Computation I 5EIA0
Homework 6: Navigation (v2.4, October 9, 2022)
Deadline Tuesday 18 October 13:30

function	1	2	3	4	5	6	7	8	9	10	11	12	% per fn	cumulative %
quit	1	1	1	1	1	1	1	1	1	1	1	1	8%	8%
input grid		1	1	1	1	1	1	1	1	1	1	1	8%	17%
print grid			1	1	1	1	1	1	1	1	1	1	17%	33%
start position					1	1	1	1	1	1	1	1	8%	42%
find path						1	1	1	1			1	25%	67%
reset path									1			1	8%	75%
find longest path										1	1	1	25%	100%

Figure 1: Test cases and points per task.

Task 1. Write a C program that asks the user to select the command that needs to be performed. The commands that need to be supported are listed in the following table:

command	operation
q	quit program
i	input grid
p	print grid
s	start of path
f	find a path
r	reset path
l	find the longest path

In this task you only need to implement the quit command. In later tasks you will implement the remainder. Hint: make a loop in which you ask for a single character command, like this:

```
char cmd;
do {
  printf ("Command? ");
  scanf(" %c",&cmd); // notice the space before the %
  ...
} while (cmd != 'q');
```

If an invalid character is given then print the error message `Unknown command 'Z'` (with Z replaced by the unknown command):

```
Command? X
Unknown command 'X'
Command? q
Bye!
```

Your program must produce the *exact* output, including all spaces, capitalisation, quotes, etc.

In this homework you will develop a navigation system that can be used by a car driver to find a route from a starting point to his/her destination while navigating a rectangular (Manhattan-like) grid filled with blockages. All roads can be driven in both directions when they are not blocked. Positions are identified by integer (x,y) coordinates in zero-based coordinates (i.e., `0...maxX-1`, where `maxX` is the size of the grid in the x-dimension, similarly `maxY`). The `(0,0)` position is in the upper-left corner. At any given moment, the car can only move one step in one of four directions provided that the position is without obstacles and lies within the grid. The car can thus move in the following directions:

```
Go North: (x,y) -> (x,y-1)
Go East:  (x,y) -> (x+1,y)
Go South: (x,y) -> (x,y+1)
Go West:  (x,y) -> (x-1,y)
```

The navigation system should search for a path from the starting position to the destination position (a so called solution path) until it finds one or until it exhausts all possibilities. In former case it should mark the path on the map.

The map is represented in the program as a grid. Below you see an example 6x6 input grid, followed by one with a partial path, and one with a solution path:

```
S#####      S#####      S#####
.....#      ++++.#      ++...#
#.####      #.####      #+####
#.####      #.####      #+####
...#.D      ...#.D      .++#+D
##...#      ##...#      ##+++#
```

The starting position of the car is indicated with the letter 'S', the destination is indicated with the letter 'D', positions where the car is allowed to drive are indicated with a '.', and obstacles (blocked positions) are marked with a '#'. A position on the path in the grid can be marked by the '+' symbol. A path refers to either a partial path, marked while the system is still searching (i.e., one that may or may not lead to a solution), or a solution path which leads from start to destination.

This is a recursive problem, like the merge sort: when the system is exploring a position it is each time confronted with the same problem: "find a path to the destination D if the position itself is not D". It may of course also conclude that the destination is not reachable from that position. Assume that there is a function `findPath(grid, x, y, l)` that finds a path from some point `(x,y)` in a `grid` to a destination. `l` is the current length of the path. Also suppose that the system reached from the start to position `x=1, y=2` in the grid (by some method). The grid (including the partial path) then looks as follows:

```
S#####
++...#
#+####
#.####
...#.D
##...#
```

So the question is then whether there is a path from `x=1, y=2` to the destination. If there is then there is a path from the start to the destination (since a path from the start to `x=1, y=2` is already found). To find a path from position `x=1, y=2` to the destination, it can just ask `findPath` to try to find a path from the North, East, South, and West of `x=1, y=2`:

```
findPath(x=1, y=1) North
findPath(x=2, y=2) East
findPath(x=1, y=3) South
findPath(x=0, y=2) West
```

Generalizing this, `findPath` can be called recursively to move from any location in the grid to adjacent locations. In this way, it moves through the grid. That is, until it has to stop. All the time invalid positions have to be accounted for. So, a call to go North of the current position, should be disregarded when that North position is illegal. In order words, base questions are "is the position within the grid (or outside its

