210411100085_Modul3_StackInfix

April 12, 2022

1 Konsep

Ekspresi Aritmatik Infix

Ekspresi ini merupakan ekspresi aritmatik yang sering kita jumpai. Dalam konsep Infix operator berada di antara bilangan. Contohnya 8+2, di ekspresi ini juga dikenalkan dengan operator precedence yang digunakan untuk menentukan ekspresi mana yang harus diselesaikan terlebih dahulu. Contoh 8-9x7 berarti 9x7 diselesaikan terlebih dahulu karena mempunyai derajat lebih tinggi. Untuk urutan pengerjaan dalam ekspresi infix dimulai dari x atau / kemudian - atau +, dan jika tidak ada ekspresi di dalam kurung ( ) berarti urutannya sesuai operator precedence.

Ekspresi Aritmatik Prefix

Ekspresi aritmatik prefix merupakan ekspresi yang mempunyai pola operatornya terletak di depan bilangan. Contohnya +89 bila dalam infix menjadi 8+9, untuk bagaimana cara konversinya dari infix ke prefix seperti berikut. Contoh ekspresi 7x9+9/3 maka cara pengerjaanya 7x9 diubah menjadi x79 kemudian pada cari derajat operator yang tertinggi yaitu ditemukan / jadi menjadi /93 setelah itu kemudian pada + di letakkan pada depan x79 menjadi +x79 dan hasilnya +x79/93.

Ekspresi Aritmatik Postfix

Ekspresi aritmatik postfix merupakan kebalikan dari ekspresi prefix yang mempunyai pola operatornya terletak di belakang bilangan. Contohnya 98x, untuk konversinya dari infix ke postfix caranya sebagai berikut. Pada ekspresi 8x2-1x(9-1) pengerjaannya cari operator derajat tertinggi terlebih dahulu yaitu ( ) berarti 91- kemudian pada 8x2 menjadi 82x dan 1 9 1-x kemudian keduanya digabungkan menjadi 82x191-x-.

2 Implementasi

2.1 Konversi Infix ke Postfix

[41]:
```
from modules import stack as st

def printToken(numstr,stro,stOp) :
    print("""
-> Read Token %d : %s
Stack = %s """ % (numstr,stro,stOp) )

def convertPostfix(strOp) :
    stOperator = st.stack()
```

```
    operator    = {"+":0,"-":0,"*":1,"/":1}
    kurung      = ("(",")")
    strRes      = ""
    num         = 1

    for sto in strOp :
        if sto == kurung[0] :
            st.push(stOperator,sto)

        elif sto == kurung[1] :
            while not(st.isEmpty(stOperator)) and st.peek(stOperator) !=␣
 ↪kurung[0] :
                strRes+=st.pop(stOperator)
            st.pop(stOperator)

        elif sto not in operator.keys() and sto not in kurung :
            strRes+=sto

        elif sto in operator.keys() :
            while not(st.isEmpty(stOperator)) and st.peek(stOperator) !=␣
 ↪kurung[0] and operator[st.peek(stOperator)] >= operator[sto] :
                strRes+=st.pop(stOperator)
            st.push(stOperator,sto)

        printToken(num,sto,stOperator)
        num+=1


    while not(st.isEmpty(stOperator)) :
        strRes+=st.pop(stOperator)

    return strRes


print("==> ((a+b)*(c-d))/f : ",convertPostfix("((a+b)*(c-d))/
 ↪f"),end="\n---------------\n\n")
print("==> a+b*c/d-f+g*h   : ",convertPostfix("a+b*c/
 ↪d-f+g*h"),end="\n---------------\n\n")
print("==> (A*(B+C))-D :␣
 ↪",convertPostfix("(A*(B+C))-D"),end="\n---------------\n\n")
```

```
-> Read Token 1 : (
Stack = ['(']

-> Read Token 2 : (
Stack = ['(', '(']
```

```
-> Read Token 3 : a
Stack = ['(', '(']

-> Read Token 4 : +
Stack = ['(', '(', '+']

-> Read Token 5 : b
Stack = ['(', '(', '+']

-> Read Token 6 : )
Stack = ['(']

-> Read Token 7 : *
Stack = ['(', '*']

-> Read Token 8 : (
Stack = ['(', '*', '(']

-> Read Token 9 : c
Stack = ['(', '*', '(']

-> Read Token 10 : -
Stack = ['(', '*', '(', '-']

-> Read Token 11 : d
Stack = ['(', '*', '(', '-']

-> Read Token 12 : )
Stack = ['(', '*']

-> Read Token 13 : )
Stack = []

-> Read Token 14 : /
Stack = ['/']

-> Read Token 15 : f
Stack = ['/']
==> ((a+b)*(c-d))/f :  ab+cd-*f/
---------------


-> Read Token 1 : a
Stack = []

-> Read Token 2 : +
Stack = ['+']
```

```
-> Read Token 3 : b
Stack = ['+']

-> Read Token 4 : *
Stack = ['+', '*']

-> Read Token 5 : c
Stack = ['+', '*']

-> Read Token 6 : /
Stack = ['+', '/']

-> Read Token 7 : d
Stack = ['+', '/']

-> Read Token 8 : -
Stack = ['-']

-> Read Token 9 : f
Stack = ['-']

-> Read Token 10 : +
Stack = ['+']

-> Read Token 11 : g
Stack = ['+']

-> Read Token 12 : *
Stack = ['+', '*']

-> Read Token 13 : h
Stack = ['+', '*']
==> a+b*c/d-f+g*h   :  abc*d/+f-gh*+
---------------


-> Read Token 1 : (
Stack = ['(']

-> Read Token 2 : A
Stack = ['(']

-> Read Token 3 : *
Stack = ['(', '*']

-> Read Token 4 : (
Stack = ['(', '*', '(']
```

```
-> Read Token 5 : B
Stack = ['(', '*', '(']

-> Read Token 6 : +
Stack = ['(', '*', '(', '+']

-> Read Token 7 : C
Stack = ['(', '*', '(', '+']

-> Read Token 8 : )
Stack = ['(', '*']

-> Read Token 9 : )
Stack = []

-> Read Token 10 : -
Stack = ['-']

-> Read Token 11 : D
Stack = ['-']
==> (A*(B+C))-D :  ABC+*D-
---------------
```

