BITS PILANI, DUBAI CAMPUS
DUBAI INTERNATIONAL ACADEMIC CITY, DUBAI

FIRST SEMESTER 2023 – 2024
COURSE: CSF301 (Principles of Programming Languages)
COMPONENT: Tutorial Sheet 2 **DATE:** 25th September 2023

Q. 1 Convert the given prefix expression to infix notation.

i. Input : Prefix : *+AB-CD

Ans:

Output : Infix : ((A+B)*(C-D))

Algorithm for Prefix to Infix:

1. Read the Prefix expression in reverse order (from right to left)
2. If the symbol is an operand, then push it onto the Stack
3. If the symbol is an operator, then pop two operands from the Stack
4. Create a string by concatenating the two operands and the operator between them.
5. **string = (operand1 + operator + operand2)**
6. And push the resultant string back to Stack
7. Repeat the above steps until end of Prefix expression.

Stack contents at each step:

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
Push D	Push C	Pop C,D	Push B	Push A	Pop A,B	Pop (A-B), (C-D)
		Add - operator in between			Add + operator in between	Add * operator in between them
		Push the String			Push the String	Push the String
				A		
	C		B	B	(A+B)	
D	D	(C-D)	(C-D)	(C-D)	(C-D)	((A+B)*(C-D))

ii. **Input : Prefix : *-A/BC-/AKL**

Output : Infix : ((A-(B/C))*((A/K)-L))

Q. 2 Convert the given postfix expression to infix notation.

i. Input : abc++

Ans.

Output : (a + (b + c))

Algorithm for Postfix to Infix:

1. Read the Postfix expression from left to right
2. If the symbol is an operand, then push it onto the Stack
3. If the symbol is an operator, then pop two operands from the Stack
4. Create a string by concatenating the two operands and the operator between them.
5. **string = (operand2 + operator + operand1)**
6. And push the resultant string back to Stack
7. Repeat the above steps until end of Postfix expression.

Step 1	Step 2	Step 3	Step 4	Step 5
Push A	Push B	Push C	Pop C,B	Pop (C+B), A
			Add + operator in between	Add + operator in between
			Push the String	Push the String
		C		
	B	B	(B+C)	
A	A	A	A	(A+(B+C))

ii. Input : ab*c+
Output : ((a*b)+c)

Q. 3. Convert infix to postfix

i. ((A + B) - C * (D / E)) + F

Ans:

A B + C D E / * - F +

Algorithm

1. Scan the infix expression from left to right.

2. If the scanned character is an operand, output it.

3. Else,

…..3.1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.

…..3.2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is an '(', push it to the stack.

5. If the scanned character is an ')', pop the stack and and output it until a '(' is encountered, and discard both the parenthesis.

6. Repeat steps 2-6 until infix expression is scanned.

7. Print the output

8. Pop and output from the stack until it is not empty.

Q. 4 Show a rightmost and leftmost derivation of "abba" in the grammar below.

<S> -> a<A> | b<A> | <A><B>

<A> -> a<A> | b | <S>

<B> -> <B>a | b

Ans. Right most derivation:

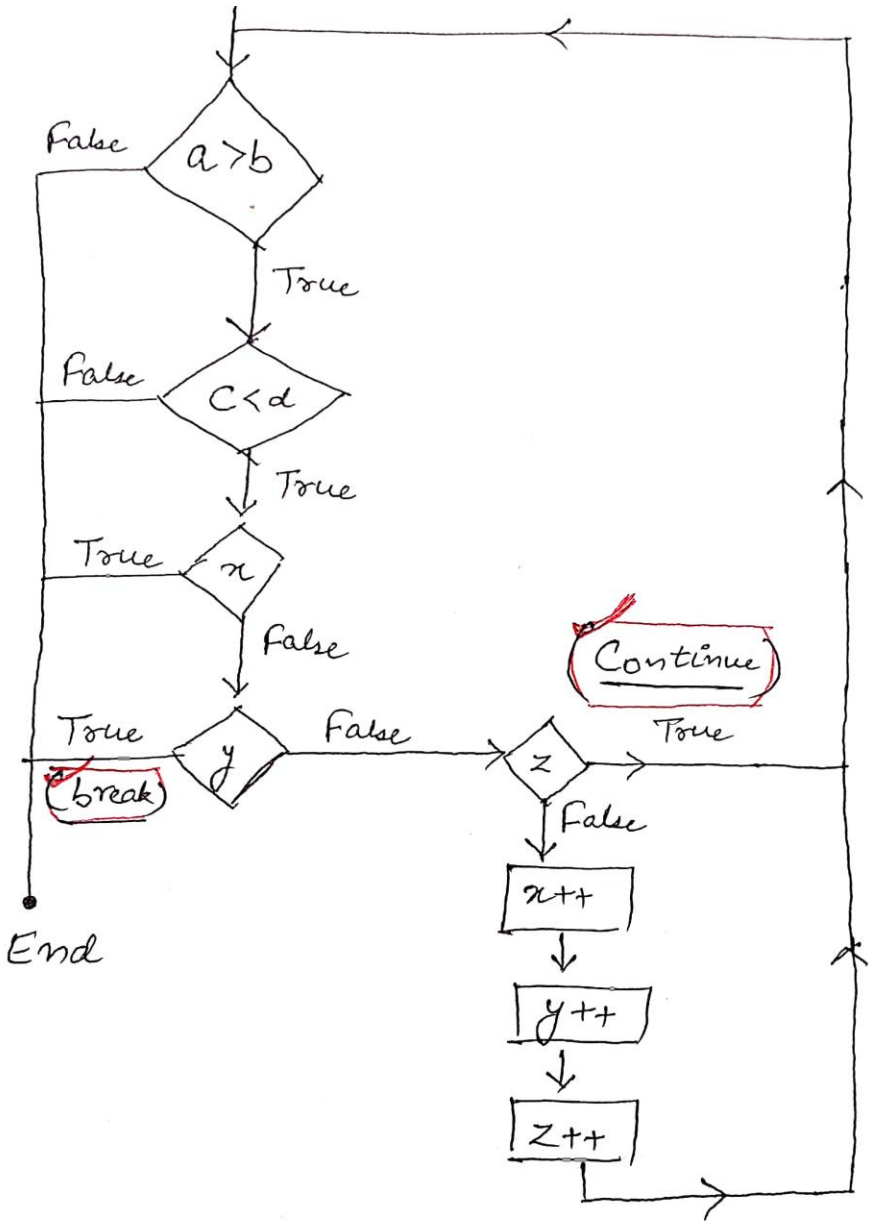
S => <A><B>
=> <A><B>a
=> <A>ba
=> a<A>ba
=> abba

Left most derivation:
S => <A><B>
=> a<A><B>

=> ab<B>
=> ab<B>a
=> abba

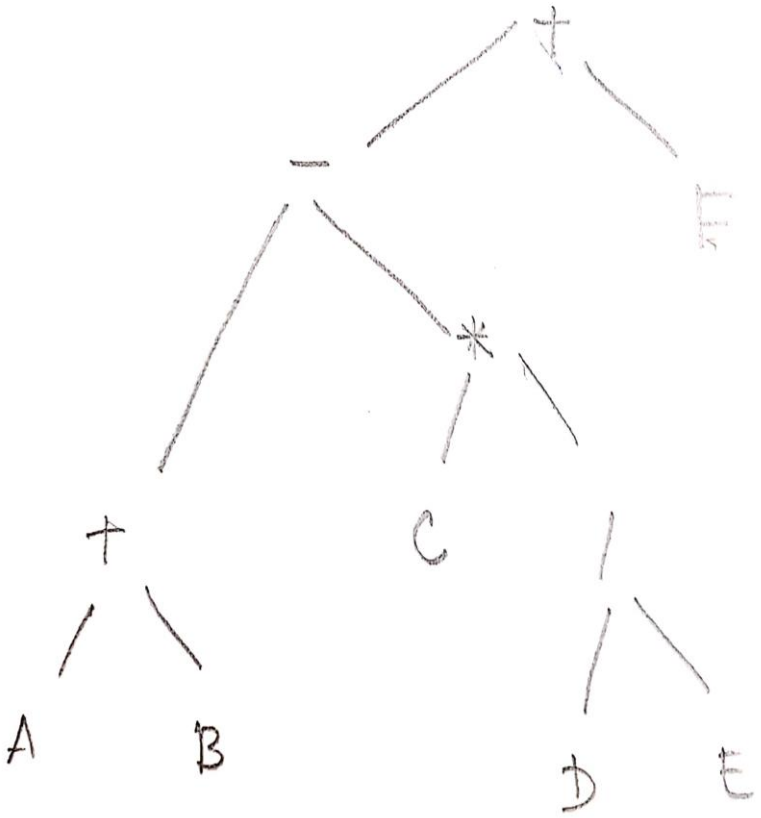
Q. 5 For the C code shown below draw the corresponding flow chart/graph.

```
int a,b,c,d,x,y,z;
while (a > b && c < d ) {
   if ( x || y )break;
   if (z) continue;
   x++; y++; z++;
}
```



