```
char    *remove_string(char *tmp)
{
    size_t      len;
    char        *new_tmp;
    size_t      n;

    len = ft_strchr(tmp, '\n', ft_strlen(tmp));
    n = ft_strlen(tmp);
    if (len == n)
    {
        free (tmp);
        tmp = NULL;
        return (tmp);
    }
    else
    {
        new_tmp = (char *)malloc(sizeof (char) * (n - len + 1));
        if (!new_tmp)
            return (ft_free(tmp));
        ft_strlcpy(new_tmp, tmp, n - len + 1, len);
        free (tmp);
        tmp = NULL;
    }
    return (new_tmp);
}
```

```
if (ft_strchr(tmp, '\n', ft_strlen(tmp)) != 0)
    {
      line = create_string(tmp, '\n', ft_strlen(tmp));
      if (!line)
        tmp = ft_free(tmp);
      tmp = remove_string(tmp);
      return (line);
    }
```

lo que hace esto es:
Si dentro de tmp encuentra \n y a la vez el total de caracteres de tmp es diferente de 0, entramos en bucle:

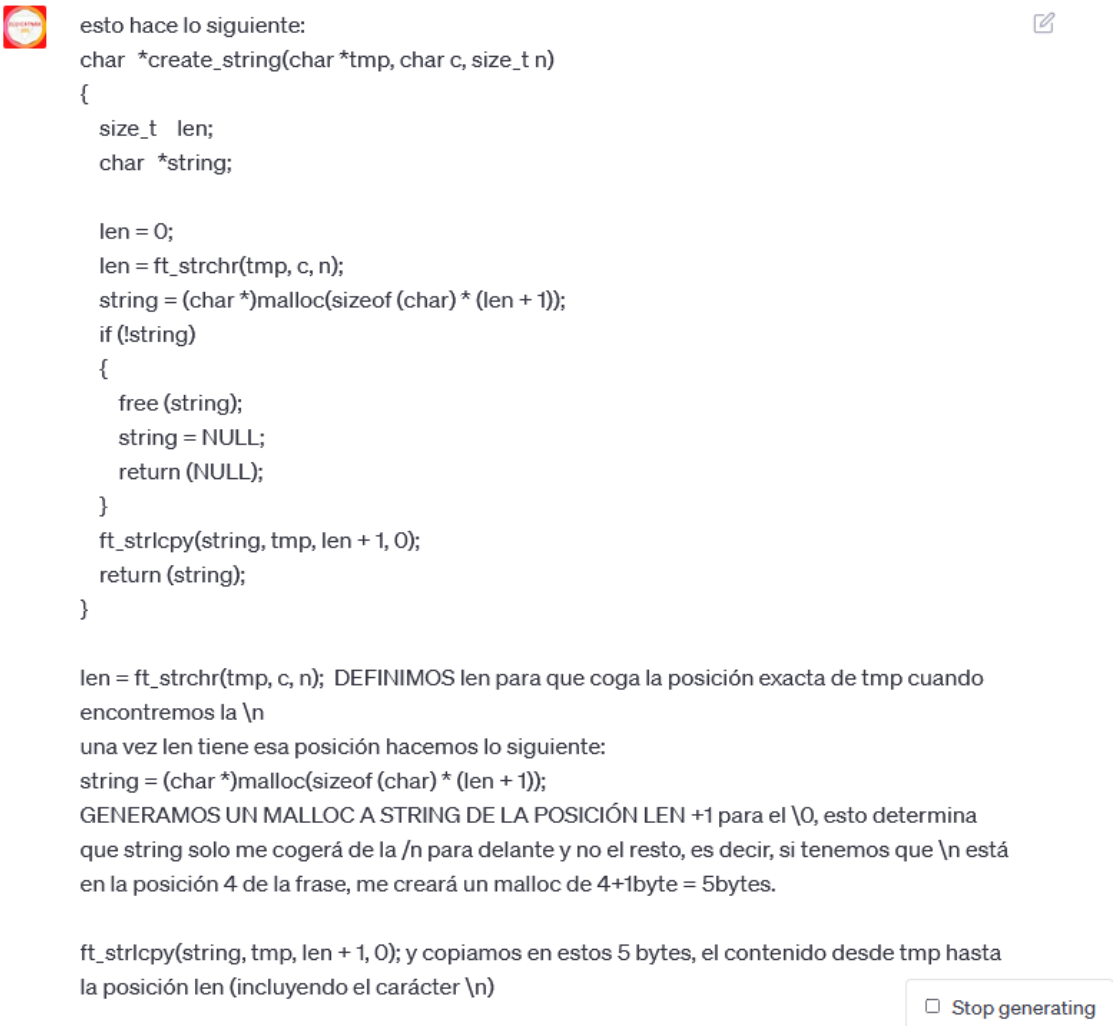
dentro de line creamos la linea que contiene lo de tmp hasta encontrar el \n con un total de tmp.
Si la linea "line" es vacía, entonces liberamos tmp y lo ponemos a null
si no, llamamos a remove_string que hará lo siguiente:

```
char  *remove_string(char *tmp)
{
  size_t      len;
  char     *new_tmp;
  size_t      n;

  len = ft_strchr(tmp, '\n', ft_strlen(tmp));
  n = ft_strlen(tmp);
  if (len == n)
  {
    free (tmp);
    tmp = NULL;
    return (tmp);
  }
```

```
  else
  {
    new_tmp = (char *)malloc(sizeof (char) * (n - len + 1));
    if (!new_tmp)
      return (ft_free(tmp));
    ft_strlcpy(new_tmp, tmp, n - len + 1, len);
    free (tmp);
    tmp = NULL;
  }
  return (new_tmp);
}
```

creamos un puntero a una nueva string llamada new_tmp.
En len = ft_strchr(tmp, '\n', ft_strlen(tmp));, cogemos en len la posición exacta dentro de tmp donde se encuentra el \n con un total de tmp
En n = ft_strlen(tmp); cogemos el total de caracteres de tmp
if (len == n) esto significa que si es lo mismo, quiere decir que no hay nada que sobre para copiar ya que el total es igual a la posición de len y lo ponemos a NULL y liberamos.
Si no: new_tmp = (char *)malloc(sizeof (char) * (n - len + 1)); generamos una reserva de memoria para la nueva tmp del resto que sobra después de \n, ya que restamos n - len+1 (el +1 es el \0)
Luego ponemos a null y liberamos y finalmente devolvemos



esto hace lo siguiente:

```
char  *create_string(char *tmp, char c, size_t n)
{
  size_t   len;
  char  *string;

  len = 0;
  len = ft_strchr(tmp, c, n);
  string = (char *)malloc(sizeof (char) * (len + 1));
  if (!string)
  {
    free (string);
    string = NULL;
    return (NULL);
  }
  ft_strlcpy(string, tmp, len + 1, 0);
  return (string);
}
```

len = ft_strchr(tmp, c, n); DEFINIMOS len para que coga la posición exacta de tmp cuando encontremos la \n
una vez len tiene esa posición hacemos lo siguiente:
string = (char *)malloc(sizeof (char) * (len + 1));
GENERAMOS UN MALLOC A STRING DE LA POSICIÓN LEN +1 para el \0, esto determina que string solo me cogerá de la /n para delante y no el resto, es decir, si tenemos que \n está en la posición 4 de la frase, me creará un malloc de 4+1byte = 5bytes.

ft_strlcpy(string, tmp, len + 1, 0); y copiamos en estos 5 bytes, el contenido desde tmp hasta la posición len (incluyendo el carácter \n)

Stop generating

es decir el create string lo que hace es crear una string nueva con malloc de un total de bytes hasta encontrar el \n y lo pasa de tmp a string, "solo ese trocito" hasta llegar al \n

CREATE STRING COPIA SIMPLEMENTE DEL INICIO HASTA \N EN STRING.

ejemplo: hola\n, pues string cogería hola\n

STRJOIN

```
char  *ft_strjoin(char *tmp, char *buf, size_t n)
{
   char     *join;
   int      i;
   int      j;

   j = 0;
   i = 0;
   if (ft_strlen(tmp) == 0)
      return (ft_strdup((const char *) buf, n, tmp));
   join = (char *)malloc(sizeof (char) * (ft_strlen(tmp) + n + 1));
   if (!join)
      return (ft_free(tmp));
   while (tmp[j])
      join[i++] = tmp[j++];
   j = 0;
   while (n)
   {
      join[i++] = buf[j++];
      n--;
   }
   join[i] = '\0';
   free (tmp);
   tmp = NULL;
   return (join);
}
```

IO QUE HACE ESTE CÓDIGO ES LO SIGUIENTE:

este codigo:
if (ft_strlen(tmp) == 0)
    return (ft_strdup((const char *) buf, n, tmp));

hace lo siguiente?

si el total de tmp es igual a 0, es decir, no tiene nada, entonces entra y me retorna, una nueva string creada con malloc dentro de strdup con lo que tenga buf en ese momento, con el tamaño n y tmp no se que hace en este caso

Estás en lo correcto. Esa parte del código:

```c
if (ft_strlen(tmp) == 0)
    return (ft_strdup((const char *) buf, n, tmp));
```

Verifica si la longitud de la cadena `tmp` es igual a 0, lo que significa que `tmp` no contiene ningún carácter. Si esta condición se cumple, entra en el bloque y hace lo siguiente:

IO QUE HACE ESTE CÓDIGO ES LO SIGUIENTE:

Aquí entra si tmp es cero, es decir, vacío:
return (ft_strdup((const char *) buf, n, tmp));
y esta línea lo que hace es crear nueva array con malloc y poner lo que hay en BUF, con n de tamaño de bytes, ya que viene del otro fichero .c
while (tmp[j]): aquí entra si tmp en la posición 0 es diferente de nullo:
join[i++] = tmp[j++];: copia lo que hay en tmp en join sumando i++

inicializamos j a 0 parqa comenzar de nuevo y copiar:
join[i++] = buf[j++]; mientras n "que es bytes" sea diferente de null, entra y copia con la j a 0 buf en join "que ya teníamos copiado tmp", por tanto quedan concatenadas las 2 en join.

le ponemos \0, liberamos y retornamos join

1. `ft_strlen` - Cuenta la longitud de una cadena de caracteres:

Esta función toma una cadena de caracteres (`s`) como entrada y cuenta cuántos caracteres hay en esa cadena hasta llegar al final. Básicamente, está contando la cantidad de letras en la palabra.

2. `ft_strlcpy` - Copia parte de una cadena en otra cadena:

Esta función copia una parte de una cadena de caracteres (`src`) en otra cadena (`dst`). Se copian `dstsize - 1` caracteres de `src` en `dst`, y luego se agrega un carácter nulo `\0` al final para asegurarse de que `dst` sea una cadena válida.

3. `ft_strchr` - Encuentra la posición de un carácter en una cadena:

Esta función busca un carácter específico (`c`) en una cadena de caracteres (`s`). Comienza a buscar desde el principio de la cadena y detiene su búsqueda cuando encuentra el carácter o cuando alcanza una cantidad máxima de caracteres (`n`). Si encuentra el carácter, devuelve la posición en la que lo encontró.

4. `ft_strdup` - Duplica una parte de una cadena:

Esta función crea una copia de una parte específica (`n` caracteres) de una cadena de caracteres (`s1`). Utiliza `malloc` para asignar memoria para la nueva cadena (`tmp`) y luego utiliza `ft_strlcpy` para copiar la parte de la cadena original en la nueva cadena. Devuelve la nueva cadena.

5. `ft_strjoin` - Une dos cadenas de caracteres:

Esta función toma dos cadenas de caracteres (`tmp` y `buf`) y las une en una nueva cadena (`join`). Si `tmp` es una cadena vacía, simplemente copia `buf` en `join`. Si no, asigna memoria para `join`, copia el contenido de `tmp` en `join`, luego agrega el contenido de `buf` al final de `join`. Finalmente, libera la memoria de `tmp` (para que no haya fugas de memoria) y devuelve la nueva cadena `join`.

La función `ft_strjoin` tiene como objetivo combinar dos cadenas de caracteres en una sola cadena nueva. Una cadena es `tmp`, que contiene datos previos que se han leído de algún lugar, y la otra cadena es `buf`, que contiene datos recién leídos. La función crea una nueva cadena, `join`, que contendrá la combinación de los contenidos de `tmp` y `buf`.

Pasos generales de lo que hace:

1. Verifica si la cadena `tmp` está vacía. Si lo está, significa que no hay datos previos y la función simplemente crea una copia de la cadena `buf` y la devuelve como resultado.
2. Si `tmp` no está vacía, la función asigna memoria para la cadena `join`, que será lo suficientemente grande para contener los contenidos de `tmp` y `buf`.
3. Copia los caracteres de `tmp` a `join` para mantener los datos anteriores.
4. Luego, copia los caracteres de `buf` a `join` para agregar los nuevos datos al final.
5. Asegura que `join` sea una cadena válida agregando un carácter nulo al final (esto es importante para que sepa dónde termina la cadena).
6. Libera la memoria ocupada por `tmp`, ya que ya no es necesaria.
7. Devuelve la cadena `join`, que contiene la combinación de los datos de `tmp` y `buf`.

