Project 1 documentation

I started by reading the requirements, and decided that a linked list is how I want to implement the storing of data (the lines). This was the best choice in my opinion, since we dont know the length of the file. This way we can just append more and more lines into the linked list.

Input.txt

input.txt

```
1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin imperdiet justo sit amet enim vulputate tincidunt. Fusce commodo hendrerit auctor. Nunc
2. lacinia nulla sed metus interdum, at feugiat ligula consectetur. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

3. Maecenas finibus mollis placerat. Aenean rutrum sit amet lorem eget semper. Pellentesque et felis ac quam dapibus lobortis. Maecenas ac nisi libero. Donec

4. neque enim, rhoncus quis lacus non, porta fringilla risus. Aliquam nisi nibh, molestie a elementum ac, dignissim sit amet odio. Nam ac tellus metus. Aenean rhoncus, risus pulvinar efficitur lacinia, nisi mauris vestibulum turpis, sed maximus neque nulla quis nisl. Aliquam tempor at orci sed dapibus. Duis

5. lacinia lacinia lorem quis vulputate.

6. Fusce vitae turpis rutrum, lacinia libero vel, auctor massa. Phasellus euismod ac nulla at pretium. Nam sollicitudin vulputate lacus, vitae commodo libero
```

Output.txt

output.txt

```
6. Fusce vitae turpis rutrum, lacinia libero vel, auctor massa. Phasellus euismod ac nulla at pretium. Nam sollicitudin vulputate lacus, vitae commodo libero
5. lacinia lacinia lorem quis vulputate.

4. neque enim, rhoncus quis lacus non, porta fringilla risus. Aliquam nisi nibh, molestie a elementum ac, dignissim sit amet odio. Nam ac tellus metus. Aenean rhoncus, risus pulvinar efficitur lacinia, nisi mauris vestibulum turpis, sed maximus neque nulla quis nisl. Aliquam tempor at orci sed dapibus. Duis

3. Maecenas finibus mollis placerat. Aenean rutrum sit amet lorem eget semper. Pellentesque et felis ac quam dapibus lobortis. Maecenas ac nisi libero. Donec

2. lacinia nulla sed metus interdum, at feugiat ligula consectetur. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.
1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin imperdiet justo sit amet enim vulputate tincidunt. Fusce commodo hendrerit auctor. Nunc
```

Creation of the nodes. Here we create a node structure, as well as a function, that creates a new node and returns a pointer to it.

```c
// Define a node in a linked list
typedef struct Node {
    char *line;
    struct Node *next;
} Node;

// Function to create a new node with the given line
Node* createNode(char *line) {
    Node *new_node = malloc(sizeof(Node));
    if (new_node == NULL) {
        fprintf(stderr, "malloc failed.\n");
        exit(1);
    }
    new_node->line = line;
    new_node->next = NULL;
    return new_node;
}
```

The function that actually reverses the file. The lines are read and put into the linked list with the newest at the front. So when the list is printed out, the file is reversed.

```c
// Function to reverse the lines by reading them into a linked list
Node* read_lines(FILE *input) {
    Node *head = NULL;
    Node *current = NULL;
    char *buffer = NULL;
    size_t buffer_size = 0;
    ssize_t line_length;

    while ((line_length = getline(&buffer, &buffer_size, input)) != -1) {
        // Create a copy of the line
        char *line = strdup(buffer);
        if (line == NULL) {
            fprintf(stderr, "malloc failed\n");
            exit(1);
        }

        // Create a new node and append it to the list
        Node *new_node = createNode(line);
        if (head == NULL) {
            head = new_node;
        } else {
            new_node->next = head;
            head = new_node;
        }
    }

    free(buffer);
    return head;
}
```

Printing the linked list out:

```c
// Function to print the lines in reverse order from the linked list
void print_lines(Node *head, FILE *output) {
    Node *current = head;
    while (current != NULL) {
        fprintf(output, "%s", current->line);
        Node *temp = current;
        current = current->next;
        free(temp->line);
        free(temp);
    }
}
```

Handling the conditions and errors. Here we check the amount of arguments, and handle them properly. Also here we handle opening the files and deal with errors with them.

```c
int main(int argc, char *argv[]) {
    FILE *input = stdin;
    FILE *output = stdout;

    if (argc > 3) {
        fprintf(stderr, "Usage: reverse <input> <output>\n");
        exit(1);
    }

    // Open input file if specified, otherwise use stdin
    if (argc >= 2) {
        input = fopen(argv[1], "r");
        if (input == NULL) {
            fprintf(stderr, "Error: cannot open file '%s'\n", argv[1]);
            exit(1);
        }
    }

    // Open output file if specified, otherwise use stdout
    if (argc == 3) {
        if (strcmp(argv[1], argv[2]) == 0) {
            fprintf(stderr, "Input and output file must differ\n");
            exit(1);
        }

        output = fopen(argv[2], "w");
        if (output == NULL) {
            fprintf(stderr, "Error: cannot open file '%s'\n", argv[2]);
            exit(1);
        }
    }

    // Read and reverse the lines
    Node *head = read_lines(input);
    print_lines(head, output);

    // Clean up and close files if necessary
    if (input != stdin) fclose(input);
    if (output != stdout) fclose(output);

    return 0;
}
```