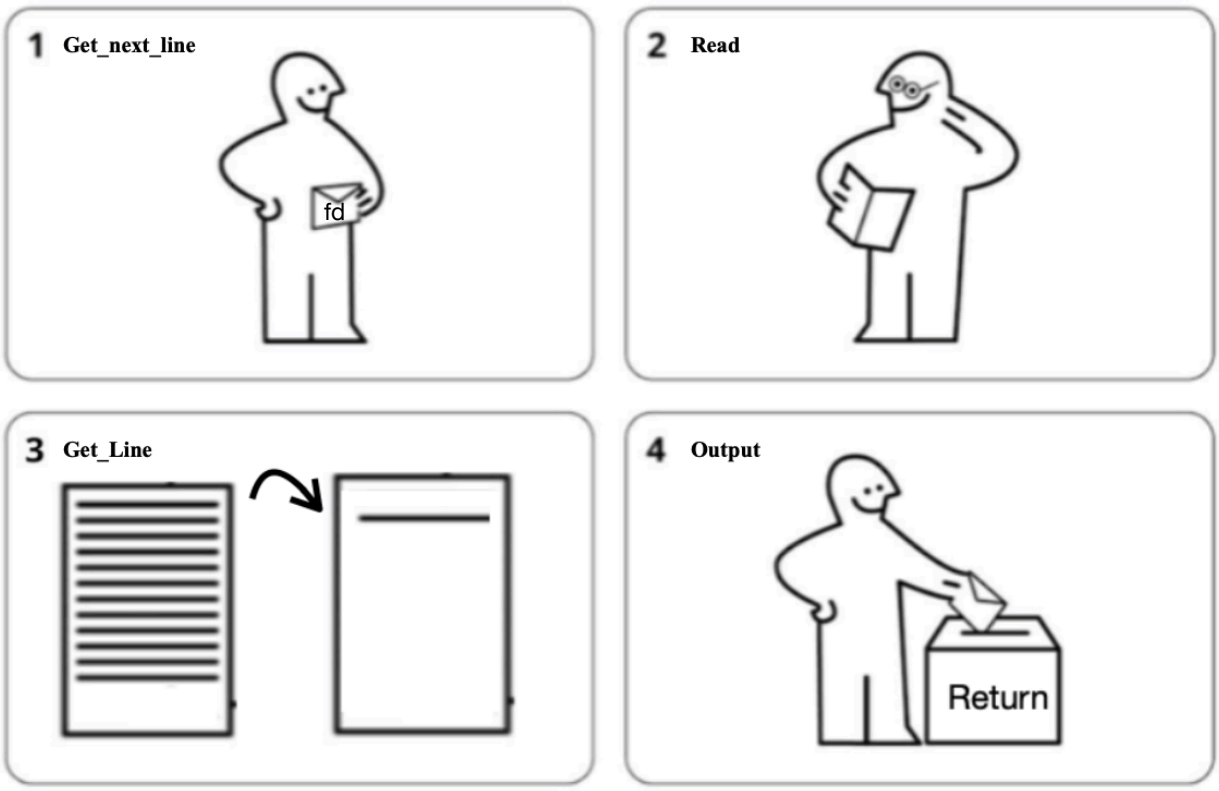
GET_NEXT_LINE

IDEA



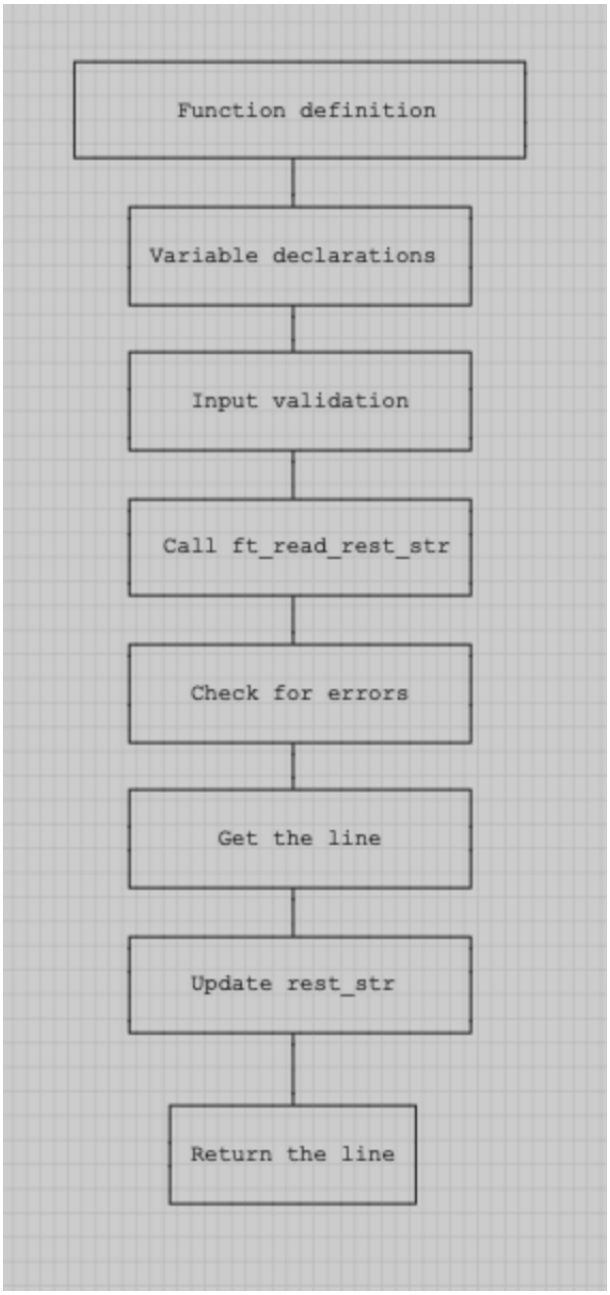
Imagine you have a big book with lots of stories, and you want to read one story at a time.

There's a magical helper named "get_next_line" who helps you do that.

Each time you ask for a story, the helper reads a little bit from the book and looks for the end of a story.
When they find the end, they give you that story to read.

Sometimes, the helper reads too much and has extra words left over.
They save those words and check them first the next time you ask for a story.
If they find a whole story in the leftover words, they give it to you.
If not, they read more from the book and find the next story for you.

