

<http://www.cs.arizona.edu/classes/cs227/spring07/>

Program #8: Bemazed!

Due Date: April 2nd, 2007, at 10:00 p.m. MST

Overview: Using a stack to help solve (that is, to find a path through) a maze is a common programming assignment. Less common is an assignment that asks students to use a stack to construct the maze. We're going to be less common.

To construct a maze using a stack, we will select a location in a matrix, and 'dig' a random path until we reach a dead-end and can dig no further. As we dig, we'll push onto the stack the locations that comprise the path we've followed. When we reach a dead-end, we'll back up (by popping locations from the stack) until we reach a location from which we can dig in a new direction. We continue this digging and retreating until we're back at the starting location and there are no more directions to try. When that happens, the maze is almost done. All that's left to do is select the entrance and exit, and display the result for the user.

Here's a rather detailed outline of the algorithm:

```
create: a 2*r+1 by 2*c+1 array representing an r by c maze
randomly select a starting location within the r by c maze, and clear it
push that location on the stack
while the stack is not empty do:
  access top of stack to learn current location
  if there is at least one new direction in which to dig:
    randomly select one of the available directions
    dig from the current location to the next cell in that direction
    push the location of that next cell on the stack
  otherwise:
    discard the location at the top of the stack
  end if
end while
select and place maze entrance and exit
display maze
```

There are a variety of possible maze representations. To keep things simple, we want you to use a $2r + 1$ by $2c + 1$ array of characters to represent an r by c maze. Here's an example:

	0	1	2	3	
0					0 #####
1					1 # # #
2					2 # # # #
					3 # # # #
					4 # # ### #
					5 # #
					6 #####

This representation allows alternate array locations to represent maze locations, with the in-between locations available to represent the walls between maze locations. Think of the maze as being created from solid rock. Initially, the diggers start somewhere within the maze, pick a direction, and dig in that direction until they dig out the next location in that direction, along with the path between the locations. In the example shown, the random starting location was (0,3). The random path dug a spiral to (1,2), backed up to (0,1), and dig some more to (2,0) before backing up to the starting location.

Assignment: Write a complete, well-documented Java program that uses the algorithm described above to create mazes with a number of rows and a number of columns entered by the user (have your program prompt the user for these values). For this assignment, we want the maze creation algorithm to be somewhere in your `main()` method's class (`Prog08`). You are to write a separate maze representation class that provides all of the methods the creation algorithm needs, such as digging to the next cell in a given direction, placing the entrance, etc. Note that as far as your `Main()` is concerned, a maze is just r by c in size. From the representation class' perspective, the maze is $2r + 1$ by $2c + 1$ in size.

Digging the maze doesn't provide the maze entrance and exit; we have to do that separately, after the maze has been created. For this program, we want you to pick a random cell along the left side of the maze and remove the wall to its left to form the entrance. Do the same on the right side to form the exit.

To get some experience with linked lists, your stack of locations is to be represented using a singly-linked list. You MAY NOT use Java's `Stack` class, nor Java's `LinkedList` class. You must create your own stack class, and an accompanying node class.

Data: Your program is to prompt the user for the numbers of rows and columns in the maze. As the content of the maze will be determined randomly, there is no other data to be gathered from the user.

Output: We'll try something a little different on this assignment.

If you are content to earn no higher than 85% on this assignment, you may output just the $2r + 1$ by $2c + 1$ character dump of the internal maze representation. (That is, the 7x9 representation you see in the figure, and you don't even have to display the numbers.) This will be very easy to do, hence the reduced number of possible points.

For a shot at 100%, you will need to augment the basic program using the ACM graphics package to produce a 'thin-walled' graphic representation, like the 3x4 figure on the left of the above image. (It's fairly easy to produce a graphic version in which each character position is a square. That's why we want to see the fancier version (with thin walls) for full credit.) Of course, if you use the graphics package, you'll have to replace your `main()` with the `run()` method, as you did in Program #4.

Turn In: For this assignment, you may place all of your classes in the `Prog08.java` file, or you may divide them among several files; it's your choice. However you arrange the code, be sure to electronically submit all of your files using the `turnin` program and the usual

`[section leader first name]_S[your section #]_P[program #]_[your last name]_[your first name]` naming convention.

Hints, Reminders, and Other Requirements:

- Forgotten how to use the ACM graphics package? There's no shame; it's been a while. Refer to the Program #4 handout, as well as the documentation and tutorial linked to the class web page below the link to that assignment.
- We encourage you to get the maze algorithm working before you worry about the graphical representation of the maze. The text version of the maze is easy to display, and you can use it to help debug your algorithm before worrying about adding the graphics.
- We anticipate that the 'external' vs. 'internal' representation issues will be a significant source of confusion for this assignment. Remember that your application (which has the maze creation logic, including the stack) sees the maze as having r rows and c columns, while the class maintaining the maze believes that there are $2r + 1$ rows and $2c + 1$ columns. Make sure you are mentally in the appropriate frame of mind when working in a particular area of your code.
- Helpful hint: Due to the coordinate system used by the ACM graphics package, to avoid producing a 'mirror-image' graphical maze, you'll need to think (column,row) instead of (row,column) when drawing the lines that form the maze.