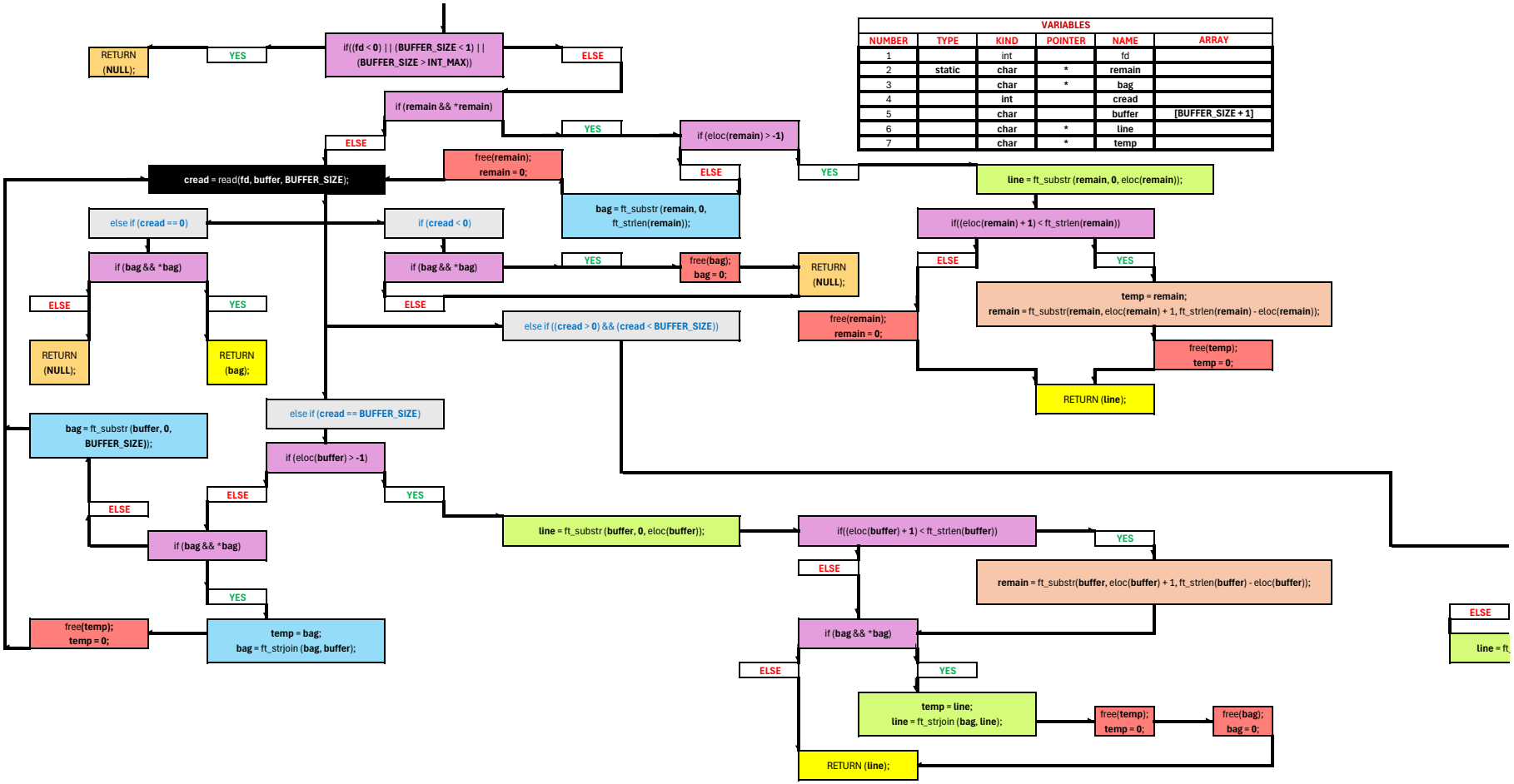
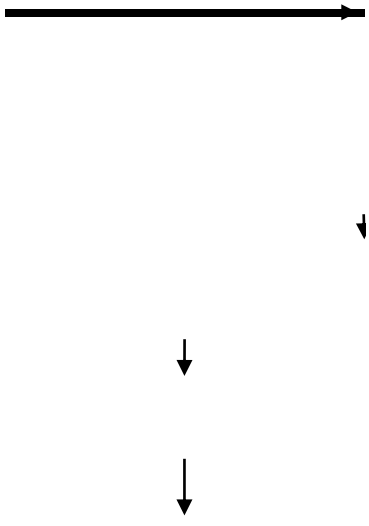
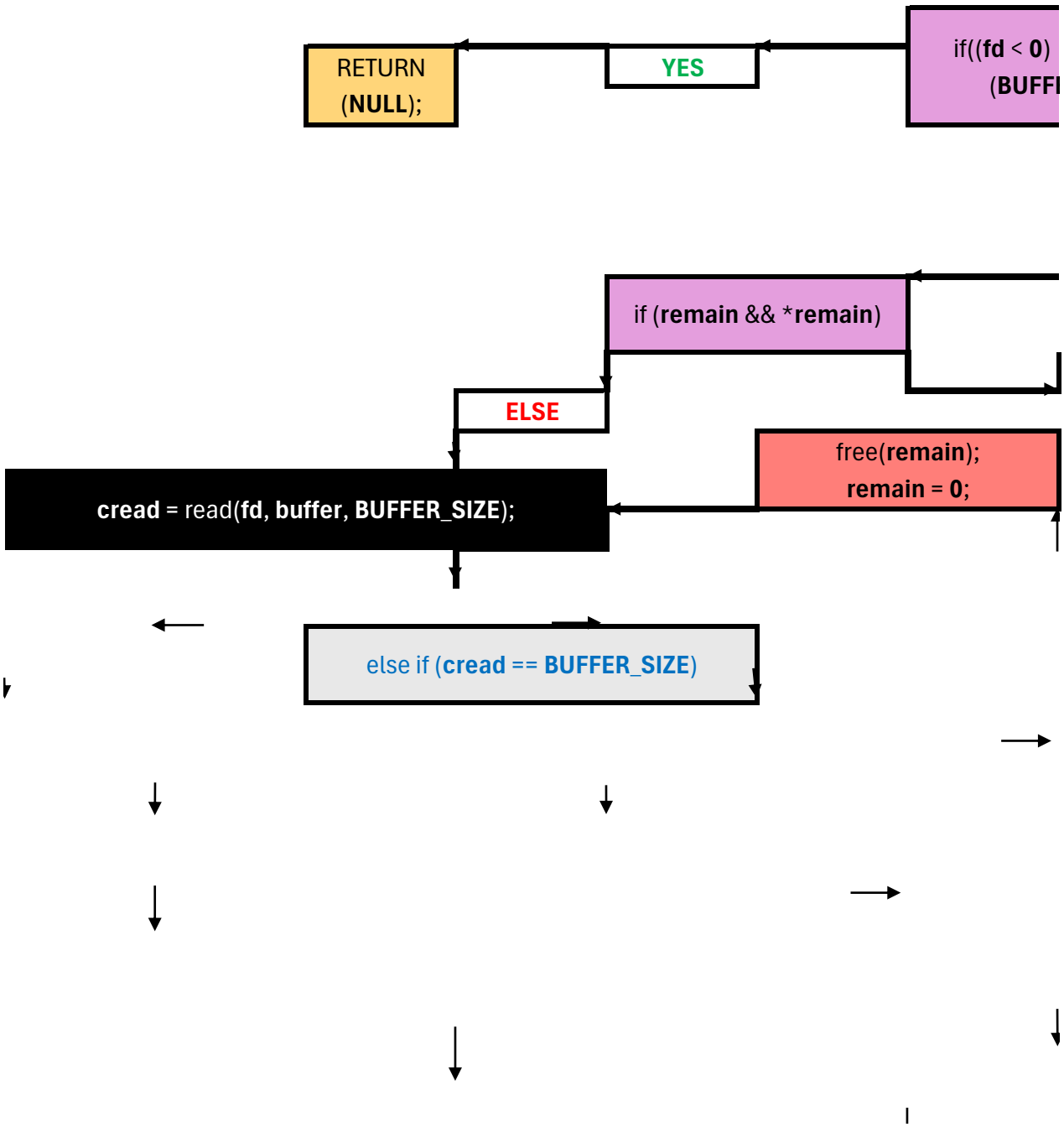
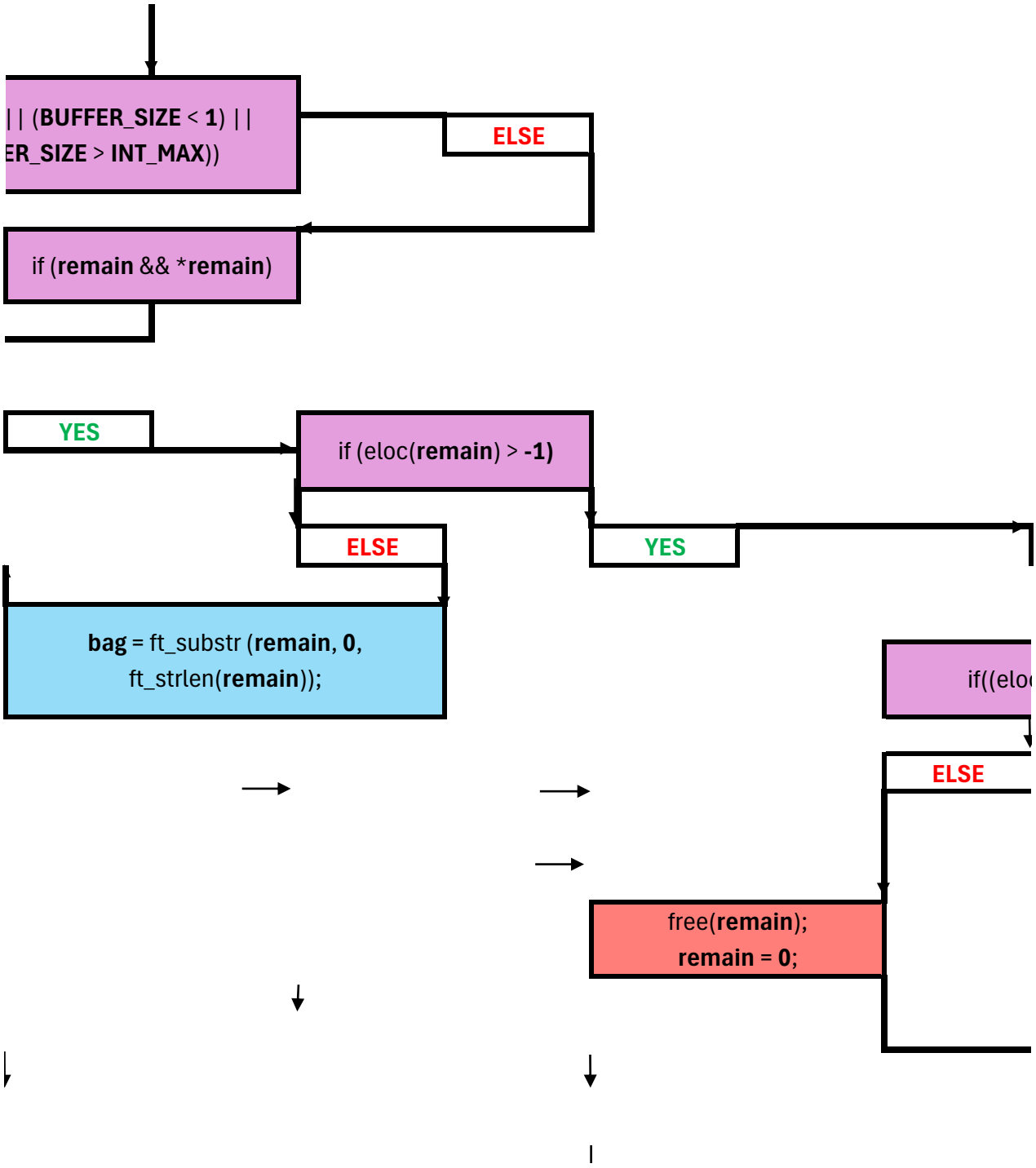
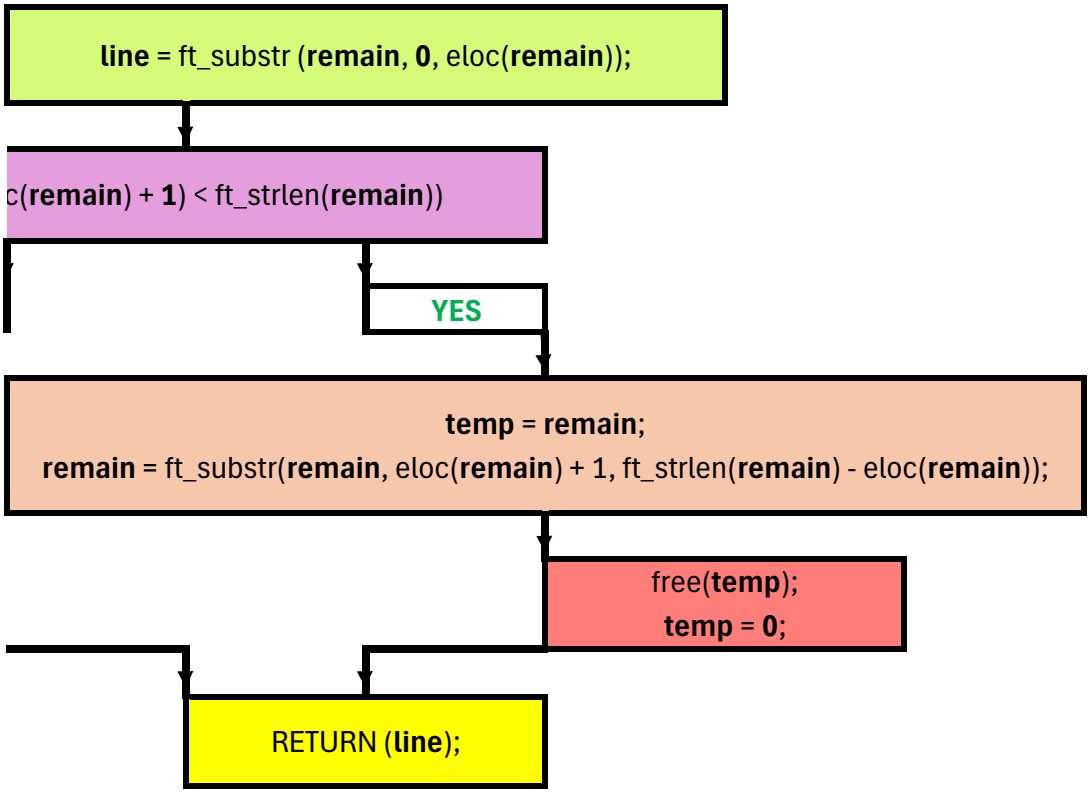


