Homework  
Subject: **Data structures and Algorithms 1**

**Topics: Convert from Infix to Postfix***(Students carefully read all instructions in this topic before doing the test)*

**I. Introduction to the problem**

With simple expressions performed from left to right, the computer can understand to perform and produce results, for example 10 + 3 - 8, the computer will take 10 add 3, and subtract 8. However, with example 10 + (3 - 8), how can the computer understand it to implement the expression in parentheses first? One of the ways to solve this problem is to use reverse Polish notation.

Reverse Polish notation (RPN) is an expression expressed in postfix form. For example, the infix expression:



can be converted to postfix form like this:



We will use Stack to convert by the following steps:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tokens from infix string** | **Steps** | **Current Stack** | **Write postfix string** |
| 10 | Operand (number): write string |  | 10 |
| + | Operator (sign): push to Stack | + |  |
| ( | “(“ : push to Stack | + ( |  |
| 3 | Operand: write string | + ( | 10 3 |
| - | Operator: push to Stack | + ( |  |
| 8 | Operand: write string | + ( - | 10 3 8 |
| ) | “)“ : pop all the elements in Stack and write to the string until the sign “(“ and pop the sign “(“ out off the Stack | + | 10 3 8 - |
| No more token | Write all the signs in Stack to the string |  | 10 3 8 - + |

In the process of pushing operators to the Stack, if the pushed operator has a lower priority than the top operator, then we must get all the higher priority operators to write to the string before push the considered operator to Stack.

Example of expression: (2 \* 4 - 3) + 6

When “-” is considered, “\*” has already been in the Stack. Because “-” has lower priority than “\*”, we take “\*” out of the Stack to write in string 2 4 \*, before push “-” into the Stack. Hence, the postfix result of this example is 2 4 \* 3 - 6 +

To calculate the postfix notation of the above example, a Stack will be used as follows:



|  |  |  |
| --- | --- | --- |
| **Tokens from infix string** | **Steps** | **Current Stack** |
| 10 | Operand: push 10 to Stack | 10 |
| 3 | Operand: push 3 to Stack | 10, 3 |
| 8 | Operand: push 8 to Stack | 10, 3, 8 |
| - | Operator: Take the second number from the top of the Stack (i.e., 3) to perform the operator (i.e., -) with the number from the top of the Stack (i.e., 8). Push the result to to Stack, 3 - 8 = -5 | 10, -5 |
| + | Operator: Take the second number from the top of the Stack (i.e., 10) to perform the operator (i.e., +) with the number from the top of the Stack (i.e., -5). Push the result to to Stack, 10 + -5 = 5 | 5 |

Now the result will be the last number in the Stack.

Students may refer to the math problem and algorithm at:   
RPN problem: https://en.wikipedia.org/wiki/Reverse\_Polish\_notation  
RPN string generation algorithm: <https://en.wikipedia.org/wiki/Shunting-yard_algorithm>

**II. Provided resources**

The provided source code includes these files:

- Input and expected output files:

o *input1.txt*: contains example infix expressions for the first requirement.  
o *input2.txt*: contains example postfix expressions for the second requirement.  
o *output* folder includes: *output1.txt*: contains sample results of the first requirement và *output2.txt*: contains the sample result of the second requirement.

**- Do not edit these files:**

o *ReadFile.java*: class of reading input file.  
o *Main.java*: calls the objects required by the exercise and writes to the output files.   
o *Requirement1\_OutputGetter.java*: used for evaluation, no edition is allowed.  
o *Requirement2\_OutputGetter.java*: used for evaluation, no edition is allowed.

