0512-1820 Fall 2024
Home Assignment #3

Yehonatan Kohavi - kohaviy@mail.tau.ac.il

Due date: 21/12/2024

In this assignment, you will get hands-on experience with **arrays & strings** in the C programming language.

Submission Guidelines

- Due date is **21/12/2024**
- Submission by **pairs**. Only one student of the pair should submit the submission file.
- Submission file is: **hw3_ID1_ID2.zip**

  The zip should include the following files:

  - **ex1.c**
  - **ex2.c**

  E.g. for a pair of students with IDs 123456789 and 987654321 the zip file should be named:

  hw3_123456789_987654321.zip

  And for example, the first source file should be named: **ex1.c**. This is merely an example.
- Do not forget to zip all of the required files as mentioned above. No internal directories and no other irrelevant files should be included in your zip, only the source files we requested!
- Please use the standard C libraries only! No other libraries should be installed or used unless you have been specifically instructed.
  The only header file you should include are `<stdio.h>`, `<ctypes.h>`, `<string.h>`. Other header files are not allowed in this home assignment.

- For any issues & questions you can use the forum in moodle, consult with your peers and also use google.
- **Warning 1**: If your code doesn't compile you will get 0, regardless of the amount of work you've put into coding.
- **Warning 2**: Do not cheat or use any automatic code generator to complete this home work! It is for your own good. Caught cheaters will be **punished**!
- **Warning 3**: Your code would be tested automatically. Failing to comply with naming conventions will result in **points deduction**!
- **Warning 4**: Mismatches during outputs comparison (due to different formatted prints) will cause failing tests and therefore lead to **points deduction**!

Suggested workflow

1. Generate a new **git** repository for this home assignment / Add a new directory to your existing HW git repository.
2. If there are any attached files, **download them** from moodle into your repository.
3. **Open & read the given files** for this home assignment.
4. **Read this entire document** before writing a single line of code.
5. Write some basic **tests** to make sure your code will work (TDD).
6. Let the **coding** begin!
   Don't forget to **commit & push your progress** in git for version control & collaboration.
7. Make sure your **code compiles** in the testing environment.
8. **Add more tests** with all of the corner cases you could think.
9. Make sure your **code runs properly** and correctly, and that all of your tests pass.
   Debug your code and fix it accordingly (you might find "rubberducking" pretty useful).
10. **Re-read this document** to make sure you have not forgotten anything.
11. **Check the moodle for any updates** regarding this assignment in the Q&A forum and in the Announcements forum.
12. **Zip your code** according to the submission guidelines above.

13. **Unzip your code** and repeat steps 7 & 9 to make sure everything is OK, and that the zip file and its content comply with the naming conventions specified above.
14. **Submit the zip file** to moodle.
15. Congratulations! you have **completed** the home assignment!

Good luck!

Exercise 1 - NxN Tic-Tac-Toe

In this exercise, you will implement an NxN board of Tic-Tac-Toe with 2 opponents:

- Player 1 would be marked as 'X'
- Player 2 would be marked as 'O'
- A free/empty cell would be marked as '_'.

The program would request the board size (N) from the user as follows:

```
Please enter the board size N [1-10]:
```

Then the program prints to the screen a welcome message and an empty board, e.g.:

```
Welcome to 7x7 Tic-Tac-Toe:
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
- - - - - - -
```

Now the game starts! In each round the program would ask for each player (in their turn) indices to their liking to mark their symbol as follows:

```
Player _, please insert your move:
```

In each round the program will:

1. Receive the player's indices (separated by a single comma).
2. Mark their symbol.
3. Print the board new board state.
4. Check if there is a winner.

A little reminder: a player can win if they are the first to complete a single row/column/diagonal with their own symbol.

- If there is a **winner** the program would declare the winner by printing the following message to the screen the and exit the program (return 0):

```
Player _ is the winner!
```

- If there is a **tie** (full board with no winner) the program would print the following message and exit the program (return 0):

```
There is a Tie!
```

- **Otherwise** the program would continue to the next round and switch the players turns.

Assumptions & Hints:

1. Assume board size from the user is a valid decimal integer in the range 1-10. No need to check this.
2. The board should be allocated **statically**.
3. If the user enters a board less than 10 (namely N), you should use the first N rows and N columns, i.e. rows 0,...,N-1 and columns 0,...,N-1. In any case you should print to the screen a NxN board and not 10x10. The board would be printed with a single space between the cells.
4. You should think about all of the **corner cases** (N=1, N=2, etc..).
5. You might want to generate some random tests and check your code coverage.
6. Players' IDs are **1 and 2**, not 0 and 1!
7. You can assume the input indices are **separated with ','** and both decimal integers in the valid integer range, but you should ignore white spaces!

