get next line

▼ GNL CODE

```
#include "get_next_line.h"

int find_newline_index(char *buff)
{
  int i;

  i = 0;
  while (buff[i])
  {
    if (buff[i] == '\n')
      return (i);
    i++;
  }
  return (-1);
}

size_t  ft_strlen(char *str)
{
  size_t  i;

  i = 0;
  while (str[i])
    i++;
  return (i);
}

char  *get_remain_str(char *str, int nl_index, int i)
{
  char  *remain_str;

  if (nl_index == -1)
  {
    free(str);
    return (NULL);
  }
  nl_index++;
  remain_str = malloc(ft_strlen(str) - nl_index + 1);
  if (!remain_str)
    return (NULL);
  while (str[nl_index + i])
  {
    remain_str[i] = str[nl_index + i];
    i++;
  }
```

```
  remain_str[i] = '\0';
  free(str);
  return (remain_str);
}

char  *get_fline(char *str)
{
  char  *line;
  int   i;

  if (!str || str[0] == '\0')
    return (NULL);
  i = 0;
  while (str[i] && str[i] != '\n')
    i++;
  if (str[i] == '\n')
    i++;
  line = malloc(sizeof(char) * (i + 1));
  if (!line)
    return (NULL);
  i = 0;
  while (str[i] && str[i] != '\n')
  {
    line[i] = str[i];
    i++;
  }
  if (str[i] == '\n')
    line[i++] = '\n';
  line[i] = '\0';
  return (line);
}

char  *merge(char *str, char *buffer, int i, int j)
{
  char  *merge_str;

  if (str == NULL)
  {
    str = malloc(sizeof(char) * 1);
    str[0] = '\0';
  }
  if (str == NULL || buffer == NULL)
    return (NULL);
  merge_str = malloc(ft_strlen(str) + ft_strlen(buffer) + 1);
  if (!merge_str)
    return (NULL);
  while (str[i])
  {
    merge_str[i] = str[i];
    i++;
  }
  while (buffer[j])
    merge_str[i++] = buffer[j++];
  merge_str[i] = '\0';
  free(str);
  return (merge_str);
}
```

```
char  *gnl_helper(char **str)
{
  int   nl_index;
  char  *line;

  nl_index = find_newline_index(*str);
  line = get_fline(*str);
  *str = get_remain_str(*str, nl_index, 0);
  return (line);
}

char  *get_next_line(int fd)
{
  static char *str = NULL;
  char    *buffer;
  int     read_counter;

  if (fd < 0 || BUFFER_SIZE <= 0)
    return (NULL);
  buffer = malloc(sizeof(char) * (BUFFER_SIZE + 1));
  if (!buffer)
    return (NULL);
  read_counter = read(fd, buffer, BUFFER_SIZE);
  if (read_counter == -1)
  {
    free(str);
    str = NULL;
    free(buffer);
    return (NULL);
  }
  buffer[read_counter] = '\0';
  str = merge(str, buffer, 0, 0);
  free(buffer);
  if (find_newline_index(str) != -1 || read_counter == 0)
    return (gnl_helper(&str));
  else
    return (get_next_line(fd));
}
```

```
int main(void)
{
    char *line;

    int fd = open("elif.txt", O_RDONLY);
    while ((line = get_next_line(fd)) != NULL)
        printf("%s", line);
}
//Her satırı okur.
```

```
#include <stdio.h>
#include <fcntl.h>
#include "get_next_line.h"

int main(void)
{
    char *line;

    int fd = open("elif.txt", O_RDONLY, 777);
    line = get_next_line(fd);
    printf("%s", line);
}
//Dosyayı kendin oluşturmalısın. 'open' ile oluşturulan dosyalar okunmuyor.
//2.satır için yeni printf yazman lazım.
```

1. `get_next_line` fonksiyonu çağrıldığında, statik bir bellek alanında tutulan `str` işaretçisi, dosyadan daha önce okunan verileri tutar. Bu bellek alanı ilk olarak `NULL` olarak tanımlanır.
2. Bellekten `BUFFER_SIZE` boyutunda bir tampon alan `buffer` işaretçisine atanır.
3. Dosyadan `read` fonksiyonu kullanılarak `BUFFER_SIZE` kadar veri okunur ve `buffer` tampon alanına yazılır.
4. Tampon alanındaki veriler, daha önce okunan verilerle birleştirilir ve `str` işaretçisine yazılır.
5. `str` işaretçisi, `find_newline_index` fonksiyonu kullanılarak kontrol edilir. Eğer tampon alanında bir satır sonu karakteri (`\n`) varsa, bu karakterin pozisyonu döndürülür, aksi takdirde `-1` döndürülür.
6. `get_fline` fonksiyonu, `str` işaretçisi içindeki tam bir satırı `line` işaretçisine kopyalar ve bu satırın sonunda bir satır sonu karakteri (`\n`) varsa, bu karakteri de dahil eder.
7. `get_remain_str` fonksiyonu, `str` işaretçisi içinde kalan verileri `remain_str` işaretçisine kopyalar. Bu fonksiyon, `str` işaretçisini günceller ve tampon alanındaki okunan verileri temizler.
8. `buffer` tampon alanı bellekten serbest bırakılır.
9. `line` işaretçisi döndürülür.
10. Eğer `read` fonksiyonu 0 döndürürse, dosyanın sonuna gelindiği anlamına gelir ve `NULL` işaretçisi döndürülür.
11. Eğer `read` fonksiyonu -1 döndürürse, bir hata oluşmuş demektir ve `NULL` işaretçisi döndürülür.

12. Eğer `str` işaretçisi `NULL` değilse, `get_next_line` fonksiyonu tekrar çağrılır ve sonraki satır okunur.