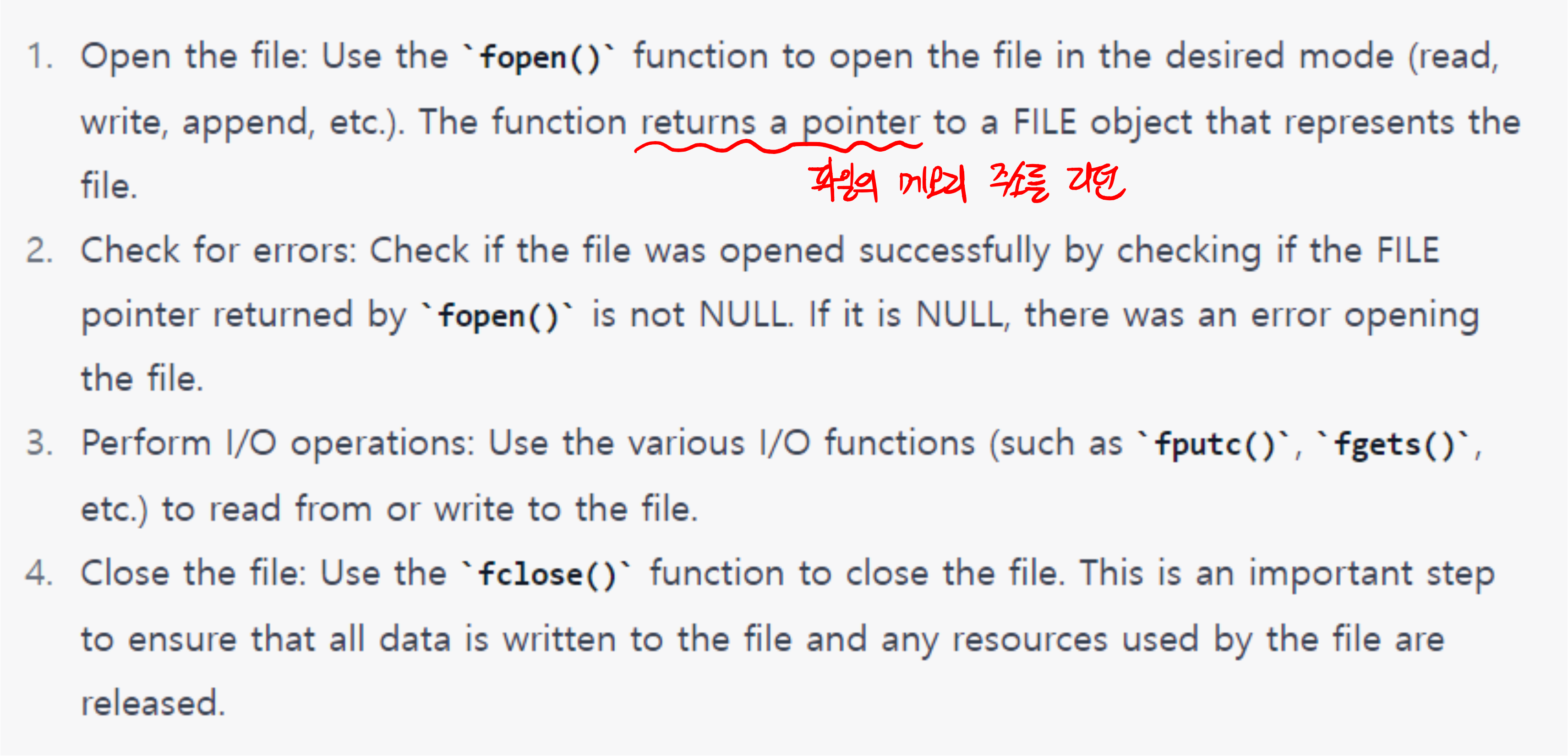
File I/O

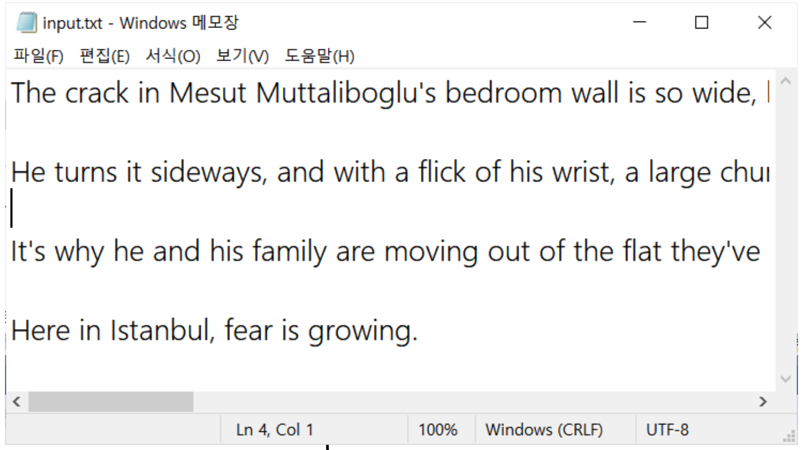
- The general **steps for performing file I/O operations** in C :



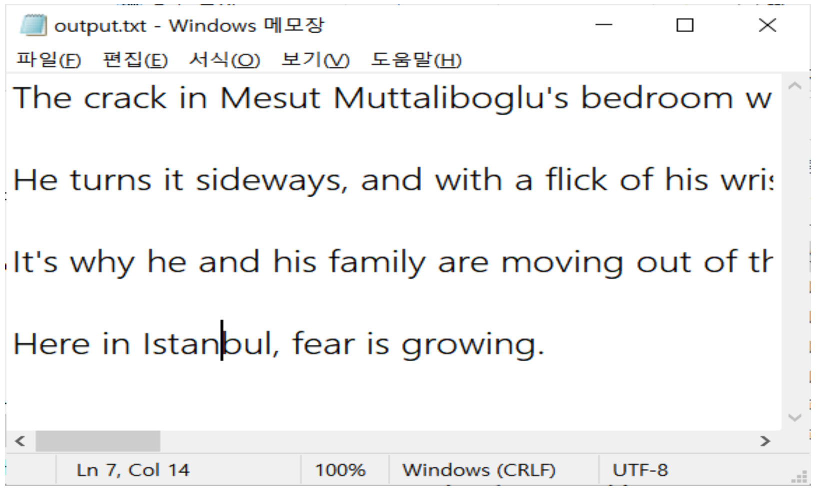
1. Open the file: Use the `fopen()` function to open the file in the desired mode (read, write, append, etc.). The function returns a pointer to a FILE object that represents the file.

