# EXPERIMENT - VIII
# OPERATOR PRECEDENCE PARSER

September 20, 2020

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

## AIM

To develop an operator precedence parser for a given language.

# THEORY

## Operator Precedence Parsing

A grammar that is generated to define the mathematical operators is called operator grammar with some restrictions on grammar. An operator precedence grammar is a context-free grammar that has the property that no production has either an empty right-hand side (null productions) or two adjacent non-terminals in its right-hand side.
An operator precedence parser is a one of the bottom-up parser that interprets an operator precedence grammar. This parser is only used for operator grammars. Ambiguous grammars are not allowed in case of any parser except operator precedence parser. There are two methods for determining what precedence relations should hold between a pair of terminals:

- Use the conventional associativity and precedence of operator.
- The second method of selecting operator-precedence relations is first to construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse trees.

This parser relies on the following three precedence relations:

- a < b This means a yields precedence to b.
- a > b This means a takes precedence over b.
- a = b This means a has precedence as b.

# ALGORITHM

**Algorithm 1** Algorithm for Operator Precedence Parser

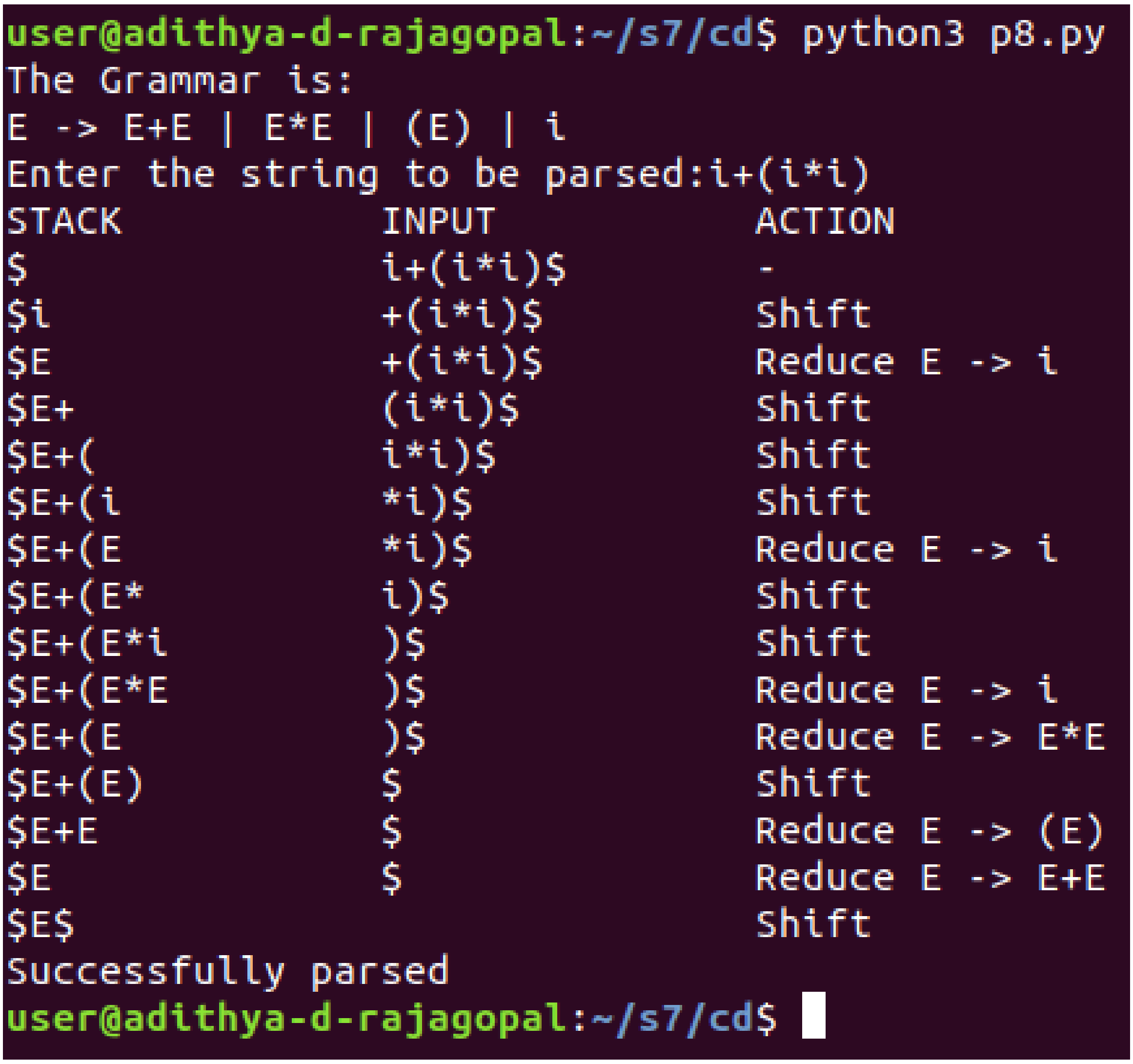
```
Start
Read the string to be parsed ($w$$).
Set ip to point to the first symbol of the input string w$.
Initialize flag=0
while flag=0 do
    Let b be the top stack symbol.
    Let a be the input symbol pointed to by ip.
    if a=$ and b=$ then
        flag=1.
    else
        if a>b or a=b then
            Stack.push(a)
            Advance ip to the next input symbol.
        else if a<b then
            c=Stack.pop()
            while c<b or c=b do
                c=Stack.pop()
            end while
        else
            flag=-1.
        end if
    end if
end while
if flag=1 then
    print "SUCCESS".
else
    print "ERROR".
end if
Stop
```