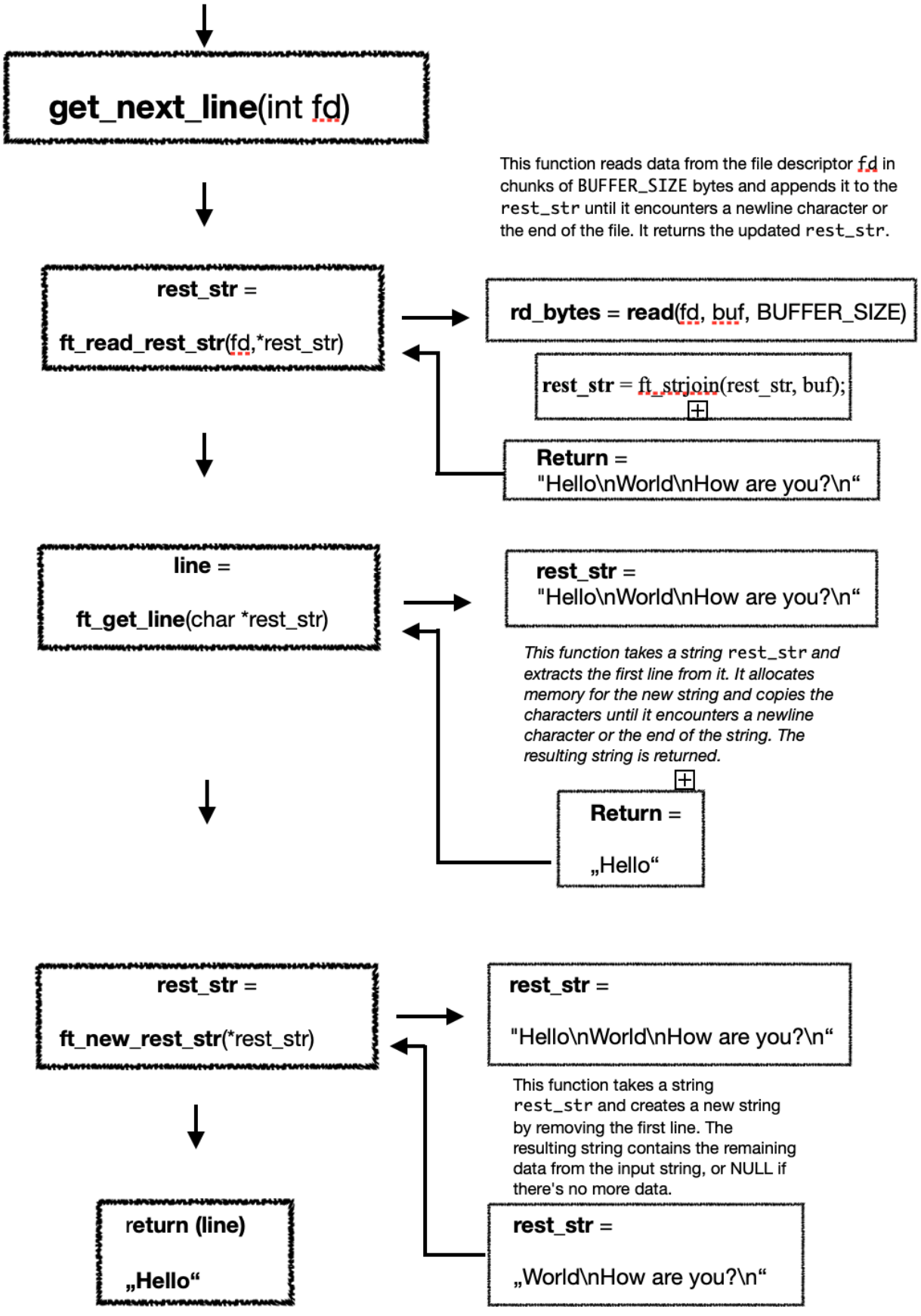
The helper has three special tools, „ft_read_rest_str“, "ft_get_line"
and „ft_new_rest_str“ , to make sure they give you the right story and save any extra words for later.
When there are no more stories left in the book, the helper tells you there’s nothing more to read.

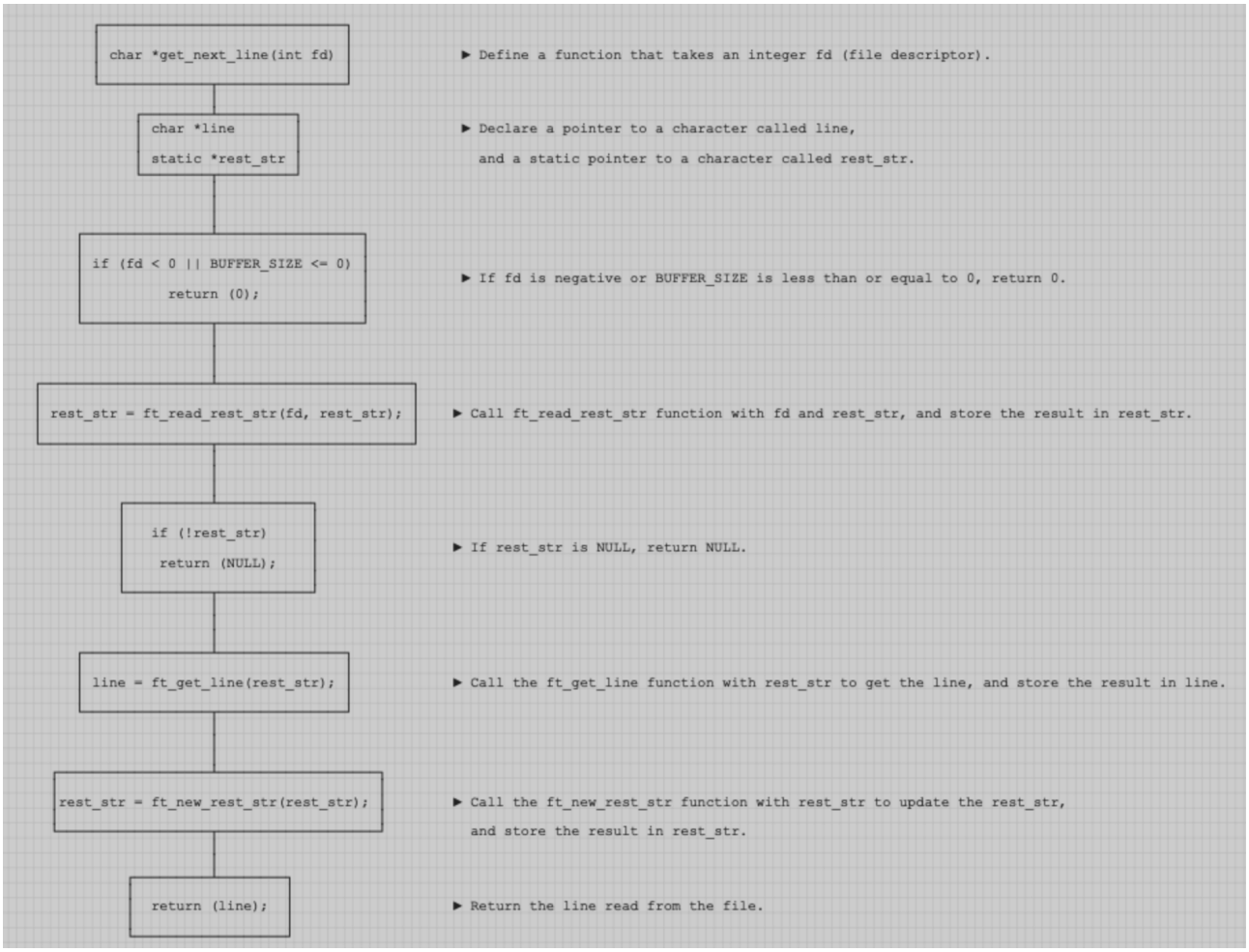


1. `get_next_line` *calls* `ft_read_rest_str` *to read the file content and update the* `rest_str` *variable.*
2. `ft_read_rest_str` *reads the file content into a buffer and appends it to* `rest_str`*.*
3. `get_next_line` *calls* `ft_get_line` *to extract the current line from* `rest_str`*.*
4. `get_next_line` *calls* `ft_new_rest_str` *to update* `rest_str` *by removing the line that was just read.*

Text in Textfile

- "Hello\nWorld\nHow are you?\n“.





`get_next_line` is a function that reads a file descriptor `fd` and returns the next line in the file.

The function reads data from the file descriptor in chunks of `BUFFER_SIZE` bytes, and returns the next line of data whenever it encounters a newline character.

The function uses a static variable `rest_str` to keep track of any data that has been read from the file descriptor but not yet returned to the caller.

When `get_next_line` is called again, it first checks whether there is any data in `rest_str`.