2.2 Evaluasi Postfix

[40]:
```python
from modules import stack as st

def printMath(num,op1,op2,opr,res) :
    print("[%d]: %s %s %s = %s " % (num,op1,opr,op2,res))


def evaluatePostfix(strPostfix):
    stOp       = st.stack()
    operator   = ("+","-","*","/")
    lstPostfix = strPostfix.split(" ")
    numSg      = 1

    for stpf in lstPostfix :

        if stpf not in operator:
            st.push(stOp,int(stpf))

        else:
            operand2 = st.pop(stOp)
            operand1 = st.pop(stOp)
```

```
            if stpf == '+' :
                hasil = operand1 + operand2

            elif stpf == '-' :
                hasil = operand1 - operand2

            elif stpf == '*' :
                hasil = operand1 * operand2

            else :
                hasil = operand1 / operand2

            printMath(numSg,operand1,operand2,stpf,hasil)
            numSg+=1

            st.push(stOp, float(hasil))

    return (st.pop(stOp))



postFix = "2 6 +"
print("\nHasil dari %s adalah = %s " %␣
 ↪(postFix,evaluatePostfix(postFix)),end="\n\n\n")

postFix1 = "10 2 3 + * 4 -"
print("\nHasil dari %s adalah = %s " %␣
 ↪(postFix1,evaluatePostfix(postFix1)),end="\n\n\n")

postFix2 = "2 3 4 * 10 / + 11 - 9 2 * +"
print("\nHasil dari %s adalah = %s " %␣
 ↪(postFix2,evaluatePostfix(postFix2)),end="\n\n\n")
```

```
[1]: 2 + 6 = 8

Hasil dari 2 6 + adalah = 8.0


[1]: 2 + 3 = 5
[2]: 10 * 5.0 = 50.0
[3]: 50.0 - 4 = 46.0

Hasil dari 10 2 3 + * 4 - adalah = 46.0


[1]: 3 * 4 = 12
```

```
[2]: 12.0 / 10 = 1.2
[3]: 2 + 1.2 = 3.2
[4]: 3.2 - 11 = -7.8
[5]: 9 * 2 = 18
[6]: -7.8 + 18.0 = 10.2

Hasil dari 2 3 4 * 10 / + 11 - 9 2 * + adalah = 10.2
```