   파일의 메모리 주소를 리턴

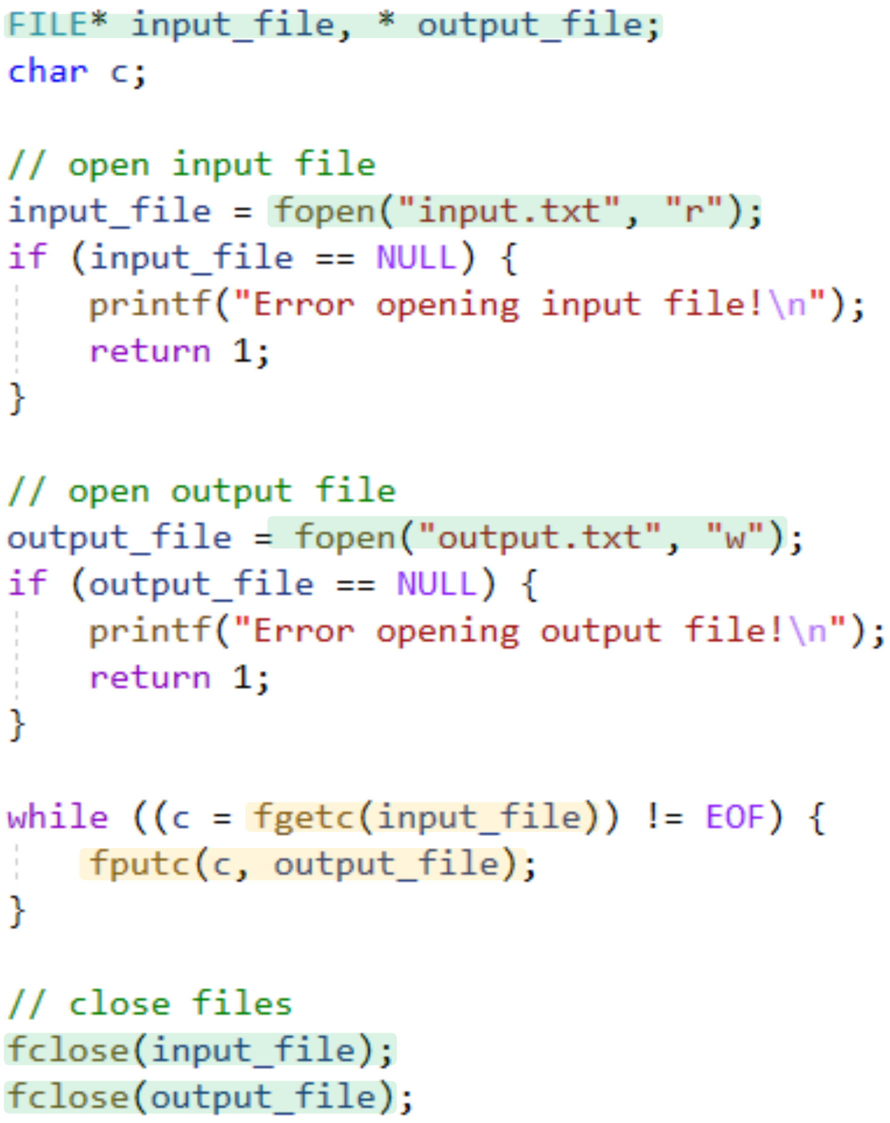
2. Check for errors: Check if the file was opened successfully by checking if the FILE pointer returned by `fopen()` is not NULL. If it is NULL, there was an error opening the file.
3. Perform I/O operations: Use the various I/O functions (such as `fputc()`, `fgets()`, etc.) to read from or write to the file.
4. Close the file: Use the `fclose()` function to close the file. This is an important step to ensure that all data is written to the file and any resources used by the file are released.


input.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 4, Col 1
100%
Windows (CRLF)
UTF-8

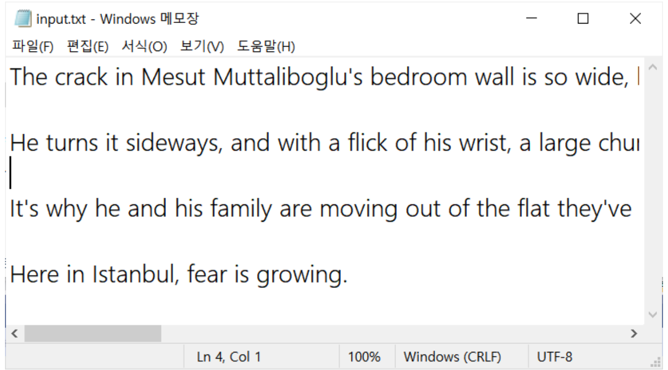
```
The crack in Mesut Muttaliboglu's bedroom wall is so wide, I

He turns it sideways, and with a flick of his wrist, a large chu

It's why he and his family are moving out of the flat they've

Here in Istanbul, fear is growing.
```