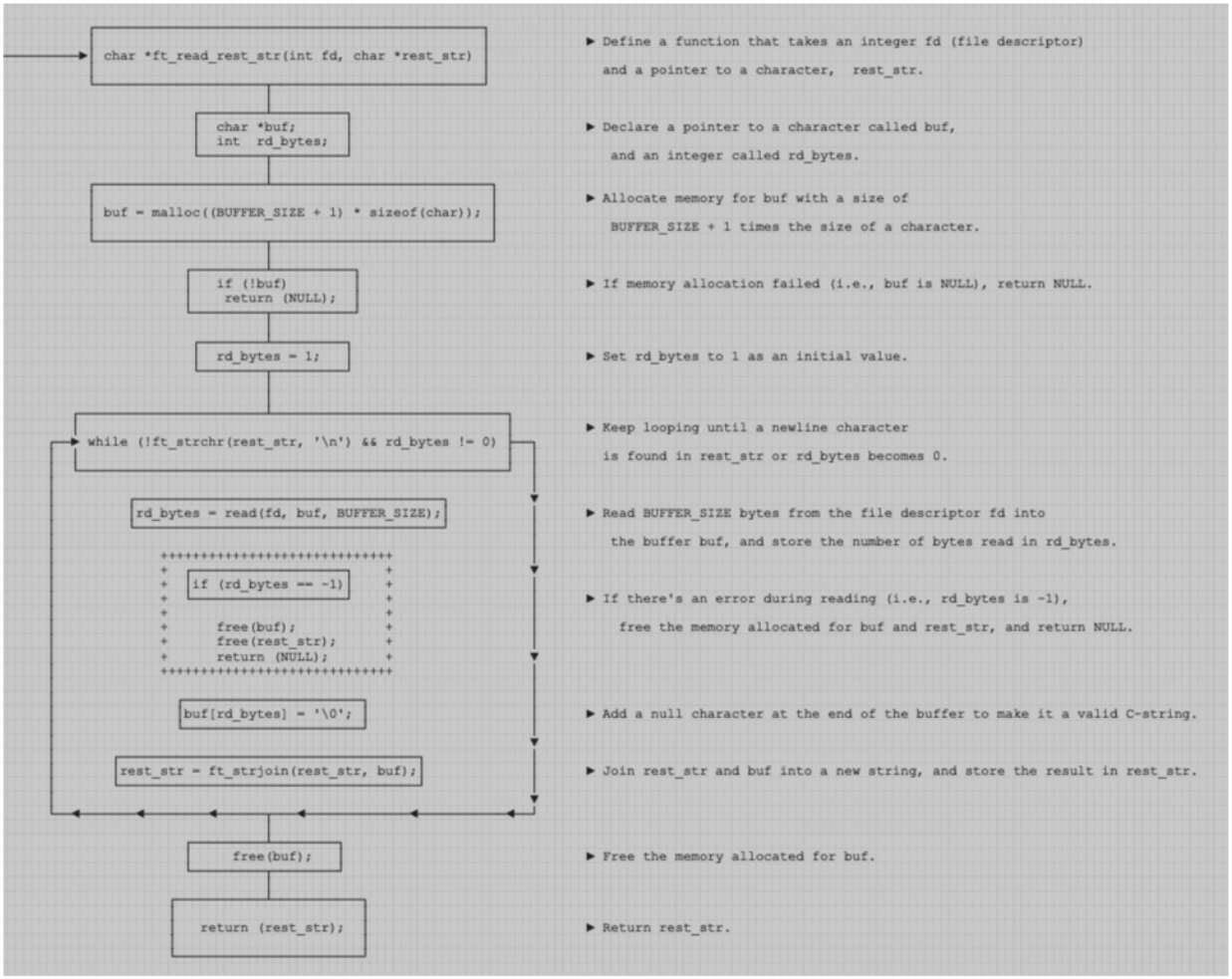
If there is, it extracts the next line of data and returns it.
If not, it reads data from the file descriptor and extracts the next line of data.

The function first calls `ft_read_rest_str` to read any remaining data from `rest_str` and concatenate it with the next chunk of data from the file descriptor.

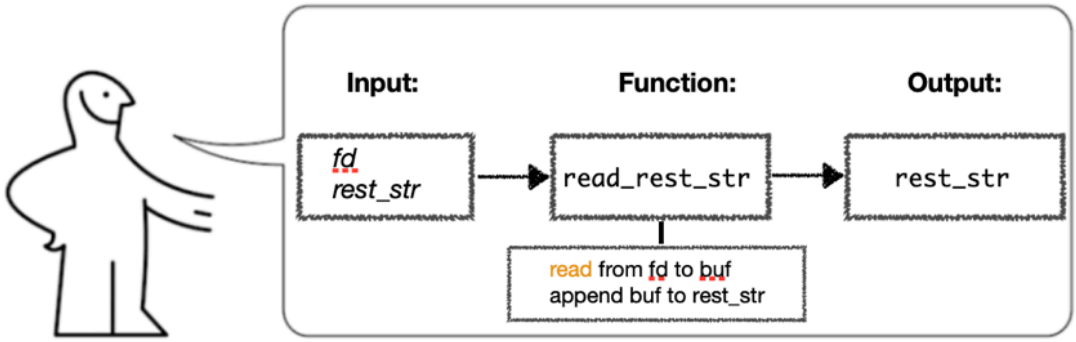
It then calls `ft_get_line` to extract the next line of data from the concatenated string, and updates `rest_str` to contain any remaining data.

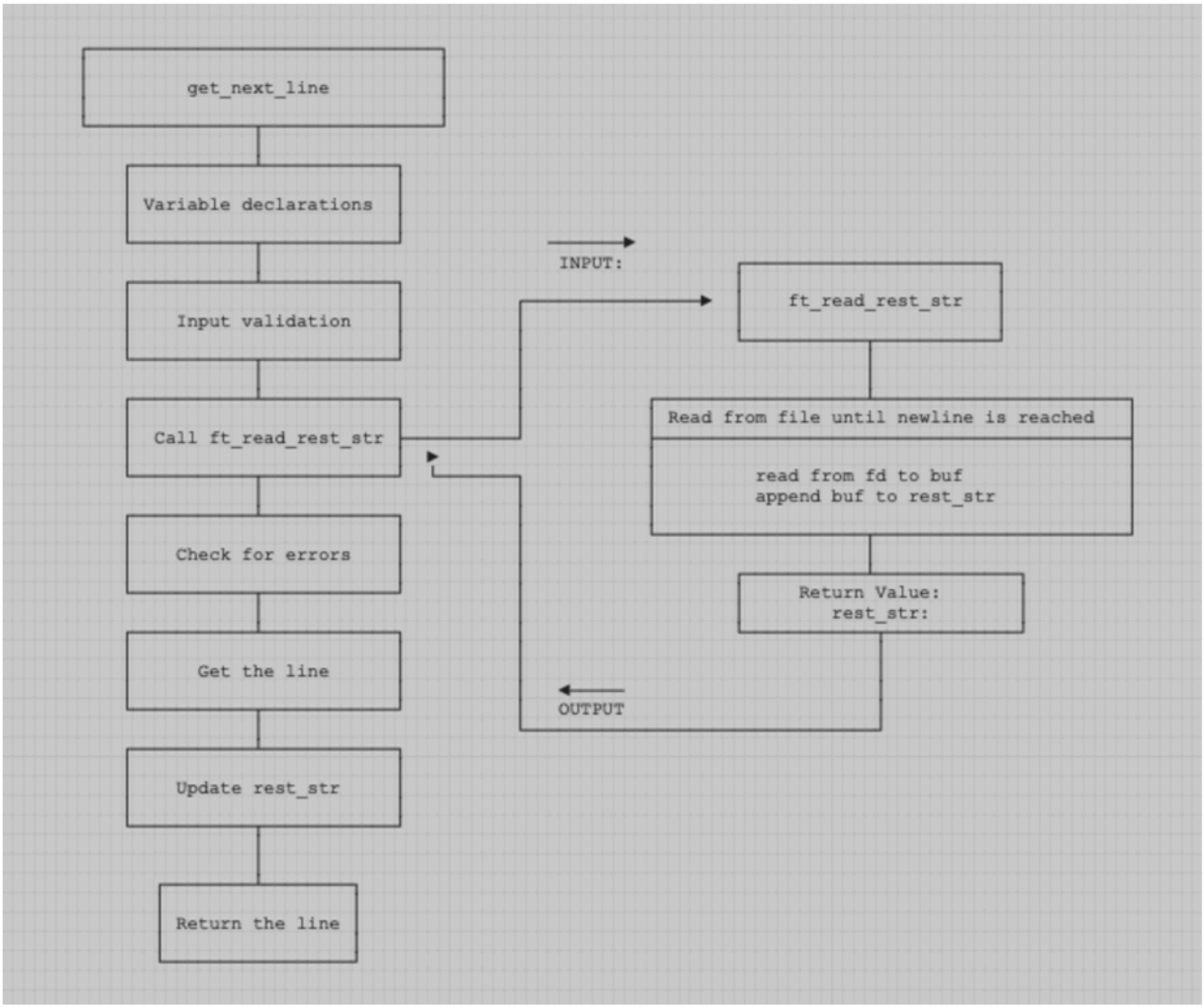
Finally, the function returns the extracted line of data. If there is no more data to be read from the file descriptor, the function returns `NULL`.