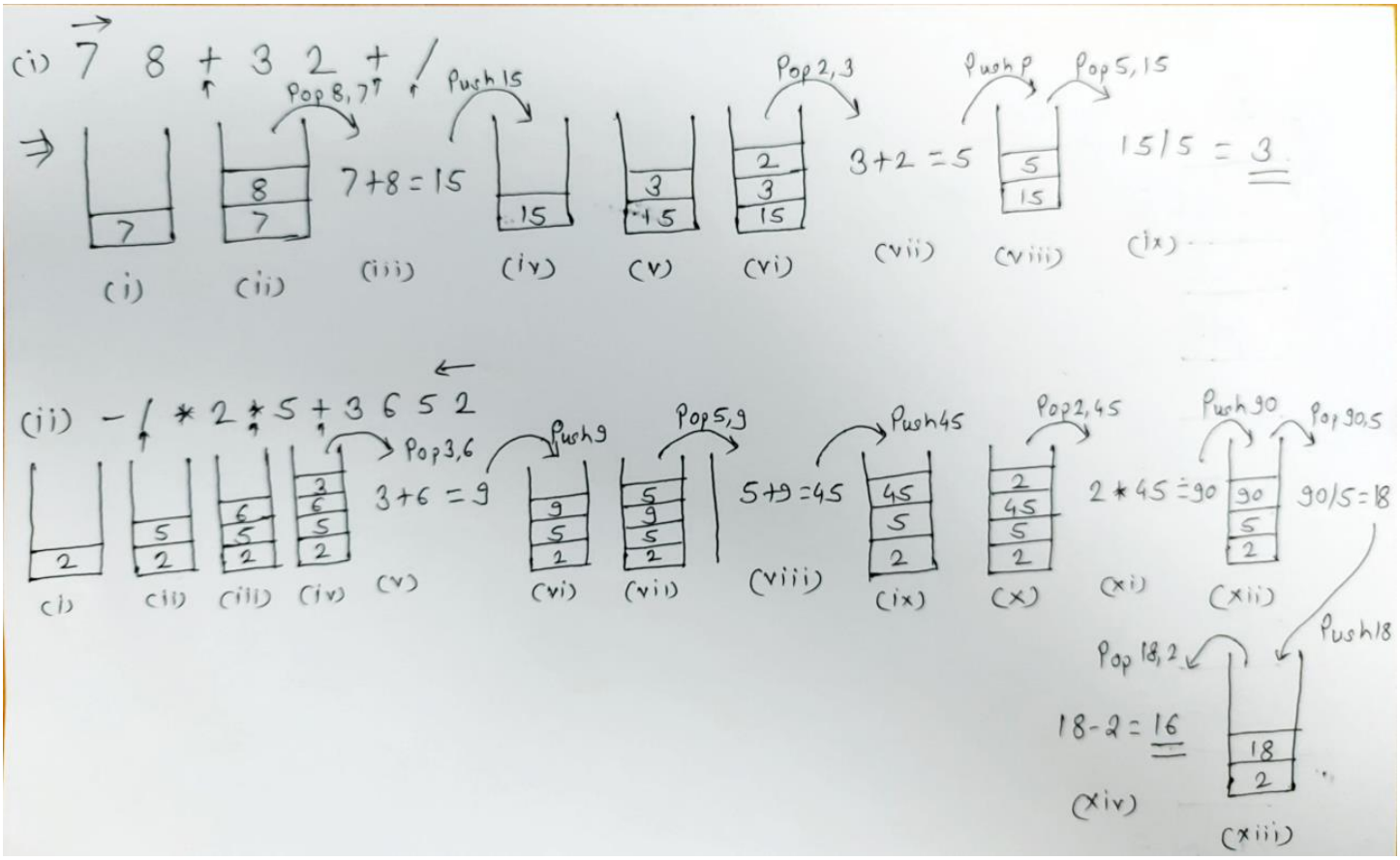
Q.6 Draw the abstract syntax tree for the following expression

((A + B) - C * (D / E)) + F



Q 7. Evaluate the following expressions

i. Postfix: 7 8 + 3 2 + /
ii. Prefix: - / * 2 * 5 + 3 6 5 2



Q8. Separate the token from following C program

```
int main()
{
    int x, y, total;
    x = 10, y = 20;
    total = x + y;
    printf (total);
}
```

Answer:

<int, keyword> <main, keyword> <(, punctuation> <),punctuation>

<{,punctuation>

<int, keyword> <x,id> <y,id> <total,id> <;,punctuation>

<x,id> <=,op> <10, number> <,,punctation> <y,id> <=,op> <20, number> <;,punctuation>

<total,id> <=,op> <x,id> <+,op> <y,id> <;,punctuation>

<printf, keyword> <(, punctuation> <total, id> <),punctuation> <;,punctuation>

<{,punctuation>