Assignment-1
CS F214
Logic in Computer Science

Total Weightage = 8% **Total Marks = 24**

General Instructions:

- The assignment is divided into four days. You need to create separate functions for each day.
- Function signatures and the main function will be given to you. You just need to write the function definitions and upload it.
- Your code will be tested with the given main function for each day.
- **You should strictly follow the filename nomenclature for each day.**
- For each day, the main file is named as "main_No.c" and the header file is named as "dayNo.h". (eg: "main_1.c" and "day1.h"). You will create a new program file for each day and save it as "dayNo.c" (e.g. day1.c).
- If you fail to submit any task before the deadline, you can continue with the next task but you won't be given marks for the task you didn't complete.
- Include appropriate header files in your code whenever necessary.
- You should comment your code properly.
- **Assignment should be done sitting in Systems Lab or Data Science (I014 and I015) labs only.**
- The connective symbols you will use for this assignment is as follows.
  1. ~ for negation
  2. V for OR
  3. ^ for AND
  4. > for implication.

Other Important Instructions:

- Please work as a team. There should be **only one submission per team**.
- **Do not share your code with other team members. Copied codes will be awarded zero marks for the entire assignment. Expecting all of you to be honest.**

Definition of Propositional Logic Formula-

<statement> ::= p | (¬p) |(~(<statement>)) | (<statement> Λ <statement>) | (<statement> V <statement>) |
(<statement> > <statement>)

Day 1 (14th Oct 2019)

Marks = 4

Learn how to initialize a stack to store characters. Implement the follow operations in stack through separate functions:

1. push(x)
2. pop()
3. isEmpty()
4. isFull()
5. top()

Your code should comply with the main function.

The filename of the header file will be "day1.h". The "day1.h" code is given to you. Please use these function signatures only.

Create a separate file - "day1.c" where you will write all the function definitions.

The "main_1.c" is the main file to test your code. Use it to test your code.
Make sure in all files you have put all the group members' name along with group ID.

Input -

MAX, the first line contains the maximum number of elements in the stack.
N, the second line contains the number of operations user wants to execute.
The next N lines contain various choice of operations ranging from 1 to 5 as mentioned above.
For push operation, your code should take a character as input.

Output -

Your function should generate appropriate outputs for isEmpty(), isFull() and top() functions.

Sample Test Case -

Input -

```
10
8
1 a
1 b
1 c
2
3
4
5
1 d
```

Output -

```
false
false
b
```

Day 2 (15th Oct 2019)

Marks = 6

The task of Day 2 is to be able to convert a well-defined fully parenthesized propositional logic formula given in infix notation to convert it into post-fix notation.

Infix notation means that the operator is written between two operand for example p ^ q.
Post fix notation means the operator is written after the two operands for example above formula can be written as pq^.

Now that you have learnt how to use a stack, implement the following functionality using stack.

Write a function to convert an infix expression to a postfix expression. The function signature is given in the header file “day2.h” :

You should name your program code as “day2.c” where you will complete the function definition.

The main_2.c is the file to test your code.

Input -
A string containing a well formed infix expression.
Output -
Postfix expression.

Sample Test Case -
Input -
((pVq)^r)
Output -
pqVr^

Algorithm for converting from infix to postfix notation.
1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. If the scanned character is an operator, push it into the stack.
4. If the scanned character is ‘(‘, push it into the stack.
5. If the scanned character is an ‘)’ , output the stack and pop until a ‘(‘ is found.
6. Repeat the steps until scanning is finished.
7. Pop out all the leftover elements in the stack.
8. You will have to include “day1.h” to use stack operations.

Day 3 (16th Oct 2019)

Marks = 6 marks

By now you have setup appropriate data structure of stack which you need to modify today to achieve the following task:

- Write a function to convert the postfix expression into a rooted binary parse tree.
- Write a function to traverse the parse tree to output the infix expression by inorder traversal of the parse tree.
- You will have to modify stack functions so that it works with a different data type.

The functions you need to implement today:

- The header file("modifiedDay1.h"): A structure to store each node of the tree has been declared. You need to create a stack of node pointers.
- The header file("day3.h") : Description is given in the header file comments.

Complete the function definitions. You should name the new files as "modifiedDay1.c" and "day3.c" only.

Input -
String containing a postfix expression.

Output -
The infix expression.

Sample Test Case -

<u>Input -</u>

pqVrp^>

<u>Output -</u>

((pVq)>(r^p))

<u>Algorithm sketch for converting from post-fix-expression to parse tree-</u>

1. Initialize a stack to store pointers to a node. (You will need to modify the functions used in day1).
2. Scan the postfix expression from left to right.
3. If the scanned character is an operand, push it into the stack.
4. If character is operator, pop two values from stack make them its child and push current node again.
5. At the end only element of stack will be root of expression tree.
6. You will have to include "modifiedDay1.h" to use stack operations and to use the structure.

<u>Algorithm sketch for inorder traversal of parse tree-</u>

1. After getting the root of the tree, first print the left subtree. Then print the node value and then print the right subtree by calling the function recursively.

Day 4 (8 marks)

Today is the last task day of assignment.
Objective of today’s task is to write a function to evaluate a parse tree given a particular valuation.

Today you will use yesterday’s code files to make the parse tree from the postfix expression.
Then write the code in “day4.c” to implement the function declared in “day4.h”. The objective of this function is to return the value of the entire input propositional logic formula given a Model or Valuation.

Finally use the given driver code “main_4.c” to test your code.

In your final submission all the required header files, their respective definition files and one single driver code “main_4.c” should be present. Make sure all these codes compile and give correct output.

Input -

First line contains the string containing the postfix expression. The next line, N denotes the number of propositional variables in the expression. The next N lines contain a character followed by 0 or 1 - denoting the propositional variable in the postfix expression and it’s value respectively.

Output -

Infix expression by inorder traversal of the tree constructed.
Print the evaluation of the given expression.

Sample Test Case-

<u>Input -</u>

```
pqVpr^>
3
p 0
q 1
r 1
```

<u>Output -</u>

```
((pVq)>(p^r))
0
```

<u>Algorithm sketch for finding the truth value of the propositional logic formula given a valuation</u>

1. Construct a parse tree using the postfix expression.
2. Use a recursive approach evaluate it.
3. If root->character is an operand, return operand’s value(0 or 1).
4. If root->character is an operator, solve both (or only right child in case of ~) children of the tree and return value of
   (LeftChildResult) operator (RightChildResult).
5. To evaluate the Left and Right child, use recursion.
6. You will need to include “modifiedDay1.h” to use the structure.