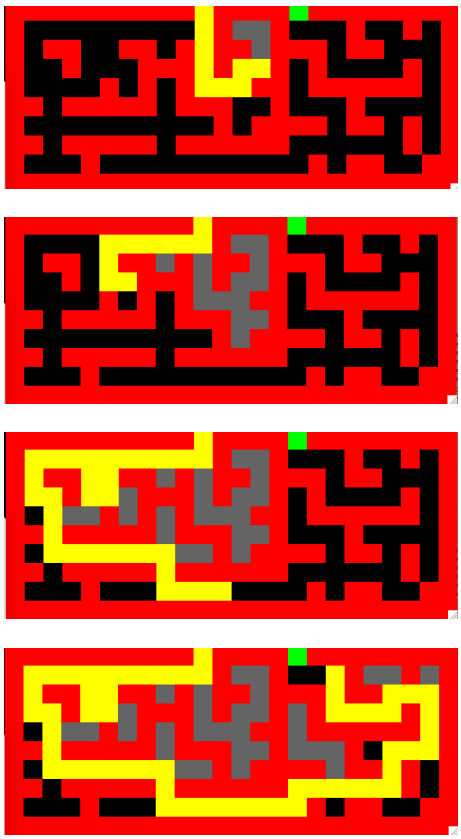
bounds)", and "is the position open (or is it blocked with an obstacle)"? Further the path that is still under consideration must be marked, and stay marked until it is clear that the destination cannot be reached from that position. In essence a complete algorithm that finds and marks a path to the destination (if any exists) and tells us whether a path was found or not (i.e., returns the length of the path or 0), may look like the following program:

```
findPath(x, y, length)
{
  if (x,y outside grid)       return 0
  if (x,y not open)           return 0
  if (x,y is the destination) return length

  mark x,y as part of solution path (if x,y isn't the start)

  // try to find a path in North, East, South, and West directions
  // note the clever use of = instead of ==
  // the extra brackets are to avoid a warning
  if ((nlength = findPath(North of x,y,length+1))) return nlength;
  if ((elength = findPath(East  of x,y,length+1))) return elength;
  if ((slength = findPath(South of x,y,length+1))) return slength;
  if ((wlength = findPath(West  of x,y,length+1))) return wlength;

  // if we get here we didn't find a solution in N,E,S, or W directions
  mark x,y as seen (if x,y isn't the start)
  return 0;
}
```

It is important that the algorithm stops when a path to the destination has been found, i.e., if going North of x,y finds a path (i.e., returns the length of the path that it has found), then from the current position (i.e., current call of `findPath`) there is no need to check East, South or West. Instead, `findPath` just needs to return the length of the current path to the previous call. Path marking can be done with the '+' symbol and marking as seen but not part of the solution path with the '~' symbol.

When no path to the destination can be found from the current position, i.e. findPath fails in all four directions, then findPath marks the current position as currently not being on the solution path, and continues searching in the remaining directions from the previous position. For example, if we're at `x=2, y=3`, we first call findPath to North (`x=2, y=2`). At `x=2, y=2` try all four directions: North `x=2, y=1`, East `x=3, y=2`, South `x=2, y=3`, West `x=1, y=2`. If all fail, we backtrack to `x=2, y=3` and try East. Another example is given on the next page.

The search is illustrated by the following figures. The destination is shown with a green pixel. The gray pixels are pixels that are SEEN, i.e. where the car passed during its search but are not part of the current path. They give a nice impression of how much of the grid has been explored. The first figure shows that the search went into a dead end and is now backtracking (since there are three gray pixels). The second and third figures show that the search has done some backtracking (there are several gray pixels) and is currently exploring a new path (since there are no gray pixels at the head of the current yellow path).



The algorithms that you have to implement are also illustrated in this screen recording:
`www.es.ele.tue.nl/~kgoossens//teaching/5eia0/2022/06-navigation.mp4`
If we would reverse the order of the following statements, what would happen to the solution path and the explored positions? Will we find the same path? If there are multiple paths, will we find the same one? Will it take longer or shorter? If these questions are too hard now, then look at them again later when you have the algorithms implemented.

```
if ((nlength = findPath(North of x,y,length+1))) return nlength;
if ((elength = findPath(East  of x,y,length+1))) return elength;
if ((slength = findPath(South of x,y,length+1))) return slength;
if ((wlength = findPath(West  of x,y,length+1))) return wlength;
```


