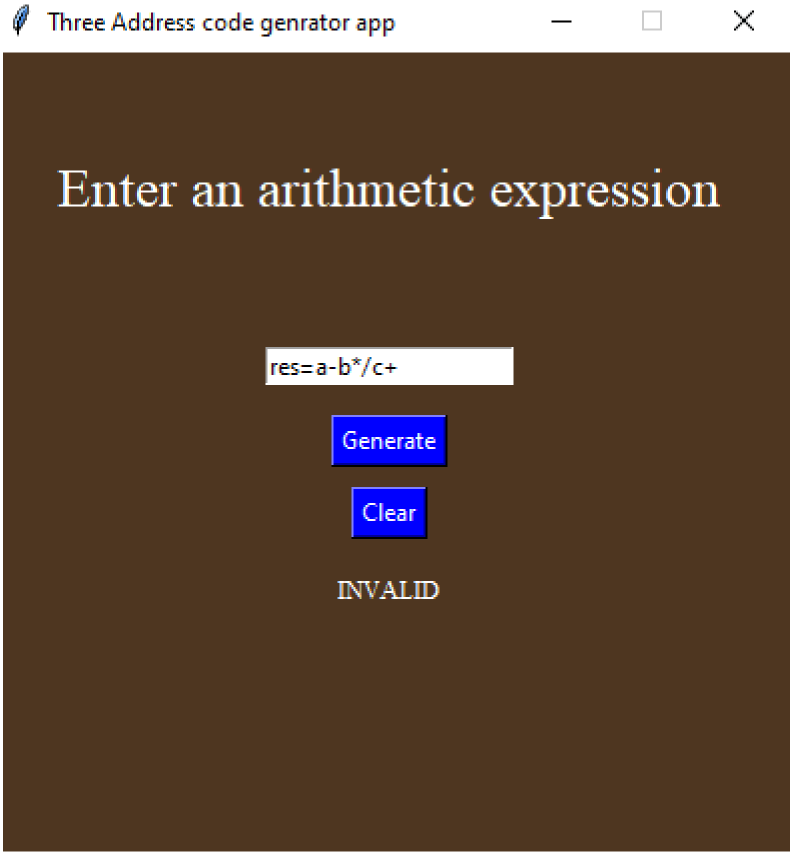
```
from tkinter import *

class Stack:

    def __init__(self,max_lim):
        self.__lis_of_eles=[]
        self.__max_lim=max_lim
        self.__top=-1

    def get_stack_eles(self):
        return self.__lis_of_eles


    def is_full(self):
        if(self.__top==self.__max_lim-1):
            return True
        else:
            return False

    def is_empty(self):
        if(self.__top==-1):
            return True
        else:
            return False

    def push(self,E):
        if(self.is_full()):
            print("Stack is full")
            return
        else:
            self.__top+=1
            self.__lis_of_eles.insert(self.__top,E)
            #print("Pushed ele: ",self.__lis_of_eles[self.__top])

    def pop(self):
        if(self.is_empty()):
            print("Stack is empty")
            return
        else:
            del_ele=self.__lis_of_eles.pop(self.__top)
            #print("Deleted element: ",del_ele)
            self.__top-=1
            return del_ele


    def fetch_top(self):
        return self.__top

    def display(self):
        i=self.__top
        for j in range(i,-1,-1):
            print(self.__lis_of_eles[j])



class Equation:

    #list of arithmetic operators according to precedence
    operators1=["*","/"]
    operators2=["-","+"]

    #only alphabet-operands:
    operands_lower_case=[chr(i) for i in range(ord("a"),ord("z")+1)]
    operands_upper_case=[chr(i) for i in range(ord("A"),ord("Z")+1)]

    def __init__(self,given):

        self.__c=0
        lis=given.split("=")  #lis[0]-location to store answer,lis[1]-location to store expression
        self.__res=lis[0]
        self.__expr=lis[1]
        self.__last_symbol=lis[1][-1]

    def get_c(self):
        return self.__c
```

```
  def get_result(self):
      return self.__res

  def get_expression(self):
      return self.__expr

  def get_last_symbol(self):
      return self.__last_symbol


  def is_operand(self,ele):
      if((ele not in Equation.operands_lower_case) and (ele not in Equation.operands_upper_case)):
          return False
      return True

  def is_operator(self,op):
      if((op in Equation.operators1) or (op in Equation.operators2)):
          return True
      else:
          return False


  def validate(self):

      f=self.__expr[0]
      n=len(self.__expr)

      if((f not in Equation.operands_lower_case and f not in Equation.operands_upper_case) or
        (self.__last_symbol not in Equation.operands_lower_case and self.__last_symbol not in Equation.operands_upper_case)):

          return False

      s=Stack(n)

      for ele in self.__expr:
          if(s.is_empty()):
              if(self.is_operand(ele)):
                  s.push(ele)
                  self.__c+=1
              else:

                  return False
          else:
              top_pos=s.fetch_top()
              stack_elements=s.get_stack_eles()

              if(self.is_operand(stack_elements[top_pos])):
                  if(self.is_operator(ele)):
                      s.push(ele)

                  else:
                      return False

              elif(self.is_operator(stack_elements[top_pos])):
                  if(self.is_operand(ele)):
                      s.push(ele)
                      self.__c+=1
                  else:

                      return False
              else:

                  return False


      return True


class ThreeAddressCodeGenerator(Equation):

  def __init__(self,eqn):
      super().__init__(eqn)
```

