National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B. Tech.(CSE)
CS2092D Programming Laboratory
Assignment #7

Submission deadline (on or before):

- 31.10.2022, 11:55 PM

Policies for Submission and Evaluation:

- You must submit your assignment in the Eduserver course page, on or before the submission deadline.
- Ensure that your programs will compile and execute without errors using gcc compiler.
- During the evaluation, failure to execute programs without compilation errors may lead to zero marks for that evaluation.
- Detection of ANY malpractice related to the lab course can lead to awarding an F grade in the course.

Naming Conventions for Submission

- The source codes must be named as

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example: $ASSG1\_BxxyyyyCS\_LAXMAN\_1.c$).

  Create separate files for each question and place all of them in a single folder named

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>`

  (Example: $ASSG1\_BxxyyyyCS\_LAXMAN$). Then Zip the **folder**.

- Submit the single ZIP (.zip) file you have created(do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip`

  (Example: $ASSG1\_BxxyyyyCS\_LAXMAN.zip$). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

  If you do not conform to the above naming conventions, your submission might not be recognized by our automated tools, and hence will lead to a score of 0 marks for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

- Violation of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: https://minerva.nitc.ac.in/?q=node/650.

General Instructions

- Programs should be written in C language and compiled using gcc compiler. **Submit the solutions to the questions through the submission link in Eduserver**.
- Check your programs with sufficiently large values of inputs with in the range as specified in the question.
- Global and/or static variables should not be used in your program.
- You shouldn't change the function prototype specified in the question.

QUESTIONS

1. Write a program for converting a given infix expression to postfix form using a STACK. Write a function InfixToPostfix(e) which converts the given infix expression e to postfix expression and returns the postfix expression.
   In the infix expression, operands are represented by characters a to z (without any extra spaces). Only the binary operators $+, -, *, /$ need to be considered. There can be parenthesized sub-expressions (including the entire expression). Follow the precedence and associativity of the operators defined in C programming language and this would apply to the other questions as well.

   Input format:
   - The first line of the input contains an integer $n \in [1, 10^4]$, the number of infix expressions to be converted.
   - The next n lines contains infix expressions.

   Output format:
   - The output is the corresponding postfix expressions printed in n lines.

   Sample Input:
   ```
   3
   a+b
   a+b*(c-d)
   a*(c+d)/(e+f*g-k*p)
   ```

   Sample Output:
   ```
   ab+
   abcd-*+
   acd+*efg*+kp*-/
   ```

2. An **Expression Tree** is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand. Write a menu driven program to construct an Expression Tree T for a given postfix expression e and perform the tree traversal operations (Inorder, Preorder and Postorder) on T. Each node x of T is an object with an attribute *data*, which is either an operator or an operand of the expression and two pointer attributes: $left$ and $right$ pointing to the left and right children of x respectively. An attribute $T.root$ points to the root node of the tree T. Your program must contain the following functions:

   - Main() - repeatedly reads a character 'e', 'i', 'p', 's', or 't' from the terminal and calls the sub-functions appropriately until character 't' is entered.
   - Construct-Tree(e) that takes as input a postfix expression e, converts it into an expression tree T and returns a pointer to the root of T.
   - Inorder(T) that takes as input an expression tree T and prints the data in the nodes of T in inorder.
   - Preorder(T) that takes as input an expression tree T and prints the data in the nodes of T in preorder.
   - Postorder(T) that takes as input an expression tree T and prints the data in the nodes of T in postorder.

   Input format:

   - Each line starts with a character from *'e', 'i', 'p', 't', 's'* .
   - Character *'e'* is followed by a postfix expression, which is a combination of characters (operands) from a to z and binary operators $+, -, *$, and $/$. Create an Expression Tree for the given postfix expression using the function Construct-Tree().
   - Character *'s'* is to perform postorder traversal of expression tree T by calling the function Postorder().

- Character *'i'* is to perform inorder traversal of expression tree T by calling the function INORDER().
- Character *'p'* is to perform preorder traversal of expression tree T by calling the function PREORDER().
- Character *'t'* is to terminate the program.
  Note: Option 'e' will always be coming first before 'i', 'p' or 's'.

Output Format:

- The output (if any) of each command should be printed on a separate line.
- For option *'i'*, print the expression obtained from inorder traversal of the expression tree.
- For option *'p'*, print the expression obtained from preorder traversal of the expression tree.
- For option *'s'*, print the expression obtained from postorder traversal of the expression tree.

Sample Input:

```
e ab+
i
p
e ab+cd-*
i
p
s
t
```

Sample Output:

```
a+b
+ab
a+b*c-d
*+ab-cd
ab+cd-*
```

3. Write a program to evaluate a given postfix expression. Write a function EVALUATEPOSTFIX(e), that evaluates the given postfix expression e and returns the result of evaluation.

Input format:

- The first line of the input contains an integer $n \in [1, 10^4]$, the number of postfix expressions to be converted.
- The next n lines contains postfix expressions, with integer operands (in the range [0, 100]) and binary operators $+, -, *, /$. Operands/operators are separated by a space.

Output format:

- The output is the resultant integer value of expression after evaluation, printed one by one in n lines.

Sample Input:

```
3
6 4 -
9 2 1 * + 9 - 4 *
10 7 3 2 + * - 7 2 * -
```

Sample Output:

```
2
8
-39
```