Task 2. The navigation grid will be dynamically sized, depending on user input. We will use an array of characters that represents a two-dimensional-grid, together with the dimensions of the grid. To treat these three as a single entity, e.g. to pass fewer arguments to functions, declare a new type at the start of your program:

```
typedef struct {
  char *grid;
  int maxX;
  int maxY;
} navigation_t;
```

Inside your main function declare a variable called `nav` of type `navigation_t` and initialise the fields to zero. Create a function `void newGrid(navigation_t *nav)` that initialises a grid by asking the user for the size of the grid and malloc'ing the required space. Because this can be done multiple times during a program run you should first check if the input (nav) doesn't already contain a valid grid. If it does, then you need to free the `nav->grid` array and set the `maxX` and `maxY` fields to zero.

After that ask the user to input the number of rows and columns of the grid. These values should be assigned to the fields `maxY` and `maxX` of the nav structure, respectively. When `maxY` and `maxX` are less than two then you should print the error message: `The number of rows and columns must be at least two` and exit the function. Otherwise, malloc space for `nav->grid`.

```
Number of rows? 1
Number of columns? 4
The number of rows and columns must be at least two
```

Hint: You may get an error like this:

```
Number of rows? 2
Number of columns? 4
Input row 0: ....
Input row 1: ....
Command? q
Bye!

=================================================================
==87645==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 35 byte(s) in 1 object(s) allocated from:
    #0 0x7fef9cc38808 in __interceptor_malloc ../../../../src/libsanitizer/asan/asan
    #1 0x5604bf6da559 in newGrid /home/p11645/tets.c:44
    #2 0x5604bf6db8c0 in main /home/p11645/tets.c:189
    #3 0x7fef9c8090b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x240b2)

SUMMARY: AddressSanitizer: 35 byte(s) leaked in 1 allocation(s).
```

Remember that everything that has been malloc'd must be freed before exiting the program!

Task 3. Write a function `void inputGrid(navigation_t nav)` that asks the user to input `maxY` lines of `maxX` characters each. (You should skip blank spaces such as newlines.) These strings should be stored in the `grid` field of `nav`. The strings should be composed of the following characters:

- 'S' - starting position
- 'D' - destination
- '+' - part of the current path of the car
- '.' - location where car is allowed to drive
- '#' - location where car is not allowed to drive (obstacle)

Additionally, `'~'` is a location where car has driven in past searches; it is not a valid user input because it is generated by the search algorithm. I defined them like this:

```
#define START           'S'
#define DESTINATION     'D'
#define PATH            '+'
#define WALL            '#'
#define UNSEEN          '.'
#define SEEN            '~'
```

If the grid contains more than one starting point then the warning message `Extra starting point` should be given. All starting points except for the first one in the input are replaced by walls. The grid is allowed to have multiple destinations.
Add the command 'i' to your main function loop that first calls newGrid and then inputGrid. The output of your program should look as follows:

```
Command? i
Number of rows? 4
Number of columns? 4
Input row 0: ...D
Input row 1: .SS.
Extra starting point
Input row 2: .###
Input row 3: ...D
Command? p
...D
.S#.
.###
...D
Command? s
The start is at x=1 and y=1
Command? i
Number of rows? 1
Number of columns? 4
The number of rows and columns must at least two
Command? q
Bye!
```

Hint: If newGrid fails then you should not call printGrid. You'll get a Segmentation fault otherwise. (Why?)

Hint: The grid field of the navigation structure is a one-dimensional array of characters but we would like to use it as a two-dimensional grid. The easiest way to model `nav.grid[y][x]` (which doesn't compile) is to use `nav.grid[y*nav.maxX+x]` instead.

Task 4. Does your program contain a memory leak, i.e. is all space that is malloc'd freed before the end of the program? A leak can occur at two places: when calling newGrid more than once, and when quitting the program after newGrid has been called at least once. Fix your memory leaks.

```
Command? i
Number of rows? 6
Number of columns? 6
Input row 0: S#####
Input row 1: .....#
Input row 2: #.####
Input row 3: #.####
Input row 4: ...#.D
Input row 5: ##...#
Command? i              -- did you free the space of the 6x6 grid?
Number of rows? 3
Number of columns? 3
Input row 0: S##
Input row 1: ..#
Input row 2: #D#
Command? q              -- did you free the space of the 3x3 grid?
Bye!
```


Task 5. Add the command 'p' to the main function that calls the function `void printGrid(navigation_t nav)` which prints the grid to the output. Remember that the `maxX` and `maxY` fields of `nav` indicate the dimensions of the grid. (The command does nothing if the grid has not been initialised, i.e. nav.grid is NULL.)

