**ICS PROJECT REPORT**

**BATCH: 2016-2017**

**CLASS: FIRST YEAR**

**SEMESTER: 2nd SEMESTER**

**SECTION: A**

**COURSE: BSCS-308 PHYSICS-II**

**SUBMITTED TO: MISS YUSRA**

**DATED: November 3, 2017**

**GROUP MEMBERS:**

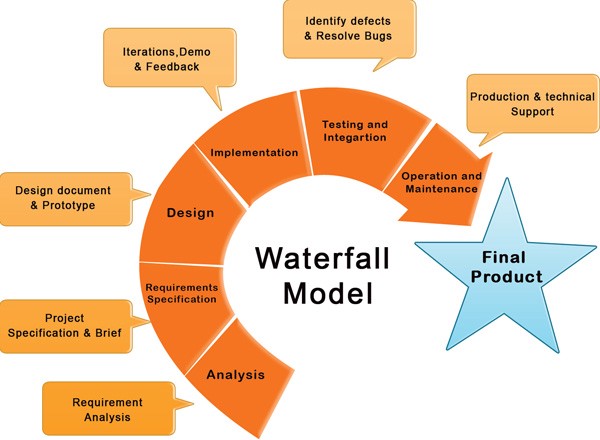
1. **AHMER AYAZ**
2. **TAHA IMTIAZ**
3. **MOHAMMAD ASHAR SARWAR**

**INTRODUCTION**

There are three forms of expressions which are prefix, infix and postfix notation.

The project will evaluate infix and postfix expressions then convert one to the other.

**THE WATER FALL MODEL**



**FEATURES**

The source code of the Applications of stack conversions uses the following features of the C-Language:

* **Operators**

An *operator* is a symbol that directs the computer to perform certain mathematical or logical manipulations and is usually used to manipulate data and variables. The prominently used operators in our program are *Arithmetic Operators*, *Relational Operators*, *Logical Operators*,and *Assignment Operators*.

* **Decision Control Statements**

*Decision Control Statements* enable us to specify the flow of program control i.e. the order in which the instructions in a program must be executed. They make it possible to make decisions. The used types of the decision control statements in our program are *if statement*, *if-else statement*, and *Nested if-else*.

* **Switch Statement**

The *switch statement* allows a user to select from multiple choices based on a set of fixed values for a given expression.

* **Loops**

A *loop* is a sequence of instructions that is continually repeated until a certain condition is reached. The *for-loop* is used in our program.

* **Arrays**

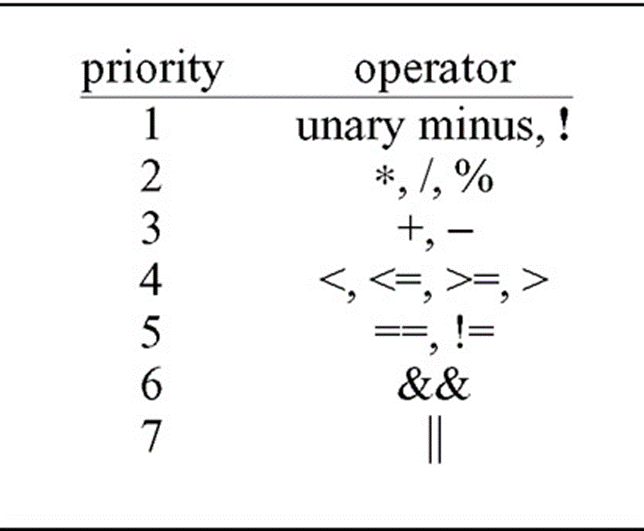
*Array* is a collection of data that holds fixed number of values belonging to the same data type. *1D-Array* is used in our program.

* **Functions**

*Function* is a small piece of code or module from which a large program is constructed. The structure of function consists of three parts i.e. *Function Declaration*, *Function Definition*, and *Function Call*.