output.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 7, Col 14
100%
Windows (CRLF)
UTF-8

```
The crack in Mesut Muttaliboglu's bedroom w

He turns it sideways, and with a flick of his wri

It's why he and his family are moving out of th

Here in Istanbul, fear is growing.
```



```c
FILE* input_file, * output_file;
char c;

// open input file
input_file = fopen("input.txt", "r");
if (input_file == NULL) {
    printf("Error opening input file!\n");
    return 1;
}

// open output file
output_file = fopen("output.txt", "w");
if (output_file == NULL) {
    printf("Error opening output file!\n");
    return 1;
}

while ((c = fgetc(input_file)) != EOF) {
    fputc(c, output_file);
}

// close files
fclose(input_file);
fclose(output_file);
```

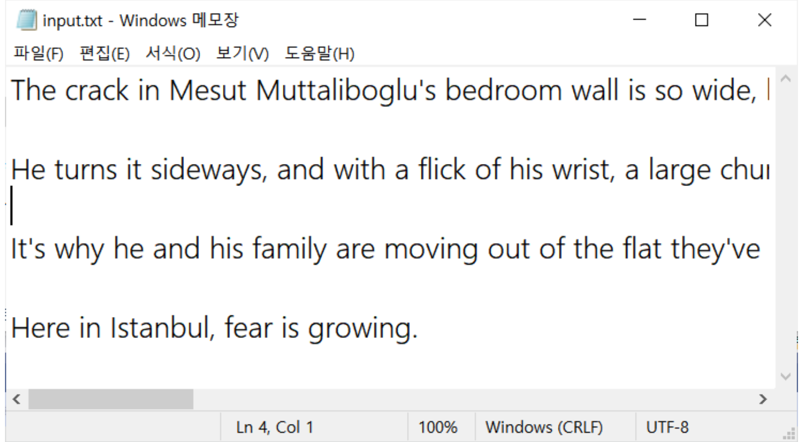

input.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 4, Col 1
100%
Windows (CRLF)
UTF-8

The crack in Mesut Muttaliboglu's bedroom wall is so wide, I

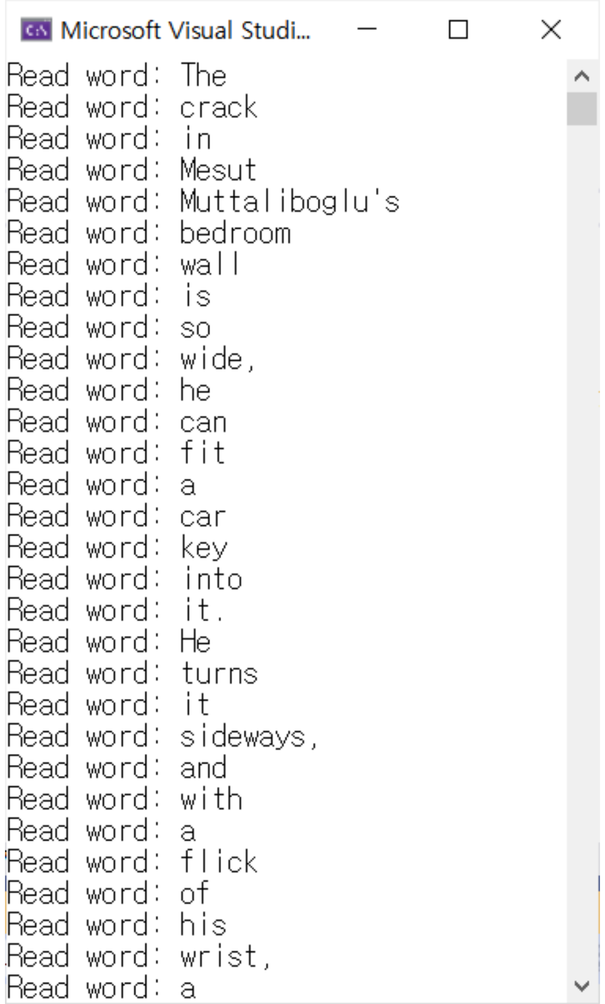
He turns it sideways, and with a flick of his wrist, a large chu

It's why he and his family are moving out of the flat they've

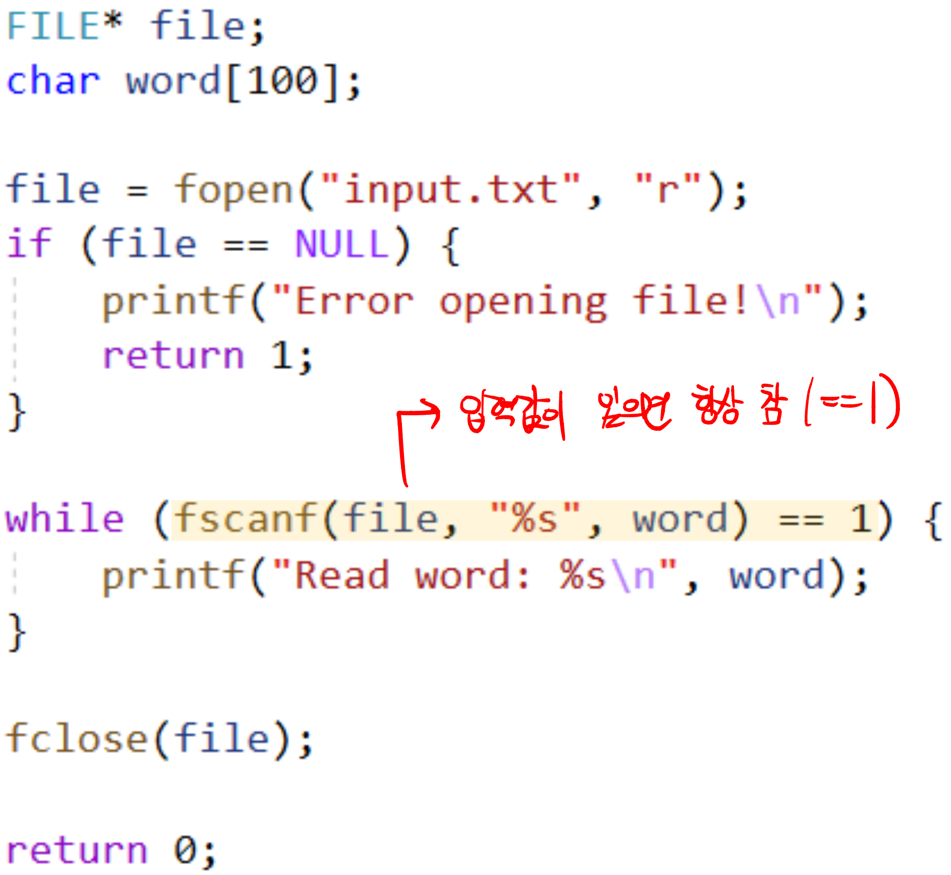
Here in Istanbul, fear is growing.


input.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 4, Col 1
100%
Windows (CRLF)
UTF-8

```
The crack in Mesut Muttaliboglu's bedroom wall is so wide, l

He turns it sideways, and with a flick of his wrist, a large chu

It's why he and his family are moving out of the flat they've

Here in Istanbul, fear is growing.
```

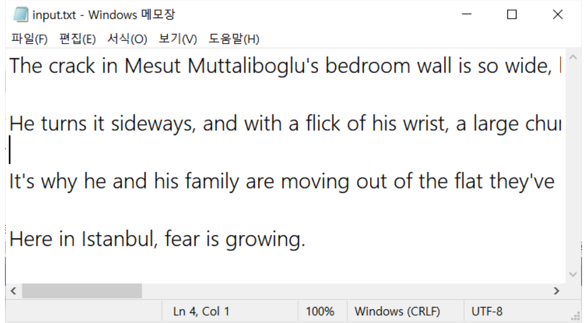


```
Read word: The
Read word: crack
Read word: in
Read word: Mesut
Read word: Muttaliboglu's
Read word: bedroom
Read word: wall
Read word: is
Read word: so
Read word: wide,
Read word: he
Read word: can
Read word: fit
Read word: a
Read word: car
Read word: key
Read word: into
Read word: it.
Read word: He
Read word: turns
Read word: it
Read word: sideways,
Read word: and
Read word: with
Read word: a
Read word: flick
Read word: of
Read word: his
Read word: wrist,
Read word: a
```



