Exercise:6.1.1

OPERATORS=set(['+','-','\*','/','(',')','^'])

PRIORITY={'+':1,'-':1,'\*':2,'/':2,'^':3}

def infix\_to\_postfix(expression):

stack=[]

output=''

for ch in expression:

if ch not in OPERATORS:

output+=ch

elif ch=='(':

stack.append('(')

elif ch==')':

while stack and stack[-1]!='(':

output+=stack.pop()

stack.pop()

else:

while stack and stack[-1]!='(' and PRIORITRY<=PRIORITY[stack[-1]]:

output+=stack.pop()

stack.append(ch)

while stack:

output+=stack.pop()

return output

expression=input('Enter infix expression')

print('infix expression:',expression)

print('postfix expression:',infix\_to\_postfix(expression))

output:

Enter infix expression:a+b(a-b\*a)^a/b

infix expression: a+b(a-b\*a)^a/b

postfix expression:ababa\*-a^b/+

Exercise:6.1.2

class Queue:

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

self.queue=[]

def is\_empty(self):

return len(self.queue)==0

def enqueue(self,item):

self.queue.append(item)

def dequeue(self):

if self.is\_empty():

raise IndexError("Queue is empty")

return self.queue.pop(0)

def size(self):

return len(self.queue)

def is\_palindrome(s):

queue=Queue()

for char in s:

queue.enqueue(char)

while queue.size()>1:

if queue.dequeue()!=queue.queue[-1]:

return False

queue.queue.pop()

return True

test\_string="radar"

if is\_palindrome(test\_string):

print(f'"{test\_string}"is a palindrome.')

else:

print(f'"{test\_string}"is a palindrome.')

test\_string="hello"

if is\_palindrome(test\_string):

print(f'"{test\_string}"is a palindrome.')

else:

print(f'"{test\_string}"is not a palindrome.')

output:

"radar"is a palindrome.

"hello"is not a palindrome.