Answers to End of Section and Review Exercises for Chapter 7

Exercises 7.1

1. Table:

Operation	State of the Stack After the Operation	Value Returned	Comment
`s = <Stack Type>()`			Initially, the stack is empty.
`s.push(a)`	`a`		The stack contains the single item `a`.
`s.push(b)`	`a b`		`b` is the top item on the stack.
`s.push(c)`	`a b c`		`c` is the top item.
`s.pop()`	`a b`	`c`	Remove the top item from the stack and return it. `b` is now the top item.
`s.pop()`	`a`	`b`	Remove the top item from the stack and return it. `a` is now the top item.
`s.peek()`	`a`	`a`	Return the top item on the stack without removing it.
`s.push(x)`	`a x`		`x` is the top item.
`s.pop()`	`a`	`x`	Remove the top item from the stack and return it. `a` is now the top item.
`s.pop()`		`a`	Remove the top item from the stack and return it.
`s.pop()`		`Exception`	Popping an empty stack raises an exception.

2. Here is the code:

```
def bracketsBalance(exp, beginning, ending):
   """exp is a string that represents the expression.
   beginning and ending are lists of matching brackets."""
    stk = LinkedStack()                        # Create a new stack
    for ch in exp:                    # Scan across the expression
        if ch in beginning:              # Push an opening bracket
```

```
            stk.push(ch)
        elif ch in ending:        # Process a closing bracket
            if stk.isEmpty():                  # Not balanced
                return False
            chFromStack = stk.pop()
            # Brackets must be of same type and match up
            # Now check the positions of the brackets in their
            # lists - they must be the same
            endPos = ending.index(ch)
            beginPos = beginning.index(chFromStack)
            if beginPos != endPos:
                return False
    return stk.isEmpty()                   # They all matched up
```

3. This strategy works correctly as long as there is just one type of brackets—say, just parentheses. Adding a second type of bracket requires a second counter, and so on.

Exercises 7.2

1. a. 90
   b. 44
   c. 2.75
   d. 39
2. The running time of the postfix evaluator algorithm is essentially O(*n*), where *n* is the number of tokens in the expression. The position of the numeric tokens within the postfix expression determines the maximum depth of the stack. In the worst case, the growth of stack memory to accommodate the operands is O(*n* / 2 + 1), when all the numeric tokens occur at the beginning of the expression. In the best case, in which each pair of numbers immediately precedes an operator, the stack growth is O(1).

Exercises 7.3

1. a. 33 15 6 * -
   b. 11 6 2 + *
   c. 17 3 + 5 -
   d. 22 6 - 33 4 / +
2. The running time of the infix to postfix conversion algorithm is essentially O(*n*), where *n* is the number of tokens in the expression. The order of the operators within the infix expression determines how many operators are pushed onto the stack. The stack tends to receive more operators if they rank in precedence from lowest to highest in the infix expression. In the worst case, the growth of stack

memory to accommodate these operators is O(*n*), and in the best case, where the operators rank from highest to lowest, it is O(1).

Exercises 7.4

1. When using a Python list to implement an array-based stack, you do not have to worry about resizing the list because that is done automatically. Also, the top item is always the one at the end of the list, so you do not need an extra variable to track this position. There do not seem to be any costs.
2. Here is the code:

```
if self.isEmpty():
    raise KeyError("Stack is empty")
```

3. Here is the code:

```
if len(self) <= len(self._items) // 4 and \
   len(self._items) >= 2 * ArrayStack.DEFAULT_CAPACITY:
    # Shrink the size by half but not below default capacity
    # and remove those garbage cells from the underlying list
    temp = Array(len(self._items) // 2)   # Create new array
    for i in range(len(self)):     # Copy data from old array
        temp [i] = self._items[i]  # to new array
    self._items = temp # Reset old array variable to new array
```

Answers to Review Questions

1. b
2. b
3. b
4. b
5. a
6. b
7. b
8. b
9. a
10. b