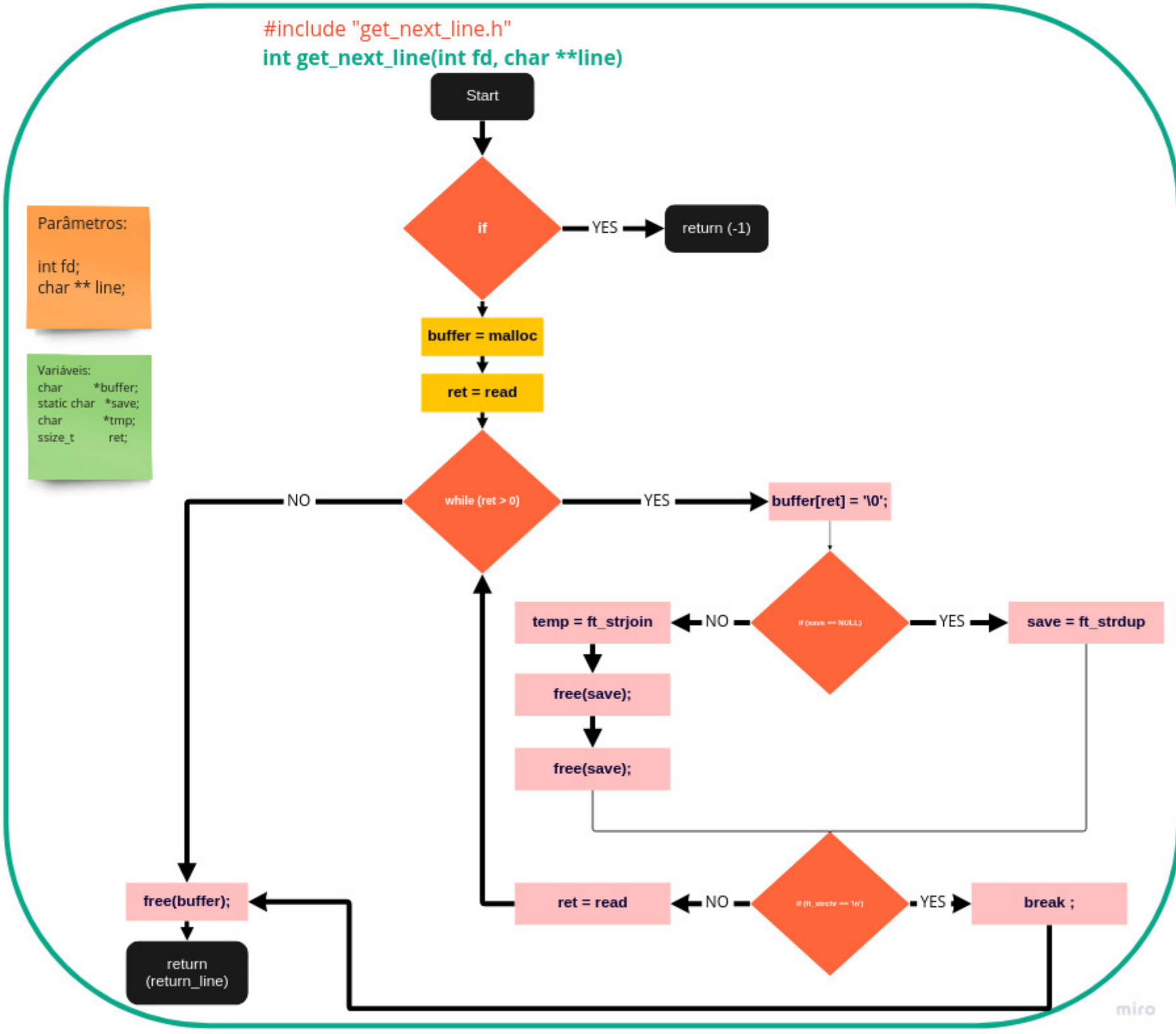
Labs42

Determinar a frequência de cada elemento
Montar árvore binária agrupando símbolos de acordo com a frequência
Percorrer a árvore para codificar e comprimir os dados
Criar espaço de memória compartilhado para troca de informações entre processos
Acessar a memória compartilhada obtendo os dados de entrada, descrever as informações e conteúdo descodificado
Leitura
Lista
Árvore
Tabela
Codificação
Compressão
Shared Memory
Descodificação
ler o texto e gerar uma tabela de frequência
gerar uma lista com nós de uma folha árvore, ordenada pela frequência de ocorrência
percorrer a tabela de frequências e para cada posição maior que zero, criar um nó e inserir na lista
cada nó terá um caracter, uma frequência e um ponteiro para esquerda e para direita, que serão os filhos de cada nó
gerar a árvore de Huffman
remover os nós com menor frequência
carrega com um caracter e nó, tornando as frequências e valores movimentos na lista
repete até a estrutura ter apenas um nó
Gerar a tabela de códigos (dicionário)
Percorrer a árvore, atingindo todas as folhas e criar uma tabela de códigos
Codificar o texto
Percorrer o texto original, para cada símbolo do texto original, pegar o código correspondente no dicionário e concatenar o resultado numa string
Enviar a concatenação da string codificada através de operações binárias
escrita em arquivo para memória compartilhada
Passa um ponteiro para memória compartilhada
Descomprimir o texto
Decodificar o texto
Passa as informações para encoder: Dados comprimidos, quantidade total de bytes na lista, texto e comprimidos, tempo de operação de descompressão
ENCODING
DECODING
miro


#include "get_next_line.h"
int get_next_line(int fd, char **line)
Start
if
YES
return (-1)
Parâmetros:
int fd;
char ** line;
buffer = malloc
ret = read
Variáveis:
char *buffer;
static char *save;
char *tmp;
ssize_t ret;
while (ret > 0)
NO
YES
buffer[ret] = '\0';
if (save == NULL)
NO
YES
temp = ft_strjoin
save = ft_strdup
free(save);
free(save);
if (ft_strchr == '\n')
NO
YES
ret = read
break ;
free(buffer);
return
(return_line)
miro

