Project 1: Warmup to C and Unix programming

Daniil Komov, Igor Zimarev

Link to source code Github repository: https://github.com/IkomovONE/C-Unix_SimpleLReverser

For this mini-project, we made a simple C program which reverses lines of strings. The program works as a Unix utility and takes in lines of text as input (either manual input or through a file), then produces the lines in reversed order (either on screen or to output file).

The reverse.c is the only codebase of the program. The program can be compiled with gcc using this command: “gcc -o reverse reverse.c”

The repository already includes a compiled program. There are 3 ways to run the program:

-0 arguments:

prompt> ./reverse

This command will run the program in manual input mode, after that it will be possible to write as many lines as wished until the input is stopped (ctrl+D). The program will then print the result on the screen.

-1 argument:

prompt> ./reverse <input>

This prompt will run the program in file input mode, it will open the input file, read the lines and display the resulting reversing order on the screen. Paste input file name instead of <input>.

-2 arguments:

prompt> ./reverse <input> <output>

This prompt will run the program in file input/output mode, it will open the input file, read the lines and write the resulting reversed order into the provided file (a new file will be created in case an existing one is not found). Paste input file name instead of <input>, and output file name instead of <output>

Error handling

- Input and output files have to be different, otherwise error will occur “Input and output file must differ”.

- If the file is corrupt of unavailable, error will pop up “error: cannot open file ‘<file>’”.

- In case of internal memory allocation issue, a error message will appear “malloc failed”.

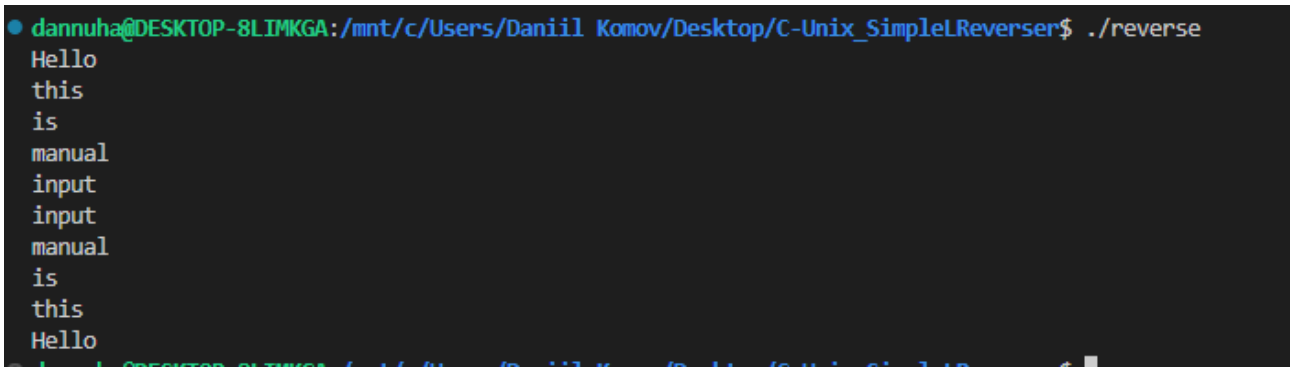
The program has no line length or file size limitations, however, very large file may affect system performance.

Use cases examples

-0 arguments:

prompt> ./reverse

The program runs in manual input mode, few line input gives result:

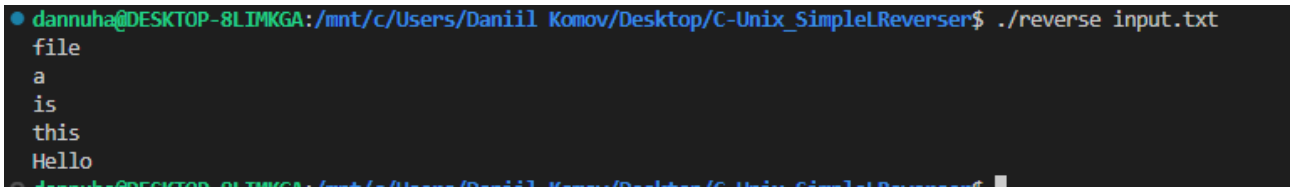


First 5 lines are the manually entered input lines, below is the output of the program.

-1 argument:

prompt> ./reverse input.txt

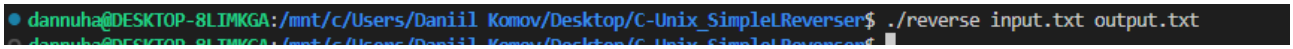
The program runs in file input mode, input file and resulting output:

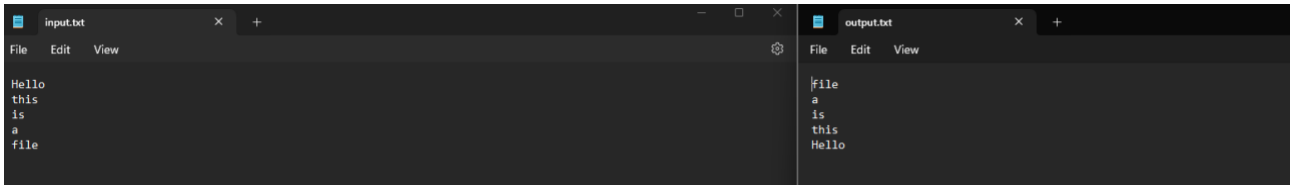


-2 arguments:

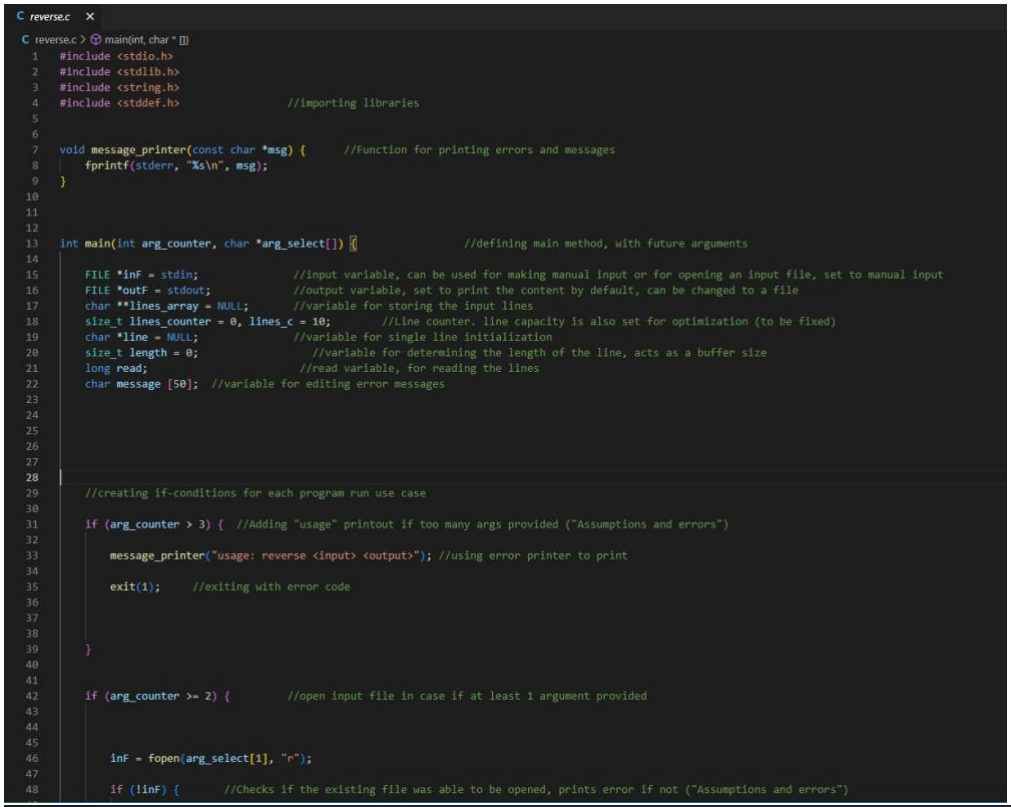
prompt> ./reverse input.txt output.txt

The program runs in file input/output mode, input file and resulting output file:

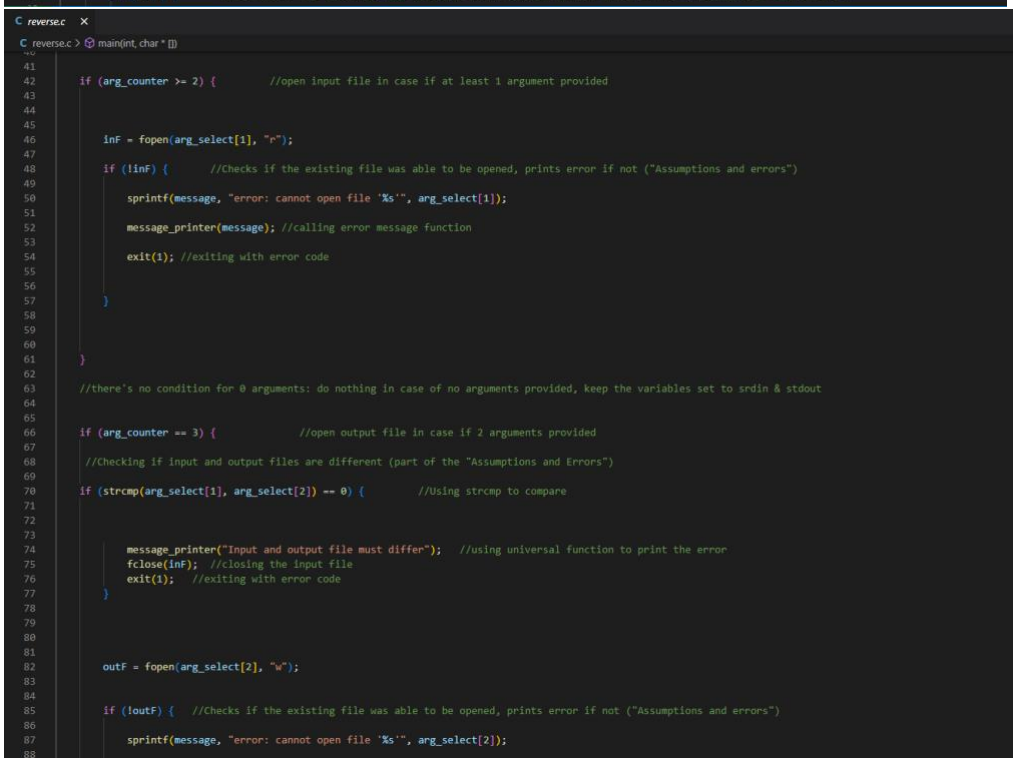




Screenshots of the reverse.c code:



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>                //importing libraries


void message_printer(const char *msg) {      //Function for printing errors and messages
    fprintf(stderr, "%s\n", msg);
}



int main(int arg_counter, char *arg_select[]) {                //defining main method, with future arguments

    FILE *inF = stdin;              //input variable, can be used for making manual input or for opening an input file, set to manual input
    FILE *outF = stdout;            //output variable, set to print the content by default, can be changed to a file
    char **lines_array = NULL;      //variable for storing the input lines
    size_t lines_counter = 0, lines_c = 10;        //Line counter. line capacity is also set for optimization (to be fixed)
    char *line = NULL;              //variable for single line initialization
    size_t length = 0;                 //variable for determining the length of the line, acts as a buffer size
    long read;                      //read variable, for reading the lines
    char message [50];  //variable for editing error messages







    //creating if-conditions for each program run use case

    if (arg_counter > 3) {  //Adding "usage" printout if too many args provided ("Assumptions and errors")

        message_printer("usage: reverse <input> <output>"); //using error printer to print

        exit(1);     //exiting with error code



    }


    if (arg_counter >= 2) {         //open input file in case if at least 1 argument provided



        inF = fopen(arg_select[1], "r");

        if (!inF) {       //Checks if the existing file was able to be opened, prints error if not ("Assumptions and errors")
```



```c
    if (arg_counter >= 2) {         //open input file in case if at least 1 argument provided



        inF = fopen(arg_select[1], "r");

        if (!inF) {       //Checks if the existing file was able to be opened, prints error if not ("Assumptions and errors")

            sprintf(message, "error: cannot open file '%s'", arg_select[1]);

            message_printer(message); //calling error message function

            exit(1); //exiting with error code



        }



    }
    //there's no condition for 0 arguments: do nothing in case of no arguments provided, keep the variables set to srdin & stdout


    if (arg_counter == 3) {             //open output file in case if 2 arguments provided

     //Checking if input and output files are different (part of the "Assumptions and Errors")

    if (strcmp(arg_select[1], arg_select[2]) == 0) {         //Using strcmp to compare



            message_printer("Input and output file must differ");   //using universal function to print the error
            fclose(inF);  //closing the input file
            exit(1);   //exiting with error code
        }



        outF = fopen(arg_select[2], "w");

        if (!outF) {   //Checks if the existing file was able to be opened, prints error if not ("Assumptions and errors")

            sprintf(message, "error: cannot open file '%s'", arg_select[2]);
```

reverse.c

reverse.c > main(int, char * [])

```c
        if (!outF) {   //Checks if the existing file was able to be opened, prints error if not (`Assumptions and errors`)

            sprintf(message, "error: cannot open file '%s'", arg_select[2]);

            message_printer(message); //calling error message function

            exit(1);  //exiting with error code


        }
    }









    lines_array = malloc(lines_c * sizeof(char *));   //Allocating memory for the array

    if (!lines_array) {             //print malloc failed error if allocation is failed ("Assumptions and errors")
        message_printer("malloc failed");    //calling message printer

        exit(1);  //exiting with error code
    }



    //Creating a while loop for reading the lines, adding each line to array

    while ((read = getline(&line, &length, inF)) != -1) {

        //Adding handling for large input files("Assumptions and errors")

        if (lines_counter >= lines_c) {  //Checking of line count exceeds the line capacity
            lines_c *= 4;   // multiply the capacity x4 in case if it's exceeded

            lines_array = realloc(lines_array, lines_c * sizeof(char *));  //Reallocate the memory for the array
```

reverse.c

reverse.c > main(int, char * [])

```c
            lines_array = realloc(lines_array, lines_c * sizeof(char *));  //Reallocate the memory for the array

            if (!lines_array) {              //print malloc failed error if allocation is failed ("Assumptions and errors")
                message_printer("malloc failed");            //calling message printer

                exit(1);    //exiting with error code
            }


        }

        lines_array[lines_counter++] = strdup(line);  //add the line to the array

    }









    // Printing the lines using for-loop
    for (long i = lines_counter - 1; i >= 0; i--) {


        fprintf(outF, "%s", lines_array[i]);  //printing

        free(lines_array[i]);    //Freeing the array
    }


    free(lines_array);   //Freeing the array
    free(line);       //Freeing the line variable

    if (inF != stdin) fclose(inF);  //closing input file

    if (outF != stdout) fclose(outF); //closing output file


    return 0;    //returning the function



}
```