My code:

include "get\_next\_line.h"

/\*

\*\*allocates memory, takes value position and length (ft\_strsub)

\*/

char \*mallocsizelensub(char const \*s, unsigned int start, size\_t len)

{

char \*str;

size\_t i;

if (s == NULL)

return (NULL);

if ((str = (char \*)malloc(sizeof(char) \* (len + 1))))

{

i = 0;

while (i < len)

{

str[i] = s[start + i];

i++;

}

str[i] = '\0';

return (str);

}

else

return (NULL);

}

/\*

\*\*builds a string by concatenating the elements in a set of data & adding a separator betwn them

\*\*(ft\_strjoin)

\*/

char \*fd\_joinbuff(char const \*s1, char const \*s2)

{

char \*str;

int i;

if ((s1 != NULL && s2 != NULL) && \

(str = (char \*)malloc(sizeof(char) \* (1 + ft\_strlen(s1) + ft\_strlen(s2)))))

{

i = 0;

while (\*s1)

{

str[i] = \*s1;

i++;

s1++;

}

while (\*s2)

{

str[i] = \*s2;

i++;

s2++;

}

str[i] = '\0';

return (str);

}

else

return (NULL);

}

/\*

\*\*LINE EVALAUTATION:

\*\*\*-Reads file line by line up to \n.

\*\*\*-Finds the length of a line.

\*\*\*-Creates a new substring allocates memory

\*/

static int evaluatesentence(char \*\*s, char \*\*line)

{

int len;

char \*temp;

len = 0;

//while string len is not a newline and not a NULL increment

while ((\*s)[len] != '\n' && (\*s)[len] != '\0')

len++;

//if string len is a newline then the following:

if ((\*s)[len] == '\n')

{

//allocates memory, takes value position and length (ft\_strsub)

\*line = mallocsizelensub(\*s, 0, len);

//ft\_strdup duplicates the strings address includes the length and the null term

temp = ft\_strdup(&((\*s)[len + 1]));

//frees the strings memory

free(\*s);

\*s = temp;

//if string zero equals NULL then delete,free & set to NULL

if ((\*s)[0] == '\0')

ft\_strdel(s);

}

//if the \*line equals the duplicated string delete,free & set to NULL

else

{

\*line = ft\_strdup(\*s);

ft\_strdel(s);

}

//return 1 means the line has been read.

return (1);

}

/\*

\*\*OUTPUTS:

\*\*\*-Has the line been read -RETURN (1)

\*\*\*-If any error(nothing in file) -RETURN (-1)

\*\*\*-If the entire file has been Read -RETURN (0)

\*/

static int output(char \*\*s, char \*\*line, int ret, int fd)

{

//if the return is less than 0 return -1

if (ret < 0)

return (-1);

//if the return is equal to zero or the string FD equals NULL, return 0

else if (ret == 0 && s[fd] == NULL)

return (0);

else

// or else go back to the function evaluate function to get a return value

return (evaluatesentence(&s[fd], line));

}

/\*

\*\*READS:

\*\*\*-Reads the file

\*\*\*-Reads and gives it a buff size

\*\*\*-Reading line by line, memory allocation & freeing it

\*/

int get\_next\_line(const int fd, char \*\*line)

{

int ret;

static char \*s[FD\_SIZE];

char buff[BUFF\_SIZE + 1];

char \*temp;

//if file descriptor is less than 0 or the line is equal to NULL RETURN -1

if (fd < 0 || line == NULL)

return (-1);

//while return read fd, buff, BUFF\_SIZE is greater than zero. return to the function output.

while ((ret = read(fd, buff, BUFF\_SIZE)) > 0)

{

//buff return is NULL

buff[ret] = '\0';

//if string file descriptor is NULL

if (s[fd] == NULL)

//string fd is ft\_strdup duplicating the strings address includes the length

s[fd] = ft\_strdup(buff);

else

{

//temporary is concatenating string FD & buff freeing it

temp = fd\_joinbuff(s[fd], buff);

free(s[fd]);

//string fd is temporary file stored space

s[fd] = temp;

}

// searches the occurrence of newline in the string file descriptor and breaks the loop

if (ft\_strchr(s[fd], '\n'))

break ;

}

return (output(s, line, ret, fd));

}