* **Priority of operators**

Within any pair of parentheses the operators have the highest priority

****

**REPORT BASED ON THE WATERFALL MODEL:**

**1. ANALYSIS:**

The basic purpose of writing this program is to explain and demonstrate one of the very important Applications of stack i.e. Convert one type of expressions into another form or vice-versa. Expressions can be represented in prefix, postfix or infix notations and conversion from one form to another may be accomplished using a **stack**. Many compilers use a **stack** for parsing the syntax of expressions, program blocks etc. before translating into low level code.

**2. REQUIREMENT SPECIFICATION**

We need the infix, postfix and prefix expressions and vice versa for the following conversions

**3. ALGORITHM**

**i)Infix to Postfix Conversion**

Let Q be any infix expression and we have to convert it to postfix expression P. For this the following procedure will be followed.

1. Push left parenthesis onto STACK and add right parenthesis at the end of Q.

2. Scan Q from left to right and repeat step 3 to 6 for each element of Q until the STACK is empty.

3. If an operand is encountered add it to P.

4. If a left parenthesis is encountered push it onto the STACK.

5. If an operator is encountered, then

Repeatedly pop from STACK and add to P each operator

Which has same precedence as or higher precedence than the operator

Encountered. Push the encountered operator onto the STACK.

6. If a right parenthesis is encountered, then

Repeatedly pop from the STACK and add to P each operator

Until a left parenthesis is encountered.

Remove the left parenthesis; do not add it to P.

7. Exit

**ii) Infix to Prefix Conversion Algorithm**

Step 1. Push “)” onto STACK, and add “(“to end of the A

Step 2. Scan A from right to left and repeat step 3 to 6 for each element of A until the STACK is empty

Step 3. If an operand is encountered add it to B

Step 4. If a right parenthesis is encountered push it onto STACK

Step 5. If an operator is encountered then:

a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has same or higher precedence than the operator.

b. Add operator to STACK

Step 6. If left parenthesis is encountered then

a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a left parenthesis is encountered)

b. Remove the left parenthesis and reverse the expression.

Step 7. Exit

**iii)Algorithm of Postfix to Infix**

1. Scan the expression from left to right.

2. If the symbol is an operand

3. Push it onto the stack.

4. Otherwise,

5. The symbol is an operator.

6. If there are fewer than 2 values on the stack

7. Show Error /\* input not sufficient values in the expression \*/

8. Else

9. Pop the top 2 values from the stack.

10. Put the operator, with the values as arguments and form a string.

11. Encapsulate the resulted string with parenthesis.

12. Push the resulted string back to stack.

13. If there is only one value in the stack

14. That value in the stack is the desired infix string.

15. If there are more values in the stack

16. Show Error /\* the user input has too many values \*/

**iv)Algorithm of Prefix to Infix**

1. Scan the expression from right to left.

2. If the symbol is an operand

3. Push it onto the stack.

4. Otherwise,

5. The symbol is an operator.

6. If there are fewer than 2 values on the stack

7. Show Error /\* input not sufficient values in the expression \*/

8. Else

9. Pop the top 2 values from the stack.

10. Put the operator, with the values as arguments and form a string.

11. Encapsulate the resulted string with parenthesis.

12. Push the resulted string back to stack.

13. If there is only one value in the stack

14. That value in the stack is the desired infix string.

15. If there are more values in the stack

16. Show Error /\* the user input has too many values \*/

**v)Algorithm of Prefix to Postfix**

**1. Scan each character of the input from right to left.**

**2. If the character is operand then push it onto the stack.**

**3. If the character if an operator, then pop two operands from stack and concatenate them as (operand1, operand2, operator).**

**4. Push this result onto the stack.**

**5. After parsing the complete input, if still any elements exit in the stack, pop them out and concatenate.**

**5. The result obtained is the postfix expression of given input prefix expression.**

**vi)Algorithm of Postfix to Prefix**

1. Scan the Postfix Expression from Left To Right.
2. If the character is an Operand, then push it on to the Stack.
3. If the character is an Operator, then Pop Operand 1 and Operand 2 and concatenate them and Push the result on the Stack.
4. Repeat the above steps until the Postfix Expression is scanned completely.
5. To get the Prefix Expression, Pop the remaining elements of the Stack.