8. Valid cell indices from user are in the range **[1,...,N]**, however if a player has inserted indices of an **occupied cell/out of bounds**, the program should prompt to the player the following message and ask for a new set of indices repeatedly until the input is valid:

```
Invalid indices, please choose your move again:
```

9. All of the printed messages are without trailing spaces, but end with a new line. Regardless, don't forget to compare your outputs with the example files to be aligned with print formatting.

Examples:

```
Please enter the board size N [1-10]:
2
Welcome to 2x2 Tic-Tac-Toe:
_ _
_ _
Player 1, please insert your move:
0,0
Invalid indices, please choose your move again:
1,2
_ X
_ _
Player 2, please insert your move:
-3,1
Invalid indices, please choose your move again:
1,1
O X
_ _
Player 1, please insert your move:
2,1
O X
X _
Player 1 is the winner!
```

Figure 1: Exercise 1 - 2_winner_1_diag_up

```
Please enter the board size N [1-10]:
3
Welcome to 3x3 Tic-Tac-Toe:
_ _ _
_ _ _
_ _ _
Player 1, please insert your move:
3,3
_ _ _
_ _ _
_ _ X
Player 2, please insert your move:
2,2
_ _ _
_ O _
_ _ X
Player 1, please insert your move:
1,3
_ _ X
_ O _
_ _ X
Player 2, please insert your move:
2,3
_ _ X
_ O O
_ _ X
Player 1, please insert your move:
2,1
_ _ X
X O O
_ _ X
Player 2, please insert your move:
1,1
O _ X
X O O
_ _ X
Player 1, please insert your move:
1,2
O X X
X O O
_ _ X
```

```
Player 2, please insert your move:
3,1
O X X
X O O
O _ X
Player 1, please insert your move:
3,2
O X X
X O O
O X X
There is a Tie!
```

Figure 2: Exercise 1 - 3_tie

Exercise 2 - Search String Permutation Counter in a String Pool (case insensitive)

In this exercise, you will return the number of case insensitive permutations of a search string in a pool of strings.

1. First the program would request the search string from the user as follows:

   ```
   Enter the search string:
   ```

2. Then the program will request the pool of strings from the user as follows:

   ```
   Enter the strings pool:
   ```

3. The program will read each string of the string pool (strings are separated by a new line), until there are no more strings from the user (**EOF**).

4. The program will count the number of case insensitive permutations of the search string within the pool and prints the result to the screen the following message:

   ```
   Number of permutations of "_" in the strings pool is: _
   ```

Naturally you should format the search string and the result in the reasonable '_' placeholders.

A little reminder: A permutation of a string S is a **rearrangement** of the characters of an ordered string S into a one-to-one correspondence with S itself. e.g.:

```
"stop" and "spot"
"yes" and "sey"
"a" and "a"
```

- A string of length N has N! permutations.
- Permutations are naturally **case sensitive.**

Assumptions & Hints:

1. In this assignment we are not **case-sensitive**, hence we treat lower case characters and upper case characters the same! so for example the following strings will be considered as valid permutations:

   ```
   "FG27j" and "J7f2g"
   "main*INT" and "*intMain"
   ```

2. Assume all of the input strings (in both search and pool) are of **maximum length of 10** characters. No need to validate this.

3. Assume the **maximum number of strings in the pool is 20**. No need to validate this. However the pool could be empty (corner case - 0 hits).

4. **EOF** in windows via command line can be generated by CTRL+Z, but you might want to insert tests via redirection from a file. You might also want to look at `scanf`s return value.

5. Strings might contain spaces! You might want to look at different **scanf spesifiers** in order to read the strings properly, and maybe flush the new line '\n'.

6. You might want (and should!) to use library functions from `<ctypes.h>` and `<string.h>` (namely look at `memmove`).

7. Think of all of the corner cases, and you might want to generate random tests.

8. All of the printed messages are without trailing spaces, but end with a new line. However don't forget to check the example files to be aligned with print formatting.

Examples:

```
Enter the search string:
bitcoin
Enter the strings pool:
Money
Value
100K
&hkjfh3
iL0VeCode
bI0tinC23
rfa
a
9031
^Z

Number of permutations of "bitcoin" in the strings pool is: 0
```

Figure 3: Exercise 2 - example 1

```
Enter the search string:
 Giant
Enter the strings pool:
st3ps
R
WHAT
yUO_OU
ta-_-ke
waLk1nggg
o n*)!
tHE
m0On
    3
buzzz
st1ng
giant
911
#100
A ginT
Major
Tom
LIfeOn
MArs S
^Z

Number of permutations of " Giant" in the strings pool is: 1
```

Figure 4: Exercise 2 - example 2