به نام خدا

پروژه ی ماشین حساب

استاد فاطمی

دانشجو:سارا رئوفی

در ابتدای برنامه کلاس مربوط به استک برای پیاده سازی آن را با استفاده از لیست نوشتم.که شامل توابع (push, pop isEmpty , peak) است.

```python
class stack:
    # Stack function implementation
    def __init__(self):
        self.stack=[]
        self.top=None

    def push(self,item):
        self.top=item
        self.stack.append(item)

    def pop(self):
        return self.stack.pop()

    def isEmpty(self):
        return (self.stack==[])

    def peak(self):
        if not self.isEmpty():
            temp=self.pop()
            self.push(temp)
            return temp
        else:
            return 0
```

سپس تابع روش محاسبه عبارت(جمع، ضرب، تقسیم، توان، منها) را نوشتم. و برای تقسیم خطای تقسیم بر صفر را در نظر گرفتم. بعد از این تابع چک کردن پرانتز ها را نوشتم که ببیند آیا با هم تطایق دارند یا نه و محل قرارگیری آن ها درست است یا نه.

```python
#The function related to checking the matching of parentheses
    def check(self,p1,p2):
        parentheses=[("(",")"),("[","]"),("{","}")]
        if p1 in open and p2 in close:
            if p1==parentheses[0][0] and p2==parentheses[0][1]:
                return True
            if p1==parentheses[1][0] and p2==parentheses[1][1]:
                return True
            if p1==parentheses[2][0] and p2==parentheses[2][1]:
                return True
            else:
                print("Error because of the matching of parentheses")
        else:
            print("Error because of the order of parentheses")
```

```python
# The function related to the calculation of operators
    def calculator(self,op,n1,n2):
        if op=="+":
            return float(n2)+float(n1)
        elif op=="-":
            return float(n2)-float(n1)
        elif op=="*":
            return float(n2)*float(n1)
        elif op=="/":
         if float(n2)==0:
            error="Error because of number/0"
            return error
         else:
            return float(n1)/float(n2)
        elif op=="^":
            return float(n1)**float(n2)
```

بعد از آن تابع تبدیل infix به postfix را نوشتم .برای این تبدیل یک لیست در نظر گرفتم که عبارت تبدیل شده را در آن بریزد. روش تبدیل هم به این صورت است که از ابتدای عبارت اگر به پرانتز و یا عملوند برخوردیم آن ها را در یک پشته ریخته و اگر به عملگر رسیدیم در لیست میریزد. اعضای پشته هم هرگاه که به پرانتز بسته رسیدیم یا با توجه به ترتیب عملوندها یکی یکی حذف شده و به لیست اضافه میشوند.

```
#def for convert infix expression to postfix
    def infix_to_postfix(self,num):
        postfix=[]
        flag=0
        i=0
        operator_n=stack()
        while i<len(num) and flag==0:
            if num[i] in open:
               st_operator.push(num[i])
               if not operator_n.isEmpty():
                operator_n.push(num[i])
               i=i+1
            elif num[i] in close:
                if st_operator.isEmpty():
                    flag=1
                    print( "Error because of the order of parentheses" )
                else:
                    p2=num[i]
                    while not st_operator.isEmpty() and st_operator.peak() not in open:
                     postfix.append(st_operator.pop())
                    p1=st_operator.pop()
                    c=self.check(p1,p2)
                    if c!=True:
                        flag=1
                    if not operator_n.isEmpty():
                        if operator_n.peak()!="+" or operator_n.peak()!="-":
                            operator_n.pop()
                        if operator_n.peak()=="-" or operator_n.peak()=="+":
                           postfix.append(operator_n.pop()+"1*")
                    i=i+1
```

```
        elif num[i] in op1:
          if num[i-1] not in open and num[i+1] not in open :
            if st_operator.top in op2:
                while not st_operator.isEmpty() and flag==0:
                   if st_operator.peak() in close:
                      p2=st_operator.pop()
                   while not st_operator.isEmpty() and st_operator.peak() not in open:
                      postfix.append(st_operator.pop())
                   if not st_operator.isEmpty() and st_operator.peak() in open :
                        p1=st_operator.peak()
                        c=self.check(p1,p2)
                        if c!=True:
                          flag=1
                        else:
                          st_operator.pop()
                   else:
                      break
                st_operator.push(num[i])
                i=i+1
            elif st_operator.top in op3:
                while not st_operator.isEmpty() and flag==0:
                   if not st_operator.isEmpty() and st_operator.peak() in close:
                      p2=st_operator.pop()
                   while not st_operator.isEmpty() and st_operator.peak() not in open:
                      postfix.append(st_operator.pop())
                   if  not st_operator.isEmpty() and st_operator.peak() in open :
                        p1=st_operator.peak()
                        c=self.check(p1,p2)
                        if c!=True:
                          flag=1
                        else:
                          st_operator.pop()
```

سپس تابع محاسبه عبارت postfix به دست آمده را نوشتم تا حاصل عبارتمان به دست آید. در اینجا اعضای لیست را از اول یکی یکی به پشته اضافه میکنیم و اگر به عملوند رسیدیم، عملوند را بر روی دو عدد قبل در پشته اجرا میکنم و شپس حاصل را دوباره در پشته میریزیم و...

```
def calculate_postfix(self,postfix):
    i=0
    flag=0
    while i<len(postfix) and flag==0:
        while i<len(postfix) and postfix[i] not in operator:
            st_number.push(postfix[i])
            i=i+1
        if postfix[i] in operator:
            op=postfix[i]
            if not st_number.isEmpty():
              n2=st_number.pop()
            else:
                error="Error because Existence of an operator without an operand"
                break
            if not st_number.isEmpty():
              n1=st_number.pop()
            else:
                error="Error because Existence of an operator without an operand"
                break
            cal=self.calculator(op,n1,n2)
            if cal=="Error because of number/0":
                flag=1
                error=cal
            i=i+1
            if i<len(postfix) and postfix[i]=="+1*":
                cal=cal*1
                i=i+1
            elif i<len(postfix) and postfix[i]=="-1*":
                cal=cal*(-1)
                i=i+1
            st_number.push(cal)
```

در اخر هم درواقع در تابع main یا قسمت اصلی برنامه عبارت را از کاربر دریافت کرده و با استفاده از توابع بالا و کلاس پشته حاصل را به دست آوردم.

```
open={"(","[","{"}
close={"}",")","]"}
operator=("+","/","-","*","^")
op1={"+","-"}
op2={"*","/"}
op3={"^"}
st_operator=stack()
st_number=stack()
number=input("Enter expresion:")
stack_num=stack()
postfix=stack_num.infix_to_postfix(number)
if postfix!=False:
  print(stack_num.calculate_postfix(postfix))
```