The output of your program should look as follows:

```
Command? i
Number of rows? 6
Number of columns? 6
Input row 0: S#####
Input row 1: .....#
Input row 2: #.####
Input row 3: #.####
Input row 4: ...#.D
Input row 5: ##...#
Command? p
S#####
.....#
#.####
#.####
...#.D
##...#
Command? q
Bye!
```

Although it is not yet shown in the grid above, the PATH and SEEN characters must also be printed later, e.g.

```
Command? f
Found a path of length 59
##########S####D########
D++++++++++#~~#+++#~~#~#
#+###++##~#~##~###+#####+#
#++#++~###~#~~#~#++++#+#
#.+~~#~#~#~~~##~######+#
##+#####~###~~#~~~#.+++#
#.+++++++~~#~####~##+#.#
##.#####+######++++++#.#
#...#...++++++++#.##..##
######################
Command? q
Bye!
```

Task 6. Add the command 's' to the main function that calls the function `position_t findStart(navigation_t nav)` which finds the starting position in the grid. (The command does nothing if the grid has not been initialised.) This point is marked with the character 'S' in the array `grid`. Define the new type `position_t` at the start of your program.

```
typedef struct {
  int x;
  int y;
} position_t;
```

When the grid contains no starting position, the error message `Grid contains no starting point` should be output. When the grid contains no destination, the error message `Grid contains no destination` should be output. In both cases the function returns a position with the x and y fields set to -1. The output of your program should look as follows:

```
Command? i
Number of rows? 4
Number of columns? 6
Input row 0: ##..#.
Input row 1: ..S.#D
Input row 2: .###..
Input row 3: .....#
Command? s
The start is at x=2 and y=1
Command? i
Number of rows? 5
Number of columns? 5
Input row 0: DDDDD
Input row 1: D...D
Input row 2: D.S.D
Input row 3: D...D
Input row 4: DDDDD
Command? s
The start is at x=2 and y=2
Command? f
Found a path of length 2
DDDDD
D.+.D
D.S.D
D...D
DDDDD
Command? i
Number of rows? 3
Number of columns? 3
Input row 0: ...
Input row 1: .#.
Input row 2: ...
Command? s
Grid contains no starting point
Grid contains no destination
Command? q
Bye!
```


Task 7. Add the command 'f' to the main function that calls the function `int findPath(navigation_t nav, int x, int y, int length)`. (The command does nothing if the grid has not been initialised.) The function recursively calls itself and which tries to find a path from the position `x`, `y` to the destination. The recursive calls should be made in such a way that the search from this point is performed in the following order: North, East, South, and finally West. The search should start at the starting point, i.e. the 'S' character. In other words, in the main function you should call `findPath` with the coordinates of the start position. (Hint: use the findStart function!)

If a path is found, its length and the grid should be printed. The output of your program should look as follows:

```
Command? i
Number of rows? 6
Number of columns? 6
Input row 0: S#####
Input row 1: .....#
Input row 2: #.####
Input row 3: #.####
Input row 4: ...#.D
Input row 5: ##...#
Command? p
S#####
.....#
#.####
#.####
...#.D
##...#
Command? f
Found a path of length 11
S#####
++...#
#+####
#+####
.++#+D
##+++#
Command? q
Bye!
```

When no path is found, the error message `No path found` should be output. The grid should be printed to show the parts of the grid that were visit. The output of your program should look then as follows:

```
Command? i
Number of rows? 6
Number of columns? 6
Input row 0: S#####
Input row 1: .....#
Input row 2: #.####
Input row 3: #.####
Input row 4: ...#.D
Input row 5: ##.#.#
Command? f
No path found
S#####
~~~~~#
#~####
#~####
~~~#.D
##~#.#
Command? q
Bye!
```


Below is a larger example that also shows parts of the grid that were explored but are not part of the final path. The grid contains two destinations; notice that the closest destination is bypassed. Can you explain why? What would happen if we removed the further destination?

```
Command? p
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#...#...........#.##..##
########################
Command? s
The start is at x=10 and y=0
Command? f
Found a path of length 59
##########S####D########
D++++++++++#~~#+++#~~#~#
#+###++##~#~##~###+######+#
#++#++~###~#~~#~#++++#+#
#.+~~#~#~#~~~##~######+#
##+#####~###~~#~~~#.+++#
#.+++++++~~#~####~##+#.#
##.#####+######++++++#.#
#...#...++++++++#.##..##
########################
Command? q
Bye!
```

Task 8. Add the command 'r' to the main function that calls the function
`void resetPath(navigation_t nav)`
that resets all SEEN and PATH characters in the grid to UNSEEN, allowing findPath (and longestPath) to be run again.