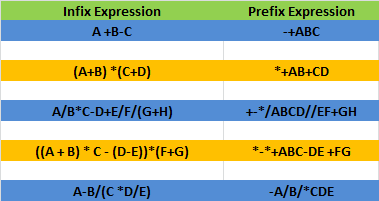
**5. SOFTWARE IMPLEMENTATION AND TESTING**

**a)Converting between Infix and Postfix**

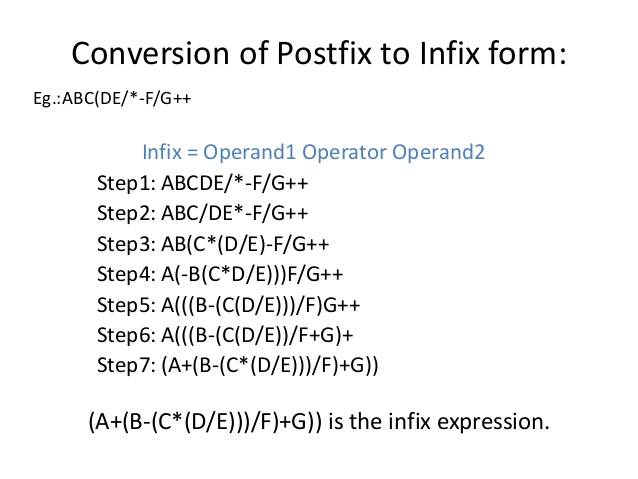
**Infix**

|  |  |
| --- | --- |
| **Infix**  **Infix** | **Postfix** |
| ( (A \* B) + (C / D) ) | ( (A B \*) (C D /) +) |
| ((A \* (B + C) ) / D) | ( (A (B C +) \*) D /) |
| (A \* (B + (C / D) ) ) | (A (B (C D /) +) \*) |

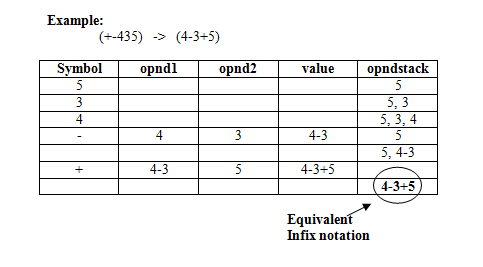
**b) Converting between Infix and Prefix**



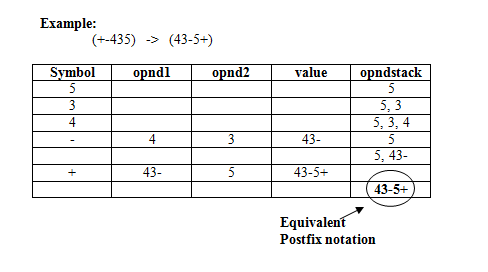
**c) Converting between Postfix to Infix**



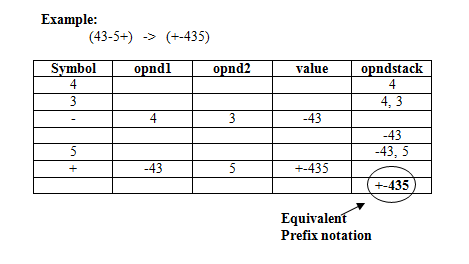
**d) Converting between prefix to infix**



**e) Converting between Prefix to Postfix**



**f) Converting between Postfix to Prefix**



***Conclusion***

1. We have understand the theory and basic concepts first. After researching on websites and reading the textbook, we totally understand the concepts and know how to code the program.
2. We are working on our program which can convert infix form and postfix form of expressions one another.
3. After the finishing of the program, we will complete a final report at the end of this semester.