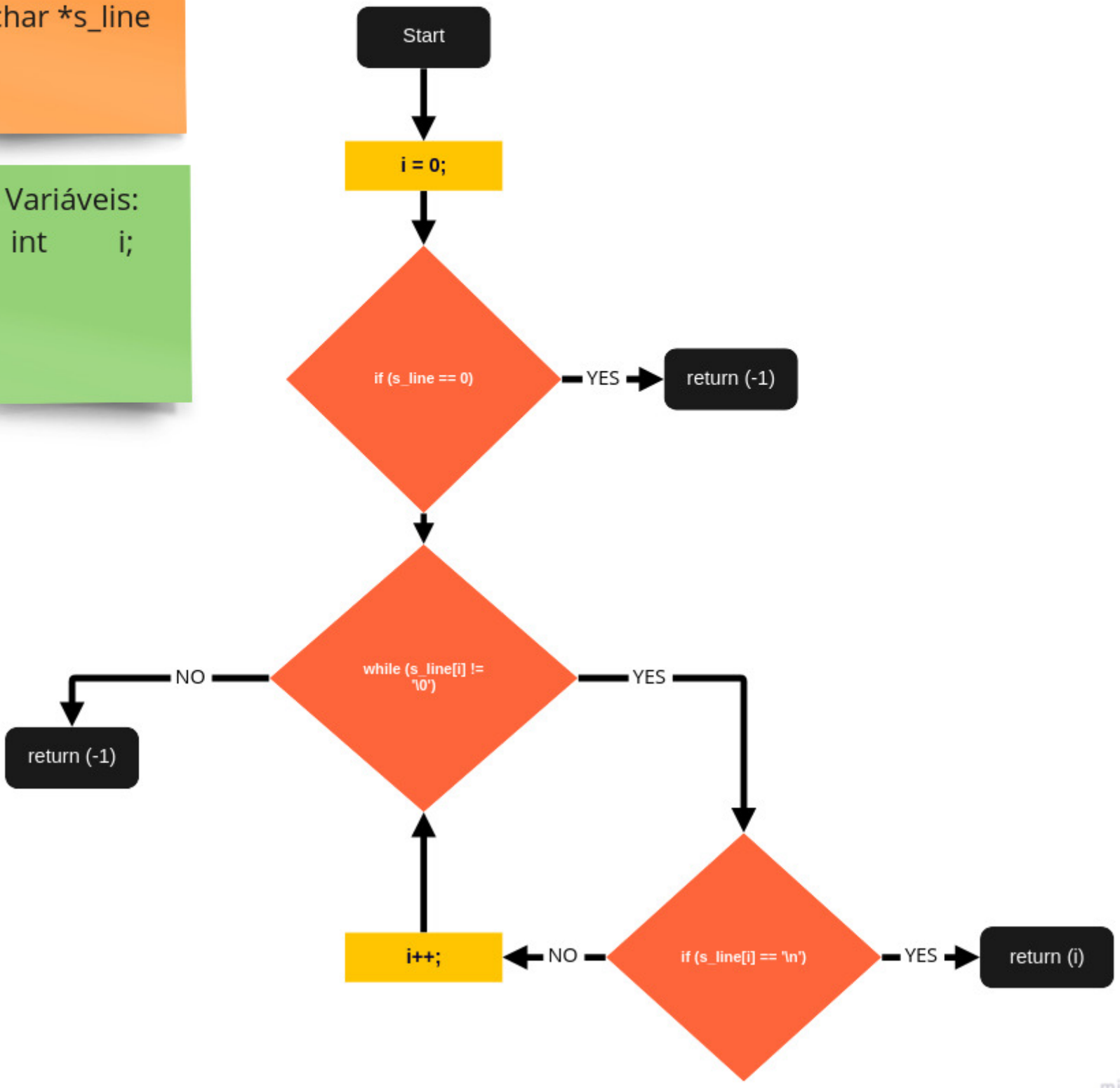
Parâmetros:

char *s_line

Variáveis:

int i;

static int find_line_break(char *s_line)



Parâmetros:

```
char     **save;
char     **line;
ssize_t    ret;
```

Variáveis:

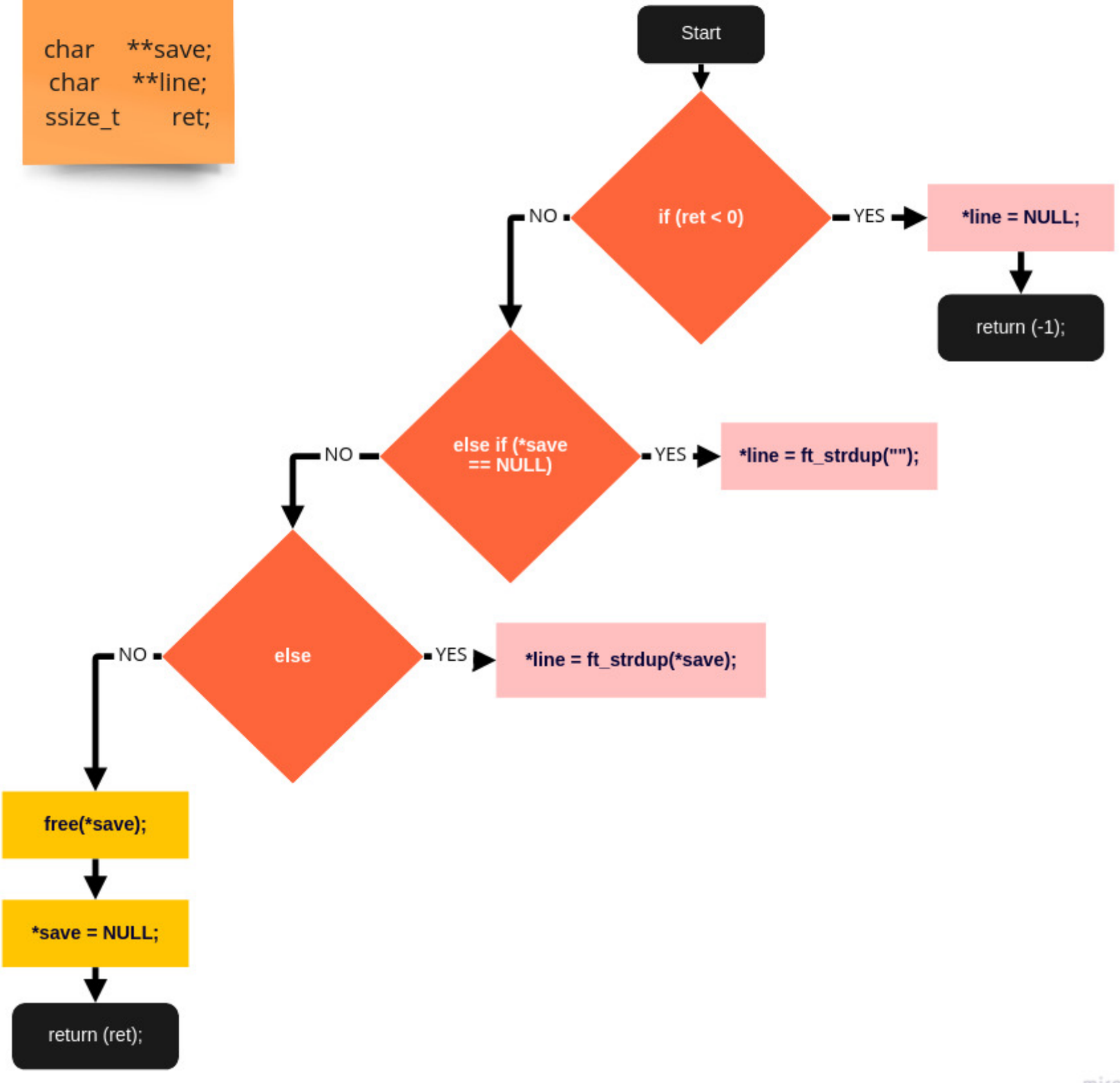
```
int          i;
char       *tmp;
```

