Advanced computer programming

Exercise session 3: Stack, queue, list, vector and sequence

Jean-Michel Begon

November 2014

Exercise 1

(a) Let `n` and `m` be two nodes of a singly linked list. Whats is the results of the following operations on a list where `n` belongs (but not `m`) ?

1. `n.next = n.next.next`
2. `m.next = n.next; n.next = m`
3. `n.next = m; m.next = n.next`

(b) Let `S` be a singly linked list. Write a function `BeforeBeforeLast(S)` which returns the before-before last node of `S`.

(c) Let `S` be a singly linked list. Write a function `Reverse(S)` which inverts `S` in place.

(d) Let `S` be a singly linked list. Write a function `Delete(S, n)` which removes node `n` from `S`.

Exercise 2

(a) Let `S` be a stack. Illustrate the following operations (the state of the stack and the sequence of outputs).

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

(b) Let `Q` be a queue. Illustrate the following operations (the state of the queue and the sequence of outputs).

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

Exercise 3

Describe a structure which provides two stacks with a single array as backup structure. Implement the `Pop` and `Push` operations.

Exercise 4

Rewrite the `Enqueue(Q, x)` and `Dequeue(Q)` functions from the course so as to manager the error (overflow/underflow/memory allocation errors). Consider the array implementation and the linked list implementation.

Exercise 5

- Implement a queue with two stacks. What is the complexity of the `Enqueue` and `Dequeue` operations ?
- Implement a stack with two queues. What is the complexity of the `Push` and `Pop` ? operations ?

Exercise 6

Modify the implementation of `RemoveAtRank(V, r)` so as to reduce memory consumption by 2 if the number of elements becomes less than `V.c / 4` (where `V.c` is the current capacity of `V`).

Exercise 7

Implement an efficient data structure for the following sequence TDA:

- `Insert-Before(S, p, x)` : insert x before p in the sequence.
- `Insert-After(S, p, x)` : insert x after p in the sequence.
- `Remove(S, p)` : remove the element at position p.
- `Replace(S, p, x)` : replace by x the object at position p.
- `First(S)`, `Last(S)` : return the first, resp. last, position in the sequence.
- `Prev(S, p)`, `Next(S, p)` : return the preceding (resp. following) position p in the sequence.
- `Elem-At-Rank(S,r)` : return the element at rank r in the sequence.
- `Replace-At-Rank(S,r, x)` : replace the element at rank r by x and return this object.
- `Insert-At-Rank(S,r, x)` : insert the element x at rank r, increasing the rank of all the following elements.
- `Remove-At-Rank(S,r)` : retrieve and remove the element at rank r, decreasing the rank of all the following elements.
- `Size(S)` : return the size of the sequence
- `At-Rank(S,r)` : return the position of the element at rank r.
- `Rank-Of(S, p)` : return the rank of the element situated at position p.

Note : you can resort to other data structure to implement the sequence.

Bonus

Bonus 1

In many fields of applied computer science (bioinformatics, optimization,...), there is a need for a model of a fixed size memory where new data replace old ones (As in a queue). For example, a memory with a capacity of 7 would yield:

```
M = create-memroy(7)
store(M, 1)
store(M, 2)
store(M, 3)
store(M, 4)
store(M, 5)
store(M, 5)
store(M, 7)
store(M, 8)
print-memory(M)
>>> 2, 3, 4, 5, 5, 7, 8 //The order is preserved
```

(a) What kind of data structures would be adequate for this application (we would like fast access to the elements through their index) ?

(b) Let us now imagine that we would like to be able to double spontaneously the memory capacity if there are $2n$ consecutive insertions without any retrieval. What data structure would you resort to ?

Bonus 2

We would like a simplified version of a priority queue. The TDA is:

- `append(Q, x)` : insert x at the *end* of the queue `Q`.
- `preprend(Q, x)` : insert x at the *start* of the queue `Q`.
- `dequeue(Q)` : return the first element of the queue `Q`.

What data structures are well suited for this task ?