Task 9. Write a function `int longestPath(navigation_t nav, int x, int y, int length)` that recursively calls itself and which finds *all paths* from the start to the destination. (The command does nothing if the grid has not been initialised.) It prints the length of every path it finds, and the end prints the length of the longest path(s). Hint: it is only a small change from `findPath`, which stops as soon as it finds a path, to not stop at the first match but keep going.
The algorithm that you have to implement is also illustrated in this screen recording: `www.es.ele.tue.nl/~kgoossens//teaching/5eia0/2022/06-navigation.mp4`

```
Command? p
#+++++
S+###D
#.....
..###.
#.....
Command? f
Found a path of length 7
Command? r
Command? l
Found a path of length 7
Found a path of length 11
The length of the longest path is 11
Command? q
Bye!
```

On the next page is an example output, when there are two destinations. Can you explain why the path to the closest destination is now found as the third path?


```
Command? p
#.....##...####.....#
..##S.#.D#......###..
#.....##########....#
..###.#.#........####
.#...#....######.....
...#...##........#.#.
Command? f
Found a path of length 52
#~~~++##+++##########+#
~~##S+#.D#+++++++###+.
#+++++############++++#
++###~#~#++++++++####
+#+++#++++######.....
+++#+++##........#.#.
Command? r
Command? l
Found a path of length 54
Found a path of length 50
Found a path of length 52
Found a path of length 52
Found a path of length 54
Found a path of length 50
Found a path of length 52
Found a path of length 54
Found a path of length 56
Found a path of length 48
Found a path of length 50
The length of the longest path is 56
Command? q
Bye!
```

At this point in time you may want to revisit the questions on page 4 on the order of the search statements.

Submission: Submit your file that implements the last task on Oncourse. You can resubmit as often as you want until the deadline.

- 3/8, v2.0: Removed PYNQ. Added command loop, more functions, malloc, and structs.
- 11/8 v2.1: More instructions for when using libasan.
- 14/9 v2.2: Minor clarifications.
- 5/10 v2.3: Now also print the grid when no path is found.
- 9/10 v2.4: Improved the example for task 6.

Input / output test cases

Long lines have been wrapped at 70 characters for legibility. When your program output is compared to the expected output lines will not be wrapped.

Case 01

Input:

```
x
q
```

Output:

```
Command? Unknown command 'x'
Command? Bye!
```


Case 02

Input:

```
i
1
2
i
2
2
..
.D
i
2
2
S.
..
i
2
2
S.
.D
i
2
2
SS
.D
i
5
7
S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
q
```

Output:

```
Command? Number of rows? Number of columns? The number of rows and
columns must be at least two
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Command? Number of rows? Number of columns? Input row 0:
Input row 1: Input row 2: Input row 3: Input row 4: Command? Bye!
```


Case 03

Input:

```
i
2
2
..
.D
p
i
2
2
S.
..
p
i
2
2
S.
.D
p
i
2
2
SS
.D
p
i
2
2
S.
DD
p
i
6
6
S#S###
.....#
#.####
#.####
...S.D
##...#
p
q
```


Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? ..
.D
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.
..
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.
.D
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Command? S#
.D
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.
DD
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Input row 2: Input row 3: Input row 4: Extra starting
point
Input row 5: Command? S#####
.....#
#.####
#.####
...#.D
##...#
Command? Bye!
```


Case 04

Input:

```
i
6
6
S#S###
.....#
#.####
#.####
...S.D
##...#
p
i
10 24
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#..........#.##..##
########################
p
i
6 21
#.....#....####.....#
..##S.#.D#......###..
#.....##########.....
..###.#.#........####
.#...#....######.....
...#...##........#.#.
p
q
```


Output:

```
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Input row 2: Input row 3: Input row 4: Extra starting
point
Input row 5: Command? S#####
.....#
#.####
#.####
...#.D
##...#
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Input row 6:
Input row 7: Input row 8: Input row 9: Command?
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#...........#.##..##
########################
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Command?
#.....#....####.....#
..##S.#.D#......###..
#.....##########.....
..###.#.#........####
.#...#....######.....
...#...##........#.#.
Command? Bye!
```


Case 05

Input:

```
i
3
3
...
.#.
...
p
s
i
6
6
S#S###
.....#
#.####
#.####
...S.D
##...#
p
s
i
10 24
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#...........#.##..##
########################
p
s
i
6 21
#.....#....####.....#
..##S.#.D#......###..
#.....##########.....
..###.#.#........####
.#...#....######.....
...#...##........#.#.
p
s
q
```

Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Command? ...
.#.
...
Command? Grid contains no starting point
Grid contains no destination
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Input row 2: Input row 3: Input row 4: Extra starting
point
Input row 5: Command? S#####
.....#
#.####
#.####
...#.D
##...#
Command? The start is at x=0 and y=0
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Input row 6:
Input row 7: Input row 8: Input row 9: Command?
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#...........#.##..##
#########################
Command? The start is at x=10 and y=0
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Command?
#.....#....####.....#
..##S.#.D#......###..
#.....##########.....
..###.#.#........####
.#...#....######.....
...#...##........#.#.
Command? The start is at x=4 and y=1
Command? Bye!
```


Case 06

Input:

```
i
2
2
S.
.D
p
s
f
i
3
3
S.#
.#.
.#D
p
s
f
i
2
2
S.
DD
p
s
f
i
2
20
D..................S
....................
p
s
f
i
2
20
D..................
...................S
p
s
f
i
2
20
S..................D
....................
p
s
f
q
```


Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.
.D
Command? The start is at x=0 and y=0
Command? Found a path of length 2
S+
.D
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Command? S.#
.#.
.#D
Command? The start is at x=0 and y=0
Command? No path found
S~#
~#.
~#D
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.
DD
Command? The start is at x=0 and y=0
Command? Found a path of length 2
S+
DD
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? D..................S
....................
Command? The start is at x=19 and y=0
Command? Found a path of length 39
D++++++++++++++++++S
++++++++++++++++++++
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? D...................
...................S
Command? The start is at x=19 and y=1
Command? Found a path of length 38
D++++++++++++++++++
.++++++++++++++++++S
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S..................D
....................
Command? The start is at x=0 and y=0
Command? Found a path of length 19
S++++++++++++++++++D
....................
Command? Bye!
```


Case 07

Input:

```
i
6
6
S#S###
.....#
#.####
#.####
...S.D
##...#
p
s
f
q
```

Output:

```
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Input row 2: Input row 3: Input row 4: Extra starting
point
Input row 5: Command? S#####
.....#
#.####
#.####
...#.D
##...#
Command? The start is at x=0 and y=0
Command? Found a path of length 11
S#####
++~~~#
#+####
#+####
.++#+D
##+++#
Command? Bye!
```


Case 08

Input:

```
i
5
7
S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
p
f
i
10 24
##########S####D########
D.........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#.........#.####.##.#.#
##.#####.######......#.#
#D..#..........#.##..##
########################
p
s
f
i
6 21
#.....#....####.....#
..##S.#.D#......###..
#.....##########.....
..###.#.#........####
.#...#....######.....
...#...##........#.#.
p
s
f
q
```


Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Command? S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
Command? Found a path of length 19
S++++++
D#~#~#+
+#~#~#+
+#~#~#+
+++++++
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Input row 6:
Input row 7: Input row 8: Input row 9: Command?
##########S####D########
D.........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#...........#.##..##
########################
Command? The start is at x=10 and y=0
Command? Found a path of length 59
##########S####D########
D++++++++++#~~#+++#~~#~#
#+####++##~#~##~###+######+#
#++#++~###~#~~#~#++++#+#
#.+~~#~#~#~~~##~######+#
##+#####~###~~#~~~#.+++#
#.+++++++~~#~####~##+#.#
##.#####+######++++++#.#
#D..#...++++++++#.##..##
########################
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Command?
#.....#....####.....#
..##S.#.D#......###..
#.....##########.....
..###.#.#........####
.#...#....######.....
...#...##........#.#.
Command? The start is at x=4 and y=1
Command? Found a path of length 52
#~~~++#.+++##########+#
~~##S+#.D#+++++++###+.
#+++++##########++++.
++###~#~#++++++++####
+#+++#+++######.....
+++#+++##........#.#.
Command? Bye!
```


