문자열 입력 함수: fgets()와 scanf() 차이점 #1

- fgets()

```
#include <stdio.h>
char *fgets(char *s, int size, FILE *stream);
```

  - stream: stdin 사용 가능
- 공백 입력을 허용
- 마지막 Enter 입력
  - 줄바꿈 문자('\n') + NULL문자('\0')를 문자열에 추가함

- scanf()

```
#include <stdio.h>
int scanf(const char *format, ...);
```

- 공백을 기준으로 입력이 분리됨
- 마지막 Enter 입력
  - 줄바꿈 문자를 문자열에 포함시키지 않고 NULL문자('\0')만 문자열에 추가

문자열 함수 fgets()와 scanf() 차이점 #2

- 사용 예제: Visual Studio에서 확인
  - 화면에서 “hello”입력 후 Enter키

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

#define BUF_SIZE 1024

int main(int argc, const char* argv[]) {
    char msg1[BUF_SIZE];
    char msg2[BUF_SIZE];

    memset(msg1, 0, sizeof(msg1));
    memset(msg2, 0, sizeof(msg2));
    // scanf() 사용
    printf("Input string1 -> ");
    scanf("%s", msg1);
    printf("msg1: %s, len: %d\n", msg1, strlen(msg1));
    getchar();

    // fgets() 사용
    printf("Input string2 -> ");
    fgets(msg2, BUF_SIZE, stdin);
    printf("msg2: %s, len: %d\n", msg2, strlen(msg2));

    return 0;
}
```

scanf()

이름	값	형식
msg1	0x004ff5c4 "hello"	char[1024]
msg2	0x004ff1bc "hello\n"	char[1024]

↳ fgets()로 "hello" 입력

scanf() 실행 결과

msg1	0x004ff5c4 "hello"	char[1024]
[0]	104 'h'	char
[1]	101 'e'	char
[2]	108 'l'	char
[3]	108 'l'	char
[4]	111 'o'	char
[5]	0 '\0'	char
[6]	0 '\0'	char
[7]	0 '\0'	char

scanf() 실행 결과
- ‘\0’만 추가
- strlen(msg1): 5

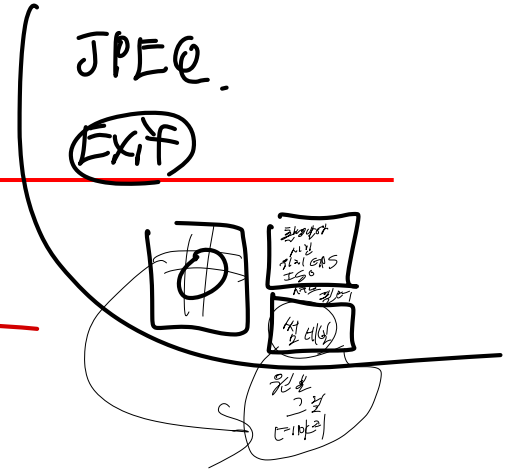
fgets() 실행 결과

msg2	0x004ff1bc "hello\n"	char[1024]
[0]	104 'h'	char
[1]	101 'e'	char
[2]	108 'l'	char
[3]	108 'l'	char
[4]	111 'o'	char
[5]	10 '\n'	char
[6]	0 '\0'	char
[7]	0 '\0'	char

fgets() 실행 결과
- ‘\n’과 ‘\0’추가
- strlen(msg2): 6

strlen(msg2)-1

파일의 끝 (EOF) 표시



- 파일의 끝 표시
  - Linux 시스템에서는 파일에 저장된 데이터를 가지고 파일의 끝을 검사하지 않음
  - File system에 해당 파일의 길이를 저장
    - 파일 내부에는 파일의 끝(EOF) 표시를 위한 special character는 없음

- 파일의 끝 감지
  - read() 함수
    - return 0 (NULL)
  - fgetc(), getc(), getchar()
    - return -1 (EOF)

#define EOF -1