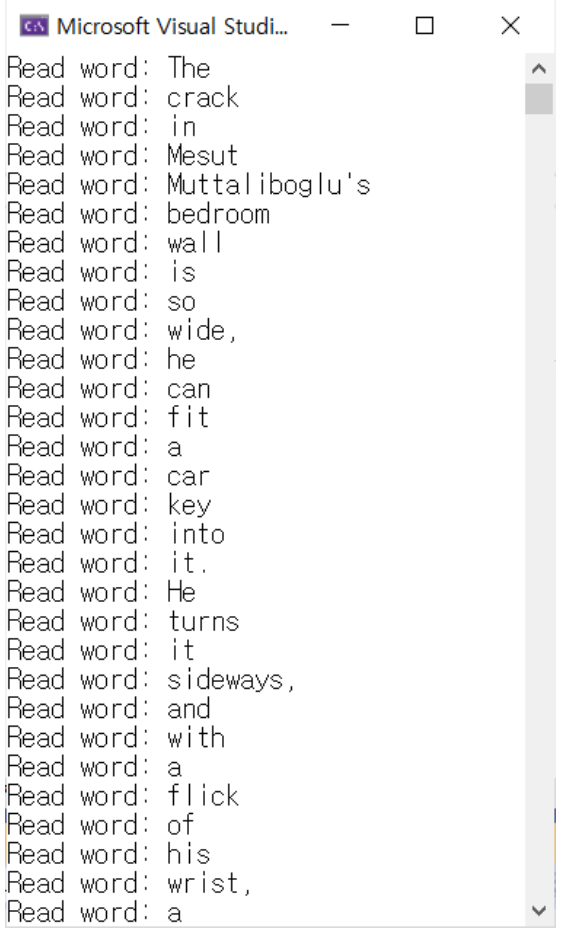
```c
FILE* file;
char word[100];

file = fopen("input.txt", "r");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

while (fscanf(file, "%s", word) == 1) {
    printf("Read word: %s\n", word);
}

fclose(file);

return 0;
```

→ 입력값이 있으면 항상 참 (==1)

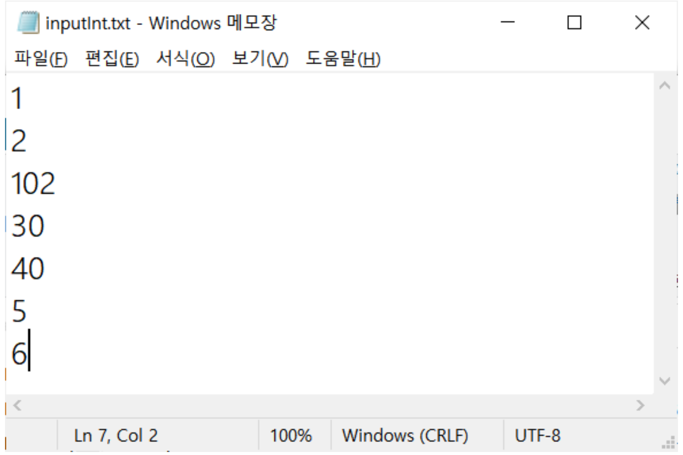


input.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 4, Col 1
100%
Windows (CRLF)
UTF-8

