Online Submission Assignment Cover Page

Name: Richard Break
ID#:
Date: 07/02/2012
Course #: CIS*3190
Course Name: Software for Legacy Systems
Instructor: Dr. Michael Wirth
Assignment #: 2
Assignment Name: Ada Programming
# Of Pages (including this one): 4

	(For instructor's use) Grade:

Design Document: Traverse a Maze with a Depth-first Search

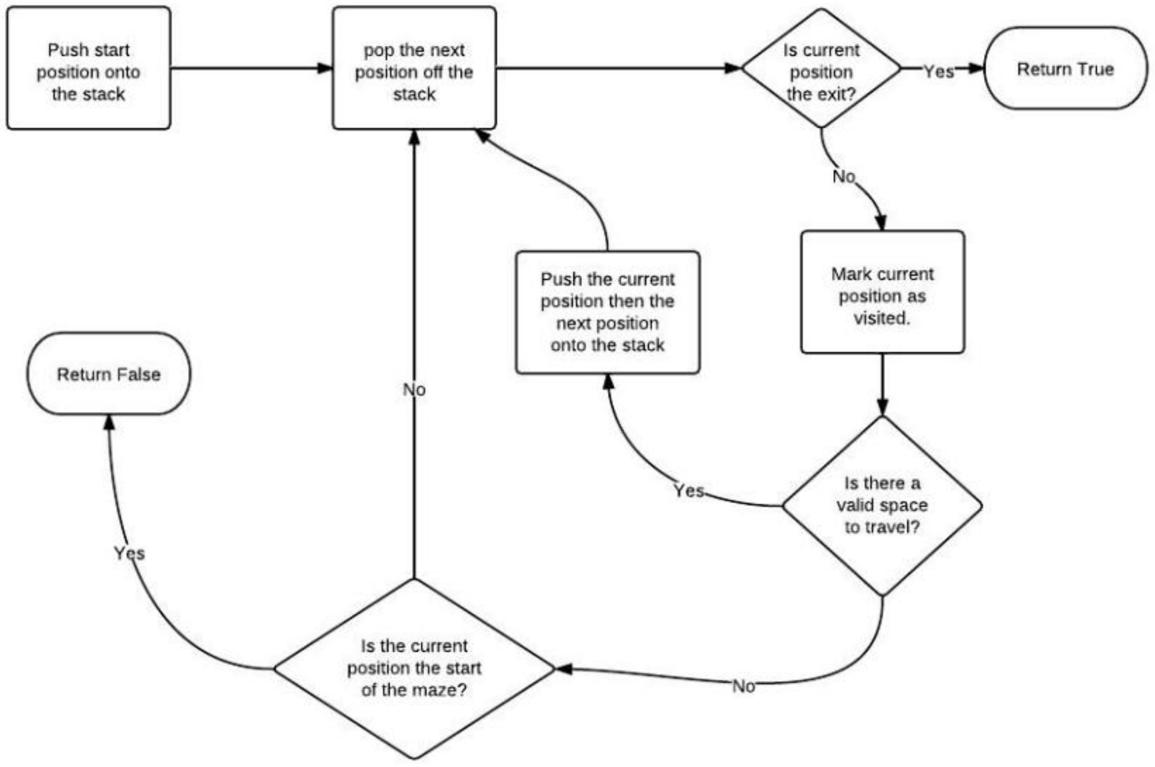
When planning the initial layout of the program, the idea was to separate the code into logical parts. The first section was the implementation of a stack to store previous locations in the maze. The stack (“stack.adb”) was put into its own package to simplify the assignment. The stack, which is the variable “stackPoints,” is an array of 2 points, x and y, corresponding to the location in the maze you are examining. This allowed for intuitive uses of pushing and popping the current position without any extra work. The push and pop functionality were implemented as a procedure to send and receive the x and y coordinates of the maze. Using a function would have resulted in 2 separate calls of the pop method to return the coordinates, or to return an array with the 2 points, which is complex and unnecessary in comparison to using a procedure.

The second section of code was the maze package (“maze.adb”). This package was intended to contain all the logic of reading in and solving the maze. The first task was to read in and store information about the maze. The user is prompt to give a file name of where the maze is located, and the file is read character by character. If the file name is not found, the program will catch the exception, display to the user the problem and will then exit. The maze is stored as a 2D character array, as parsing through the string data type caused complex statements to look at a specific character of the string.

The second task of the maze package was to traverse throughout the maze to find the exit. The findExit function contains majority of the logic, using a depth-first search and using the stack package. It first pushes the starting location onto the stack. From there, these steps are taken:

1) It pops off the current location off the stack and marks the position as a wall (which is equivalent to marking the position as previously traveled except using fewer elements).
2) It then checks if the current location is the exit. If so, the function ends, otherwise it continues.