miro

Parâmetros:

char **save;
char **line;
ssize_t ret;

static int check_ret(char **save, char **line, ssize_t ret)



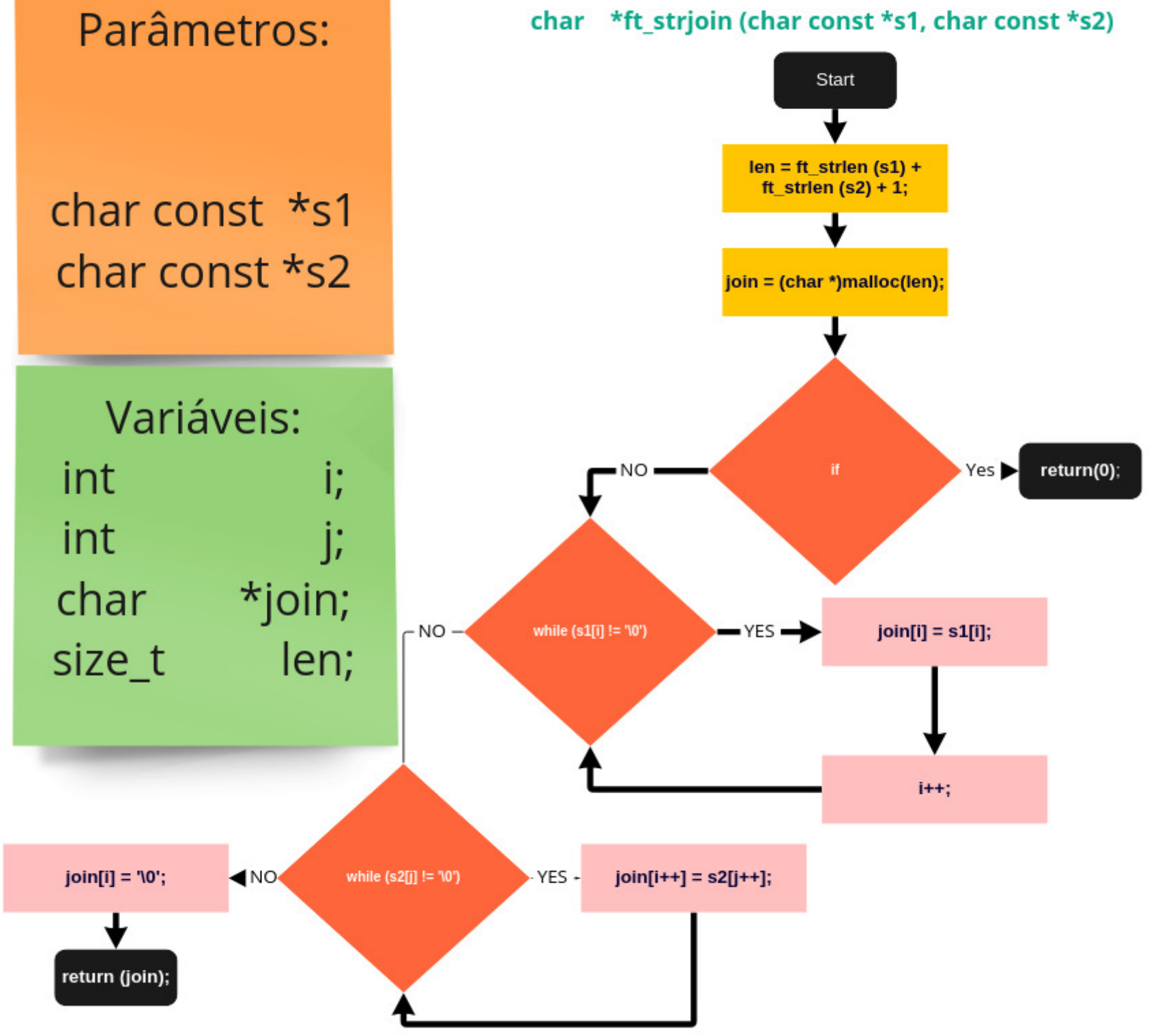
miro

Parâmetros:

char const *s1
char const *s2

Variáveis:

int i;
int j;
char *join;
size_t len;



