Informatics 2C Computer Systems - Lab 3

1 Introduction

The objective of this lab is to familiarize students with dynamic memory allocation and management and use of pointer-based data structures in C. It involves implementing a queue using a linked list. Download the supporting lab material from the course webpage.

2 Pointer-based Data Structures

As discussed in the lectures, structures can be used to aggregate data of different types. The `malloc` and `calloc` functions are used to dynamically allocate an area of memory to be used for the data structure. Pointers are used to access the dynamically allocated data structures. A simple example is that of a dynamically allocated array:

```
int *dyn_array = malloc(16 * sizeof(int));
```

Similarly, more complex data structures can be dynamically allocated. This lab focuses on one such data structure, a linked list. The structure used in this lab will be a linked list element, `elem`. The file `linkedlist.c` contains the declaration for `elem`. `head` is a pointer to the first element in the linked list.

```
typedef struct elem {
      int val;
      struct elem *nxt;
} elem;

elem *head;
```

2.1 Linked List

A linked list consists of elements containing data and a link to the next element. The first link points to the first element, second link to the second element, and so on. The last element has a reference to `NULL`. The size of the linked list can change dynamically, upon insertion or deletion of elements, during program runtime. This is a singly linked list.

Another type of linked list is a doubly linked list where each element contains two links, one to the previous element and the other to the next element.

This lab requires implementation of a queue using a singly linked list as the underlying data structure.

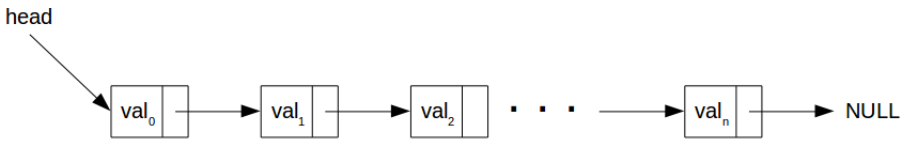

Figure 1: Queue implementation using a singly linked list.

2.2 Dynamic Memory Allocation and Management

In C, dynamically allocated memory (on the heap) must be explicitly freed. Failure to do so leads to the problem of "memory leaks" as discussed in the lectures and tutorials. When manipulating a linked list it is important to explicitly free memory upon deletion of an element.

3 Tasks

The `struct elem` has been provided to you in the file `linkedlist.c`. Using this structure, complete the following tasks.

- Create the first element in the linked list using `malloc` and assign it to `head`.
- Write the function `insertq` that adds an element to the end of the queue.
  ```
  void insertq(elem *qhead, int value);
  ```
- Write the function `removeq` that removes an element from the end of the queue.
  ```
  void removeq(elem *qhead);
  ```
- Write the function `sizeq` that finds the number of elements in the queue.
  ```
  int sizeq(elem *qhead);
  ```
- Write the function `searchq` that searches through the queue for a value `v` and returns the reference to the first matching element.
  ```
  elem* searchq(elem *qhead, int value);
  ```