# Programming Assignment 1: Linked Lists

Due: Monday, Feb 8

---

Key Concepts:

- Linked-Lists
- Memory allocation and deallocation in C
- Compilation with Multiple Files
- Runtime (a little)
- Recursion

Background Reading:

- Chapter 3 of Weiss
- Stanford C pointer tutorial
- Stanford linked-list tutorial

---

**Part I:** Examine the files `list.h` and `llist.c` in the `src` directory where you found this handout.

You will discover that it is a partially implemented linked list "module".

The lists store numeric values (the type of which can be changed by altering the `typedef` for `ElemType` in `list.h`). The idea here is to allow some degree of flexibility in what a list stores -- however, for the purposes of this assignment, you don't need to worry about this. Just continue to use `ElemType` as `int`.

The header file `list.h` file gives the interface for an ADT while the actual implementation is given in the `llist.c` file. The members of `list_struct` are also "hidden" in the `.c` file. The ADT defines many natural operations on lists -- some of these have already been implemented and will be used as motivating examples during lecture; others have not been implemented: It is your job to do the implementation! Look for `TODO` labels throughout the files.

A subtle detail: why did I decide to name the header file `list.h` (one `'l'`), but the implementation file `llist.c` (two `'l'`s)???

So… part I is completion of all of the `TODO` items specified.

**Rules**:

- **You cannot modify `list.h`**

Exceptions: if you want to fiddle with different ElemType (e.g., double) OR you want to add new sanity checkers to call externally.

All of your "real" work is in `llist.c` (except testing code).

**Discussion**: The given linked list structure has two "levels":

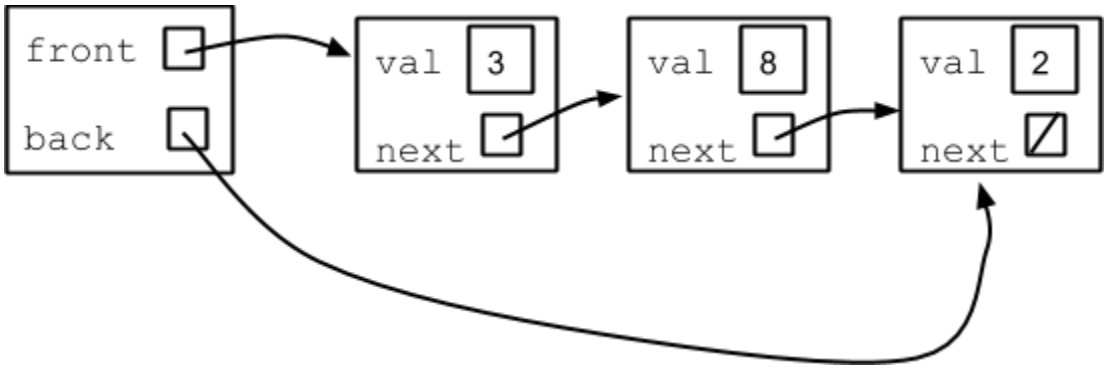
At the "lowest" level are the linked-list nodes themselves specified as:

```
typedef struct node {
    ElemType val;
    struct node *next;
} NODE;
```

However, the type `NODE` isn't even visible to a client program. Only the type `LIST` is visible to a client (just the type -- not the struct members). Through the header file, `LIST` is equivalent to a `struct list_struct` which is specified as follows:

```
struct list_struct {
    NODE *front;
    NODE *back;
};
```

Here is a diagram of a list with three entries: `<3, 8, 2>`. The `struct` at the left (a `LIST`) gives access to the actual nodes.




List of `TODO` functions:

```
lst_length, lst_count, lst_pop_back, lst_print_rev,
lst_reverse, lst_is_sorted, lst_insert_sorted,
lst_merge_sorted, lst_clone, lst_from_array, lst_to_array,
lst_prefix, lst_filter_leq, lst_concat
```

---

**Submission Details:**

You will submit the following files through Blackboard:

```
list.h (which actually should not have changed)
llist.c
my_tester.c
makefile
readme (but only if there is something unusual about your
submission that you want to point out to the TAs)
```

We should be able to create `llist.o` by simply typing:

```
make llist.o
```

We should also be able to create your tester program by typing:

```
make my_tester
```

---

## NOTES:

As suggested, you are expected to submit a tester program. Your tester program will not contribute to your score, but we want to be able to give you feedback on it.

Approximately 4 days before the assignment is due, we will release a suite of test cases. This will take the form of a driver program and they will be a ***subset*** of the test cases we will use for grading your submission.

The idea is to encourage you to think carefully about testing -- ideally, the test cases given will be redundant for you because you've already done similar tests; but if not, it is a bit of a wakeup call.

A somewhat trivial driver program has been given in `ll_tst.c`; we've also included a baseline makefile which you can modify.