Add a list function

Extend the list library covered in the lectures (in the `list/`
subdirectory) with a new function:

```C
// Insert element at end of list.  Returns 0 on success.
int list_insert_last(struct list*, void*);
```

It should insert a new element at the end of the list. Make sure to
write tests for it, and to check that it does not leak memory.

Implementing a stack

The `calc/` directory contains the code for a simple calculator that
uses Reverse Polish Notation. For example, to compute the expression
`(3 - 4) / (5 + 6) == -0.090909` and print the result, we would input:

~~~
$ ./calc
3
4
-
5
6
+
/
p
Result: -0.090909
~~~

Note the newlines after each symbol.

And to compute the expression `3 - (4 - 5) == 4` and print the result,
we would input:
~~~
$ ./calc
3
4
5
-
-
p
Result: 4
~~~

Unfortunately I ran late and forgot to finish the
implementation of `stack.h` and `stack.c`.
Finish it for me so the stack calculator will work.

List implementation

Change the implementation of the list library covered in the lectures
(in the `list/` subdirectory) to use a different representation. Do
not change `list.h`. For example, an implementation with doubly
linked lists:

```
    struct node {
      void *elem;
      struct node *next;
      struct node *prev;
    };

    struct list {
      struct node *first;
      struct node *last;
    };
```

Or one where the representation is a flat array:

```
struct list {
  void **data;
  int length;
};
```

Consider how these affect the performance of the API, without changing its semantics.