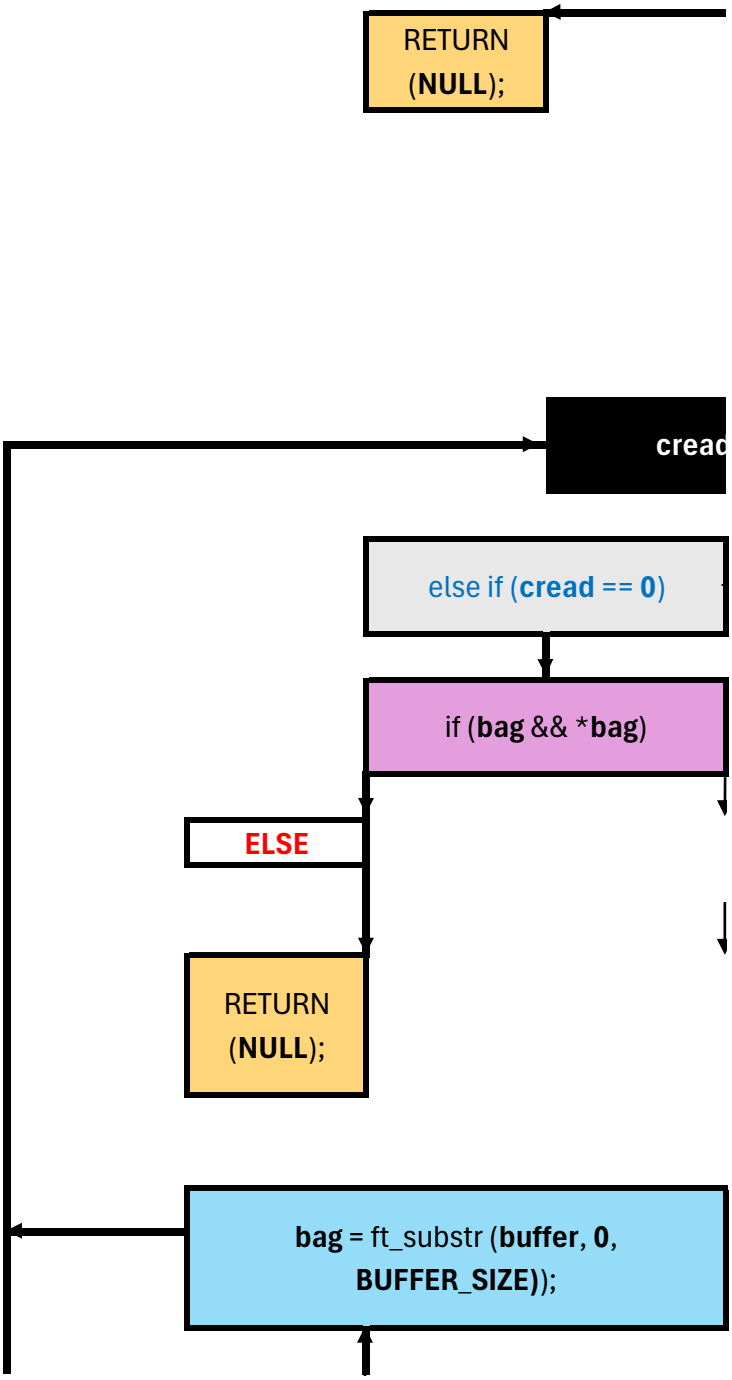
VARIABLES					
NUMBER	TYPE	KIND	POINTER	NAME	ARRAY
1		int		fd	
2	static	char	*	remain	
3		char	*	bag	
4		int		cread	
5		char		buffer	[BUFFER_SIZE + 1]
6		char	*	line	
7		char	*	temp	

