

Project 11

due at midnight on Tue Dec 13 (60 points)

For this assignment, we'll simulate a "random walk" across an $M \times N$ grid. (M and N should be *constants* defined in your code.)

Your program will start out at the upper left (row 0, column 0) and randomly walk from one square on the board to another, always going North, South, East, or West (equivalently: up, down, right, or left). As your program walks, it will mark the squares it visits with the letters A through Z, in the order visited. It may not visit the same square twice.

After the walk is finished, the program should print the resulting grid. Note that it should not output anything while it figures out where to walk: only *afterwards* will we see the trace of the walk.

Here is an example, on a 6×8 grid:

```
AB...ZY.
DC...WX.
E.MNUV..
F.LOTS..
GJKPQR..
HI.....
```

You can see where the program walked by tracing the letters A through Z from one square to the next.

Before taking each step, your program will need to make sure that it (a) won't go outside the grid, and (b) doesn't land on a square that was already visited. The walk is finished when it gets to Z or when there are no possible moves left.

Here are a few more examples, including some that get 'stuck' before reaching Z.

A...QRS.	ABCDE...	ABCDGH..
B...P.TUF...	TU.EFI..
CLMNO.WVGJKL	SVONMJ..
DK....X.HIPM	RQP.LK..
EJI...Y.ON
FGH...Z.

You should be able to change the M and N constants and recompile your program to get different shapes:

4x21:	4x10:	3x3:
A.....OPST.....	ADENOPQRS.	ABC
BC...MNQRU.....	BCFML...TU	.ED
ED.JKL...VWXYZ.....	..GHK...WV	.FG
FGHI.....	...IJ...XY	

Optionally, once your program is working as specified above, you may want to try this variation. Before taking the random walk, place K road blocks = on the board. Your walk cannot cross a road block, so you are more likely to become stuck. In the examples below, $M=6$, $N=8$, and $*K=10$.

ABCDEFGG=	ABC=..=.	ABC=SR..
.===N=HI	..=....=	.ED.PQ..
..=.MLKJ	=.....	=F=NO=..
...==...	=.....	=G=M..==
.....	=.....	.HKL==..
.=.....=.	...=..=.	.IJ.....

Your solution should use *functions* effectively to modularize its calculations. For example, my solution uses these ‘predicates’ (Boolean functions):

```
bool can_go_left(int i, int j);
bool can_go_right(int i, int j);
bool can_go_up(int i, int j);
bool can_go_down(int i, int j);
bool stuck(int i, int j);
```

and these void functions:

```
void initialize_grid(char grid[M][N]);
void print_grid(char grid[M][N]);
```

Name your program p11grid.cpp and submit to [this dropbox](#).