miro

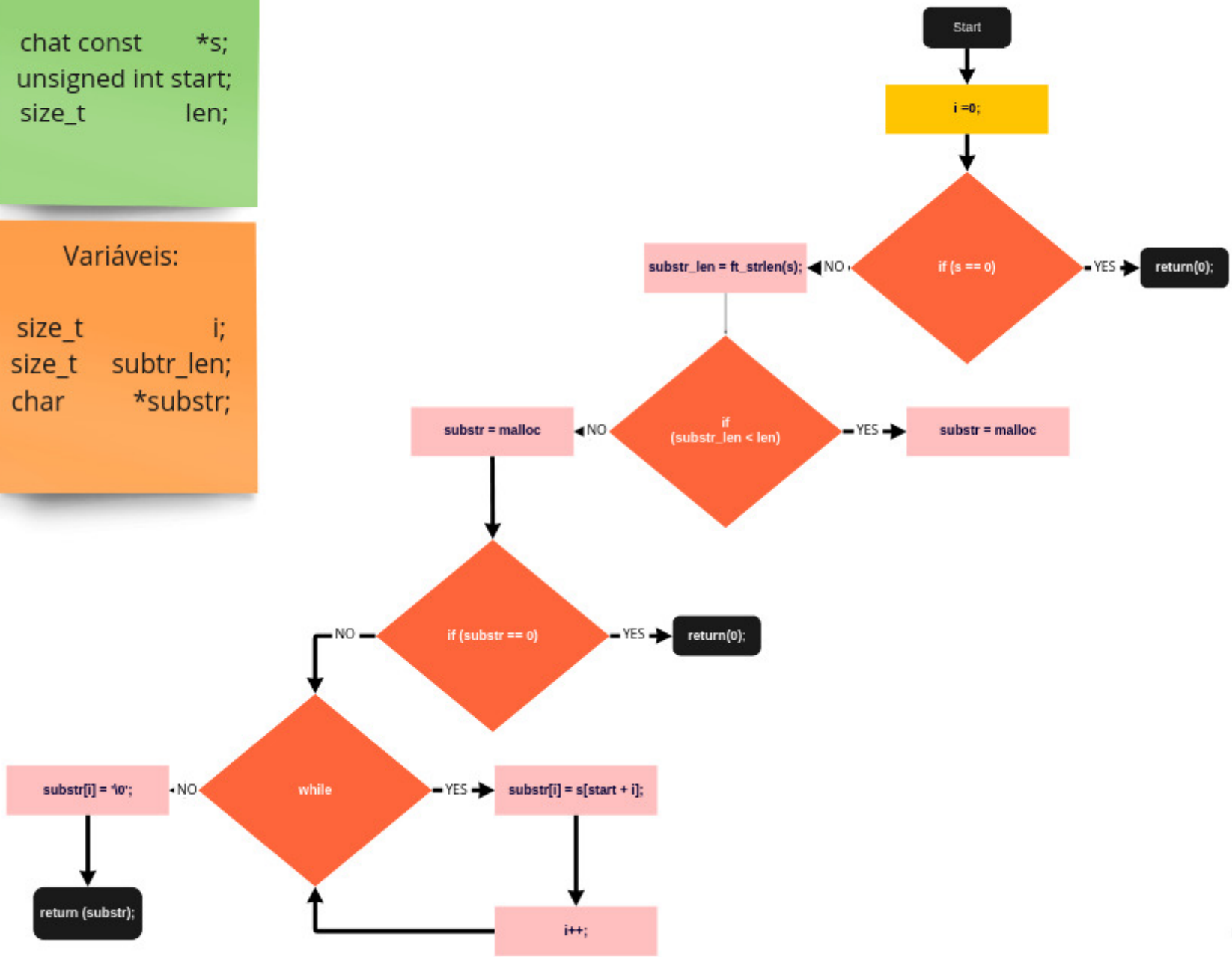
Parâmetro:

chat const *s;
unsigned int start;
size_t len;

Variáveis:

size_t i;
size_t subtr_len;
char *substr;

char *ft_substr(char const *s, unsigned int start, size_t len)



miro

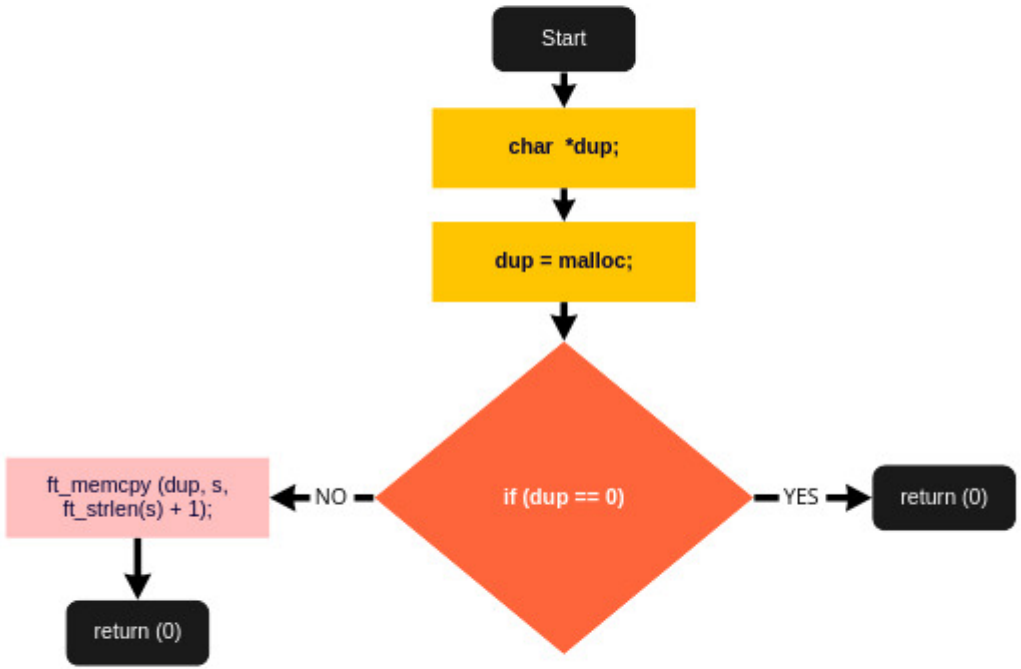
Parâmetro:

Const char *s;

Variáveis:

char *dup;

char *ft_strdup (const char *s)