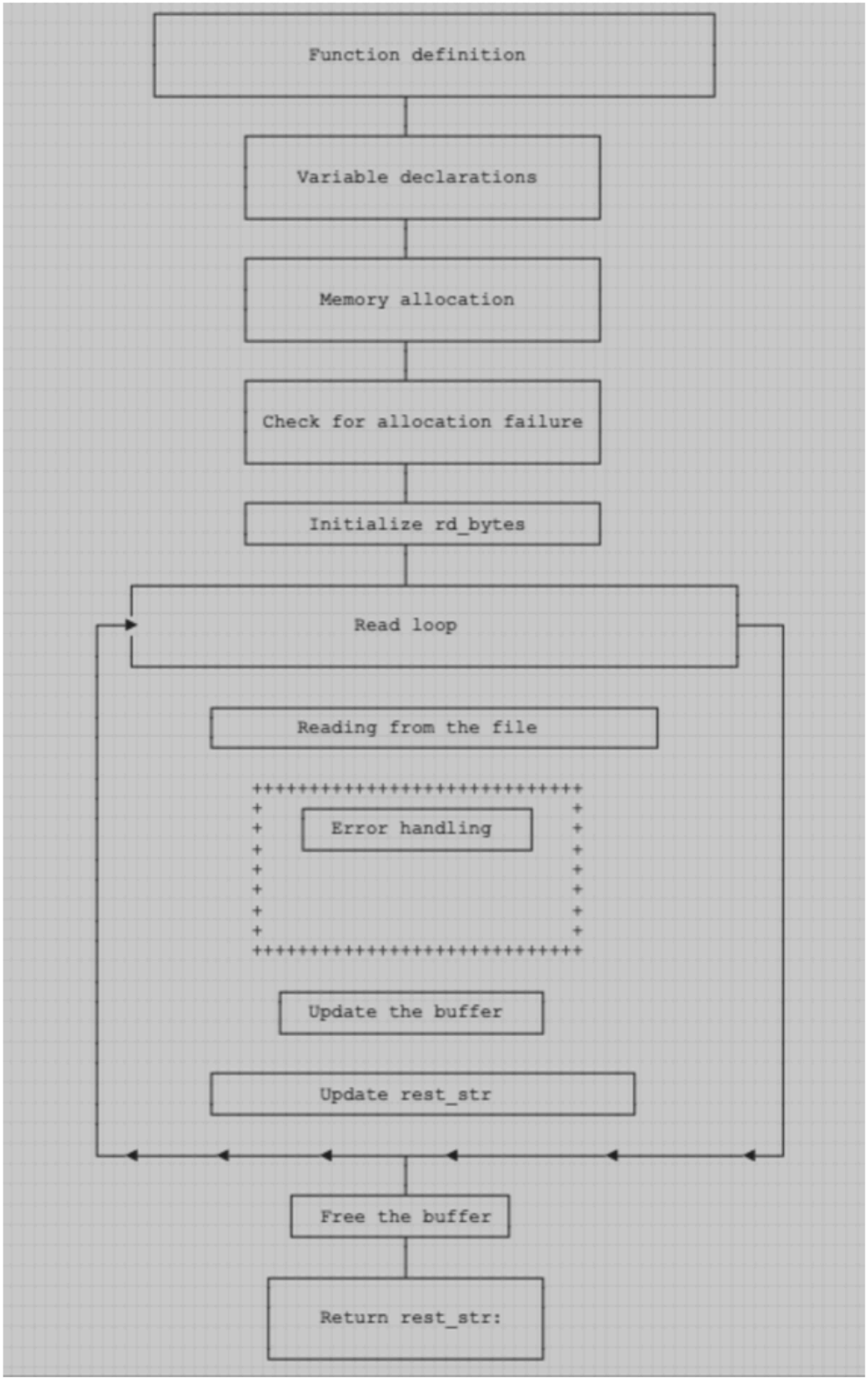
ft_read_rest_str



1. When `get_next_line` is called, it first calls `ft_read_rest_str` with the file descriptor `fd` and the static variable `rest_str`.
2. `ft_read_rest_str` reads the file content into a buffer in chunks of `BUFFER_SIZE` bytes using the `read` function, and appends the contents of the buffer to `rest_str`.
3. The function checks for errors during the read operation, such as if `read` returns -1. In such a case, the function frees the buffer and `rest_str`, and returns `NULL`.
4. The function continues to read from the file descriptor and append the content to `rest_str` until it reaches the end of the file or a newline character is encountered.
5. If the function reaches the end of the file and there is no newline character, it returns `rest_str` as is. Otherwise, it returns a substring of `rest_str` that includes everything before the newline character.
6. Back in `get_next_line`, the result of `ft_read_rest_str` is stored in `rest_str`, which now contains any remaining content from previous read operations, as well as any new content read from the file descriptor.


Input:
Function:
Output:
fd
rest_str
read_rest_str
rest_str
read from fd to buf
append buf to rest_str


get_next_line
Variable declarations
Input validation
Call ft_read_rest_str
Check for errors
Get the line
Update rest_str
Return the line
INPUT:
OUTPUT
ft_read_rest_str
Read from file until newline is reached
read from fd to buf
append buf to rest_str
Return Value:
rest_str:


Function definition
Variable declarations
Memory allocation
Check for allocation failure
Initialize rd_bytes
Read loop
Reading from the file
Error handling
Update the buffer
Update rest_str
Free the buffer
Return rest_str:

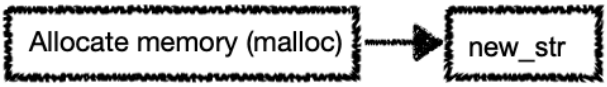
ft_get_line

7. Next, `get_next_line` calls `ft_get_line` with `rest_str` to extract the current line from `rest_str`. This function searches for the first newline character in `rest_str` and returns a substring up to that point.

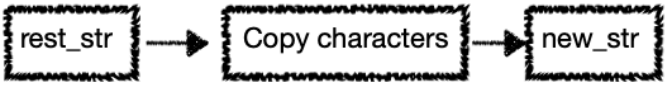
- The function loops through the characters of `rest_str` until a newline character is found or the end of the string is reached, and stores the index of the newline character in `i`.
- **If no newline character is found, the function frees the memory allocated for `rest_str` and returns `NULL`.**
- Otherwise, the function allocates memory for `str` to hold the line, including space for a null terminator and the newline character.
- The function then copies characters from `rest_str` to `str` until the newline character is found or the end of the string is reached.
- If a newline character is found, the function adds it to the line string and updates the counter.
- Finally, the function adds a null terminator to the end of the line string and returns it.
- Overall, `ft_get_line` plays a crucial role in `get_next_line` by extracting the current line from `rest_str` and returning it for further processing, such as printing to standard output or writing to a file.

1. Count characters until newline or end of string:

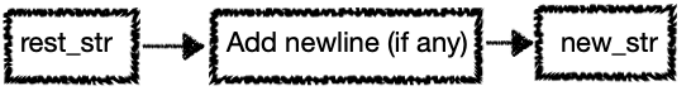
2. Allocate memory for new string (i + 2):



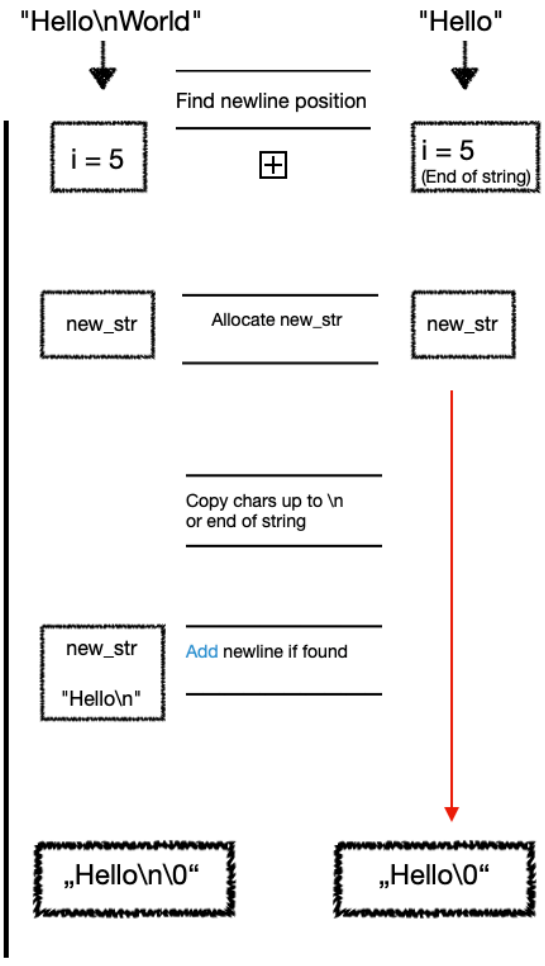
3. Copy characters to new_str:

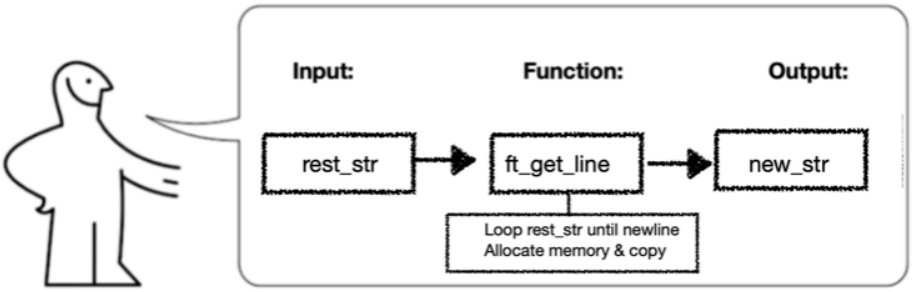


4. Add newline if found in rest_str:



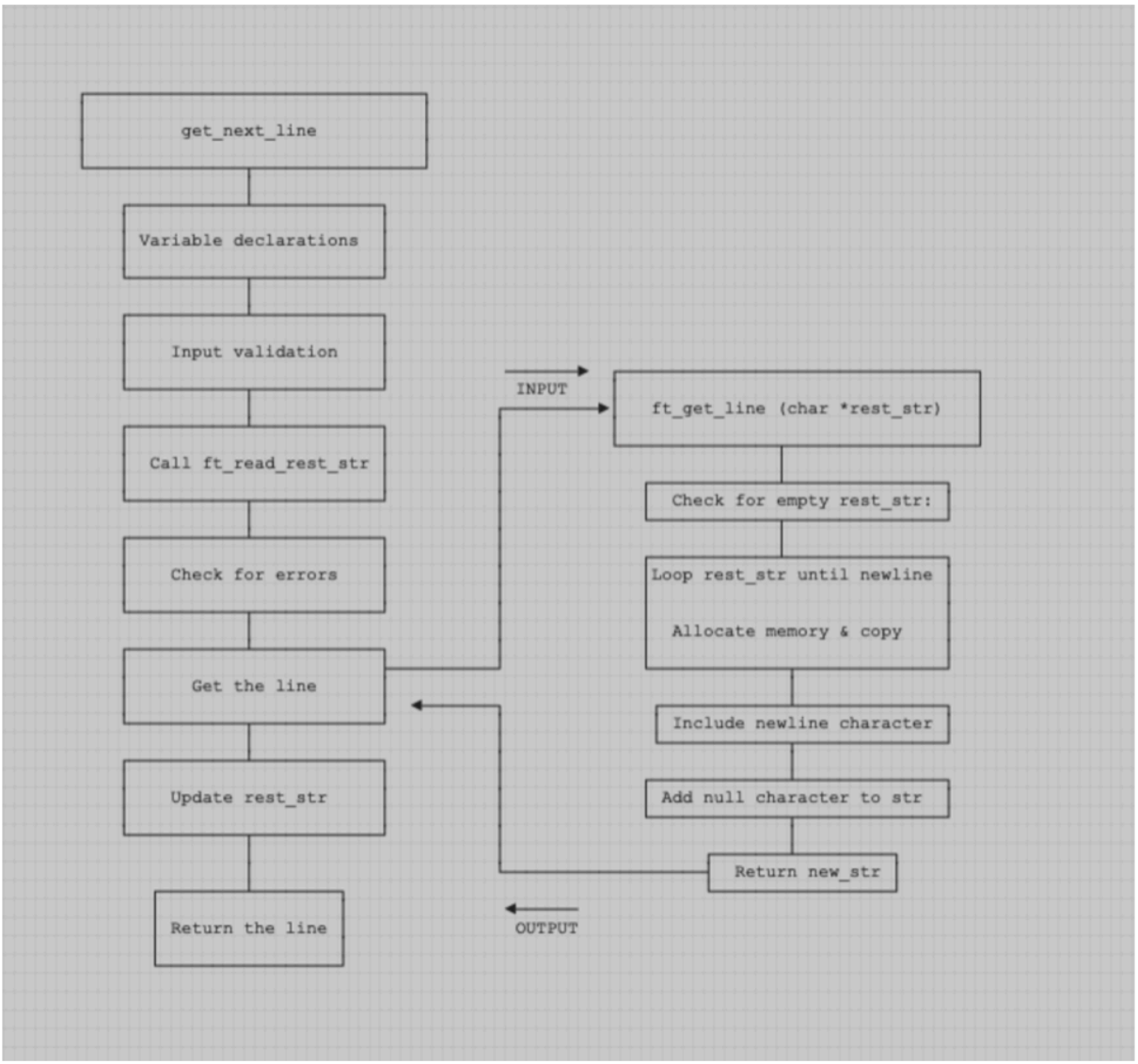
5. Add null terminator to new_str:

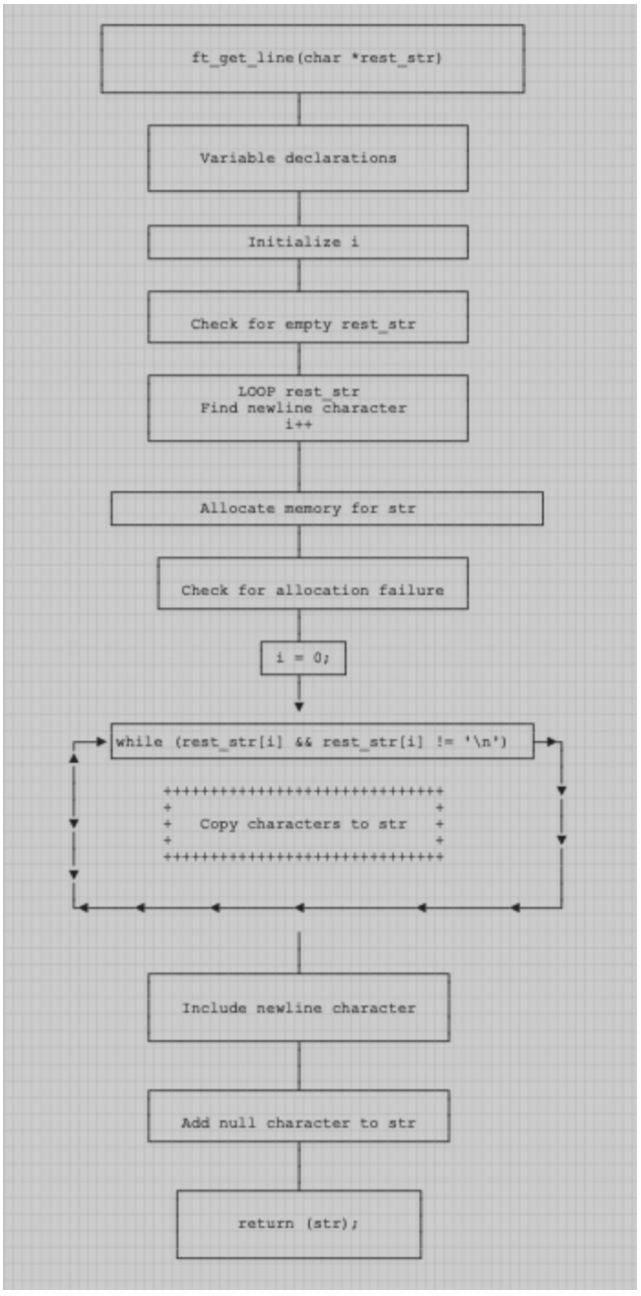


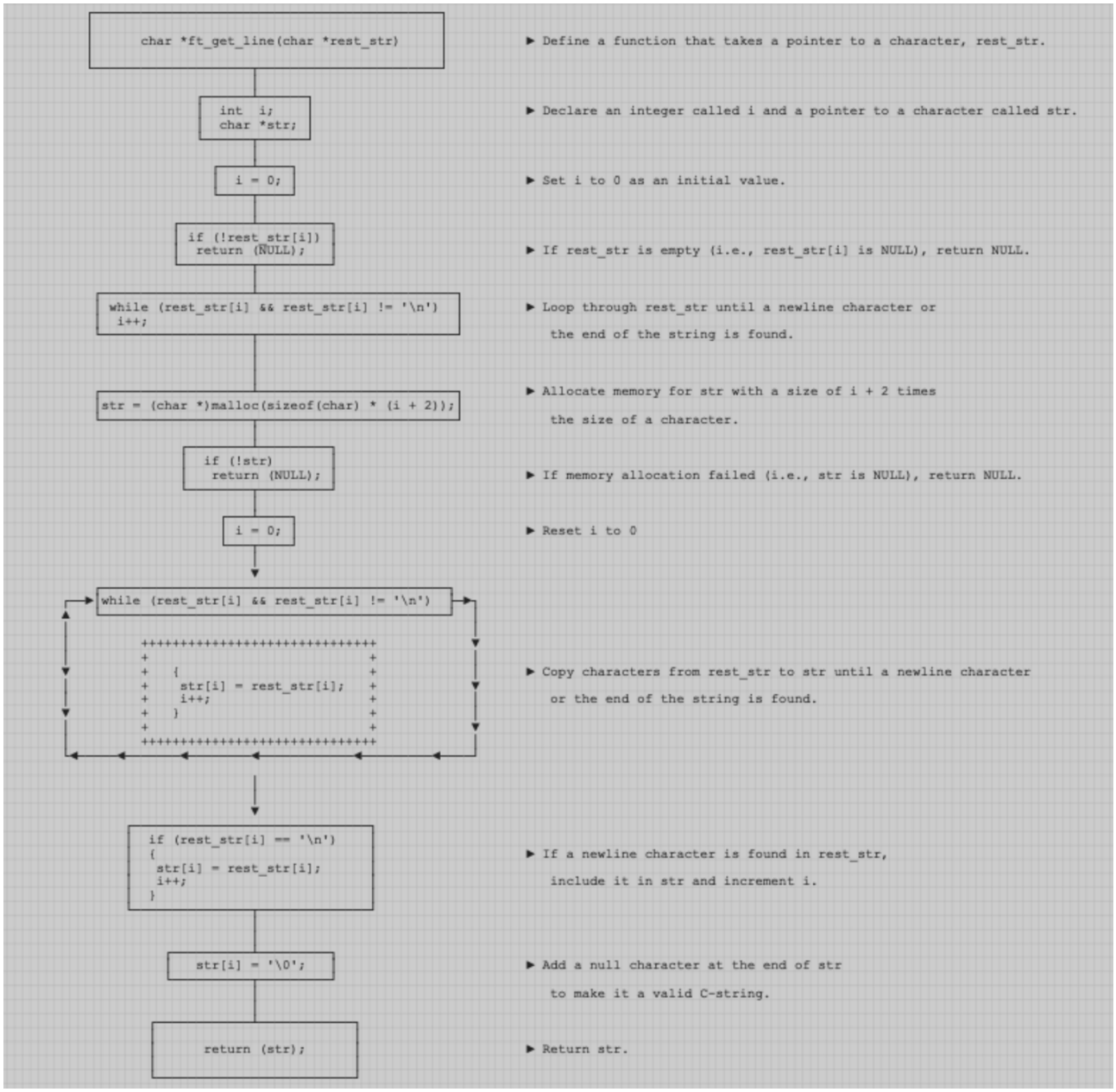


Summary:

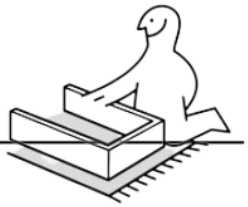
The function starts by counting the number of characters in the input string `rest_str` up to a newline character or the end of the string. Then, it allocates memory for a new string `new_str` with the size of the counted characters plus 2. The characters are copied from `rest_str` to `new_str`, and if a newline character was found in `rest_str`, it is added to `new_str`. Finally, the null terminator ('\0') is added to the end of `new_str`.




ft_get_line(char *rest_str)
Variable declarations
Initialize i
Check for empty rest_str
LOOP rest_str
Find newline character
i++
Allocate memory for str
Check for allocation failure
i = 0;
while (rest_str[i] && rest_str[i] != '\n')
Copy characters to str
Include newline character
Add null character to str
return (str);


char *ft_get_line(char *rest_str)
▶ Define a function that takes a pointer to a character, rest_str.
int i;
char *str;
▶ Declare an integer called i and a pointer to a character called str.
i = 0;
▶ Set i to 0 as an initial value.
if (!rest_str[i])
return (NULL);
▶ If rest_str is empty (i.e., rest_str[i] is NULL), return NULL.
while (rest_str[i] && rest_str[i] != '\n')
i++;
▶ Loop through rest_str until a newline character or
the end of the string is found.
str = (char *)malloc(sizeof(char) * (i + 2));
▶ Allocate memory for str with a size of i + 2 times
the size of a character.
if (!str)
return (NULL);
▶ If memory allocation failed (i.e., str is NULL), return NULL.
i = 0;
▶ Reset i to 0
while (rest_str[i] && rest_str[i] != '\n')
{
str[i] = rest_str[i];
i++;
}
▶ Copy characters from rest_str to str until a newline character
or the end of the string is found.
if (rest_str[i] == '\n')
{
str[i] = rest_str[i];
i++;
}
▶ If a newline character is found in rest_str,
include it in str and increment i.
str[i] = '\0';
▶ Add a null character at the end of str
to make it a valid C-string.
return (str);
▶ Return str.

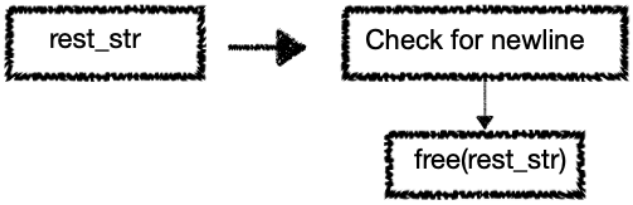
ft_new_rest_str



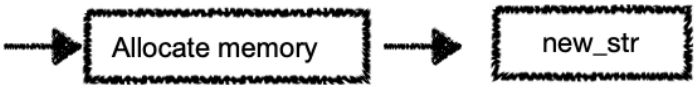
`ft_new_rest_str` takes a character string `rest_str` as input and returns a new string with the content after the first newline character, if present, or `NULL` if no newline character is found. The original `rest_str` is freed in the process.

