Department of Computer Science
Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia.

Assignment 2: Recursion

Assigned on Thursday October 06, 2022
[Deadline: Thursday October 27, 2022]

Objectives:

1. Apply recursion to solve simple problems.
2. Comprehend recursion concepts.

ABET Student Outcome (SO-2):
Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.

Read Carefully:
This assignment is worth **5%** of your final grade.

NOTE:
This is an individual assignment; you must solve it by yourself. Any form of cheating will result in receiving zero in the assignment.

The deadline for this assignment is **Thursday October 27, 2022 @ 11:59pm**. Once the clock becomes 11:59 PM, the submission will be closed!

LATE SUBMISSION: No assignment will be accepted after the deadline

Blackboard Submission:
This assignment must be submitted online via Blackboard.
The source file(s) of your program should be zipped up. You must name the zip file using the following naming convention: **SectionNumber_StudentID_ProgramNumber.zip**
Example: EA_1110348_P2.zip

Question:	1	2	3	Total
Points:	20	20	60	100

Question 1: (20 Points, SO-2)

Consider a **recursive** method that prints the riser and tread of stair. The riser is the stair's vertical surface (represented by a pipe symbol "|"), while the tread is the stair's horizontal surface and the part of the stair you step on (represented by underscore symbol "_"). The following two are sample outputs to print the stairs for number of *step* = 4 and 7:

Example 1: No. of step = 4

```
|_
  |_
    |_
      |_
```

Example 2: No. of step = 7

```
|_
  |_
    |_
      |_
        |_
          |_
            |_
              |_
```

1- Write a complete pseudocode/algorithm that accepts an integer number representing the number of *step* and outputs the stairs as explained above. **(15 points)**

Note: *write the information that you need to describe the header of the method*

2- Trace your algorithm **for length=4**. **(5 points)**

Note: You must write your tracing using recursion trace - its format is described in the course lectures.

Question 2: (20 Points, SO-2)

Consider the following recursive method and answer the questions:

```
 1 public class NewClass {
 2
 3   public static void main(String[] args) {
 4         int[][] st =new int[5][5];
 5         printX(st,0,st.length-1,7);
 6     }
 7
 8     public static void printX(int[][] st, int i, int j, int num) {
 9         if (i < st.length) {
10             st[i][i] = num;
11             st[i][j] = num;
12             for (int k = 0; k < st.length; k++) {
13                 System.out.print(st[i][k]);
```

```
14              }
15              System.out.println("");
16              printX(st, ++i, --j,num);
17          }
18      }
19  }
```

1- Trace the above code. Note: you must write your tracing using recursion trace as described in the course lectures. You must also show the final answer that is returned. **(15 points)**

2- What does the recursive method print? **(5 points)**

Question 3: (60 Points, SO-2)

Write a Java program that reads a set of commands from the input file and executes the relevant recursive method to get the required output, as shown in the attached input/output files. Each command in the input file is followed (on the same line of input) by the respective parameters required for executing the recursive method. **[15 points for writing the main method, 15 points for writing the recursive method for each of the below problems].**

⇒ **Problem 1:** Write a recursive method **characterPattern** that accepts one character parameter ***c***. The parameter ***c*** is one of the characters 'A' through 'Z'. The method uses recursion to print out a pattern of characters as follows:

- If the parameter ***c*** is the letter 'A', then the output will only be 'A'.
- For any other value of ***c***, the output (pattern printed) will have three parts:
  1. The output for the previous letter (***c*** -1)
  2. Then the letter ***c*** will be printed.
  3. Then a second copy of the output for the previous letter will be printed.

- **Example 1:**
  Command: **charPattern A**
  Output: A
- **Example 2:**
  Command: **charPattern B**
  Output: ABA
- **Example 3:**
  Command: **charPattern C**
  Output: ABACABA
- **Example 4:**
  Command: **charPattern D**
  Output: ABACABADABACABA

⇒ **Problem 2:** Write a recursive method **diamondShape** that prints a large diamond made of small stars ("* "). The size of the diamond will be determined from the maximum width of the diamond, which is given in the input file and is guaranteed to be odd integer.

Example 1:

```
Command: drawDiamond 7
Output:
       *
     * * *
   * * * * *
* * * * * * *
   * * * * *
     * * *
       *
```

Hint: you CAN use a for loop to draw one, single row of the diamond. But you need recursion to call the method on the "smaller" version of the problem.

⇒ **Problem 3:** You must write a recursive method **throwBricks**, which will count the number of ways to throw bricks. Consider a construction site worker who manually unload bricks from the truck. The worker has three options depending on his fatigue level to pick brick from truck and throw it on the ground:

1. Throw one brick at a time
2. Throw two bricks at a time
3. Throw three bricks at a time

You need to calculate how many different ways the worker can throw the bricks using the three options (trow one brick at a time, two bricks at a time, or three bricks at a time).

- **Example 1:**
  `Command: bricksToUnload 1`
  `Output: 1`

  For 1 brick, the worker has only one option to throw one brick

- **Example 2:**
  `Command: bricksToUnload 2`
  `Output: 2`

  For 2 bricks, the worker has two options: (1) throw first brick and then throw the second brick (2) Throw both bricks together

- **Example 3:**
  `Command: bricksToUnload 3`
  `Output: 4`

  For 3 bricks, the worker has four options: (1) throw one brick, then second brick and then third brick (2) Throw one brick, then two bricks together (3) Throw two bricks together, then one brick (4) Throw all three bricks together

Input File Specifications

You will read in input from a file, "**input.txt".** Have this AUTOMATED. Do not ask the user to the name of the input file. You should read in this automatically. The first line of the input file will have one positive integer, representing the number of commands (lines) inside the input file.

Each of the following n lines will have a command, and each command will be followed by appropriate data as described below (and this data will be on the same line as the command).

The commands (for the 3 recursive methods), and their relevant data, are described below:

⇒ **`charPattern`**: This command will be followed by a single character.
⇒ **`drawDiamond:`** This command will be followed by a single integer number.
⇒ **`bricksToUnload:`** This command will be followed by a single integer number.

Output Format

Your program must output to a file, called **"output.txt". You must follow the program specifications exactly.**

Grading Details

Your program will be graded upon the following criteria:

1. Adhering to the implementation specifications listed on this write-up.
2. Your algorithmic design.
3. Correctness.
4. **Use of Recursion. If your program does not use recursion, you will get a zero.**
5. The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
6. Your program should **include a header comment with the following information: your name, ID, email, course number, section number, assignment title, and date.**
7. Your output MUST adhere to the EXACT output format shown in the sample output file.

Deliverables

You should submit a folder with **TWO** files inside:

1- You must name the java file using the following naming convention:

Example: *EA_ 1110348_ P2.**java***

2- .docx or .pdf file containing solution of Question1 and 2.