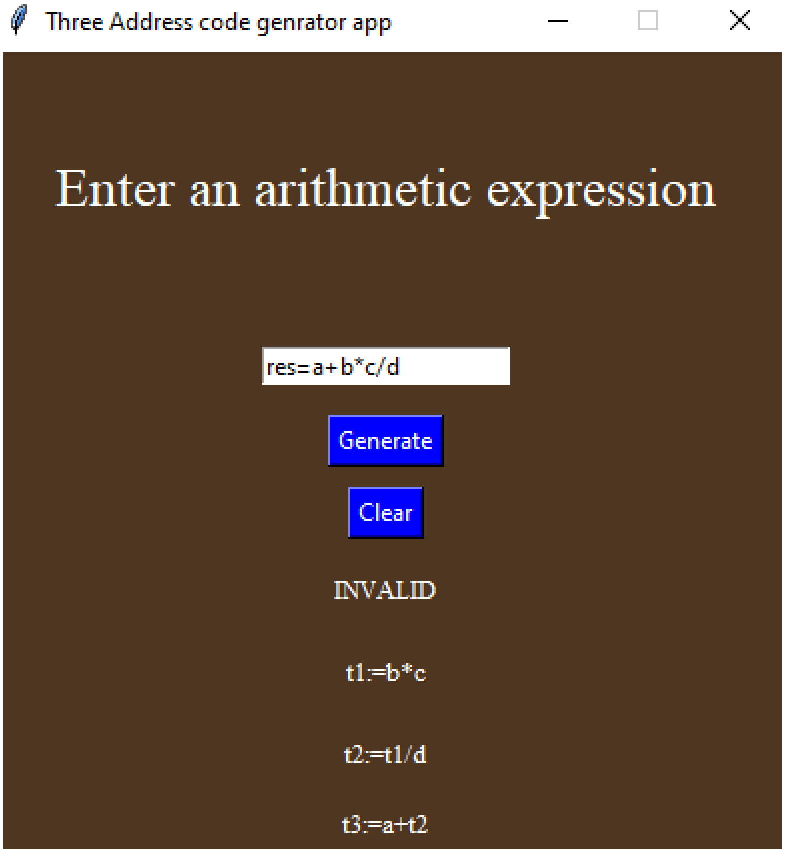
```
def generate(self):

    if(super().validate()==False):
        global label1
        label1=Label(root,text="INVALID",font=("times",10),fg="white",bg="#4e3620")
        label1.pack(pady=10)
        labels_list.append(label1)
        return

    exp=super().get_expression()
    n=len(exp)

    sob=Stack(n)


    num=1
    i=0
    track=0


    while(i<n):

        e=exp[i]

        if(super().is_operator(e)):

            if e not in super().operators1:
                sob.push(e)
                i+=1

            else:
                track+=1
                top_ele=sob.pop()


                ch=str("t"+str(num))

                sob.push(ch)

                #print(ch)

                if(track==1):
                    global label2
                    label2=Label(root,text=ch+":="+top_ele+e+exp[i+1],font=("times",10),fg="white",bg="#4e3620")
                    label2.pack(pady=10)
                    labels_list.append(label2)

                else:
                    global label3
                    label3=Label(root,text=ch+":="+prev_val+e+exp[i+1],font=("times",10),fg="white",bg="#4e3620")
                    label3.pack(pady=10)
                    labels_list.append(label3)



                prev_val=ch

                num+=1
                i+=2

        else:
            sob.push(e)
            i+=1



    track1=0
    tracker=sob.fetch_top()

    while(tracker>-1):

        track1+=1

        operand2=sob.pop()
```

```
            operator=sob.pop()
            operand1=sob.pop()

            ch=str("t"+str(num))
            sob.push(ch)
            #print(ch)

            if(track==0 and track1==1):
                prev_val=ch

            if(track>=1):
                prev_val=operand2


            if(track1>1):
                global label4
                label4=Label(root,text=ch+":="+operand1+operator+prev_val,font=("times",10),fg="white",bg="#4e3620")
                label4.pack(pady=10)
                labels_list.append(label4)



            else:
                global label5
                label5=Label(root,text=ch+":="+operand1+operator+operand2,font=("times",10),fg="white",bg="#4e3620")
                label5.pack(pady=10)
                labels_list.append(label5)
            num+=1
            tracker-=3
            prev_val=ch
root=Tk()
root.title("Three Address code genrator app")
root.geometry("400x400")

global labels_list
labels_list=[]

label=Label(root,text="Enter an arithmetic expression",font=("times",20),fg="white",bg="#4e3620")
label.pack(pady=50)

name=StringVar()

my_box=Entry(root,textvariable=name)
my_box.pack(pady=10)
my_box.focus()



def assign_given():
    global user_ip
    user_ip=my_box.get()
    T=ThreeAddressCodeGenerator(user_ip)
    T.generate()

def clear():
    name.set("")
    for label in labels_list:
        label.destroy()



button1=Button(root,text="Generate",command=assign_given,fg="white",bg="blue")
button1.pack(pady=5)

button2=Button(root,text="Clear",command=clear,fg="white",bg="blue")
button2.pack(pady=5)


root.configure(bg="#4e3620")
root.resizable(False,False)
root.mainloop()
```

Output


Three Address code genrator app
Enter an arithmetic expression
Generate
Clear

```
res=a-b*/c+
INVALID
```


Three Address code genrator app
Enter an arithmetic expression
Generate
Clear

```
res=a+b*c/d
INVALID
t1:=b*c
t2:=t1/d
t3:=a+t2
```