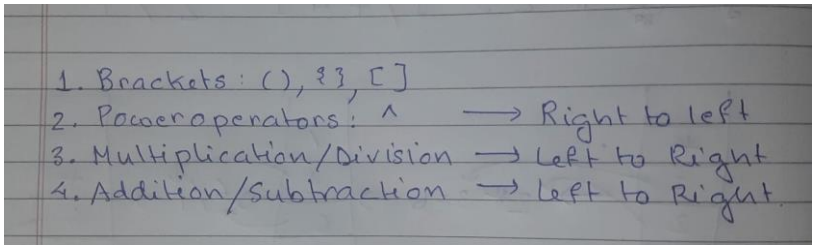
Unit 2: Data Structures Notes – 2

When dealing with or thinking about the inner workings of a computer, we realise that computer is a machine that was basically built to perform calculations and solve complex equations with ease. Obviously, it does more now but if we think the basic parts of computing an expression for example an arithmetic one, we tend to realise that computers cannot understand the expressions that we write or have learned throughout the years. For us writing complex nested equations can be solved by using parenthesis wherever required but in computers this can be inefficient.

So, to reduce the computational workload different expressions where introduced. Some of them are:

(The expressions and their meanings remain the same just the way of writing them changes)

1. Infix Expressions: This is the usual way we write expressions by hand. In these expressions we place the operator between the two operands it operates on. Infix expressions can also include parentheses to indicate the order of operations.
   For eg, 5 + 3, a * b etc.
   For evaluating these expressions, we have to follow some rules and precedence. The precedence levels (decreasing order) and related associativity are:



2. Prefix Expressions: As the name suggests we keep the operator before the operands. So A + B would be written as + AB. These expressions are also known as Polish Notation. Not much is mentioned in your syllabus about it so we won't go at it in depth.

3. Postfix Expressions: Postfix expressions, also known as reverse Polish notation, where we place the operator after the operands. For eg, if we have A + B we write it as AB+.

Converting Infix to Postfix:

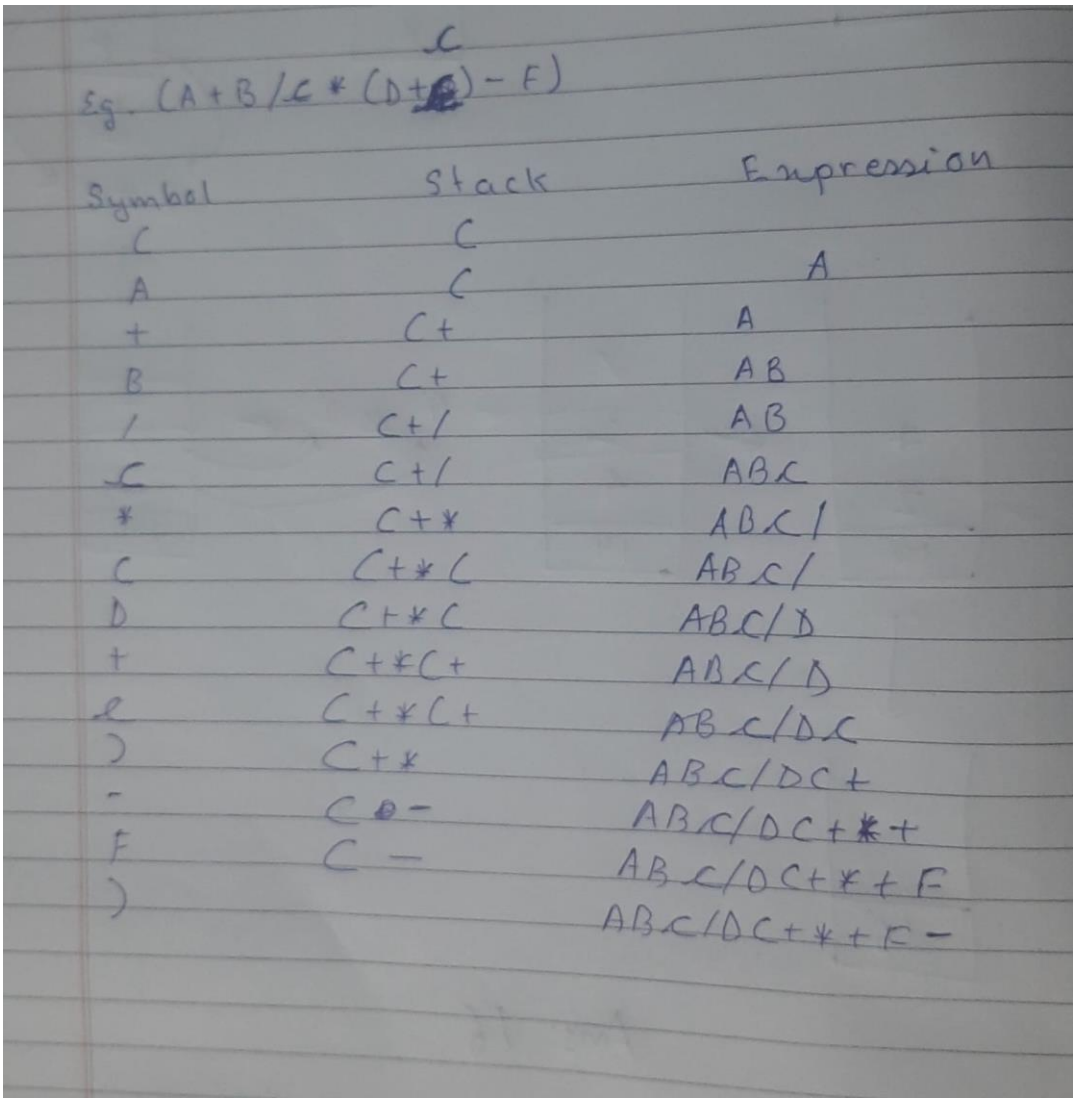
To do this, we need to follow certain rules.

1. We push and pop only operators into and from the stack. Any operand coming up during traversing the expression goes directly in the expression queue.
2. We will follow the same precedence level used in the infix operations.

3. No operator having a lower precedence level can be on top of a higher or same level precedence operator in the stack. To push that operator into the stack the other operators must be popped out and put into the new expression queue. They can be separated by another symbol but cannot stay together.

4. If we encounter an opening bracket "(" we directly push it into the stack but If we encounter a closing bracket ")" we pop elements in the stack until we reach a "(" and then cancel out the brackets.

Let us see this in working with an example:



Evaluating Post Fix Expressions: This is a very simple process. We will scan the expression from left to right and keep pushing the operands into the stack until we find an operator. Once we find the operator, we pop two values from the stack. The top value which is popped goes to the right side of the operator and the 2nd popped value will go to the left of the operator (Think in terms of normal maths expressions). Then after we find the result, we push the result into the stack and keep on scanning and continue to do so until all the operators and operands are finished in the expression.

Let us see an example: