TD 1 - Stacks and Queues

Exercise 1: Implementation of a stack or a queue

Propose an implementation of a stack and a queue by writting following functions:

```
def create_stack():
    # Return a new empty stack.
def push( P, e ):
    # Push an element 'e' in a stack 'P'.
def pop( P ):
    # Pop an element from the stack 'P' and return this element.
def create_queue():
    # Return a new empty queue.
def enqueue( F, e ):
    # Enqueue the element 'e' in the queue 'F'.
def dequeue( F ):
    # Dequeue an element from the queue 'F' and return this element.
```

Draw the stack and queue contents during the run of the following programs. You will give the contents of the terminal also.

```
theStack=create_stack()
push(theStack, 1)
push(theStack, 2)
push(theStack, 3)
print(pop(theStack))
print(pop(theStack))
print(pop(theStack))
```

```
theQueue = create_queue()
enqueue(theQueue,1))
enqueue(theQueue,2))
enqueue(theQueue,3))
print(dequeue(theQueue))
print(dequeue(theQueue))
print(dequeue(theQueue))
```

Exercise 2

Suppose we cannot create or manipulate lists, dictionaries or tuples. Suppose one give you the following primitives to manipulate stacks:

```
def create_stack()
def push( P, e )
def pop( P )
```

1) Propose an implementation of queues by only using stacks, i.e. the previous primitives. To do that, implement the following primitives:

```
def create_queue()
def enqueue( F, e )
def dequeue( F )
```

Evaluate the complexity of these functions.

2) Propose an implementation of arrays by only using previous primitives. To do that , implement the following primitives:

```
def create_array():
    # Create et return an empty array.
def insert_element( T, id, e ):
    # Insert a new element 'e' to the position 'id' in the array.
        # 'T'. The array size is increased by 1.
def suppress_element( T, id ):
    # Suppress the element at the position 'id' in the array 'T'.
    # The size of the array is decreased by 1.
def replace_element( T, id, e ):
    # Replace the element at the position 'id' in 'T' by 'e'.
    # The size of the array has not changed.
def get_element( T, id ):
    # Return the element into the position 'id' in the array.
```

Evaluate the complexity of each function.

3) Propose an implementation of dictionaries by only using previous primitives. To do that, implement the following primitives:

```
def create_dictionary():
    # Create and return a new dictionary.
def insert_association( D, key, value ):
    # Add a new association ('key','value') in the
    # dictionary D.
    # If an association ('key', 'OtherValue') already exists in D,
    # this  association is replaced by ('key', 'value') .
def suppress_association( D, key ):
    # Suppress the association with the key 'key' from D.
def is_in_dictionary( D, key ):
    # Return True if 'key' is a key in the dictionary D.
def get_value( D, key ):
    # Return the value associated to the key in D.
```

Evaluate the complexity of each function.