#Weekly Assessment 5

Instructions:

1. Sign the Honor Pledge.
2. In addition to your own notes and books, online resources can also be used but you have to provide reference(s).

Honesty Pledge:

I pledge on my honor that I have not given or received any unauthorized assistance on this exam/assignment.

Indicate your Name / Date: Kyle Matthew M. Cayanan / March 26, 2023

Week 5 Python Package to be used in this assessment:
https://drive.google.com/file/d/14_V23WaSzXww_XadaBnUtNxVEuAeqo9G/view?usp=sharing (https://drive.google.com/file/d/14_V23WaSzXww_XadaBnUtNxVEuAeqo9G/view?usp=sharing)

1. What values are returned during the following series of stack operations, if executed upon an initially empty stack? [10 pts]

push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().

###Answer:5, 8, 2, 1, 6, 7, 4, 9

2. What values are returned during the following sequence of queue opera- tions, if executed on an initially empty queue? [10 pts]

enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(), dequeue(), enqueue(4), dequeue(), dequeue().

###Answer:5,3,2,8,9,1,7,6

3. Postfix notation (also called Reverse Polish Notation) is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if "`(exp1) op (exp2)`" is a normal, fully parenthesized infix expression whose operation is `op`, the postfix version of this is "`pexp1 pexp2 op`", where `pexp1` is the postfix version of `exp1` and `pexp2` is the postfix version of `exp2`. The postfix version of a single number or variable is just that number or variable.

For example:

Infix expression: $((5 + 2) * (8 - 3))/4$

Postfix version: $5\ 2\ +\ 8\ 3\ -\ *\ 4\ /$

(a) Describe an algorithm that converts an infix expression to a postfix expression using one of the Linear ADTs discussed this week. [10 pts]

```
###Answer:
Assumptions
- The function being implemented is only for these operators(+,-,*,/,^)
- The input string will NOT have any empty space (since the sample inputs
also dont have this)
- The function must be able to perform operations on numbers higher than 1-
digit integers
- The Infinix Expression will always contain a parenthesis for any equation
with two or more operations. For example,
        Allowed: 4+(2*3)
        Not Allowed: 4+2*3

Implementation
- Using Dynamic Array ADT
- First Convert String to Characters
- Process each character one by one

Standard Approach (example: 5+3*6)
- first number is added, then space is added [5, ]
- operator is added, then space is added [5, ,+, ]
- after operator is added, set an integer equal to the position of the
operator
- second number will be inserted on to the operator's position [5, ,3, ,+, ]

Special Case (EMDAS):
- If an operator has greater priority than the previous operation,
- Then: subtract current_index by (2) "current_index-=2"
- This will cause the next few characters to be inserted into its rightful
position
[5, ,*, ,3, ,+, ]
[5, ,6, *, ,3, ,+, ]

- After they are added, change the current index back to normal
- current_index=len(dynamic_list)-1

Special Case (Parenthesis):
- If an operation is inside a parenthesis,
- It will follow the same premise where the current index will be subtracted
or left alone assuming it is already zero
    example: (5+3)*6
    [5, ,3, ,+, ]
    [5, ,3, ,+, ,6,*]

    example: 6*(5+3)
    [6,*]
    [5, ,3, ,+, ,6,*]
```

(b) Implement your algorithm in (a) for an Infix to Postfix Converter. It should accept a string (infix expression) and return a string (postfix expression). [10 pts]

Your code should reuse the module(s) in the Week 5 package given to you.

In [ ]:
```
def infix_to_postfix(s: str) -> str:
  pass

# Example:
# infix_to_postfix("2*20/2+(3+4)*3^2-6+15")

# Output should be:
# Infix: 2*20/2+(3+4)*3^2-6+15
# Postfix: 2 20 * 2 / 3 4 + 3 2 ^ * + 6 - 15 +
```

(c) Describe a nonrecursive way of evaluating a postfix expression using one of the linear ADTs discussed this week. [10 pts]

In [ ]:
```

```

```
###Answer:
I will use a stack implementation. Everytime the function detects a number,
it will push it onto the stack. When it detects the function, it will pop the
stack and use those numbers for the math equation. Since postfix already
follows PEMDAS, I no longer need to think about which operator should take
priority.
```

(d) Implement your algorithm in (c) for a Postfix Expression. It should accept a string (postfix expression) and return a float (result). [10 pts]

Your code should reuse the module(s) in the Week 5 package given to you.