## SOURCE CODE

```
def printStack():
        global Stack
        for i in Stack:
                print(i,end="")
        print("\t\t",end="")

def reduce():
        global Stack
        global handle
        global prevhandle
        if Stack[-1]=="i":
                Stack.pop()
                Stack.append("E")
                prevhandle=handle[0]
                return True
        if len(Stack)>=3:
                if Stack[-1]=="E" and Stack[-3]=="E":
                        op=Stack.pop()
                        op=Stack.pop()
                        if op=="+":
                                prevhandle=handle[1]
                        elif op=="*":
                                prevhandle=handle[2]
                        return True
                elif Stack[-1]==")" and Stack[-2]=="E" and Stack[-3]=="(":
                        op=Stack.pop()
                        op=Stack.pop(-2)
                        prevhandle=handle[3]
                        return True
        return False

def Operator_Precedence_Parser(str):
        global Stack
```

```
    global handle
    global prevhandle
    T=[’+’,’*’,’i’,’(’,’)’,’$’]
    precedence=[]
    precedence.append([’>’,’<’,’<’,’<’,’>’,’>’])
    precedence.append([’>’,’>’,’<’,’<’,’>’,’>’])
    precedence.append([’>’,’>’,’e’,’e’,’>’,’>’])
    precedence.append([’<’,’<’,’<’,’<’,’>’,’e’])
    precedence.append([’>’,’>’,’e’,’e’,’>’,’>’])
    precedence.append([’<’,’<’,’<’,’<’,’<’,’>’])
    Stack=[’$’]
    ip=0
    handle=[’i’,’E+E’,’E*E’,’(E)’]
    print("STACK\t\tINPUT\t\tACTION")
    print("$\t\t"+str+"\t-")
    while ip<len(str):
        Stack.append(str[ip])
        ip=ip+1
        printStack()
        print(str[ip:],end="\t\t")
        print("Shift")
        if ip==len(str):
            break
        tp=T.index(Stack[-1])
        curr=T.index(str[ip])
        if precedence[tp][curr]==’>’:
            while(reduce()):
                printStack()
                print(str[ip:],end="\t\t")
                print("Reduce E -> "+prevhandle)
    if Stack[0]==’$’ and Stack[1]==’E’ and Stack[2]==’$’:
            return True
    return False

global Stack
global handle
global prevhandle
print("The Grammar is:")
print("E -> E+E | E*E | (E) | i")
s=input("Enter the string to be parsed:")
w=s+'$'
if(Operator_Precedence_Parser(w)):
        print("Successfully parsed")
else:
        print("Error in parsing")
```

## SAMPLE OUTPUT



```
user@adithya-d-rajagopal:~/s7/cd$ python3 p8.py
The Grammar is:
E -> E+E | E*E | (E) | i
Enter the string to be parsed:i+(i*i)
STACK               INPUT               ACTION
$                   i+(i*i)$            -
$i                  +(i*i)$             Shift
$E                  +(i*i)$             Reduce E -> i
$E+                 (i*i)$              Shift
$E+(                i*i)$               Shift
$E+(i               *i)$                Shift
$E+(E               *i)$                Reduce E -> i
$E+(E*              i)$                 Shift
$E+(E*i             )$                  Shift
$E+(E*E             )$                  Reduce E -> i
$E+(E               )$                  Reduce E -> E*E
$E+(E)              $                   Shift
$E+E                $                   Reduce E -> (E)
$E                  $                   Reduce E -> E+E
$E$                                     Shift
Successfully parsed
user@adithya-d-rajagopal:~/s7/cd$
```

# RESULT

An operator precedence parser has been developed using Python and the outputs have been verified.