**- Fill your code in these files:**

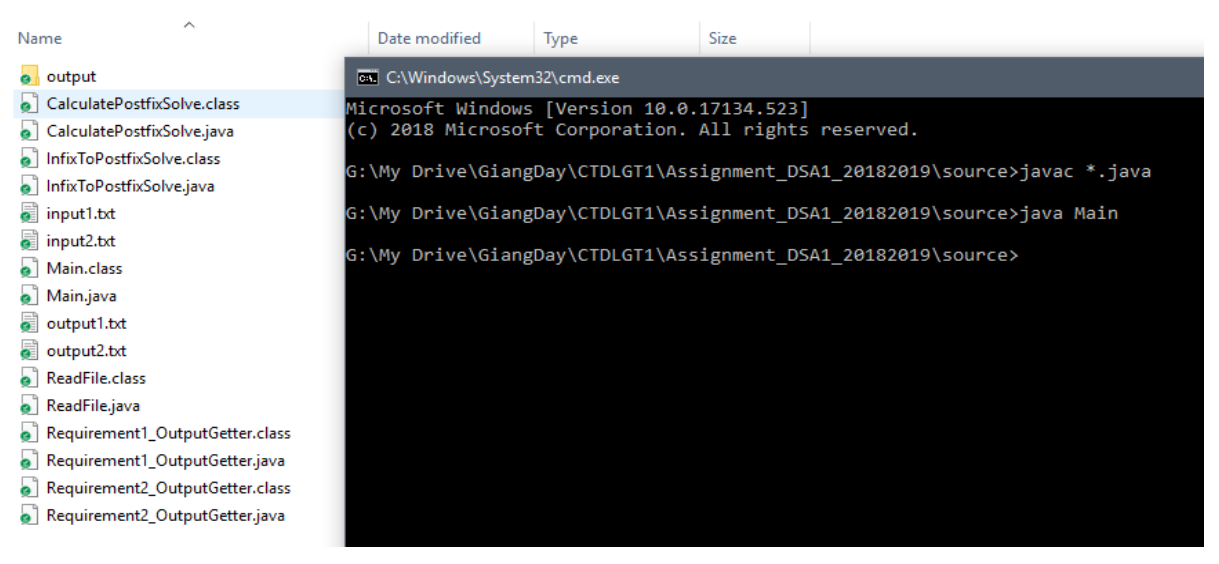
o *InfixToPostfixSolve.java*: class of converting infix string into postfix string.  
o *CalculatePostfixSolve.java*: class of converting postfix string to result of expression.

**III. How to compile the program**Students run in the command line window as they’ve learned in the practice class:

- Browse the path to the folder containing the exercise.

- Type *javac \* .java* command to compile.

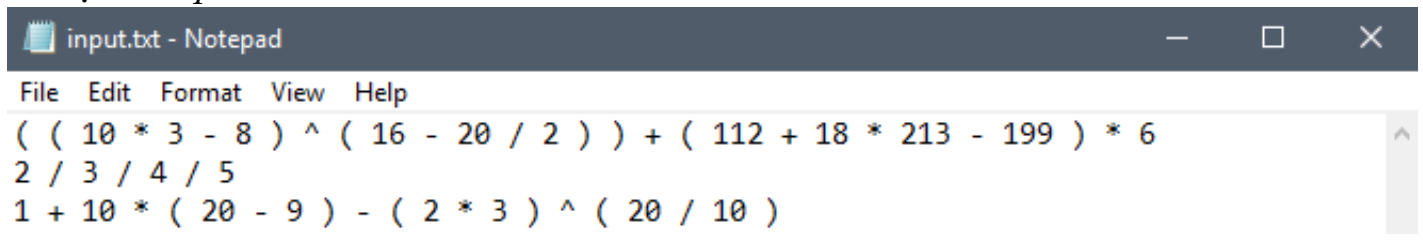
- Type *java Main* command to run the program.

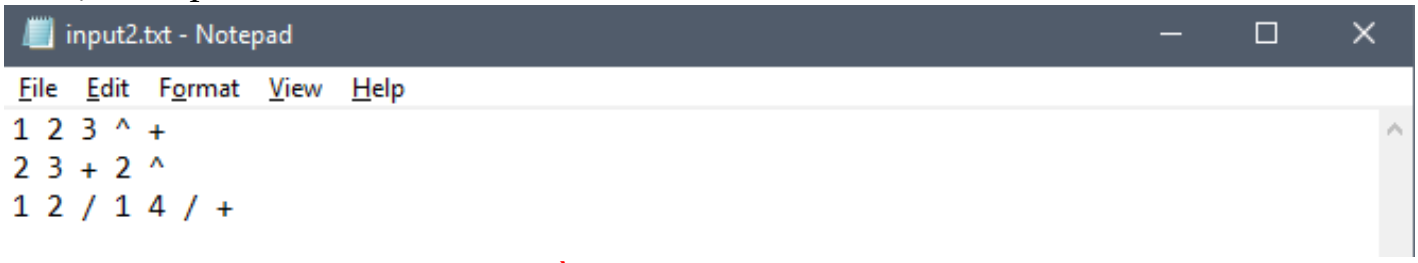


Sau khi chạy, hai file output kết quả bài làm sẽ tự sinh ra. Sinh viên thực hiện các yêu cầu ở mục dưới và so sánh kết quả output của mình với kết quả output đã được cung cấp sẵn.

**IV. Requirements of the exercise  
Note before solving the exercise:**- This exercise only requires working with expressions containing integers (int) and operators add “+”, substract “-”, multiply “\*”, divide “/” and power “^”. File *input1.txt* and *input2.txt* contains a standard-formatted expression, tokens are separated by a space. All expressions used for marking are meaningful and in right format.

