Lab 10

COMP9021, Session 1, 2014

The aim of this lab is to practice the use of linked lists and queues.

1 Polynomials as linked lists

Write a new version of the program developed in Lab 8 where polynomials are represented as linked lists of monomials rather than as arrays of monomials. In this new version, monomials and polynomials will be represented as the following structures.

```
typedef struct monomial {
    int coeff;
    int degree;
    struct monomial *pt_to_monomials;
} Monomial;

typedef Monomial *Polynomial;
```

2 Doubly linked lists

A *doubly linked list* is a linear data structure whose nodes include both a link to the previous node and a link to the next node. Implement the following interface.

```
 typedef struct DL_node {
    int value;
    struct DL_node *pt_to_previous_node;
    struct DL_node *pt_to_next_node;
} DL_node;


/* Returns the address of a newly created DL_node that stores val. */
DL_node *create_DL_node(const int val);

/* Returns the length of the list that starts at pt_to_DL_node. */
int list_length(const DL_node *pt_to_DL_node);

/* Applies the function which is the value of pt_to_function to the values stored
 * in the list that starts at pt_to_DL_node. */
void apply_to_DL_list(void (*const pt_to_function)(), const DL_node *pt_to_DL_node);

/* Inserts in the DL_list that starts at the value of pt_to_pt_to_DL_node, and that is
 * assumed to be sorted, a DL_node that stores val, so that the resulting list remains
 * sorted, and updates the value of pt_to_pt_to_DL_node to the address of the first
 * DL_node of the resulting list. */
void insert_in_sorted_DL_list(const int val, DL_node **const pt_to_pt_to_DL_node);

/* Removes in the DL_list that starts at the value of pt_to_pt_to_DL_node,
 * the first DL_node that stores val, and updates the value of pt_to_pt_to_DL_node
 * to the address of the first DL_node of the resulting DL_list.
 * If val is not stored in the DL_list then the DL_list is unchanged
 * and the function returns false; otherwise the function returns true. */
bool remove_from_DL_list(const int val, DL_node **const pt_to_pt_to_DL_node);

/* Deletes all DL_nodes of the DL_list that starts at the value of pt_to_pt_to_DL_node,
 * and sets that value to NULL. */
void delete_list(DL_node **const pt_to_pt_to_DL_node);
```

3 Queues as circular structures

A queue can be represented by an array `A` conceived of as a circular structure: if `n` is the size of `A` then `A[0]` is viewed as following `A[n - 1]` in the same way as `A[m + 1]` is viewed as following `A[m]` for all `m` smaller than `n - 1`. The queue is represented by a pair of values `front` and `rear` that indicate the position of the front and rear elements of the queue. For an empty queue `front` is the position that follows `rear` (hence either `front` is equal to `m + 1` and `rear` to `m` for some `m` smaller than `n - 1`, or `front` is equal to `0` and `rear` to `n - 1`). This representation implies that the queue is full when it contains `n - 1` elements.

Modify `tailored_queue.h` accordingly, and rewrite the implementation of that interface using this representation.