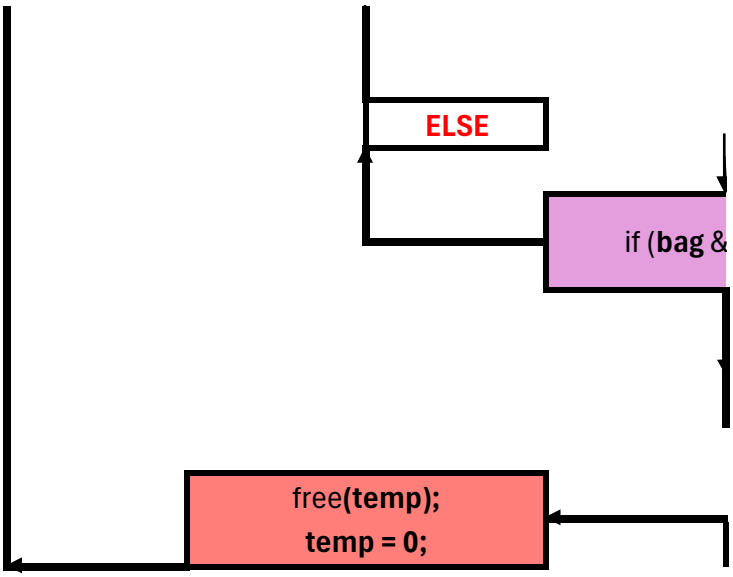



RETURN
(NULL);
YES
if((fd < 0)
(BUFFI
if (remain && *remain)
ELSE
free(remain);
remain = 0;
cread = read(fd, buffer, BUFFER_SIZE);
else if (cread == BUFFER_SIZE)