Example *input1.txt* file:

  
Example *input2.txt* file



- In this exercise, students do not need to redefine Stack by themselves but are allowed to use the Stack library available in Java. The code files provided to students have already been added the Stack library.

- Students carefully review input and output files at first. How many lines that *input1.txt* file has, the same number of result lines that *output1.txt* file will have. Similarly, *input2.txt* and *output2.txt* will have the same number of lines.

- Students can manually add testcase to input files to try many different cases but note that the added data must be in the correct format as above.

- Students should carefully read the **main** function to direct their implementation of the required classes. The main function calls *Readfile.java* to write to *output1.txt* and *output2.txt*.

- Bài làm của sinh viên **phải chạy được trên hàm main đã cung cấp sẵn**. Sinh viên không chỉnh sửa file **Readfile.java**, file **Main.java** và **2 file interface**.

- The students’ codes **must run on the provided main function.** Students are not allowed to edit **Readfile.java**, **Main.java** and **2 interface files**.

- Student implement **REQUIREMENT 1** on the *InfixToPostfixSolve.java* file and **REQUIREMENT 2** on the *CalculatePostfixSolve.java* file. Some methods in these two files are absolutely not allowed to edit the name and code inside the method – details will be mentioned in the requirement section below.

- Students can add new functions to serve the assignment, but the students' work must be run with the provided Main.java file and they are not allowed to modify at the mentioned places above.

**REQUIREMENT 1 (5/10)**

Explaination of the ***InfixToPostfixSolve*** class (students would not modified the name of the class):

Attributes:

- *inputString*: infix input string (students cannot edit attribute names)

- *outputString*: postfix output string (students cannot edit attribute names)

Methods:

- The constructors and attribute setters: students must not modify these methods.

- String infixToPostfix(): used to convert the infix string *inputString* into postfix string.

- String [] stringTokenizer(): used to separate token by token using spaces, the returned result is an array of characters.

- int priorityOfOperator(String op): returns the priority of the operator

- String getOutputString(): return result *outputString*, *main()* function will call this method to get the result. *(students cannot edit this method)*

- private boolean isNum (String c): check whether the input string c is a number or not.

Students implement the methods so that the **inputString** is the infix input string and **outputString** is the returned postfix string. The result will be written in "output1.txt" file by the main function.

**REQUIREMENT 2 (5/10)**Explaination of the ***CalculatePostfixSolve*** class (students would not modified the name of the class):

Attributes:

- String *inputString*: postfix input string (students cannot edit attribute names)

- double *resultOfExpression*: the result of the expression (students cannot edit attribute names)

Methods:

- The constructors and attribute setters: students must not modify these methods.

- double calculatePostfix(): used to convert postfix string *inputString* to result

of expression.

- String [] stringTokenizer(): used to separate token by token using spaces, the returned result is an array of characters.

- double getResultOfExpression(): returns the result of the expression, the *main()* function will call this method for getting results. (students cannot edit this method)

- private boolean isNum (String c): check whether the input string c is a number or not.

Students implement the methods so that the **inputString** is the postfix input string and **resultOfExpression** is the result of the expression. During the calculation process if you meet the case of 0/0, you can stop the program and return 0 as the result of the current expression.

The result of the requirement is written by the main function to "output2.txt" file.

**V. Instructions for submission**

- Students only submit the *InfixToPostfixSolve.java* and *CalculatePostfixSolve.java* files, **do not submit any other files and absolutly do not edit these 2 filenames.**

**- Students put the 2 files into the *StudentID\_FullName*** folder and compress with the **.zip** format before submitting to Sakai in *DSA1\_Assignment01*.

- Any case of wrong submission (wrong folder name, not put the files in the folder, redundant files, ...), students' work will get **0 points**.

**VI. Evaluation and regulation**

- The code will be automatically marked by testcases (input and output files are in the format as the attachments) so students are responsible failing the Instructions for Submission or modifying the cannot-be-edited methods, which make the code cannot be compiled.

- The number of testcases used for marking is 20, with 10 testcases *input1.txt* for REQUIREMENT 1 and 10 testcases *input2.txt* for REQUIREMENT 2.

- For any requirement, if the code make compilation error, **0 point** will be given to that requirement.

**- In case of copied code from internet or your friends or letting your friend copy your code will be given 0 point.** In those cases, students will be asked to explain themselves at the office.

**- Deadline for submission: 23h00 on April 19, 2019.**

**-- THE END --**