Maze Solver

DATA TYPES

Path (user-defined struct)

- string maze – hold layout of maze
- int width – hold integer width of each dimension of the maze (square)
- int locations[500] – hold ordered list of visited locations, as indexes in string maze (begin with start)
- int length – hold number of path locations (length of path, number of array indexes used)

Stack or Queue of Paths (provided)

FUNCTIONS

void start(Path& P, string filename)

- accept new Path and filename as arguments
- read maze from file into maze field in Path
- find starting point in maze, intialize Path to have that one location and length of 1

bool isFinished(Path P)

- accept Path as argument
- indicate (T/F) if Path has reached finish point in maze

ostream& operator<<(ostream& out, Path P)

- overload output operator to display Path
- iterate through Path locations, display each as directional step from previous location (so initial location – start – does not show in output)
- return ostream reference

void copyPath(Path a, Path& b)

- accept two Paths as arguments, copies contents of *a* into *b* so the two Paths are identical

bool isGoodStep(Path P, char direction)

- accept Path and direction (assume N/S/E/W) as arguments
- indicate (T/F) if the direction is a "good step" for the Path (not a wall, not already visited)

void takeStep(Path&, char direction)

- accept Path and direction (assume N/S/E/W) as arguments
- assume that the step is good (this function not responsible for checking isGoodStep)
- add new location to end of Path, determined by given direction from current last location

main ( )

- initialize stack (or queue) of paths
- read filename as string from standard input
- pass a Path variable and the filename to **start**, which will "initialize" the Path at start location
- while the stack (or queue) is not empty
    - pop one Path
    - check **isFinished** for that Path
        - if true, break out of the loop and output the path (using **<<**)
    - check **isGoodStep** for each of the four directions
        - if true, copy the Path, then take the step in the copy and push that onto stack
- if stack (or queue) is empty, print "No path." after loop ends

TIMELINE (in minutes)

```
 0  Stack       (use existing implementation)
 0  Path        (use definition as designed above)
 5  start
 5  isFinished
 5  operator<<
 5  copyPath
 5  isGoodStep
 5  takeStep
15  main
15  testing / debugging
---------------
60 minutes (1.0 hours) total, estimated
```