Elementary data structures (exercises)

— P.05 —

Summary:

- Stacks
- Singly-linked lists
- Queues
- Deques
- Doubly-linked lists
- Min-heap
- Priority queue
- Hash tables

Stacks

Extract the files `aStack.h` and `aStack_demo.cpp` from the archive `P05.tgz`. Study the generic implementation of a stack (file `aStack.h`). The purpose of the program `aStack_demo.cpp` is to verify if the parentheses of each of its text arguments are balanced. When called as follows (warning: copying and pasting may not work properly on the following line; if it does not work the accute accent is the culprit)

```
  ./aStack_demo 'abc' 'a(b)' 'a(b' 'a)b' 'a(b(c)(d((ef)g))h)i'
```

it should produce the output

```
abc
  good
a(b)
  '(' at position 1 and matching ')' at position 3
  good
a(b
  unmatched '(' at position 1
  bad
a)b
  unmatched ')' at position 1
  bad
a(b(c)(d((ef)g))h)i
  '(' at position 3 and matching ')' at position 5
  '(' at position 9 and matching ')' at position 12
  '(' at position 8 and matching ')' at position 14
  '(' at position 6 and matching ')' at position 15
  '(' at position 1 and matching ')' at position 17
  good
```

The code in `aStack_demo.cpp` is incomplete. Complete it using a stack.

Modify the `aStack.h` class so that the stack can grow as much as needed. (Hint: write a private member function that resizes the stack, and start with a stack with a maximum size of, say, 100.)

Singly-linked Lists

Extract the files `sList.h` and `sList_test.cpp` from the archive `P05.tgz`. The file `sList.h` implements a generic singly-linked list. Study it. Study also the file `sList_test.cpp`, that tests the correctness of the implementation in `sList.h`.

Queues

Extract the files `lQueue.h` and `lQueue_demo.cpp` for the archive `P05.tgz`. The file `lQueue.h` contains a skeleton of an implementation of a generic queue based on a singly-linked list. Complete the implementation and write code to test it.

Deque

Implement a generic deque (double-ended queue) using an array. On a deque, insertion and deletion can occur at both ends of the queue.

Doubly-linked lists

Work to be done at home: using the code in `sList.h` as starting point, implement a doubly-linked list. Hints:

- the `move()` member function can be improved, but that is not strictly necessary,
- the various `insert` and `remove` member functions have to be modified.
- the computational complexity of some of these functions may change!

Min-heap

Using the code presented in T.05 lecture as inspiration, implement a min-heap in C. Use the min-heap to sort an array of integers in decreasing order.

Priority queue

Using your min-heap, implement a priority queue (assume that lower values have higher priorities). Test it.

Hash tables

Using the code presented in the T.05 lecture as inspiration, complete the implementation (in C) of a hash table (with separate chaining) capable of storing (key,value) pairs, in which the key is a string with at most 64 characters and in which the value is of type `T`. An incomplete implementation is stored in the file `hash_table.h` (`P05.tgz`). Use the hash table to count the number of times each word occurs in the text file `SherlockHolmes.txt`[1] (get it from `P02.tgz`). Use as keys the words, and as values the number of times each one occurs. The file `count_words.c` contains an incomplete implementation; finish it! Which word appears more times?

[1]Source: `http://sherlock-holm.es`.