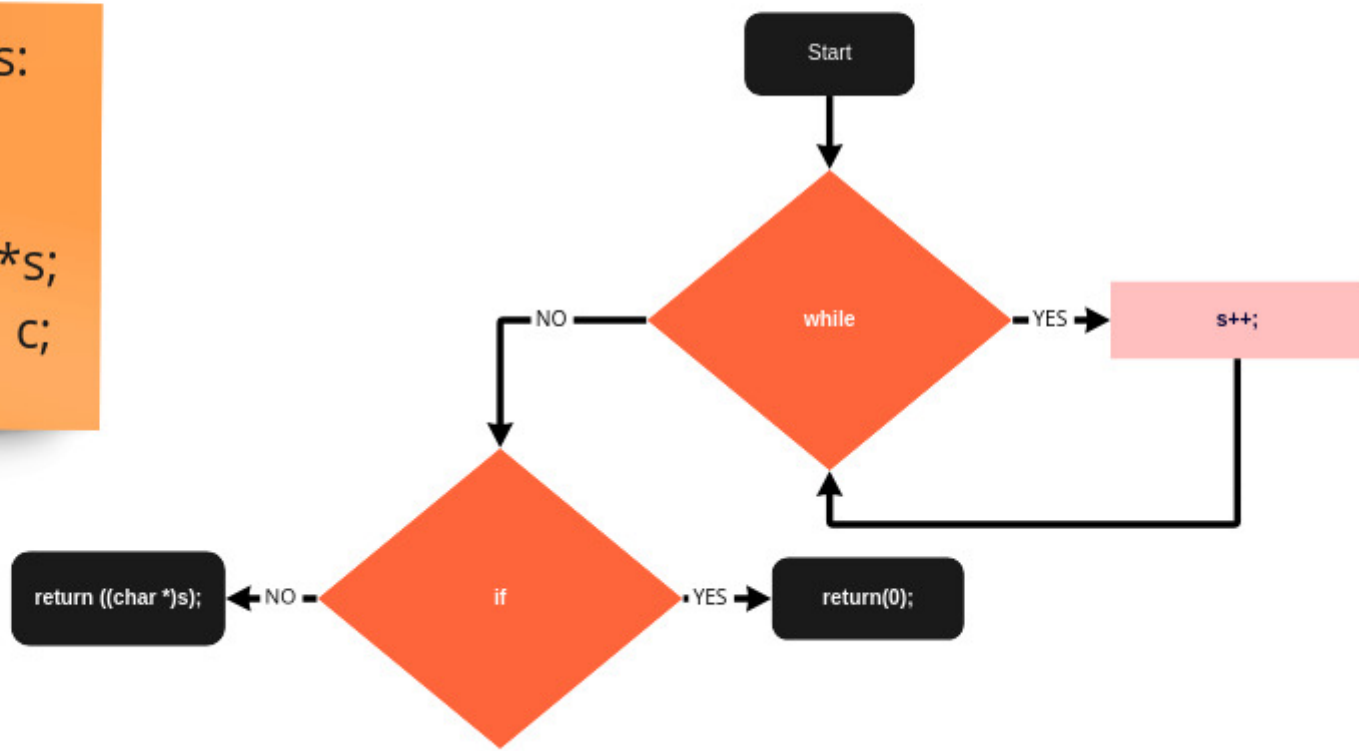


miro

char *ft_strchr (const char *s, int c)

Parâmetros:

char const *s;
int c;



miro

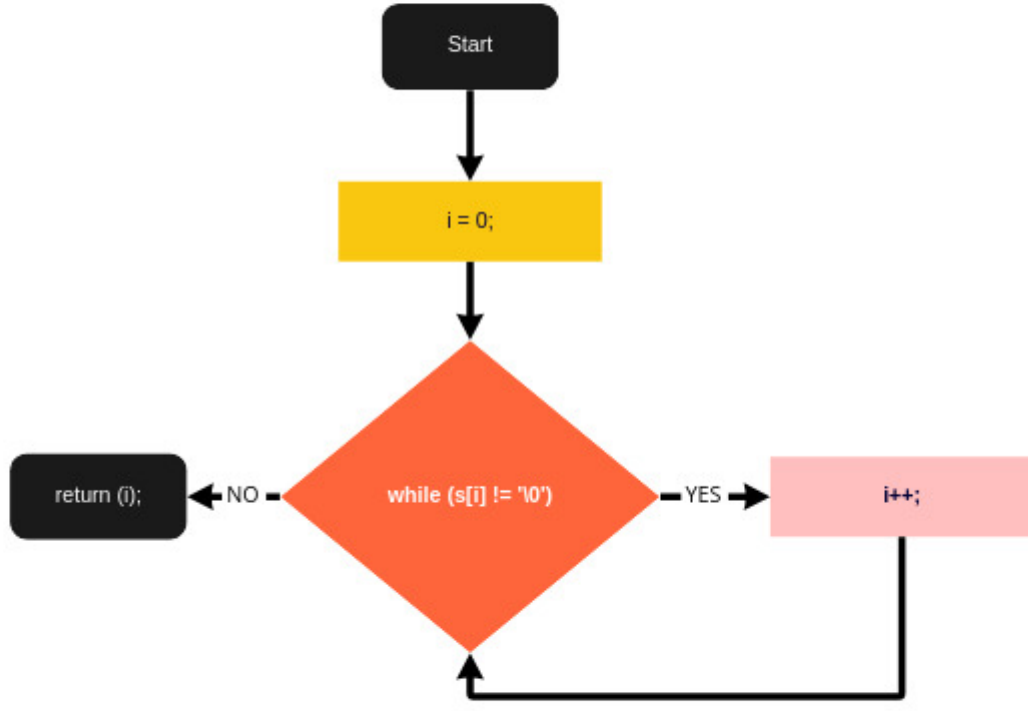
Parâmetros:

char const *s;
int c;

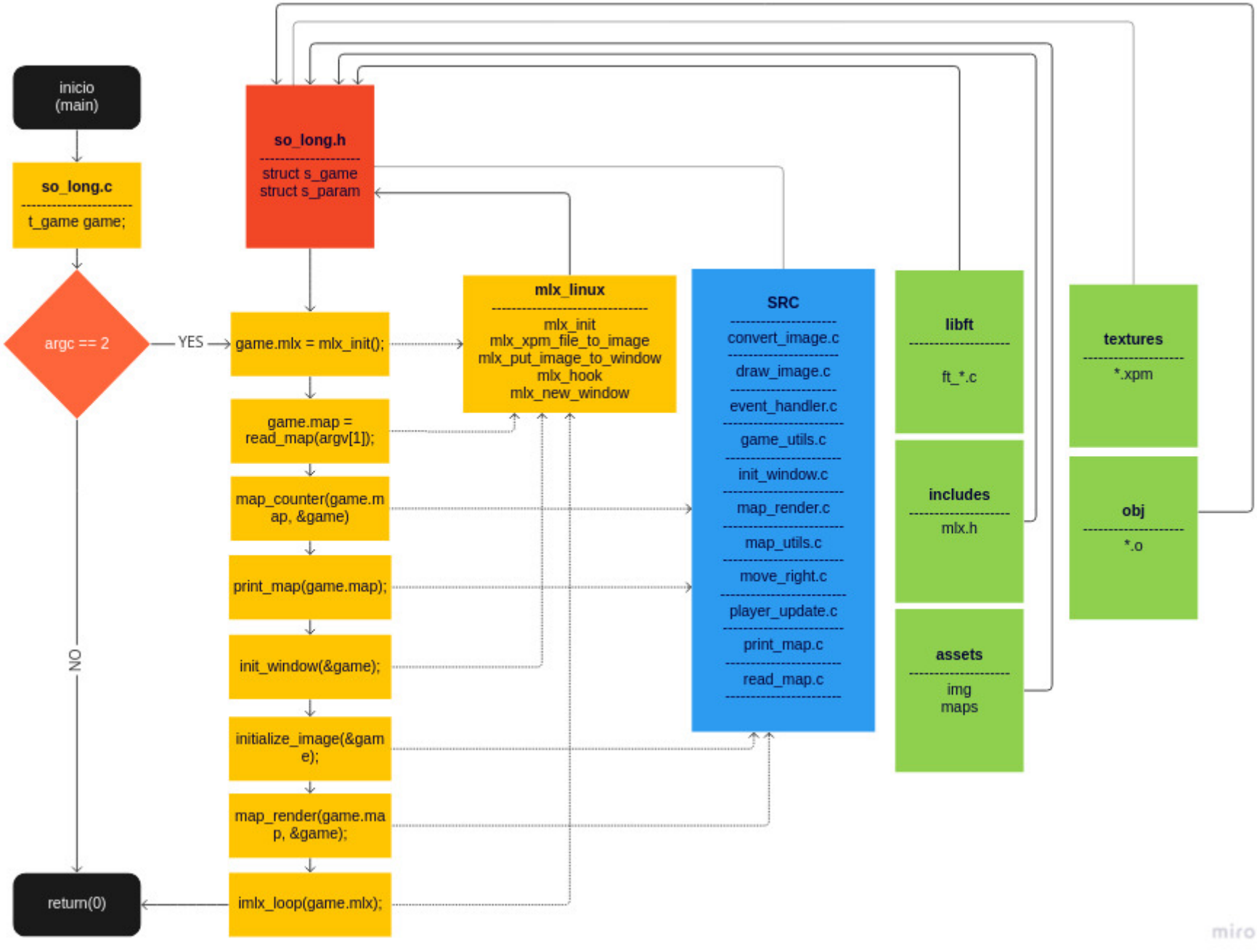
Variáveis:

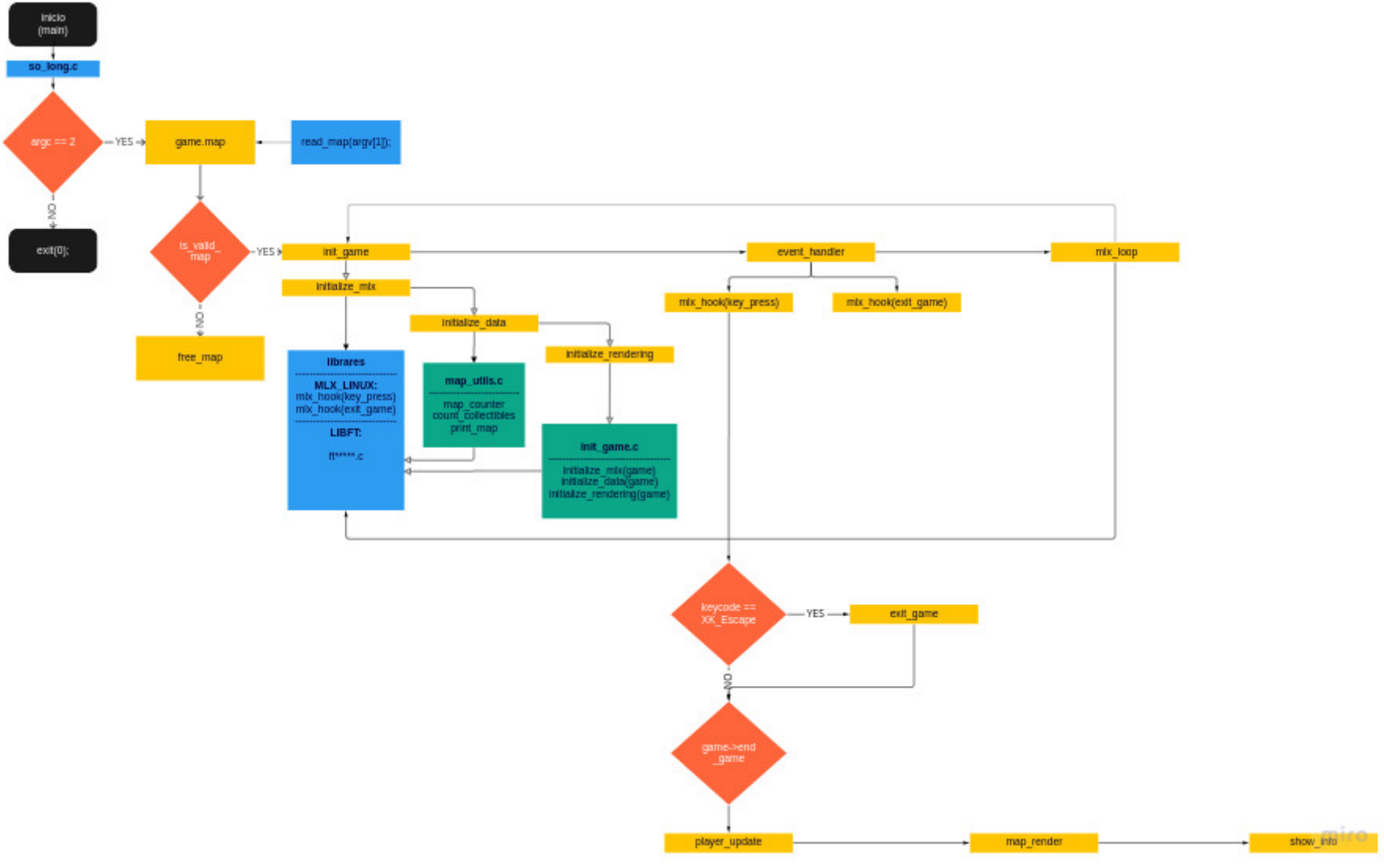
size_t i;

size_t ft_strlen (const char *s)



