Middle East Technical University   Department of Computer Engineering

CENG 140

C Programming

Spring 2015-2016

Take Home Exam 1

1 Regulations

- **Due date:** 2 May 2016, Monday, 23:55
  Not subject to postpone, please do not ask for.
- **Submission:** Electronically. You will be submitting your program source code written in a file which you will name as **the1.c** through the COW. Resubmission is allowed (till the last moment of the due date), the last will replace the previous.
- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
- **Team:** There is **no** teaming up. The take home exam has to be done/turned in individually.
- **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
- **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

2 Introduction

This assignment aims to get you familiar with multi-dimensional arrays, recursion and simple input/output in C.

Spider cannibalism is the act of a spider consuming another individual of the same species as food. In this assignment, you are going to implement a program to solve a spider game on a field. A grid-like field contains multiple spaces and spiders on it. The aim is to decide that the given field evolves to an **an** ideal field with only one spider on it or not. In order to reach an ideal field, one of the spiders will be eaten by another at each turn.

There are some criterias defining how spiders eat each other and move, which will be explained further in the following section. Your program has to act according to these specifications. You will either find an ideal field or you will conclude that there cannot be an ideal field with the given configuration.

In order to solve this problem, you have to have a good understanding of recursion and backtracking. Backtracking is a generic algorithm to generate all/some solutions to a problem. The idea is to incrementally build candidate solutions and "backtrack" (to a previous step) as soon as the candidate is determined as an invalid solution. See `https://www.cis.upenn.edu/~matuszek/cit594-2002/Pages/backtracking.html` and hint provided in the specifications for further information about backtracking.

Keywords: *Recursion, Backtracking, Arrays*

3 Specifications

- Spiders live on a V-shaped field where we show a spider with **s** and an empty space with **e**.
- At each turn, only one spider can move. This can be *any* of the spiders that is *able* to move.
- A spider can only move by jumping over a neighbor spider (i.e., by eating it) to an empty space. A spider can move at most in six different directions (i.e., diagonal and horizontal but not vertical). However, it cannot go out of the bounds of the field.
- For example, spider at 16 has 6 candidate positions to go: 03 (top-left, by eating 10), 05 (top-right, by eating 11), 14 (left, by eating 15), 18 (right, by eating 17), 23 (bottom-left, by eating 20) and 25 (bottom-right, by eating 21). However since 10 and 17 is empty (no spider to eat) it cannot move to 03 and 18; and since 25 is not empty, it cannot move to 25.

```
e e e e e e e     01  02  03  04  05  06  07
 e e e s e e        08  09  10  11  12  13
  e s s e e           14  15  16  17  18
   e s s e              19  20  21  22
    e e s                 23  24  25
     e e                    26  27
      e                       28
```

- An ideal field is the one with only one spider on it. Our goal is writing a recursive function that can determine whether a given field can evolve to an ideal field after a number of moves of the spiders.
- **Iterative solutions will get NO credits.**
- If the program can find such a sequence of moves that converts the initial field to an ideal one, it will print the solution and terminate (i.e., no need to look for any other solutions). If the field can not be converted to an ideal one, the program will give a message and terminate. See Output section for the details of the expected output for both cases.
- **Hint:** You should design a recursive function, e.g., `CHECK(field)` to decide whether a given initial field may become an **ideal** field after a number of turns (and, moves). That is, at a given turn, if the current field is **ideal**, the function can stop and return. Otherwise, the function should try **every** legal (valid) move on the current field in this turn, and after every move, check again whether the new field is either an ideal one or may evolve to an ideal one, and if not, **undo** the last move (and try another one).
- **Example:** Consider the field

```
e e s s     01  02  03  04
 e s e        05  06  07
  e e           08  09
   e              10
```

Now, for this initial field (clearly not ideal); there are two possible moves: the diagonal one and horizontal one. Assume the algorithm first tries the diagonal move (spider at position 3 moves to 8 by eating spider at 6): then it checks for the new field, since the field is not ideal and there is no more possible moves, it should undo the first move. Then it tries what happens if it makes the horizontal move (spider at position 4 moves to 2 by eating spider at 3), checks the new board, not yet ideal but there is a possible move, try the diagonal move now, and check the new board, and done; it is ideal! Note that, undoing the previous move at each turn is called backtracking.

- There can be more than one solution for finding the ideal field, i.e. the consecutive moves can be different. It is enough to come up with **one** solution.
- There can be any number of empty spaces in the initial field configuration.
- Number of rows in the field is not fixed, determined by the given value in the input. See next section for details. Hence, try to come up with a generic algorithm to find valid moves on a field.

4 Input

- Sample input:

```
4
ssss
sse
ss
s
```

- You are expected to write a program that reads from standard input. You may use redirection for simplicity when testing your code. See `https://www.cs.bu.edu/teaching/c/file-io/intro/`.
- The first line of the input contains an integer K which represents the length of the field that spiders live, i.e. the number of lines you are going to read from the standard input.
- Following K lines represents K rows of the field, each one having one less character than the line above, i.e. $K, K-1, ..., 2, 1$. On these lines you will see two different characters, where **s** being a spider and **e** being an empty space in the field.
- You are **not** allowed to use any dynamic data structures (as we have not discussed them in the class yet), so use multi-dimensional arrays to represent the field.
- There will be no erroneous input.

5 Output

- Sample output for the input above:

```
e e s e
 e e e
  e e
   e

s s e e
 e e e
```

```
  e e
   e

s e s s
 e e e
  e e
   e

s e s e
 e e s
  e s
   e

e e s e
 s e s
  s s
   e

s s e e
 s e s
  s s
   e

s e s s
 s e s
  s s
   e

s s s s
 s s s
  s e
   e

s s s s
 s s e
  s s
   s
```

- Your program will write to standard output.
- If the field evolves to an ideal one, output will contain the field after each move, in reverse order. That is, first one is the latest status of the field (only one spider on it) and the last one is the initial status given as the input.
- Note that, you are required to print the final field when your check algorithm decides that it is ideal; and the fields all the way back through the recursive call chain. However, in this case, you should not permanently change your field with a successful move (otherwise, you will only print the final (ideal) field again and again). You should also handle this situation in an appropriate way, so that when the program terminates (either finds an ideal field or not), the field array should store the initial field.

- Note that there may be more than one possible solution, you have to print one of them. As long as your moves are valid and you found the solution, it is correct.
- In some cases, given field may not evolve to an ideal field. In that case, print **"No ideal field!"** without quotes. (After trying each possible move and concluding that there is no way to go.)
- In order to print the field as above, use the following code snippet. As your code will be evaluated using black-box technique, you have print same as the sample. Replace `k` and `field` variables with yours.

```
for (int i = 0; i < k; ++i) {
    for (int j = 0; j < i; ++j) {
        printf(" ");
    }

    for (int j = 0; j < (k - i); ++j) {
        printf("%c ", field[i][j]);
    }
    printf("\n");
}
printf("\n");
```

6 Compile, Run and Test

- Make sure that you can compile and run your code on Ineks with:

```
gcc -Wall -o the1 the1.c
./the1 < sample_input.txt
```