1. **Función `ft_free`**:
    - Libera la memoria ocupada por un puntero (`tmp`) utilizando la función `free`.
    - Luego, establece el puntero `tmp` en `NULL` para evitar problemas de acceso a memoria liberada.
2. **Función `remove_string`**:
    - Esta función elimina parte del contenido de una cadena `tmp` hasta la primera aparición de un carácter de nueva línea (`'\n'`).
    - Si no encuentra un salto de línea, libera la memoria ocupada por `tmp` y lo establece en `NULL`.
    - Si encuentra un salto de línea, crea una nueva cadena (`new_tmp`) que excluye el contenido antes del salto de línea y la copia.
    - Libera la memoria de `tmp`, lo establece en `NULL` y devuelve la nueva cadena `new_tmp`.
3. **Función `create_string`**:
    - Crea una nueva cadena (`string`) a partir del contenido en `tmp` hasta la primera aparición de un carácter específico (`c`).
    - La longitud de la subcadena se calcula mediante la función `ft_strchr` que busca la posición de `c` en `tmp`.
    - Reserva memoria para `string`, copia la subcadena en `string` y agrega un carácter nulo al final.
    - Si no puede reservar memoria, libera la memoria de `string` (si se reservó) y devuelve `NULL`.
4. **Función `create_nullstring`**:
    - Crea una nueva cadena (`line`) a partir del contenido en `tmp` (si no está vacío).

- Reserva memoria para `line`, copia el contenido de `tmp` y agrega un carácter nulo al final.
- Luego, libera la memoria ocupada por `tmp` y lo establece en `NULL`.
- Si no puede reservar memoria para `line`, libera la memoria ocupada por `line` (si se reservó), la memoria ocupada por `tmp` y establece ambos punteros en `NULL`.

5. **Función `get_next_line`**:
   - Lee y procesa líneas de un archivo identificado por el descriptor `fd`.
   - Lee bloques de datos del archivo en un búfer (`buf`) utilizando `read`.
   - Acumula los datos leídos en la variable estática `tmp`.
   - Busca un carácter de nueva línea (`'\n'`) en `tmp`. Si lo encuentra, crea una línea utilizando `create_string`, la devuelve y elimina la parte procesada de `tmp` utilizando `remove_string`.
   - Si no se encuentra un salto de línea y se llega al final del archivo, crea una línea utilizando `create_nullstring` y libera la memoria de `tmp`.
   - La función se ejecuta en un bucle infinito para procesar las líneas hasta que no hay más datos para leer.
6. **Función `main`**:
   - En la función principal, se abre el archivo "1char.txt" en modo de lectura (`O_RDONLY`).
   - Utiliza un bucle `while` para leer líneas del archivo utilizando `get_next_line`.
   - Imprime cada línea y libera la memoria de cada línea después de usarla.
   - Después de salir del bucle, intenta leer una vez más utilizando `get_next_line` para verificar si se detecta correctamente el final del archivo.

1. **Función `get_next_line`**:
   - La función `get_next_line` está diseñada para leer líneas de un archivo de manera incremental, en lugar de cargar todo el contenido en memoria de una vez.
   - Utiliza un búfer `buf` para leer datos desde el archivo especificado por su descriptor (`fd`).
   - Mantiene un puntero estático `tmp` que acumula el contenido leído en lecturas anteriores.
   - La función busca caracteres de nueva línea (`'\n'`) en el contenido acumulado en `tmp`. Cuando encuentra uno, crea una nueva línea a partir del contenido antes del salto de línea y devuelve esa línea.
   - Si llega al final del archivo y no hay más líneas en el contenido acumulado, devuelve `NULL`.
2. **Función `create_string`**:
   - La función `create_string` se encarga de crear una nueva subcadena de caracteres a partir del contenido en `tmp`.
   - Esta subcadena va desde el inicio de `tmp` hasta la primera aparición del carácter `c` dentro de los primeros `n` caracteres.

    - Reserva memoria para la subcadena y utiliza `ft_strlcpy` para copiar los caracteres relevantes en la nueva cadena.
3. **Función `create_nullstring`**:
    - La función `create_nullstring` se utiliza para crear una nueva cadena de caracteres a partir del contenido almacenado en `tmp`.
    - Si `tmp` no está vacío, se reserva memoria para la nueva cadena y se copia el contenido de `tmp`.
    - Luego, la memoria ocupada por `tmp` se libera, y el puntero `tmp` se establece en `NULL`.
    - Esto permite extraer y retener el contenido acumulado en `tmp` en una nueva cadena mientras se libera la memoria original.
4. **Función `remove_string`**:
    - La función `remove_string` se utiliza para crear una nueva cadena que excluye el contenido hasta la primera aparición de `'\n'` en `tmp`.
    - Se reserva memoria para la nueva cadena y se copia el contenido de `tmp` después de la posición del primer salto de línea.
    - La memoria de `tmp` se libera, y el puntero se establece en `NULL`.
    - Esto ayuda a eliminar la parte procesada del contenido acumulado en `tmp`.
5. **Función `main`**:
    - En la función principal, se abre el archivo "1char.txt" en modo de lectura (`O_RDONLY`).
    - La función `get_next_line` se utiliza en un bucle para leer líneas del archivo.
    - Cada línea se imprime y luego se libera la memoria ocupada por esa línea.