1. Count characters until newline or end of string:

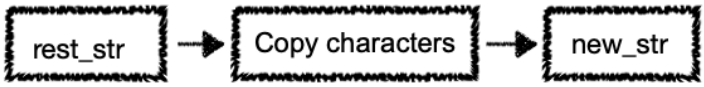
2. Check if no newline found and free rest_str:



3. Allocate memory for new string (ft_strlen(rest_str) - i + 1):



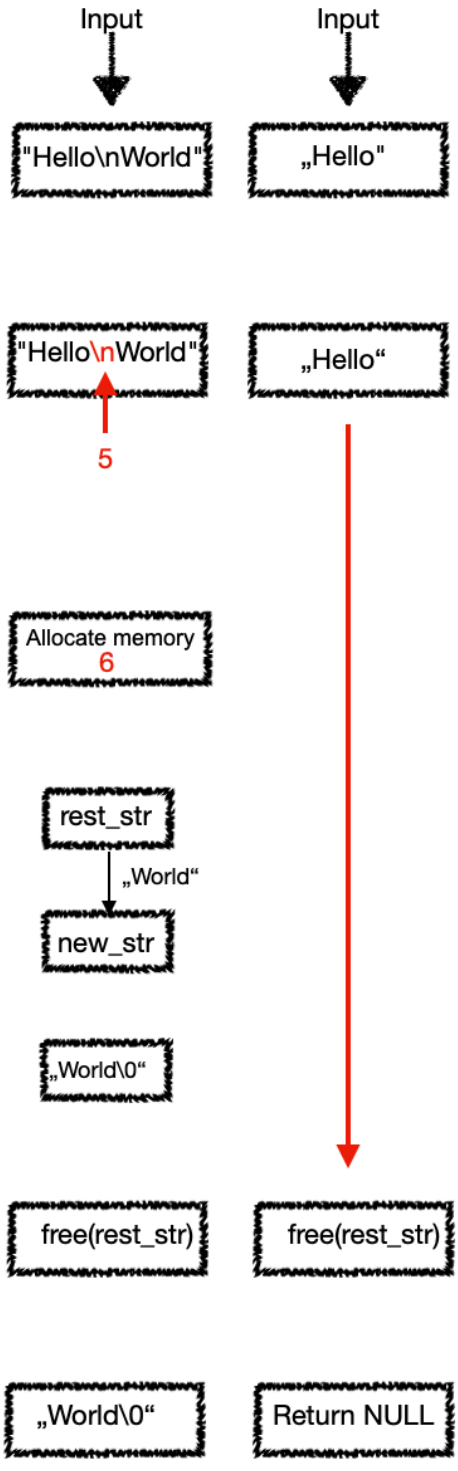
4. Copy characters after newline to new_str:



5. Add null terminator to new_str:

6. Free original rest_str:

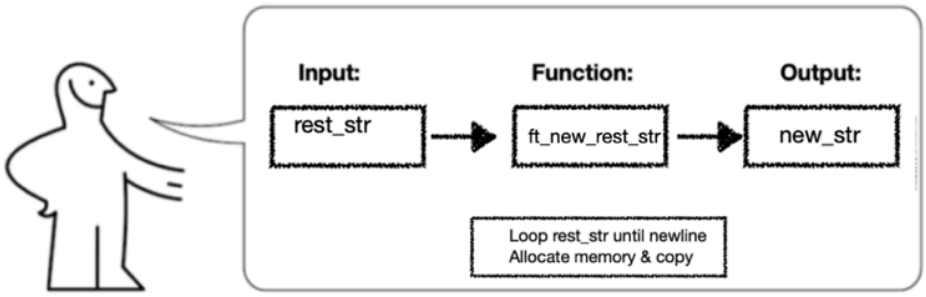
7. Return new_str , Example: return "World":

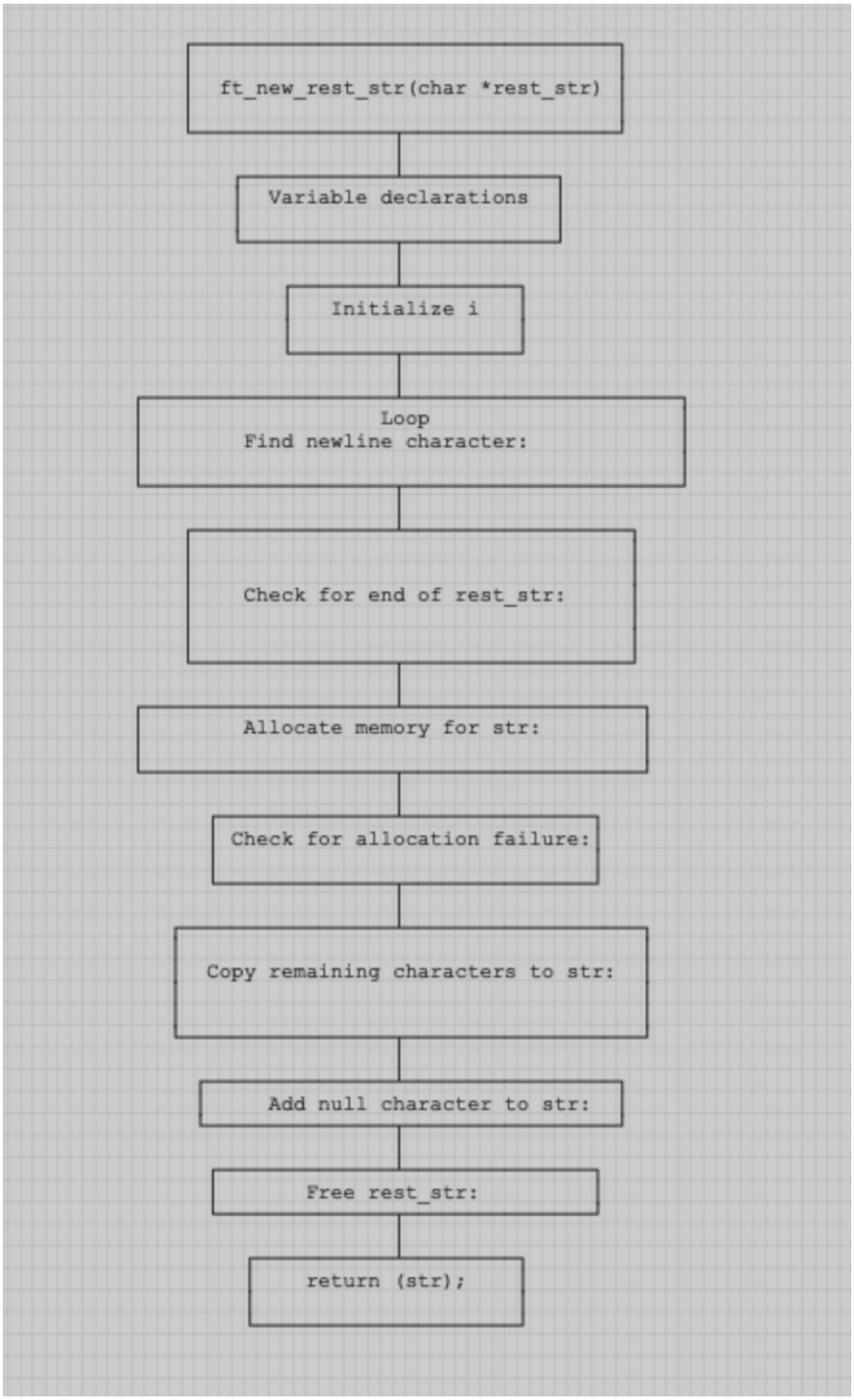


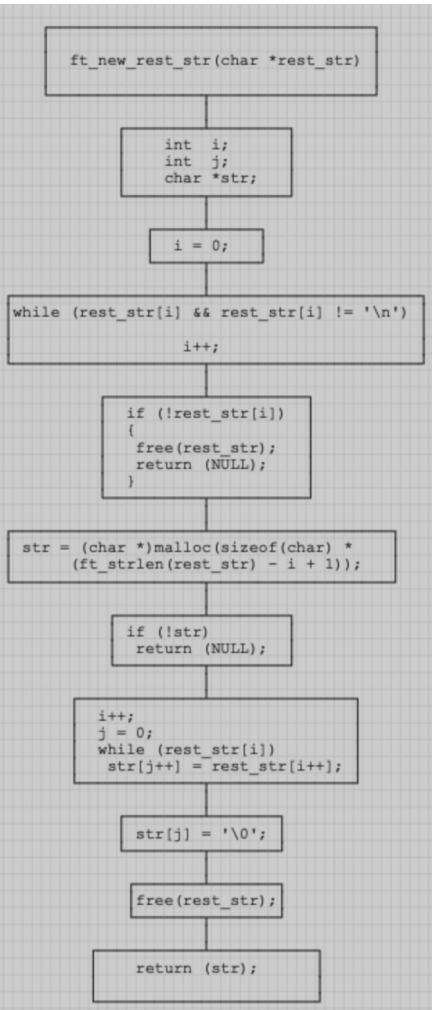
The `ft_get_line` function extracts the current line from the `rest_str`, while the `ft_new_rest_str` function updates `rest_str` by removing the line that was just read. Both functions work together with the previously explained `get_next_line` and `ft_read_rest_str` functions to read a file line by line.

