Lecture 9.1 : Object-oriented programming: Stacks and Queues

Stacks

- The *stack* is a fundamental data structure which stores a collection of objects (of arbitrary type) that are inserted and removed in a *last-in, first-out (LIFO)* order.
- Objects can always be added to the stack but the only object accessible at any time is the most recently added object (which lives at the *top* of the stack).
- The *push* operation is used to add an object to the stack (making it the new stack top) while the *pop* operation removes the object currently at the top of the stack.

Stack methods

- An instance `S` of the stack abstract data type supports at a minimum the following two methods:
  - `S.push(e)`: Add element `e` to the top of the stack `S`.
  - `S.pop()`: Remove and return the element at the top of the stack `S`; an error occurs if the stack `S` is currently empty.
- The following convenience methods are also often implemented:
  - `S.top()`: Return a reference to the top element of stack `S` without removing it; an error occurs if the stack `S` is currently empty.
  - `S.is_empty()`: Return `True` if stack `S` is empty and `False` otherwise.
  - `len(S)`: Return the number of elements in `S`.

Implementing a stack with a list

```python
class Stack(object):

    def __init__(self):
        self.l = []

    def push(self, e):
        self.l.append(e)

    def pop(self):
        return self.l.pop()

    def top(self):
        return self.l[-1]

    def is_empty(self):
        return len(self.l) == 0

    def __len__(self):
        return len(self.l)
```

Queues

- Another fundamental data structure is the *queue*. A queue stores a collection of objects (of arbitrary type) that are inserted and removed in a *first-in, first-out (FIFO)* order.
- Objects can always be added to the *back* of the queue but the only object accessible at any time is the object that lives at the *front* of the queue i.e. the one which has been longest in the queue.
- The *enqueue* operation is used to add an object to the queue (it goes to the back) while the *dequeue* operation removes the object currently at the front of the queue.

Queue methods

- An instance `Q` of the queue abstract data type supports at a minimum the following two methods:
    - `Q.enqueue(e)`: Add element `e` to the back of the queue `Q`.
    - `Q.dequeue()`: Remove and return the element at the front of the queue `Q`; an error occurs if the queue `Q` is currently empty.
- The following convenience methods are also often implemented:
    - `Q.first()`: Return a reference to the element at the front of the queue `Q` without removing it; an error occurs if the queue `Q` is currently empty.
    - `Q.is_empty()`: Return `True` if queue `Q` is empty and `False` otherwise.
    - `len(Q)`: Return the number of elements in queue `Q`.

Implementing a queue with a list

- This is left as a lab exercise.