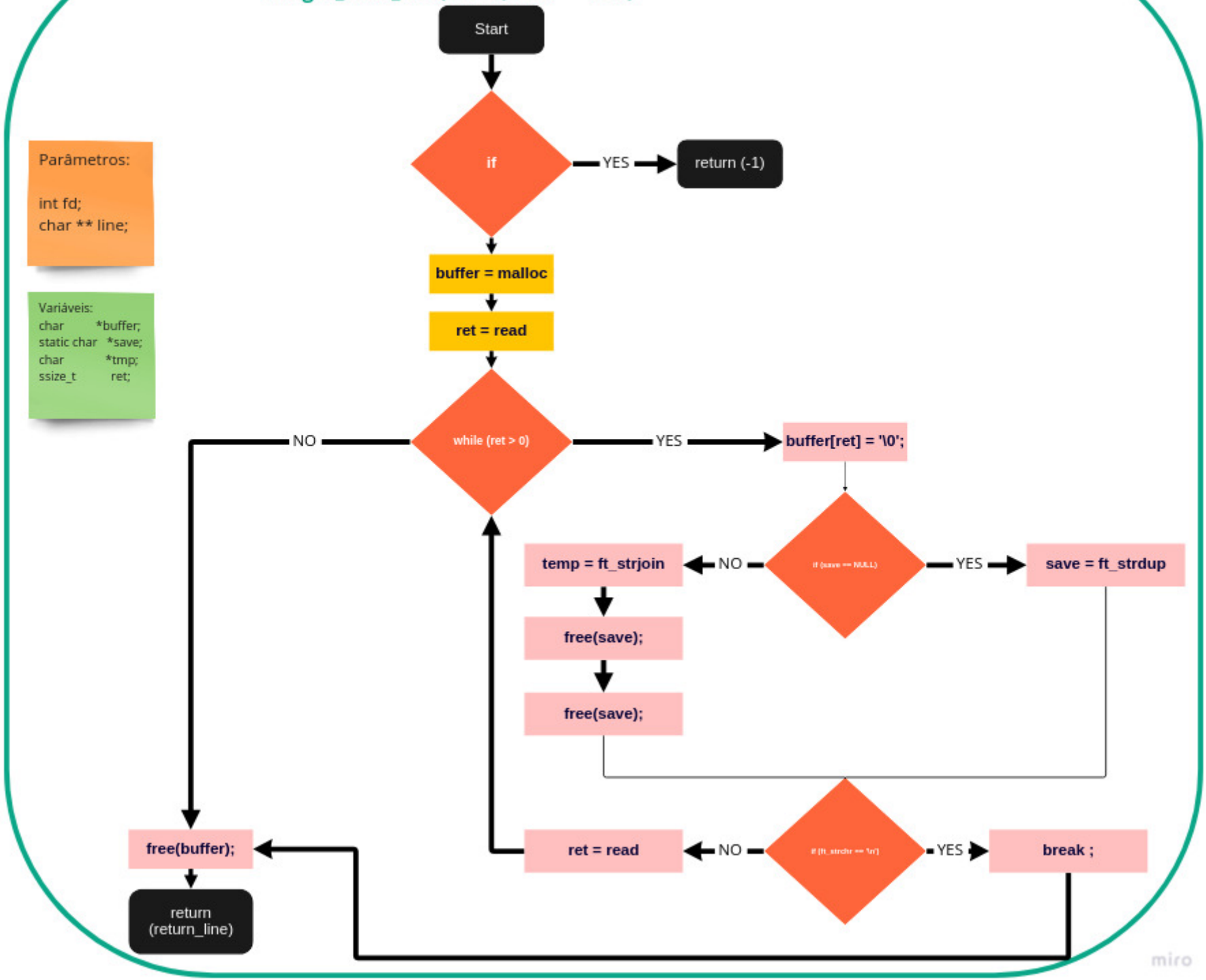
miro


inicio
(main)
so_long.c
t_game game;
argc == 2
YES
NO
return(0)
so_long.h
struct s_game
struct s_param
game.mlx = mlx_init();
game.map =
read_map(argv[1]);
map_counter(game.map, &game)
print_map(game.map);
init_window(&game);
initialize_image(&game);
map_render(game.map, &game);
imlx_loop(game.mlx);
mlx_linux
mlx_init
mlx_xpm_file_to_image
mlx_put_image_to_window
mlx_hook
mlx_new_window
SRC
convert_image.c
draw_image.c
event_handler.c
game_utils.c
init_window.c
map_render.c
map_utils.c
move_right.c
player_update.c
print_map.c
read_map.c
libft
ft_*.c
includes
mlx.h
assets
img
maps
textures
*.xpm
obj
*.o
miro


Inicio
(main)
so_long.c
argc == 2
YES
game.map
read_map(argv[1]);
NO
exit(0);
is_valid_
map
YES
NO
free_map
init_game
initialize_mlx
initialize_data
initialize_rendering
librares
MLX_LINUX:
mlx_hook(key_press)
mlx_hook(exit_game)
LIBFT:
ft*****.c
map_utils.c
map_counter
count_collectibles
print_map
init_game.c
initialize_mlx(game)
initialize_data(game)
initialize_rendering(game)
event_handler
mlx_hook(key_press)
mlx_hook(exit_game)
mlx_loop
keycode ==
XK_Escape
YES
exit_game
NO
game->end
_game
player_update
map_render
show_info

#include "get_next_line.h"

int get_next_line(int fd, char **line)



