Assignment 8 — Dynamic Allocation and Structures

Due: Monday, June 6th

Beyond the statically allocated global data that is packaged with the machine code of executables and the automatically allocated local data that is maintained on the runtime stack, C allows programmers to dynamically allocate memory at runtime on the heap. Programmers can use this memory for any form of data they desire, whether that be primitive variables, arrays, or structures.

Deliverables:

GitHub Classroom: `https://classroom.github.com/a/wJIYcD01`

Required Files: `list.c`

Optional Files: `*.c`, `*.h`

Part 1: Ground Rules

Throughout this class, any C code you write must compile using the command:

```
>$ gcc -Wall -Werror -ansi -pedantic ...
```

Furthermore, your C programs are expected to compile and run on Cal Poly's Unix servers.

Part 2: Linked Lists

Implement a linked list by completing the C functions in `list.c`:

- Your functions must dynamically allocate memory to store any lists, nodes, or arrays that they create.
- Your functions must, in the appropriate functions, free all memory that was allocated.
- You may add helper functions to `list.c`, and you may add additional C source or header files, if desired. However, you may *not* alter contents of `list.h`.

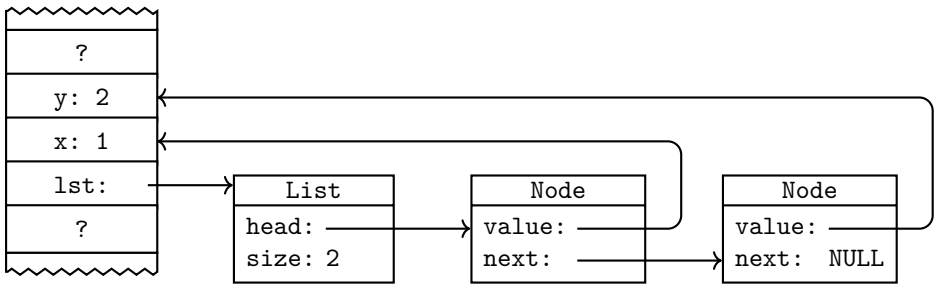
The values stored in your linked lists will be void pointers, so that your linked lists can contain data of any type. That is, each linked list node will contain a pointer to some piece of data — you may assume that your functions' callers will correctly manage the types of and memory for their data.

Thus, the following code:

```
   List *lst;
   int x = 1, y = 2;

1  lst = lstcreate()
2  lstadd(lst, 0, &x)
3  lstadd(lst, 1, &y)
```

…should produce the following pointers and dynamically allocated structures:



Part 3: Testing

C does not have a built-in unit testing framework. The C standard library does, however, provide a single lightweight function[1], `assert`, which checks to see if a boolean expression evaluates to "true".

A minimal set of tests has been provided in `lsttests.c`. If an `assert` succeeds, execution continues on; if one fails, an error message is printed and the program is terminated.

For example:

```
>$ gcc -Wall -Werror -ansi -pedantic list.c lsttests.c
>$ ./a.out
```

These tests are by no means exhaustive, and you are encouraged — though not required — to write additional unit tests. The quality of your tests will not be assessed as part of grading.

Part 4: Submission

The following files are required and must be pushed to your GitHub Classroom repository by the deadline:

- `list.c` — A working C implementation of linked lists, as specified.

The following files are optional:

- `*.c` — Any additional C source code specific to your implementation of `list.c`.
- `*.h` — Any additional C header files specific to your implementation of `list.c`.

Any files other than these will be ignored.

[1] Actually, "`assert`" is a macro — macros that take arguments are beyond the scope of this class.