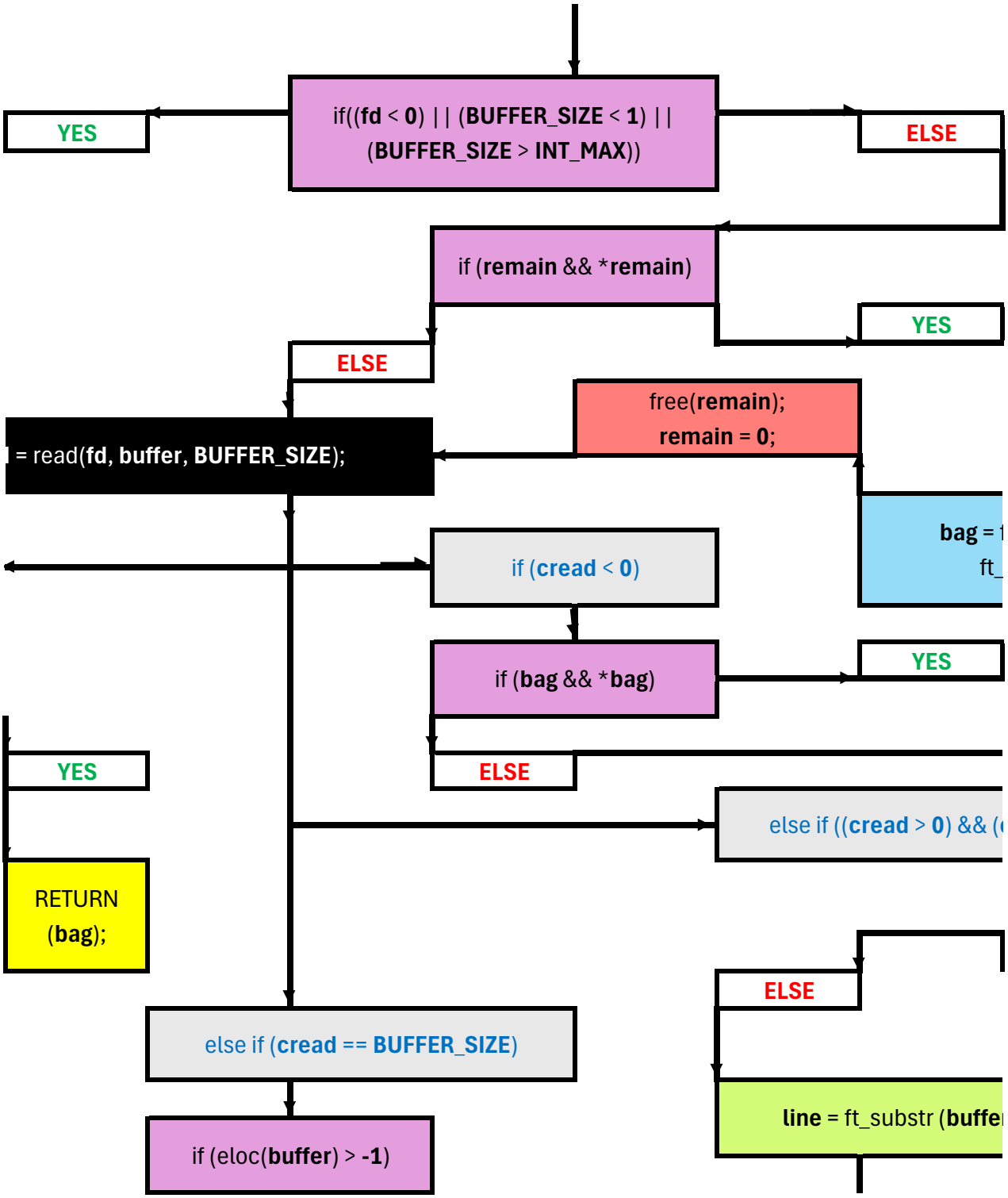

| | (BUFFER_SIZE < 1) | |
ER_SIZE > INT_MAX))
ELSE
if (remain && *remain)
YES
if (eloc(remain) > -1)
ELSE
YES
bag = ft_substr (remain, 0,
ft_strlen(remain));
if((eloc
ELSE
free(remain);
remain = 0;

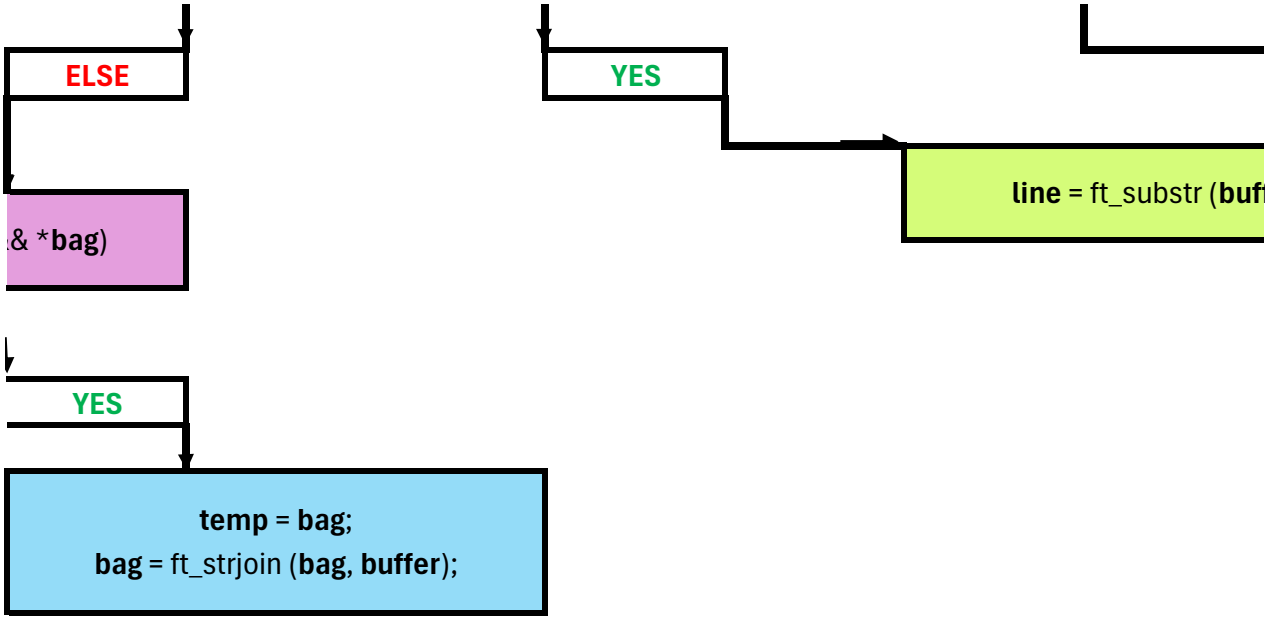

line = ft_substr (remain, 0, eloc(remain));
c(remain) + 1) < ft_strlen(remain))
YES
temp = remain;
remain = ft_substr(remain, eloc(remain) + 1, ft_strlen(remain) - eloc(remain));
free(temp);
temp = 0;
RETURN (line);


RETURN
(NULL);
cread
else if (cread == 0)
if (bag && *bag)
ELSE
RETURN
(NULL);
bag = ft_substr (buffer, 0,
BUFFER_SIZE));

