Array-Based Stack with Maximum Capacity X12072_en

Modify the ArrayStack implementation included in the Public Files section of the statement so that the stack's capacity is limited to `maxlen` elements, where `maxlen` is an optional parameter to the constructor (that defaults to `0`). If push is called when the stack is at full capacity, throw a `Full` exception (defined similarly to `Empty`). In order to avoid amortization[1], the new implementation of ArrayStack should initialize the underlying list to a list with length equal to the stack's maximum capacity specified in the call to the constructor.
1. See section 5.3 Dynamic Arrays and Amortization of the book:
M.T. Goodrich, R. Tamassia, and M.H. Goldwasser. Data Structures and Algorithms in Python. Wiley, 2013.
Observation It may be useful to run the program given in the Public Files on the public example (sample-1.inp) without modifying the class ArrayStack, and compare the result with the output the solution to this problem should generate for the same example (i.e. sample-1-cor).
In your implementation of `push` and `pop`, make sure the first line is

```
previous = sys.getsizeof(self._data)
```

and the last two lines (before `return`, if the function returns a value) are

```
    current = sys.getsizeof(self._data)
    ArrayStack._resize_check(previous, current)
```

You can use the following templates:

```
  def push(self, e):
    """Add element e to the top of the stack."""
    previous = sys.getsizeof(self._data)
    # INSERT YOUR CODE HERE
    current = sys.getsizeof(self._data)
    ArrayStack._resize_check(previous, current)

  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')
    else:
      previous = sys.getsizeof(self._data)
      # INSERT YOUR CODE HERE
      current = sys.getsizeof(self._data)
      ArrayStack._resize_check(previous, current)
      return val
```

Sample input

```
0
7
5
```

Sample output

```
len 0
stack empty
push error: stack full
len 0
top error: stack empty
len 0
top error: stack empty
pop error: stack empty
pop error: stack empty
len 0
top error: stack empty

len 0
stack empty
0 pushed
1 pushed
2 pushed
3 pushed
4 pushed
5 pushed
6 pushed
push error: stack full
len 7
top 6
6 popped
len 6
top 5
5 popped
4 popped
3 popped
2 popped
1 popped
len 1
top 0

len 0
stack empty
0 pushed
1 pushed
2 pushed
3 pushed
4 pushed
push error: stack full
len 5
top 4
4 popped
len 4
top 3
3 popped
2 popped
1 popped
0 popped
pop error: stack empty
len 0
top error: stack empty
```

Problem information

Author :
Generation : 2024-09-12 14:47:57