITCHEN)."

""