```c
#include <stdio.h>

int main()
{
    FILE *stream;
    int c;

    stream = fopen("sample.txt", "r");
    if(stream != NULL)
    {
        while((c = fgetc(stream)) != EOF)
        {
            putchar(c);
        }
    }
    return 0;
}
```

https://latedev.wordpress.com/2012/12/04/all-about-eof/

파일에 데이터 쓰기: 배열의 크기를 지정하지 않음

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
void error_handling(char* message);

int main(void)
{
    int fd;
    int size;
    char buf[]="Let's go!\n";

    fd = open("data.txt", O_CREAT|O_WRONLY|O_TRUNC, 0644);
    if(fd == -1)
        error_handling("open() error!");

    printf("file descriptor: %d \n", fd);
    size = write(fd, buf, sizeof(buf));
    printf("write size: %d\n", size);
    if(size == -1)
        error_handling("write() error!");

    close(fd);
    return 0;
}
```

크기를 지정하지 않음

⇒ 자동으로 NULL을 붙임

헷갈리지 말 것!

strlen(buf)

sizeof(buf)

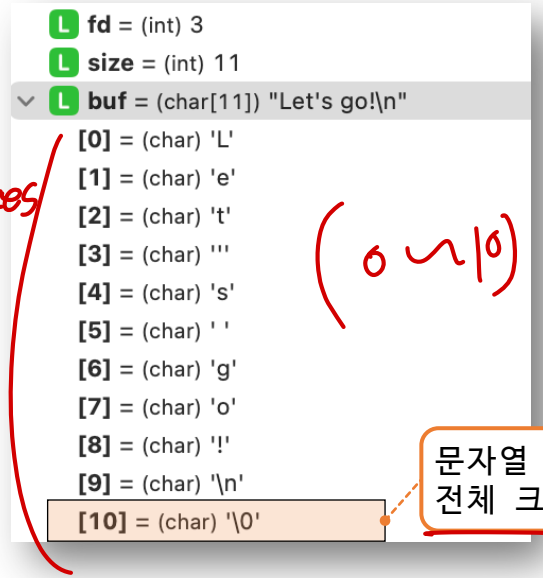
```
(base) [redacted]f$ hexyl data.txt
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────────┐
│00000000│ 4c 65 74 27 73 20 67 6f │ 21 00                   │Let's go│!0      │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
```

NULL

```c
void error_handling(char* message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

컴파일 및 실행

```
$ gcc low_open.c -o lopen
$ ./lopen
file descriptor: 3
write size: 11
```



11 bytes

(0 ~ 10)

문자열 끝에 자동으로 NULL('\0') 추가
전체 크기(size=11)

파일에 데이터 쓰기: 고정된 크기의 배열 저장

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
void error_handling(char* message);

int main(void)
{
    int fd;
    int size;
    char buf[10]="Let's go!\n";

    fd = open("data.txt", O_CREAT|O_WRONLY|O_TRUNC, 0644);
    if(fd == -1)
        error_handling("open() error!");

    printf("file descriptor: %d, strlen: %ld \n", fd, strlen(buf));

    size = write(fd, buf, sizeof(buf));
    printf("write size: %d\n", size);
    if(size == -1)
        error_handling("write() error!");

    close(fd);
    return 0;
}
```

배열의 크기를 문자열의 길이만큼 배정

```c
void error_handling(char* message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

```
$ ./low_open1
file descriptor: 3, strlen: 10
write size: 10
```

- fd = (int) 3
- size = (int) 10
- buf = (char[10]) "Let's go!\n"
  - [0] = (char) 'L'
  - [1] = (char) 'e'
  - [2] = (char) 't'
  - [3] = (char) '''
  - [4] = (char) 's'
  - [5] = (char) ' '
  - [6] = (char) 'g'
  - [7] = (char) 'o'
  - [8] = (char) '!'
  - [9] = (char) 0x0a

```
(base) [redacted]$ hexyl data.txt
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────────┐
│00000000│ 4c 65 74 27 73 20 67 6f │ 21 0a                   │Let's go│!_      │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
```

문자열 끝에 NULL 없음
전체 크기(size=10)

파일에 데이터 쓰기: 문자열 보다 큰 크기의 배열 저장

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
void error_handling(char* message);

int main(void)
{
    int fd;
    int size;
    char buf[100]="Let's go!\n";

    fd = open("data.txt", O_CREAT|O_WRONLY|O_TRUNC, 0644);
    if(fd == -1)
        error_handling("open() error!");

    printf("file descriptor: %d, strlen: %ld \n", fd, strlen(buf));

    size = write(fd, buf, sizeof(buf));
    printf("write size: %d\n", size);
    if(size == -1)
        error_handling("write() error!");

    close(fd);
    return 0;
}
```

문자열의 길이 : 10 bytes
⇒ strlen(buf)

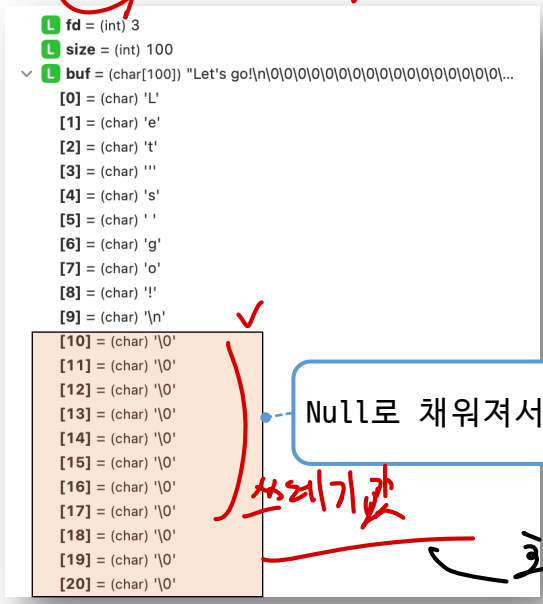
배열의 크기를 문자열 보다 크게 설정한 경우

문자열의 길이와 상관없이 buf[100]의 길이만큼 저장(100 bytes)

100 bytes의 파일 생성.

```c
void error_handling(char* message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

```
$ ./low_open1
file descriptor: 3, strlen: 10
write size: 100
```



Null로 채워져서 저장

쓰레기값

초기화를 안하면

파일에 데이터 쓰기: 문자열 보다 큰 크기의 배열 저장

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
void error_handling(char* message);

int main(void)
{
    int fd;
    int size;
    char buf[100]="Let's go!\n";

    fd = open("data.txt", O_CREAT|O_WRONLY|O_TRUNC, 0644);
    if(fd == -1)
        error_handling("open() error!");

    printf("file descriptor: %d, strlen: %ld \n", fd, strlen(buf));

    size = write(fd, buf, strlen(buf));
    printf("write size: %d\n", size);
    if(size == -1)
        error_handling("write() error!");

    close(fd);
    return 0;
}
```

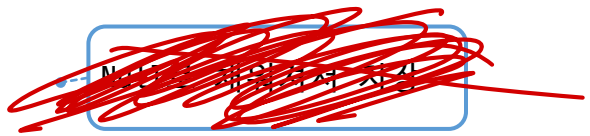
배열의 크기를 문자열 보다 크게 설정한 경우

strlen(buf)만큼 파일에 저장 (10 bytes)

```c
void error_handling(char* message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