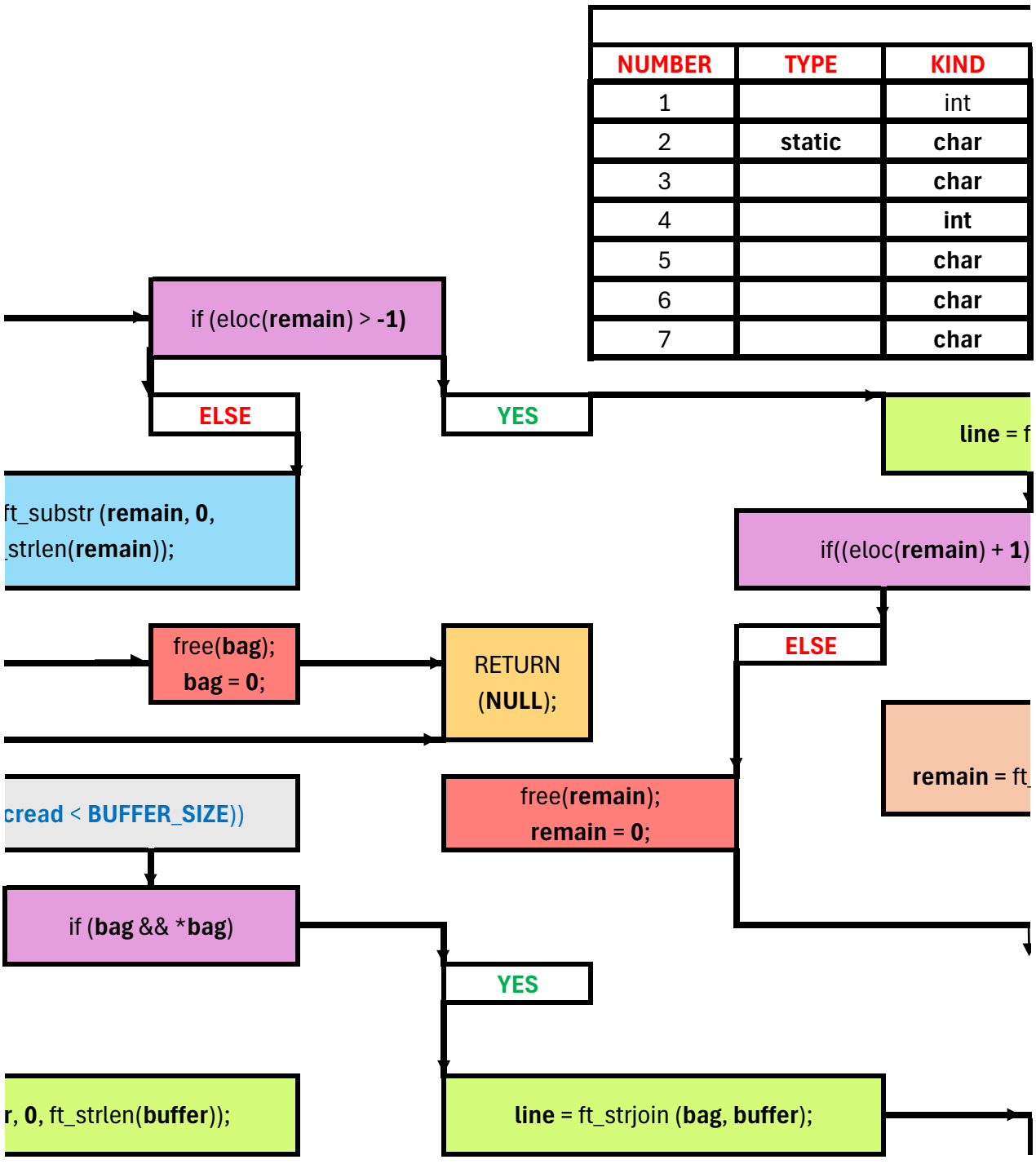


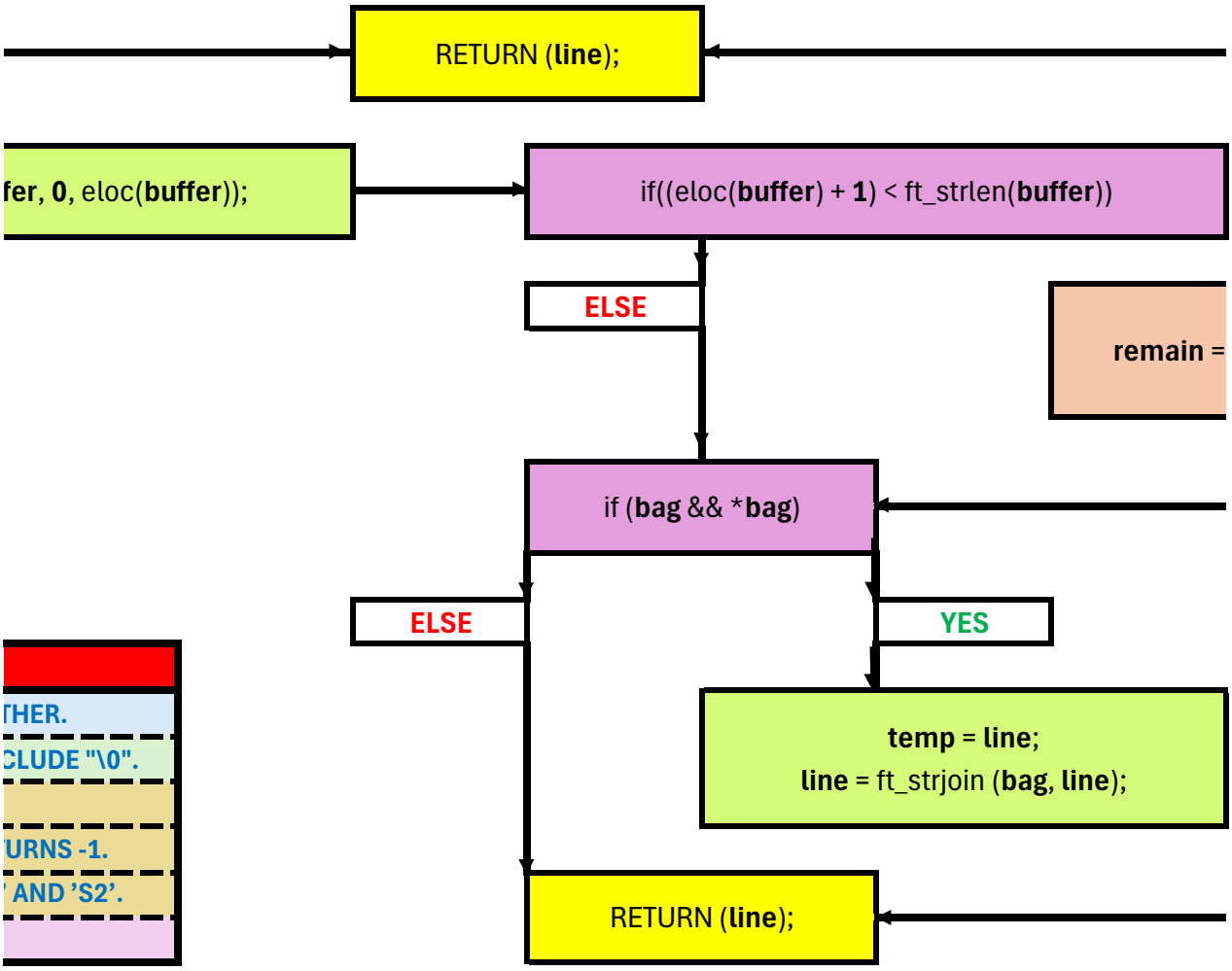
COPY STRING:	size_t ft_strlcpy(
STRING'S LENGHT:	int ft_strlen(cha
MAKE A SUB-STRING:	char *ft_substr(
LOCATE END OF LINE:	int eloc(char *tx
JOIN STRINGS:	char *ft_strjoin(
GET NEXT LINE:	char *get_next_l


if((fd < 0) | | (BUFFER_SIZE < 1) | |
(BUFFER_SIZE > INT_MAX))
YES
ELSE
if (remain && *remain)
YES
ELSE
free(remain);
remain = 0;
= read(fd, buffer, BUFFER_SIZE);
bag = f
ft_
if (cread < 0)
if (bag && *bag)
YES
YES
ELSE
else if ((cread > 0) && (
RETURN
(bag);
ELSE
else if (cread == BUFFER_SIZE)
line = ft_substr (buffe
if (eloc(buffer) > -1)

