**The code**

class i\_to\_p\_conversion:

precedence={'+':4,'-':3,'\*':2,'/':2}

def \_\_init\_\_(self):

self.stack=[]

self.size=-1

def top(self):

if self.size==-1:

return False

else:

return self.stack[self.size]

def push(self,value):

self.stack.append(value)

self.size+=1

def pop(self):

if self.size==-1:

return 0

else:

self.size-=1

return self.stack.pop()

def convert(self,expr):

postfix=""

for i in expr:

if(len(expr)%2==0):

print("please enter a proper infix expression")

return False

elif(i.isalnum()):

postfix +=i

elif(i in '+-\*/'):

while(len(self.stack) and self.precedence[i]<=self.precedence[self.top()]):

postfix+=self.pop()

self.push(i)

while len(self.stack):

postfix+=self.pop()

return postfix

f = open("three.txt","r")

for string in f:

s=i\_to\_p\_conversion()

postfix = s.convert(string)

print(postfix)

**How to run the code**

1)The infix strings that we want to parse need to be written inside the the file three.txt

2)Then we need to run the command python3 three.py on the command line

**Methodology**

In this system we use four functions - push, pop, top, convert and a constructor of class i\_to\_p\_conversion.

**Constructor/\_\_init\_\_()** initializes two variables stack and size where stack is an array where all the operators will be stored temporarily and variable size is the size of the array which is -1 initially

def \_\_init\_\_(self):

self.stack=[]

self.size=-1

**Top()** checks if the top of the stack. If the stack is empty it returns false otherwise it returns the element on the top.

def top(self):

if self.size==-1:

return false

else:

return self.stack[self.size]

**Push(value)** - adds values to the stack. And increments the size of the array to +1

|  |
| --- |
| def push(self,value):  self.stack.append(value)  self.size+=1 |

**Pop()** pops the value at the top of the stack and decrements the value of the size variable by 1.

def pop(self):

if self.size==-1:

return 0

else:

self.size-=1

return self.stack.pop()

**Convert()**

It is the main function in this program and returns the final postfix value.

It takes the string as parameter expr.

def convert(self,expr):

A variable named postfix is declared where the final string is added as the loop increments.

postfix=""

A for loop is run for every character in the string expr.

for i in expr:

If the length of the string is even the for loop closes and prints “incorrect infix expression” because number of operators is always number of operands plus one.

if(len(expr)%2==0):

print("Incorrect infix expr")

return False

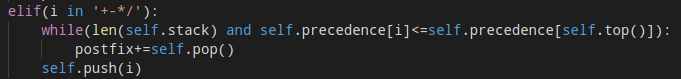
The function isalnum() checks if the character in the string is a number or alphabet. If it detects a number it adds it into the string postfix

elif(i.isalnum()):

postfix +=i

This is the most important part of the code as in this part the arrange of the operators in the postfix string is determined. If the precedence of the of the operator at hand is less than or equal to the precedence of the the operator at the top of the top element of the stack is popped and added to the postfix string. The operator at hand is appended into the top of the stack. Otherwise the operator at hand is directly pushed on top of the stack without any popping.

For example in the case of the string “2-3\*4+a-b+c\*d/e” if we had encountered \* while our stack is filled with a - we would have need to pop the minus first before pushing \* into the stack. But when we encounter the + we dont need to pop \*. Instead we push + on top of \* into the stack.



After the forloop is over and all the characters in the infix has been parsed we pop all the operators

while len(self.stack):

postfix+=self.pop()

The convert function returns the postfix string when called.

return postfix

**Main**

File is opened and each string is converted to their corresponding postfix and printed

f = open("three.txt","r")

for string in f:

s=i\_to\_p\_conversion()

postfix = s.convert(string)

print(postfix)

Sources

<https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/>