In [8]:
```
#Import of Array Stack Implementation
##############################################################################
# Copyright 2013, Michael H. Goldwasser
#
# Developed for use with the book:
#
#    Data Structures and Algorithms in Python
#    Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser
#    John Wiley & Sons, 2013
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

"""Basic example of an adapter class to provide a stack interface to Python's l

#from ..exceptions import Empty

class ArrayStack:
    """LIFO Stack implementation using a Python list as underlying storage."""

    def __init__(self):
        """Create an empty stack."""
        self._data = []                       # nonpublic list instance

    def __len__(self):
        """Return the number of elements in the stack."""
        return len(self._data)

    def is_empty(self):
        """Return True if the stack is empty."""
        return len(self._data) == 0

    def push(self, e):
        """Add element e to the top of the stack."""
        self._data.append(e)                  # new item stored at end of list

    def top(self):
        """Return (but do not remove) the element at the top of the stack.

        Raise Empty exception if the stack is empty.
        """
        if self.is_empty():
            raise Empty('Stack is empty')
        return self._data[-1]                 # the last item in the list

    def pop(self):
        """Remove and return the element from the top of the stack (i.e., LIFO)
```

```
        Raise Empty exception if the stack is empty.
        """
        if self.is_empty():
            raise Empty('Stack is empty')
        return self._data.pop()                 # remove last item from list

################################################################################
#start of my own code


def postfix_calculator(s: str) -> float:
    #convert to list of characters
    eq_list=list(s)

    #declare stack
    eval_stack=ArrayStack()

    #declare list of operations
    operators=["+","-","/","*","^"]

    #declare indexes for counting
    ind_start=0
    ind_last=0
    skip=0

    #process the entire list
    for i in eq_list:


        #in my code, I use the space to detect the end of a number
        #skip is done whenever an operation has been done. This means that the
        #which is a space
        if(skip==1):
            #reset skip
            skip=0
            #then do nothing

        #if the character is an operator, then perform the math
        elif (i in operators):

            #call the operator function
            newnum=operator(eval_stack,i)

            #return final answer back to the stack
            eval_stack.push(newnum)

            #enable skip
            skip=1

            #change the start index for the next number
            ind_start=ind_last+2


        #if the current character is a space, that means it is the end of a num
        elif(i==" "):
            #process the number
```

```
            truenum=num_maker(eq_list,ind_start,ind_last)

            #push it onto the stack
            eval_stack.push(truenum)

            #change the starting index for the next number
            ind_start=ind_last+1

        ind_last+=1

    return(eval_stack.pop())



def num_maker(list,start,finish):
    result = float(''.join(list[start:finish]))
    return result

def operator(list, op):
    if op == "+":
        num1 = list.pop()
        num2 = list.pop()
        result = num1 + num2
        return result
    elif op == "-":
        num1 = list.pop()
        num2 = list.pop()
        result = num2 - num1
        return result
    elif op == "*":
        num1 = list.pop()
        num2 = list.pop()
        result = num1 * num2
        return result
    elif op == "/":
        num1 = list.pop()
        num2 = list.pop()
        result = num2 / num1
        return result
    elif op == "^":
        num1 = list.pop()
        num2 = list.pop()
        result = num2 ** num1
        return result

# Example:
# postfix_calculator("2 20 * 2 / 3 4 + 3 2 ^ * + 6 - 15 +")

# Output should be:
# Postfix: 2 20 * 2 / 3 4 + 3 2 ^ * + 6 - 15 +
# Result: 92

#test
Result=postfix_calculator("2 20 * 2 / 3 4 + 3 2 ^ * + 6 - 15 +")
print(Result)
```

```
92.0
```

4. Consider a queue-like data structure that supports insertion and deletion at both the front and the back of the queue. Such a structure is called a double-ended queue, or deque, which is usually pronounced "deck" to avoid confusion with the `dequeue` method of the regular queue ADT, which is pronounced like the abbreviation "D.Q."

The Deque ADT is more general than both the stack and the queue ADTs. The extra generality can be useful in some applications. For example, we describe a restaurant using a queue to maintain a waitlist. Occassionally, the first person might be removed from the queue only to find that a table was not available; typically, the restaurant will re-insert the person at the first position in the queue. It may also be that a customer at the end of the queue may grow impatient and leave the restaurant. (Note that We will need an even more general data structure if we want to model customers leaving the queue from other positions.)

The Deque ADT has the following methods:

D.add_first(e): Add element e to the front of deque D.

D.add_last(e): Add element e to the back of deque D.

D.delete_first(): Remove and return the first element from deque D; an error occurs if the deque is empty.

D.delete_last(): Remove and return the last element from deque D; an error occurs if the deque is empty.

Additionally, the deque ADT will include the following accessors:

D.first(): Return (but do not remove) the first element of deque D; an error occurs if the deque is empty.

D.last(): Return (but do not remove) the last element of deque D; an error occurs if the deque is empty.

D.is_empty(): Return True if deque D does not contain any elements.

len(D): Return the number of elements in deque D; in Python, we implement this with the special method `__len__`.

The following table shows a series of operations and their effects on an initially empty deque D of integers.