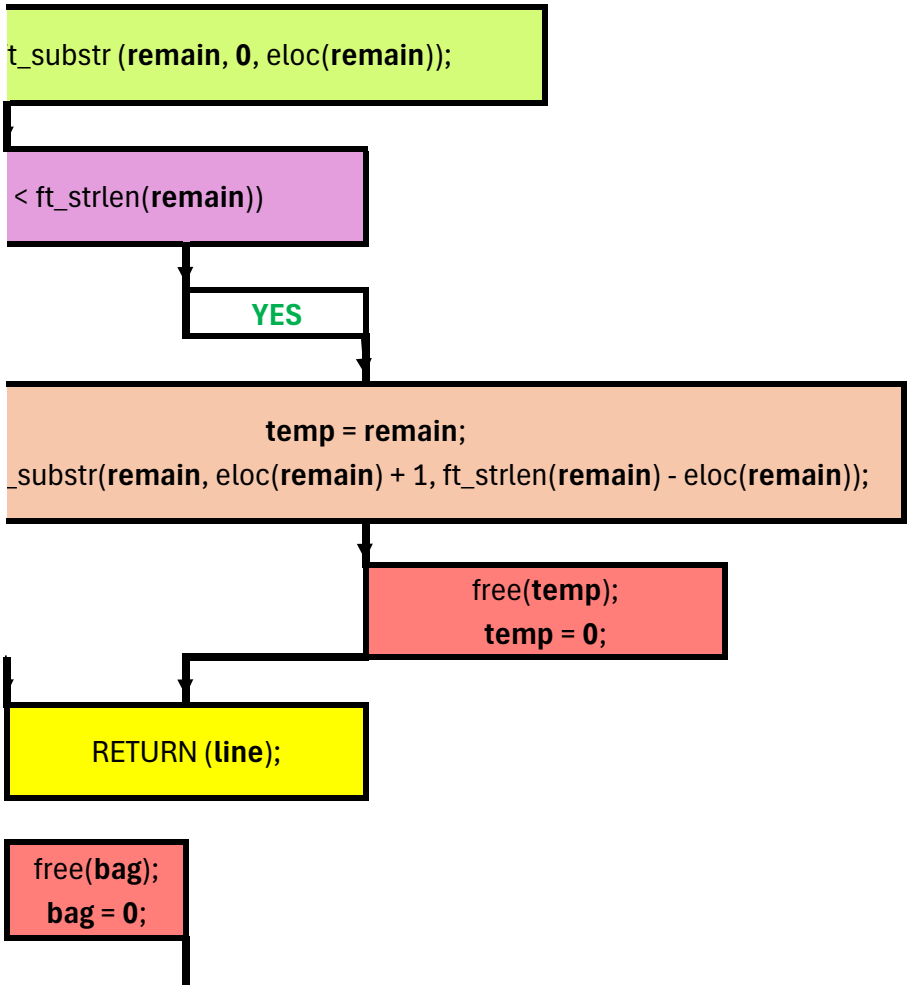


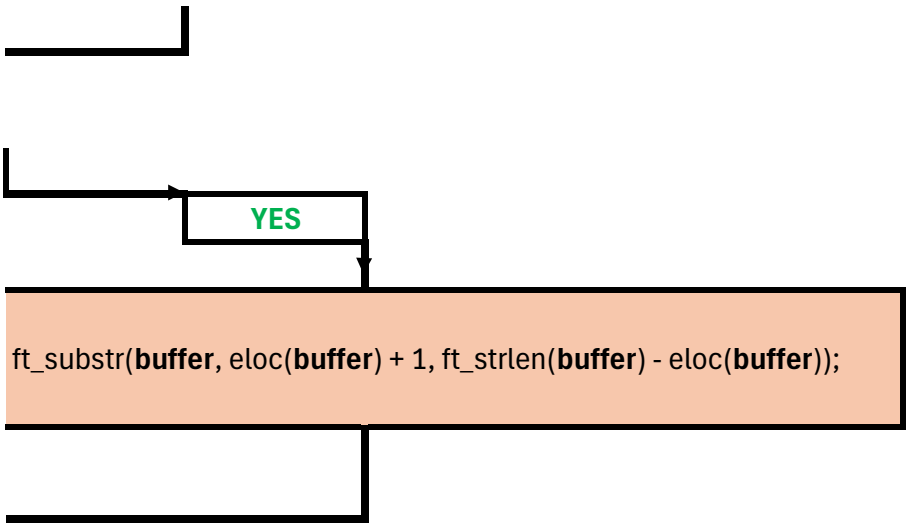
EXTRA FORMULAS:	
(char *dst, const char *src, size_t n);	COPY X NUMBER OF CHARACTERS FROM A STRING TO ANOT
r *txt);	COUNTS THE TOTAL NUMBER OF CHARACTERS, DOESNT IN
char const *s, unsigned int start, size_t len);	CREATE A SUBSTRING FROM A STRING.
t);	SEARCH FOR THE POSITION OF FIRST "\N", OTHERWISE RET
char const *s1, char const *s2);	NEW STRING, THE RESULT OF THE CONCATENATION OF 'S1'
line(int fd);	GETS NEXT LINE FROM A FILE DESCRIPTOR.

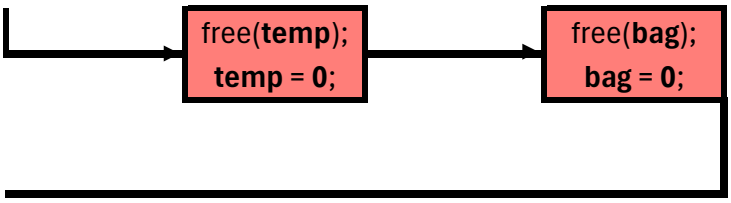

NUMBER
TYPE
KIND
1
int
2
static
char
3
char
4
int
5
char
6
char
7
char
if (eloc(remain) > -1)
ELSE
YES
line = f
ft_substr (remain, 0,
_strlen(remain));
if((eloc(remain) + 1)
free(bag);
bag = 0;
RETURN
(NULL);
ELSE
remain = ft_
cread < BUFFER_SIZE))
free(remain);
remain = 0;
if (bag && *bag)
YES
r, 0, ft_strlen(buffer));
line = ft_strjoin (bag, buffer);

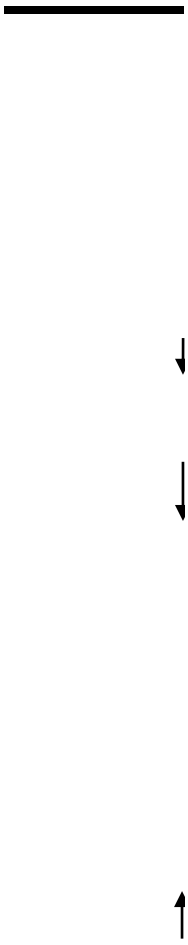

RETURN (line);
fer, 0, eloc(buffer));
if((eloc(buffer) + 1) < ft_strlen(buffer))
ELSE
remain =
if (bag && *bag)
ELSE
YES
temp = line;
line = ft_strjoin (bag, line);
RETURN (line);
THER.
CLUDE "\0".
URNS -1.
' AND 'S2'.

