National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B.Tech.(CSE)
CS2092D Programming Laboratory
Assignment #6 - Part 2

Submission deadline (on or before):

- 10.11.2021, 9:00 AM

Policies for Submission and Evaluation:

- You must submit both the parts of your assignment (Assignment 6 - Part 1 and Part 2) together 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

- Submit a single ZIP (.zip) file (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.
- The source codes must be named as

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

  (For example: *ASSG1_BxxyyyyCS_LAXMAN_1.c*). 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 here.

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.

QUESTIONS

3. 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.

   Input format:
   The input is an infix expression, with operands represented by characters a to z. Only the binary operators $+, -, *, /$ need to be considered. There can be parenthesised subexpressions (including the entire expression). The associativity of the binary operators is from left to right and this would apply to the other questions as well.

   Output format:
   The output is a postfix expression.

   Sample Input 1:
   a+b*(c-d)

   Sample Output 1:
   abcd-*+

   Sample Input 2:
   a*(c+d)/(e+f*g-k*p)

   Sample Output 2:
   acd+*efg*+kp*-/

4. Write a menu driven program to construct an $ExpressionTree$ T(it is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand) for a given postfix expression e and perform the tree traversal operations (Inorder, Preorder and Post order) 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 contains 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 $/$.
   - Character *'i'* is to perform inorder traversal of expression tree T by calling the function Inorder(T).
   - Character *'p'* is to perform preorder traversal of expression tree T by calling the function Preorder(T).

- Character *'s'* is to perform postorder traversal of expression tree T by calling the function POSTORDER(T).
- Character *'t'* is to terminate the program.

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+cd-*
i
p
s
t
```

Sample Output

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

5. 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 input consists of a postfix expression (represented as a characer string), with integer operands and binary operators $+, -, *, /$. Operands/operators are separated by a space. Assume that only positive operands are present in the expression. Evaluate the postfix expression and print the result.

   Output format: The output is the value of expression after evaluation.

   Sample Input 1:

   ```
   9 2 1 * + 9 - 4 *
   ```

   Sample Output 1:

   ```
   8
   ```

   Sample Input: 2

   ```
   10 7 3 2 + * - 7 2 * +
   ```

   Sample Output 2:

   ```
   -11
   ```