Week 6

Word Ladder

In this game, made famous by the author Lewis Caroll, and investigated by many Computer Scientists, including Donald Knuth :

`http://en.wikipedia.org/wiki/Word_ladder`

you find missing words to complete a sequence. For instance, you might be asked how go from "WILD" to "TAME" by only changing one character at a time:

W	I	L	D
W	I	L	E
T	I	L	E
T	A	L	E
T	A	M	E

We will use a heavily constrained version of this game:

- Words are always four letters long.
- We only seek ladders of five words in total.
- Only one letter is changed at a time.
- A letter is only changed from its initial sate, to its target state.

So, in the example above, it is enough to give the first word, the last word, and the position of the character which changed on each line. On line one, the fourther letter 'D' was changed to an 'E', on the next line the first character "W" was changed to a "T" and so on. The whole ladder is defined by "WILD", "TAME" and the sequence $4, 1, 2, 3$.

				Which position changes
W	I	L	D	*4*
W	I	L	E	*1*
T	I	L	E	*2*
T	A	L	E	*3*
T	A	M	E	

To help with this program, we are going to need several helper functions, including:
A function to create all 24 permutations of the numbers 1 to 4, i.e.:

```
1234
1243
1324
1342
1423
1432
.
. etc.
.
4132
4213
4231
4312
4321
```

Plus another function :

```
int edit_distance(char *s, char *t);
```

which returns the number of characters which are different between two strings of the same length. For our strings of length four (excluding the null character) this will be either $0, 1, 2, 3$ or 4.
Now, given the file of valid four letter words on the unit page, write a program which when given two words on the command line (`argv[1]` and `argv[2]`) outputs the correct solution, if available. Use an exhaustive search over all 24 permutions until one is valid. Check that the following work:

C	O	L	D	*3*
C	O	R	D	*2*
C	A	R	D	*1*
W	A	R	D	*4*
W	A	R	M	

P	O	K	E	*3*
P	O	L	E	*4*
P	O	L	L	*1*
M	O	L	L	*2*
M	A	L	L	

C	U	B	E	*4*
C	U	B	S	*1*
T	U	B	S	*3*
T	U	N	S	*2*
T	O	N	S	

Langton's Ant

See `http://en.wikipedia.org/wiki/Langton's_ant`
Write a program that show the behaviour of Langton's Ant and displays it on the screen using text. The ant will start in the middle, and each cell will have 2 states : `'#'` (white) or `'.'` (black).
At each step, the ant:

- At a white square, turn 90 right, flip the color of the square, move forward one unit
- At a black square, turn 90 left, flip the color of the square, move forward one unit

After each step, display the board, and wait for the user to type a character before repeating.
After 30 or so iterations, the board might look something like :

```
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . ## . . . . . .
. . . . . . # . . # . . . . .
. . . . . . # . ### . . . .
. . . . . . . . . ## . . . .
. . . . . . . . ## . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
```

This is a good chance to practice 2D arrays, but also enumerated types (for the direction, state of the square etc.)
Now extend this program to use SDL instead.