3) Checks north, east, west and south, in that order, if it is a legal move to travel there. If there is an open space to travel in any of these 4 directions, it will push the current location and that location onto the stack and returns to step 1.
4) If an opening is not found, then it is a dead end and the program will not push any points into the stack. If the stack is empty, this means that there is no solution and the program ends, otherwise it returns to step 1.

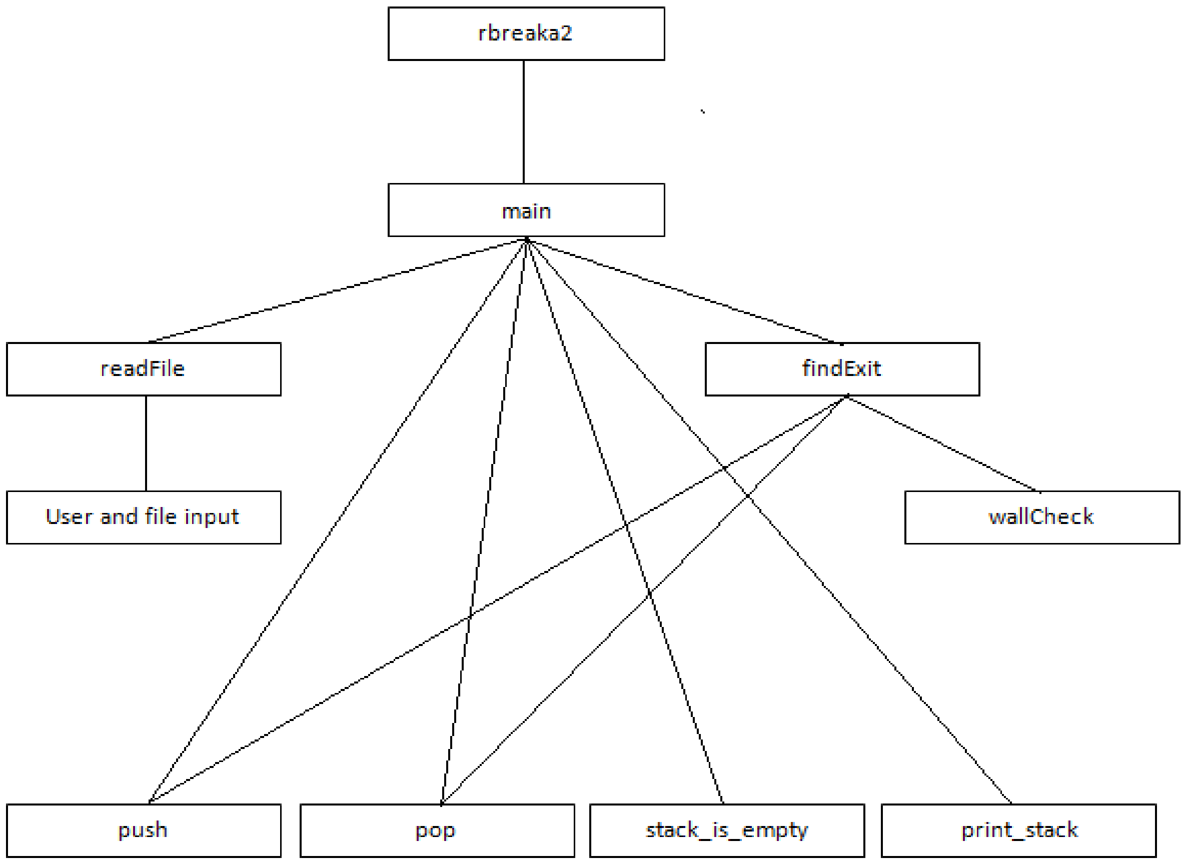

Push start position onto the stack
pop the next position off the stack
Is current position the exit?
Yes
Return True
No
Mark current position as visited.
Is there a valid space to travel?
Yes
Push the current position then the next position onto the stack
No
Is the current position the start of the maze?
Yes
Return False
No

The maze package consists of only functions. Each function only needed to return a single value which was mainly for debugging. Those return values then were used to determine the results of the maze. For example, the “checkWall” function was implemented as a way to determine if the file was being read correctly. It was then repurposed towards checking the current position in the maze. Another example is the “readFile” function’s return. It was used to determine if the file was incorrect or an exception was given. In general, functions were implemented over procedures because of the ability to call functions within other statements resulted in the code being more intuitive.

Finally, to be able to actually use the packages, the procedure, rbreaka2 (“rbreaka2.adb”), was implemented to call the “main” function of the maze package.

Designing the maze package to be separated from the stack package had its benefits. For example, it was harder to misuse the function and procedure calls of the stack package. It also allows the stack package to become reusable in other programs, and it reduced the coupling between the 2 packages. Overall, abstracting away the implementation of the stack package from the maze package allowed for a more comprehensible program.

Function and Procedure Diagram


rbreaka2
main
readFile
findExit
User and file input
wallCheck
push
pop
stack_is_empty
print_stack