For example, if the input `rest_str` is "Hello\nWorld", the function will return a new string "World". The original `rest_str` will be freed during the process.

If the input `rest_str` is "Hello". The function finds the position of the newline character, but in this case, there is no newline in the input string. After checking for the newline, the function frees the original `rest_str` and returns `NULL`.




ft_new_rest_str(char *rest_str)
Variable declarations
Initialize i
Loop
Find newline character:
Check for end of rest_str:
Allocate memory for str:
Check for allocation failure:
Copy remaining characters to str:
Add null character to str:
Free rest_str:
return (str);



- Define a function that takes a pointer to a character, rest_str.
- Declare two integers called i and j, and a pointer to a character called str.
- Set i to 0 as an initial value.
- Loop through rest_str until a newline character or the end of the string is found.
- If the end of rest_str is reached, free the memory allocated for rest_str and return NULL.
- Allocate memory for str with a size of the length of rest_str minus i plus 1, times the size of a character.
- If memory allocation failed (i.e., str is NULL), return NULL.
- Increment i to move past the newline character.
- Initialize j to 0,
- copy the remaining characters from rest_str to str.
- Add a null character at the end of str to make it a valid C-string
- Free the memory allocated for rest_str.
- Return str:

Let's go through the `get_next_line` function with an example file containing the string **"Hello\nWorld\nHow are you?\n".**
We'll assume that the file has already been opened and the file descriptor `fd` is available.

Here's a breakdown of how the function works for this example:

1. Call `get_next_line(fd)` for the first time:
    - `rest_str` is initially NULL, so `ft_read_rest_str` is called to read data from the file.
    - Since there is a newline character after "Hello", `rest_str` is set to "Hello\nWorld\nHow are you?\n".
    - `ft_get_line` extracts "Hello\n" from `rest_str` and returns it as the first line.
    - `ft_new_rest_str` updates `rest_str` to "World\nHow are you?\n".
    - The first call to `get_next_line` returns "Hello\n".
2. Call `get_next_line(fd)` for the second time:
    - The current `rest_str` is "World\nHow are you?\n".
    - `ft_get_line` extracts "World\n" from `rest_str` and returns it as the second line.
    - `ft_new_rest_str` updates `rest_str` to "How are you?\n".
    - The second call to `get_next_line` returns "World\n".
3. Call `get_next_line(fd)` for the third time:
    - The current `rest_str` is "How are you?\n".
    - `ft_get_line` extracts "How are you?\n" from `rest_str` and returns it as the third line.
    - `ft_new_rest_str` updates `rest_str` to NULL, as there is no more data remaining.
    - The third call to `get_next_line` returns "How are you?\n".
4. Call `get_next_line(fd)` for the fourth time:
    - The current `rest_str` is NULL, which means there's no more data to read.
    - `get_next_line` returns NULL, indicating the end of the file.

In this example, the `get_next_line` function successfully reads and returns each line in the file one by one: "Hello\n", "World\n", and "How are you?\n". When called again, it returns NULL, indicating that there is no more data to read.

In the example "Hello\nWorld\nHow are you?\n", let's assume that the BUFFER_SIZE **is large enough** to read the entire content of the file at once. Here's a step-by-step breakdown of how `ft_read_rest_str` processes the data:

1. `buf` is allocated with a size of BUFFER_SIZE + 1.
2. `rd_bytes` is set to 1 to enter the while loop.
3. Inside the loop, `read(fd, buf, BUFFER_SIZE)` reads data from the file and stores it in `buf`. In this case, it reads "Hello\nWorld\nHow are you?\n". The variable `rd_bytes` is set to the number of bytes read.
4. Since `rd_bytes` is not -1, the error condition is not triggered.
5. `buf[rd_bytes]` is set to the null terminator, marking the end of the string.
6. `rest_str` is updated with the data read from the file using `ft_strjoin(rest_str, buf)`.
7. If `rest_str` was initially NULL, it will now contain the data read from the file. In this case, since the entire content of the file was read, `rest_str` will be set to "Hello\nWorld\nHow are you?\n".
8. If the file was larger and the content was read in chunks, the `rest_str` would be updated incrementally with each chunk of data read until a newline character is encountered or the end of the file is reached.

After this step, the `ft_read_rest_str` function returns the updated `rest_str` to the caller, which in our example would be the entire file content: "Hello\nWorld\nHow are you?\n“

With a `BUFFER_SIZE` of 3, the `ft_read_rest_str` function will read the example file "Hello\nWorld\nHow are you?\n" in smaller chunks.
Here's a step-by-step breakdown of how `ft_read_rest_str` processes the data in this scenario:

1. Call `get_next_line(fd)` for the first time:
    - `buf` is allocated with a size of `BUFFER_SIZE + 1`, which is 4 bytes.
    - `rd_bytes` is set to 1 to enter the while loop.
    - Inside the loop, `read(fd, buf, BUFFER_SIZE)` reads 3 bytes from the file and stores it in `buf`. In this case, it reads "Hel". The variable `rd_bytes` is set to the number of bytes read.
    - Since `rd_bytes` is not -1, the error condition is not triggered.
    - `buf[rd_bytes]` is set to the null terminator, marking the end of the string.
    - `rest_str` is updated with the data read from the file using `ft_strjoin(rest_str, buf)`. Now, `rest_str` is "Hel".
    - The loop continues because no newline character was encountered.
    - In the next iteration, the `read` function reads the next 3 bytes, "lo\n".
    - The `rest_str` is updated using `ft_strjoin(rest_str, buf)`, which appends the new chunk to the existing `rest_str`. Now, `rest_str` becomes "Hello\n". The loop exits as a newline character is encountered in the updated `rest_str`.

Continuing from where we left off, after the loop in `ft_read_rest_str` exits, the function returns the updated `rest_str`, which now contains "Hello\n". Now, we can proceed with the subsequent steps:

1. Call `get_next_line(fd)` for the first time:
    - `ft_get_line` extracts "Hello\n" from `rest_str` and returns it as the first line.
    - `ft_new_rest_str` updates `rest_str` to an empty string (since there's no more data after "Hello\n" in the current `rest_str`).
    - The first call to `get_next_line` returns "Hello\n".
2. Call `get_next_line(fd)` for the second time:
    - Since `rest_str` is empty, `ft_read_rest_str` is called again to read more data from the file.
    - The next 3 bytes, "Wor", are read and appended to `rest_str`, which now becomes „Wor".
    - The loop in `ft_read_rest_str` continues because no newline character was encountered.
    - In the next iteration, the `read` function reads the next 3 bytes, "ld\n". The `rest_str` is updated to "World\n". The loop exits as a newline character is encountered in the updated `rest_str`.
    - `ft_get_line` extracts "World\n" from `rest_str` and returns it as the second line.
    - `ft_new_rest_str` updates `rest_str` to an empty string (since there's no more data after "World\n" in the current `rest_str`).
    - The second call to `get_next_line` returns „World\n".

3. Call `get_next_line(fd)` for the third time:
    - Since `rest_str` is empty, `ft_read_rest_str` is called again to read more data from the file.
    - The next 3 bytes, "How", are read and appended to `rest_str`, which now becomes "How".
    - The loop in `ft_read_rest_str` continues because no newline character was encountered.
    - In the next iterations, the `read` function reads the next chunks " ar", "e yo", "u?\n" consecutively. After each iteration, the `rest_str` is updated accordingly: "How ar",

"How are yo", "How are you?\n". The loop exits as a newline character is encountered in the updated `rest_str`.

- `ft_get_line` extracts "How are you?\n" from `rest_str` and returns it as the third line.
- `ft_new_rest_str` updates `rest_str` to NULL, as there is no more data remaining.
- The third call to `get_next_line` returns "How are you?\n“.
- Call `get_next_line(fd)` for the fourth time:
- The current `rest_str` is NULL, which means there's no more data to read.
- `get_next_line` returns NULL, indicating the end of the file.

In this example, the `get_next_line` function successfully reads and returns each line in the file one by one with `BUFFER_SIZE` set to 3: "Hello\n", "World\n", and "How are you?\n". When called again, it returns NULL, indicating that there is no more data to read.