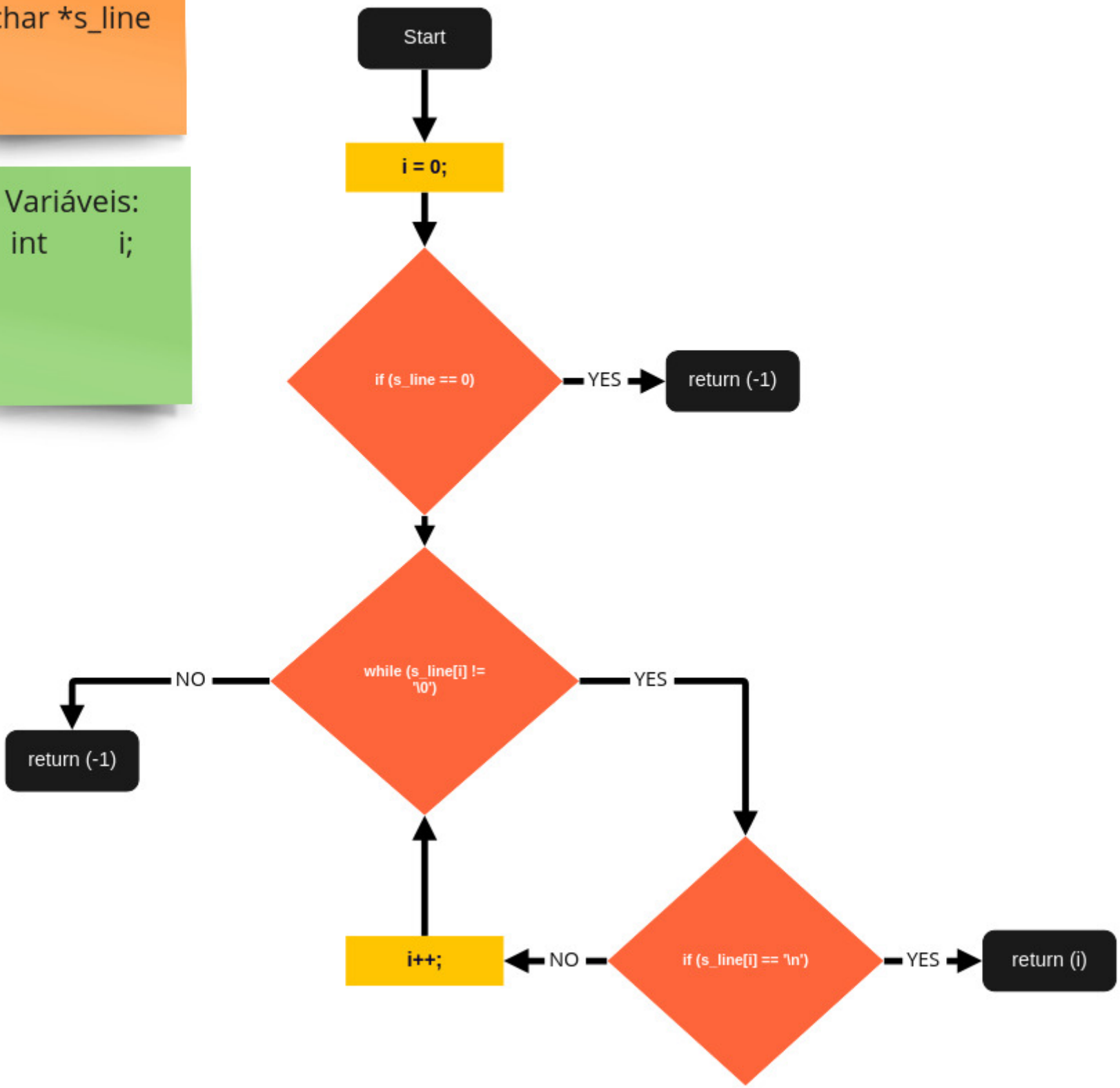
Parâmetros:

char *s_line

Variáveis:

int i;

static int find_line_break(char *s_line)



miro

Parâmetros:

```
char    **save;
char    **line;
ssize_t ret;
```

Variáveis:

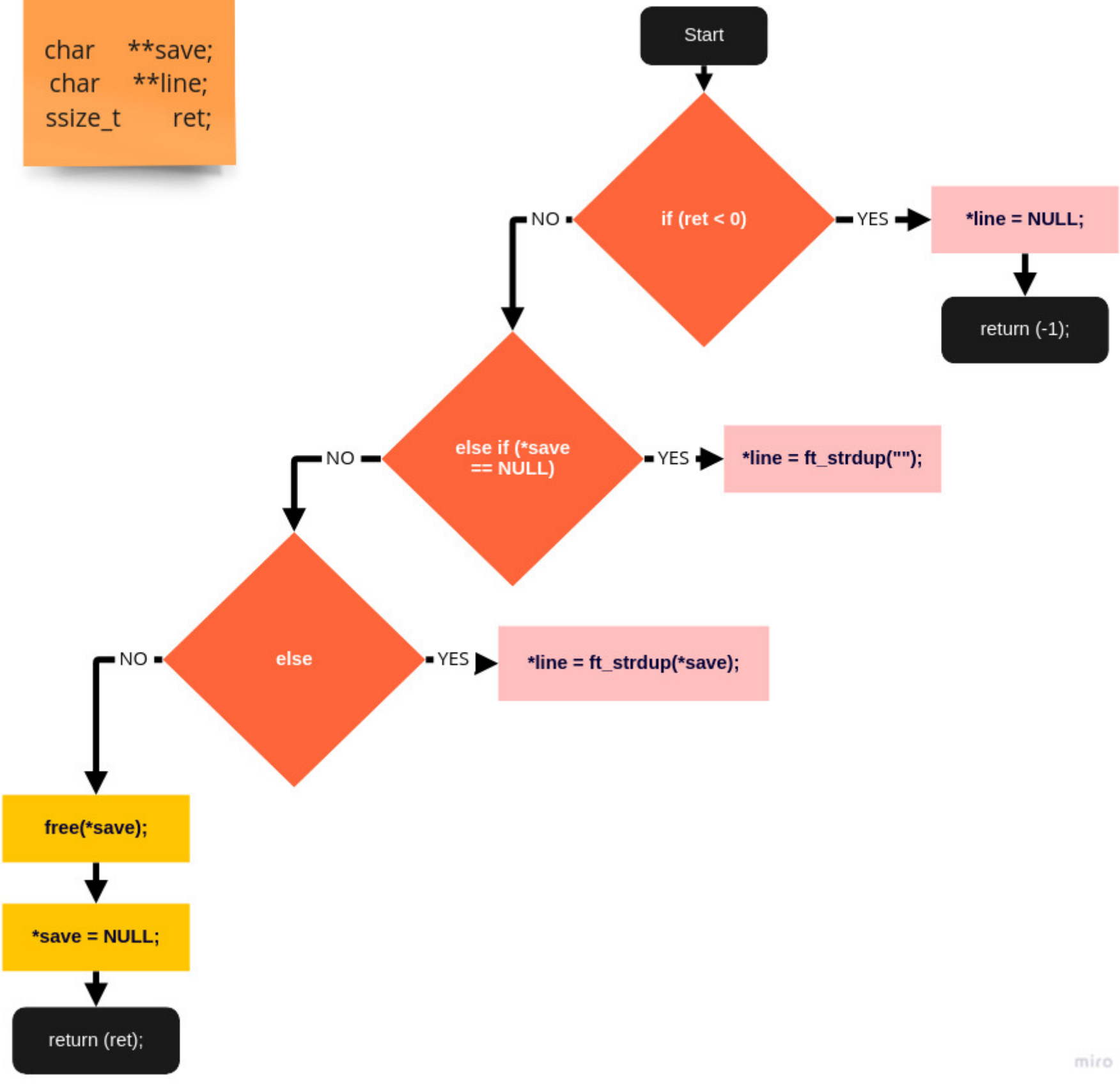
```
int     i;
char    *tmp;
```

int return_line(char **save, char **line)

Parâmetros:

char **save;
char **line;
ssize_t ret;

static int check_ret(char **save, char **line, ssize_t ret)



miro