CSE 3100 Systems Programming	Fall 2025
Lab #4	

Welcome to Lab 4! Read each question carefully before beginning work. Note that a Makefile is provided. In this assignment, you can only change the functions you are asked to implement. You may add additional functions, if necessary. However, *do not change other functions already implemented in the starter code.*

Search `TODO` to quickly find the functions you need to work on.

Part 1. Delete node. The linked list example we have studied in lecture maintains a singly linked list of integers. The program can append and prepend integers to the list. Type `help` in the program to see a list of commands. Note that if the number to be added is already on the list, the program prints the information about the node that stores the number and does not add the number twice.

The list is displayed every time an integer is processed.

Currently, the program does not support the delete function. Complete the function `delete_node()` in `linkedlist.c` so that the program can delete an integer from the list. The prototype of the function is:

```
node * delete_node(node * head, int v);
```

`head` is a pointer to the head node and `v` is the integer to be removed. The function returns the pointer to the head node of the new list, which could be the same as `head`. If `v` is not on the list, the function prints a message using the following function call: `error_message(ERR_NOTFOUND)`.

Below is an example session using the delete feature.

```
$ ./linkedlist-main
1 2 3 4 5 6 7 8 9
[1, ]
[1, 2, ]
[1, 2, 3, ]
[1, 2, 3, 4, ]
[1, 2, 3, 4, 5, ]
[1, 2, 3, 4, 5, 6, ]
[1, 2, 3, 4, 5, 6, 7, ]
[1, 2, 3, 4, 5, 6, 7, 8, ]
[1, 2, 3, 4, 5, 6, 7, 8, 9, ]
d5
[1, 2, 3, 4, 6, 7, 8, 9, ]
d3
[1, 2, 4, 6, 7, 8, 9, ]
d3
linkedlist: The number is not on the list.
[1, 2, 4, 6, 7, 8, 9, ]
```

Part 2. List reversal. In this exercise, we implement another function `reverse_list()` in `linkedlist.c`. The prototype of the function is:

```
node * reverse_list(node * head);
```

The function receives `head` as its sole argument, a pointer to the head node of the linked list. Assume the list is acyclic. The function must change the `next` fields of each node so that the nodes end up linked in reverse order. The function must return the address of the new head node (originally last in the list). You may use additional functions and local variables besides those already included in the starter code, but cannot change the values stored in the nodes or dynamically allocate new nodes. You can use either a loop or recursion.

Below is an example session using the reversal feature.

```
$ ./linkedlist-main
1 2 3 4 5 6 7 8 9
[1, ]
[1, 2, ]
[1, 2, 3, ]
[1, 2, 3, 4, ]
[1, 2, 3, 4, 5, ]
[1, 2, 3, 4, 5, 6, ]
[1, 2, 3, 4, 5, 6, 7, ]
[1, 2, 3, 4, 5, 6, 7, 8, ]
[1, 2, 3, 4, 5, 6, 7, 8, 9, ]
r
[9, 8, 7, 6, 5, 4, 3, 2, 1, ]
r
[1, 2, 3, 4, 5, 6, 7, 8, 9, ]
```