```
$ ./low_open1
file descriptor: 3, strlen: 10
write size: 10
```

```
temp % hexyl data.txt
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────────┐
│00000000│ 4c 65 74 27 73 20 67 6f │ 21 0a                   │Let's go│!_      │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
```



ASCII Code Table

ASCII CODE TABLE

DEC	HEX	OCT	CHAR
0	0x00	000	NUL(null)
1	0x01	001	SOH(start of heading)
2	0x02	002	STX(start of text)
3	0x03	003	ETX(end of text)
4	0x04	004	EOT(end of transmission)
5	0x05	005	ENQ(enquiry)
6	0x06	006	ACK(acknowledge)
7	0x07	007	BEL(bell)
8	0x08	010	BS(backspace)
9	0x09	011	HT(horizontal tab)
10	0x0A	012	LF(NL line feed, new line)
11	0x0B	013	VT(vertical tab)
12	0x0C	014	FF(NP from feed, new page)
13	0x0D	015	CR(carriage return)
14	0x0E	016	SO(shift out)
15	0x0F	017	SI(shift in)
16	0x10	020	DLE(data link escape)
17	0x11	021	DC1(device control 1)
18	0x12	022	DC2(device control 2)
19	0x13	023	DC3(device control 3)
20	0x14	024	DC3(device control 4)
21	0x15	025	NAK(negative acknowledge)
22	0x16	026	SYN(synchronous idle)
23	0x17	027	ETB(end of trans. block)
24	0x18	030	CAN(cancel)
25	0x19	031	EM(end of medium)
26	0x1A	032	SUB(substitute)
27	0x1B	033	ESC(escape)
28	0x1C	034	FS(file separator)
29	0x1D	035	GS(group separator)
30	0x1E	036	RS(record separator)
31	0x1F	037	US(unit separator)

DEC	HEX	OCT	CHAR
32	0x20	040	SP(Space)
33	0x21	041	!
34	0x22	042	"
35	0x23	043	#
36	0x24	044	$
37	0x25	045	%
38	0x26	046	&
39	0x27	047	'
40	0x28	050	(
41	0x29	051	)
42	0x2A	052	*
43	0x2B	053	+
44	0x2C	054	,
45	0x2D	055	-
46	0x2E	056	.
47	0x2F	057	/
48	0x30	060	0
49	0x31	061	1
50	0x32	062	2
51	0x33	063	3
52	0x34	064	4
53	0x35	065	5
54	0x36	066	6
55	0x37	067	7
56	0x38	070	8
57	0x39	071	9
58	0x3A	072	:
59	0x3B	073	;
60	0x3C	074	<
61	0x3D	075	=
62	0x3E	076	>
63	0x3F	077	?

DEC	HEX	OCT	CHAR
64	0x40	100	@
65	0x41	101	A
66	0x42	102	B
67	0x43	103	C
68	0x44	104	D
69	0x45	105	E
70	0x46	106	F
71	0x47	107	G
72	0x48	110	H
73	0x49	111	I
74	0x4A	112	J
75	0x4B	113	K
76	0x4C	114	L
77	0x4D	115	M
78	0x4E	116	N
79	0x4F	117	O
80	0x50	120	P
81	0x51	121	Q
82	0x52	122	R
83	0x53	123	S
84	0x54	124	T
85	0x55	125	U
86	0x56	126	V
87	0x57	127	W
88	0x58	130	X
89	0x59	131	Y
90	0x5A	132	Z
91	0x5B	133	[
92	0x5C	134	W
93	0x5D	135	]
94	0x5E	136	^
95	0x5F	137	_

DEC	HEX	OCT	CHAR
96	0x60	140	'
97	0x61	141	a
98	0x62	142	b
99	0x63	143	c
100	0x64	144	d
101	0x65	145	e
102	0x66	146	f
103	0x67	147	g
104	0x68	150	h
105	0x69	151	i
106	0x6A	152	j
107	0x6B	153	k
108	0x6C	154	l
109	0x6D	155	m
110	0x6E	156	n
111	0x6F	157	o
112	0x70	160	p
113	0x71	161	q
114	0x72	162	r
115	0x73	163	s
116	0x74	164	t
117	0x75	165	u
118	0x76	166	v
119	0x77	167	w
120	0x78	170	x
121	0x79	171	y
122	0x7A	172	z
123	0x7B	173	{
124	0x7C	174	|
125	0x7D	175	}
126	0x7E	176	~
127	0x7F	177	DEL