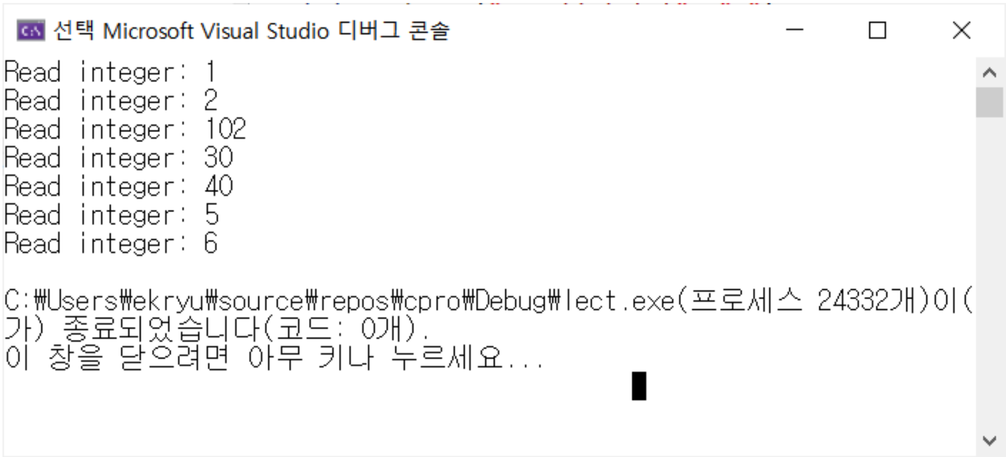
```
The crack in Mesut Muttaliboglu's bedroom wall is so wide, I

He turns it sideways, and with a flick of his wrist, a large chu

It's why he and his family are moving out of the flat they've

Here in Istanbul, fear is growing.
```



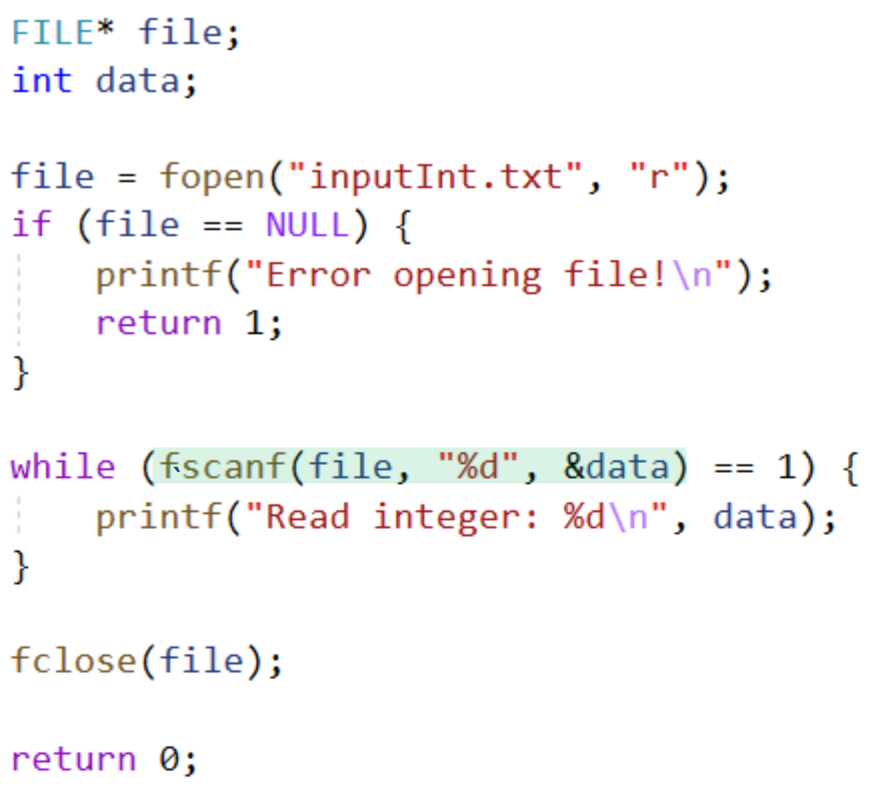
```
Read word: The
Read word: crack
Read word: in
Read word: Mesut
Read word: Muttaliboglu's
Read word: bedroom
Read word: wall
Read word: is
Read word: so
Read word: wide,
Read word: he
Read word: can
Read word: fit
Read word: a
Read word: car
Read word: key
Read word: into
Read word: it.
Read word: He
Read word: turns
Read word: it
Read word: sideways,
Read word: and
Read word: with
Read word: a
Read word: flick
Read word: of
Read word: his
Read word: wrist,
Read word: a
```