Operation	Return Value	Deque
D.add_last(5)	–	[5]
D.add_first(3)	–	[3, 5]
D.add_first(7)	–	[7, 3, 5]
D.first( )	7	[7, 3, 5]
D.delete_last( )	5	[7, 3]
len(D)	2	[7, 3]
D.delete_last( )	3	[7]
D.delete_last( )	7	[ ]
D.add_first(6)	–	[6]
D.last( )	6	[6]
D.add_first(8)	–	[8, 6]
D.is_empty( )	False	[8, 6]
D.last( )	6	[8, 6]

a) Implement that Deque ADT in much the same way as the ArrayQueue class in the Week 5 package. [30 pts]

In [2]:
```
#for this task, I will modify the provided Array_qeue function
#I will not remove the copyright since the code does not completely belong to m

#############################################################################

# Copyright 2013, Michael H. Goldwasser
#
# Developed for use with the book:
#
#    Data Structures and Algorithms in Python
#    Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser
#    John Wiley & Sons, 2013
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

#from ..exceptions import Empty

#############################################################################

# YOUR CODE HERE
class DoubleQueue:
    """FIFO queue implementation using a Python list as underlying storage."""
    DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

    def __init__(self):
        """Create an empty queue."""
        self._data = [None] * DoubleQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0
        self.back=0 #added back integer

    def __len__(self):
        """Return the number of elements in the queue."""
        return self._size

    def __str__(self):
        actual_array = [x for x in self._data if x is not None]
        return str(actual_array)

    def is_empty(self):
        """Return True if the queue is empty."""
        return self._size == 0

#############################################################################

#functionally the same
```

```
    def first(self):
        """Return (but do not remove) the element at the front of the queue.

    Raise Empty exception if the queue is empty.
    """
        if self.is_empty():
            raise Empty('Queue is empty')
        return self._data[self._front]

#last function
#returns the last value
    def last(self):
        """Return (but do not remove) the element at the back of the queue.
    Raise Empty exception if the queue is empty.
    """
        if self.is_empty():
            raise Empty('Queue is empty')
        last=(self._front + self._size) % len(self._data) - 1
        return self._data[last]


################################################################################
#changed to remove first
#functionally the same as the original dequeue
    def delete_first(self):
        """Remove and return the first element of the queue (i.e., FIFO).

    Raise Empty exception if the queue is empty.
    """
        if self.is_empty():
            raise Empty('Queue is empty')

        answer = self._data[self._front]
        self._data[self._front] = None         # help garbage collection
        self._front = (self._front + 1) % len(self._data)
        self._size -= 1
        return answer


#changed to delete_last
#modified to remove from the back of the line
    def delete_last(self):
            """Remove and return the last element of the queue

    Raise Empty exception if the queue is empty."""

            if self.is_empty():
                raise Empty('Queue is empty')

            #solve for the last integer position
            back = (self._front + self._size) % len(self._data) - 1

            #declare that integer to be the value of answer
            answer = self._data[back]

            #replace with NoneType
            self._data[back] = None         # help garbage collection
```

```
            #resize as necessary
            self._size -= 1

            #return
            return answer

################################################################################
#changed to add_last
#functionally the same as original enqueue
    def add_last(self, e):
        """Add an element to the back of queue."""
        if self._size == len(self._data):
            self._resize(2 * len(self._data))     # double the array size
        avail = (self._front + self._size) % len(self._data)
        self._data[avail] = e
        self._size += 1

#changed to add_front
#adds a number to the front of the line
    def add_first(self, e):
        """Add an element to the front of queue."""
        if self._size == len(self._data):
            self._resize(2 * len(self._data))     # double the array size

        if(self._data[0]==None):
            #this means that the actual queue may be somewhere in the middle of

            #solve for the proper front
            avail=self._front-1

            #change front variable
            self._front=avail
        else:
            #this means that the front of the queue is at the very front

            #need to shift values to the right
            self._data=[None]+self._data[0:-1]

            #avail will be the start
            avail=0

            #self.front is unchanged

        self._data[avail] = e
        self._size += 1

    def _resize(self, cap):                  # we assume cap >= len(self)
        """Resize to a new list of capacity >= len(self)."""
        old = self._data                       # keep track of existing list
        self._data = [None] * cap              # allocate list with new capacit
        walk = self._front
        for k in range(self._size):            # only consider existing element
            self._data[k] = old[walk]            # intentionally shift indices
            walk = (1 + walk) % len(old)         # use old size as modulus
        self._front = 0                        # front has been realigned
```

(b) Running your code, what values are returned during the following sequence of deque ADT operation, on an initially empty deque, D? [10 pts]

D.is_empty(), D.add_first(4), D.add_last(8), D.add_last(9), D.add_first(5), len(D), D.delete_first(), D.delete_last(), D.add_last(7), D.first(), D.last(), D.add_last(6), D.delete_first(), D.delete_first(), D.is_empty(), len(D), print(D)

Note: For the print() method, you have to implement the **str** method in the deque class such that it shows the contents of the deque as shown in the table above.

In [6]:
```
# YOUR CODE HERE

#print version for easier testing
if __name__ == '__main__':
    D=DoubleQueue()
    D.is_empty()
    D.add_first(4)
    D.add_last(8)
    D.add_last(9)
    D.add_first(5)
    len(D)
    D.delete_first()
    D.delete_last()
    D.add_last(7)
    D.first()
    D.last()
    D.add_last(6)
    D.delete_first()
    D.delete_first()
    D.is_empty()
    len(D)
    print(D)
```

```
[7, 6]
```