Case 09

Input:

```
i
5
7
S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
p
f
r
p
f
r
p
q
```

Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Command? S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
Command? Found a path of length 19
S++++++
D#~#~#+
+#~#~#+
+#~#~#+
+++++++
Command? Command? S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
Command? Found a path of length 19
S++++++
D#~#~#+
+#~#~#+
+#~#~#+
+++++++
Command? Command? S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
Command? Bye!
```


Case 10

Input:

```
i
2
20
D..................S
####################
p
s
l
i
2
2
S.
.D
p
s
l
i
2
2
SD
..
p
s
l
i
2
3
S.D
...
p
s
l
i
2
4
S..D
....
p
s
l
q
```


Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? D.................S
####################
Command? The start is at x=19 and y=0
Command? Found a path of length 19
The length of the longest path is 19
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.
.D
Command? The start is at x=0 and y=0
Command? Found a path of length 2
Found a path of length 2
The length of the longest path is 2
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? SD
..
Command? The start is at x=0 and y=0
Command? Found a path of length 1
Found a path of length 3
The length of the longest path is 3
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S.D
...
Command? The start is at x=0 and y=0
Command? Found a path of length 2
Found a path of length 4
Found a path of length 4
Found a path of length 4
The length of the longest path is 4
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Command? S..D
....
Command? The start is at x=0 and y=0
Command? Found a path of length 3
Found a path of length 5
Found a path of length 5
Found a path of length 5
Found a path of length 5
Found a path of length 7
Found a path of length 5
Found a path of length 5
The length of the longest path is 7
Command? Bye!
```


Case 11

Input:

```
i
6
6
S#S###
.....#
#.####
#.####
...S.D
##...#
p
s
l
q
```

Output:

```
Command? Number of rows? Number of columns? Input row 0: Extra
starting point
Input row 1: Input row 2: Input row 3: Input row 4: Extra starting
point
Input row 5: Command? S#####
.....#
#.####
#.####
...#.D
##...#
Command? The start is at x=0 and y=0
Command? Found a path of length 11
The length of the longest path is 11
Command? Bye!
```


Case 12

Input:

```
i
5
7
S......
D#.#.#.
.#.#.#.
.#.#.#.
.......
p
f
r
l
i
10 24
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#...........#.##..##
########################
p
s
f
r
l
i
6 21
#.....##...####.....#
..##S.#.D#......###..
#.....##########....#
..###.#.#........####
.#...#....######.....
...#...##........#.#.
p
s
f
r
l
i
5 6
#.....
S.###D
#.....
..###.
#.....
p
s
f
r
l
q
```


Output:

```
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Command? S.....
D#.#.#.
.#.#.#.
.#.#.#.
.......
Command? Found a path of length 19
S++++++
D#~#~#+
+#~#~#+
+#~#~#+
+++++++
Command? Command? Found a path of length 19
Found a path of length 15
Found a path of length 11
Found a path of length 1
The length of the longest path is 19
Command? Number of rows? Number of columns? Input row 0: Input row 1:
Input row 2: Input row 3: Input row 4: Input row 5: Input row 6:
Input row 7: Input row 8: Input row 9: Command?
##########S####D########
D..........#..#...#..#.#
#.##..##.#.##.###.##...#
#..#...###.#..#.#....#.#
#....#.#.#...##.######.#
##.#####.###..#...#....#
#..........#.####.##.#.#
##.#####.######......#.#
#D..#............#.##..##
########################
Command? The start is at x=10 and y=0
Command? Found a path of length 59
##########S####D########
D++++++++++#~~#+++#~~#~#
#+##++##~#~##~###+##+++#
#++#++~###~#~~#~#++++#+#
#.+~~#~#~#~~~##~######+#
##+#####~###~~#~~~#.+++#
#.+++++++~~#~####~##+#.#
##.#####+######++++++#.#
#D..#...++++++++#.##..##
########################
Command? Command? Found a path of length 59
Found a path of length 23
Found a path of length 59
Found a path of length 23
Found a path of length 15
Found a path of length 17
Found a path of length 53
Found a path of length 17
Found a path of length 17
Found a path of length 57
Found a path of length 21
Found a path of length 57
Found a path of length 21
Found a path of length 13
Found a path of length 17
Found a path of length 53
Found a path of length 17
Found a path of length 17
Found a path of length 19
Found a path of length 55
Found a path of length 19
```