VARIABLES		
POINTER	**NAME**	**ARRAY**
	fd	
*	**remain**	
*	**bag**	
	cread	
	buffer	**[BUFFER_SIZE + 1]**
*	**line**	
*	**temp**	

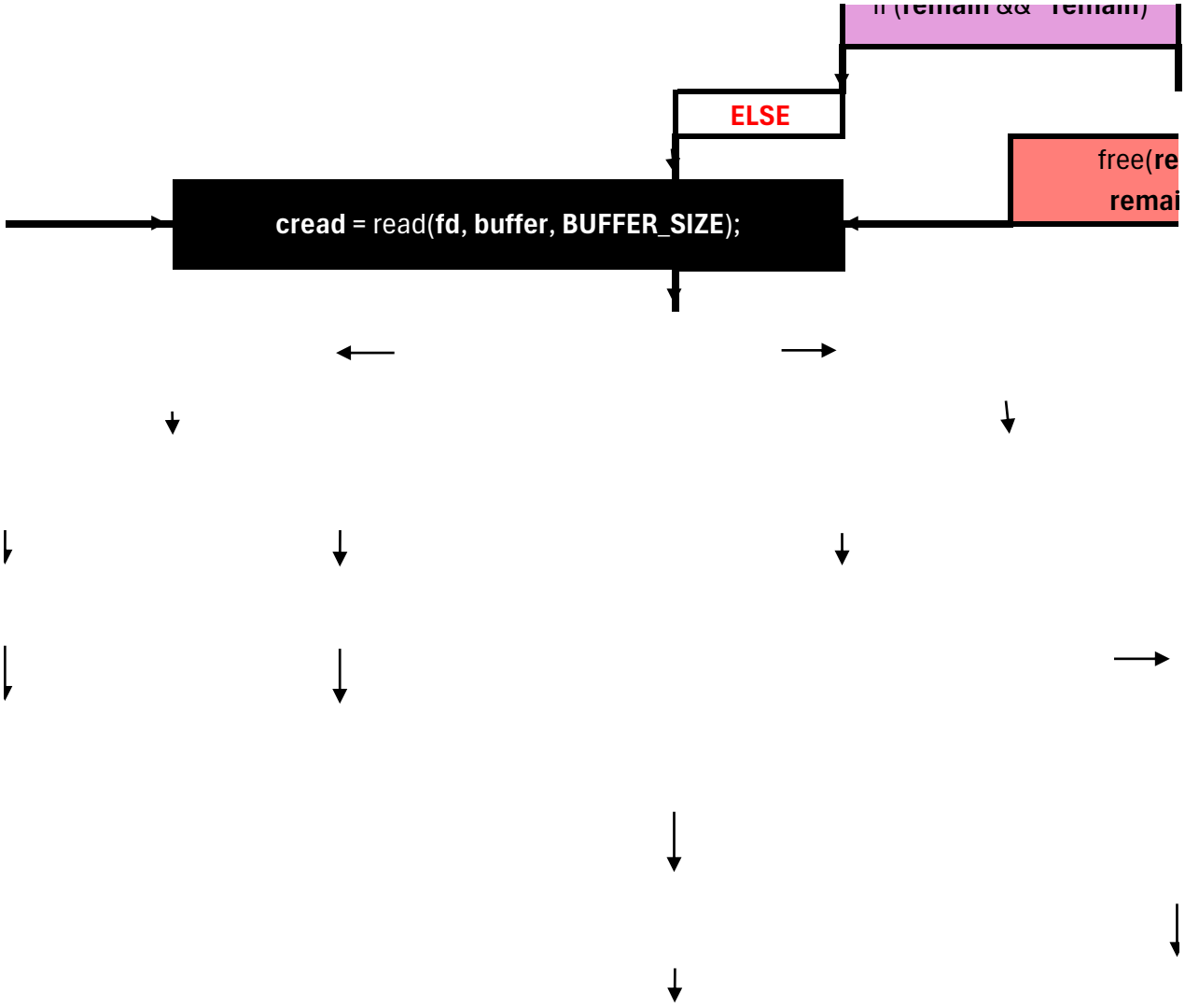


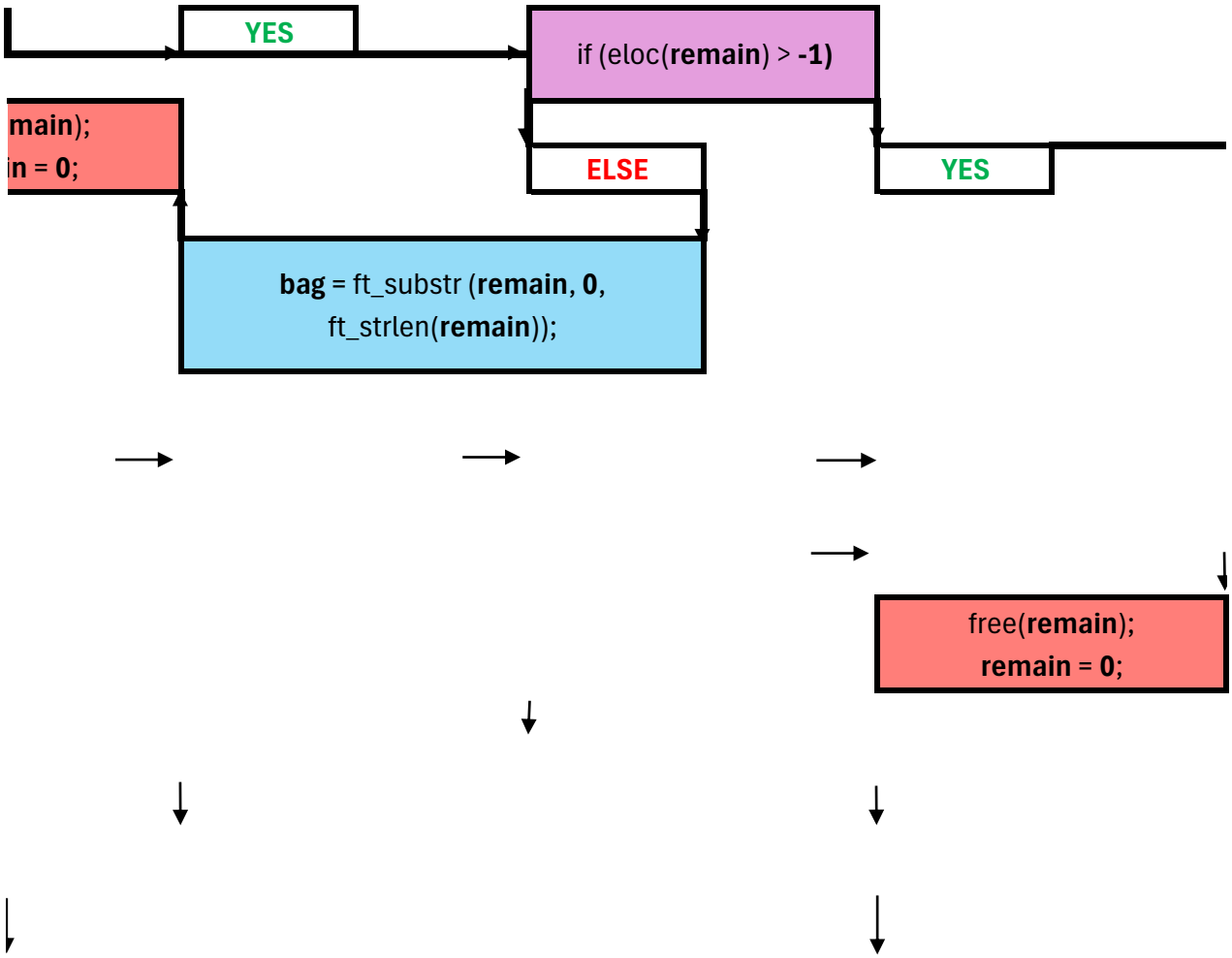

YES
ft_substr(buffer, eloc(buffer) + 1, ft_strlen(buffer) - eloc(buffer));

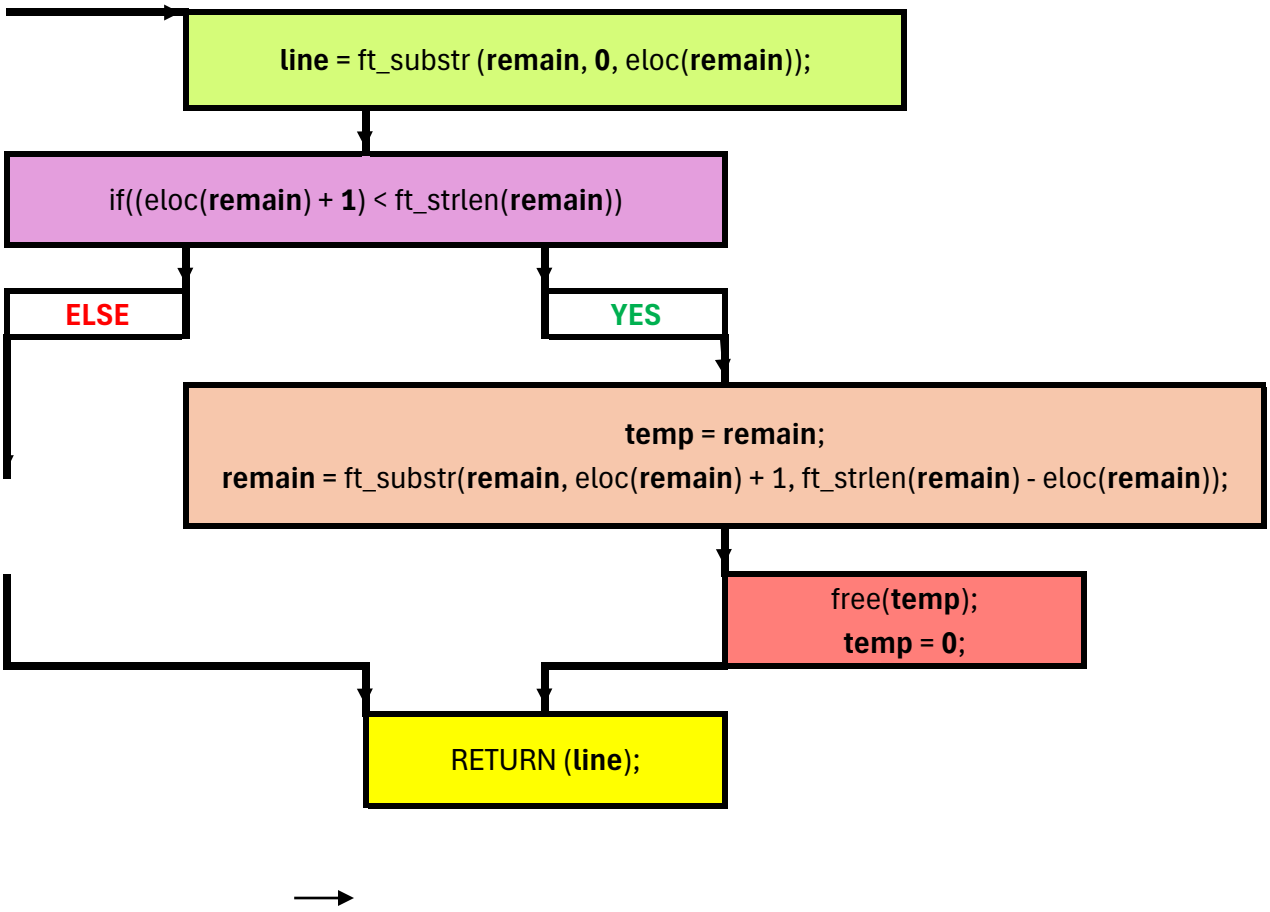

free(temp);
temp = 0;
free(bag);
bag = 0;




if (remain && *remain)


ELSE
free(re
remai
cread = read(fd, buffer, BUFFER_SIZE);


YES
if (eloc(remain) > -1)
main);
in = 0;
ELSE
YES
bag = ft_substr (remain, 0,
ft_strlen(remain));
free(remain);
remain = 0;


line = ft_substr (remain, 0, eloc(remain));
if((eloc(remain) + 1) < ft_strlen(remain))
ELSE
YES
temp = remain;
remain = ft_substr(remain, eloc(remain) + 1, ft_strlen(remain) - eloc(remain));
free(temp);
temp = 0;
RETURN (line);

```
char   *initializ
char   *read_ar
**r);
void   cleanup_
char *b);

char   *get_nex
  char   *line =
  static char

  line = initializ
  if (line != NUL
    return (line)

  line = read_ar
  cleanup_mer
  return (line);
}
```

```
e_and_preprocess(int fd);
nd_process_input(int fd, char

_memory(char *line, char *r,


t_line(int fd) {
 NULL;
*r = NULL;

e_and_preprocess(fd);
_L)
);

nd_process_input(fd, &r);
nory(line, r, NULL);
```

```
char    *initialize_and_preprocess(int fd) {
   char    *line = NULL;
   static char *r = NULL;

   if ((fd < 0) || (BUFFER_SIZE < 1) ||
(BUFFER_SIZE > INT_MAX))
      return NULL;

   if (r && *r) {
      int loc = eloc(r);
      if (loc > -1) {
         line = ft_substr(r, 0, loc);
         if (loc < ft_strlen(r)) {
            char *tmp = ft_substr(r, loc,
ft_strlen(r) - loc);
            free(r);
            r = tmp;
         } else {
            free(r);
            r = NULL;
         }
         return line;
      } else {
         free(r);
         r = NULL;
      }
   }

   return line;
}
```

```
char *read_and_process_input
**r_ptr) {
   char *line = NULL;
   char buff[BUFFER_SIZE + 1];
   int cread = 0;

   clean(buff, BUFFER_SIZE);
   cread = read(fd, buff, BUFFEI

   if (cread <= 0) {
      return NULL;
   } else {
      char *r = NULL;
      if (*r_ptr)
         r = *r_ptr;
      *r_ptr = ft_strjoin(r, buff);
      free(r);
      if (!*r_ptr)
         return NULL;

      line = ft_substr(*r_ptr, 0, el
      if (eloc(*r_ptr) >= 0) {
         char *tmp = *r_ptr;
         *r_ptr = ft_substr(*r_ptr,
ft_strlen(*r_ptr) - eloc(*r_ptr));
         free(tmp);
      } else {
         free(*r_ptr);
         *r_ptr = NULL;
      }
      return line;
   }
}
```

```
t(int fd, char

R_SIZE);

oc(*r_ptr));

eloc(*r_ptr),
```