inputInt.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 7, Col 2
100%
Windows (CRLF)
UTF-8

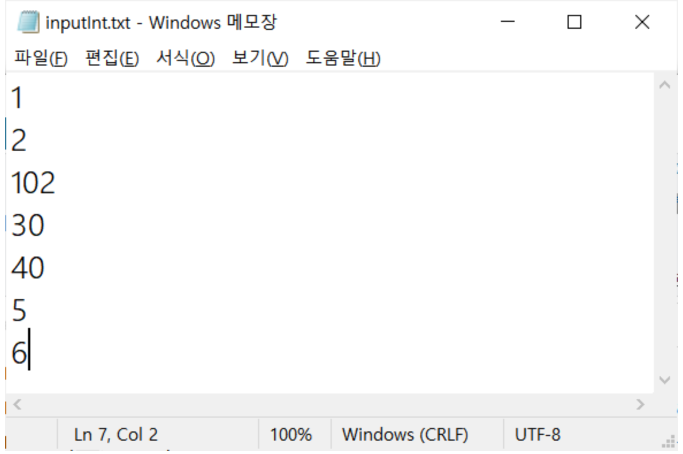
```
1
2
102
30
40
5
6
```



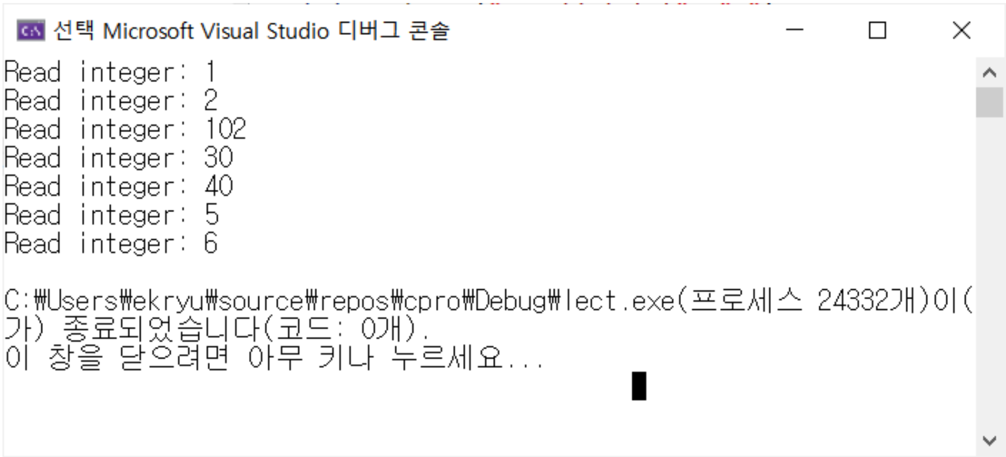
```
Read integer: 1
Read integer: 2
Read integer: 102
Read integer: 30
Read integer: 40
Read integer: 5
Read integer: 6

C:₩Users₩ekryu₩source₩repos₩cpro₩Debug₩lect.exe(프로세스 24332개)이(
가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```



```c
FILE* file;
int data;

file = fopen("inputInt.txt", "r");
if (file == NULL) {
    printf("Error opening file!\n");
    return 1;
}

while (fscanf(file, "%d", &data) == 1) {
    printf("Read integer: %d\n", data);
}

fclose(file);

return 0;
```

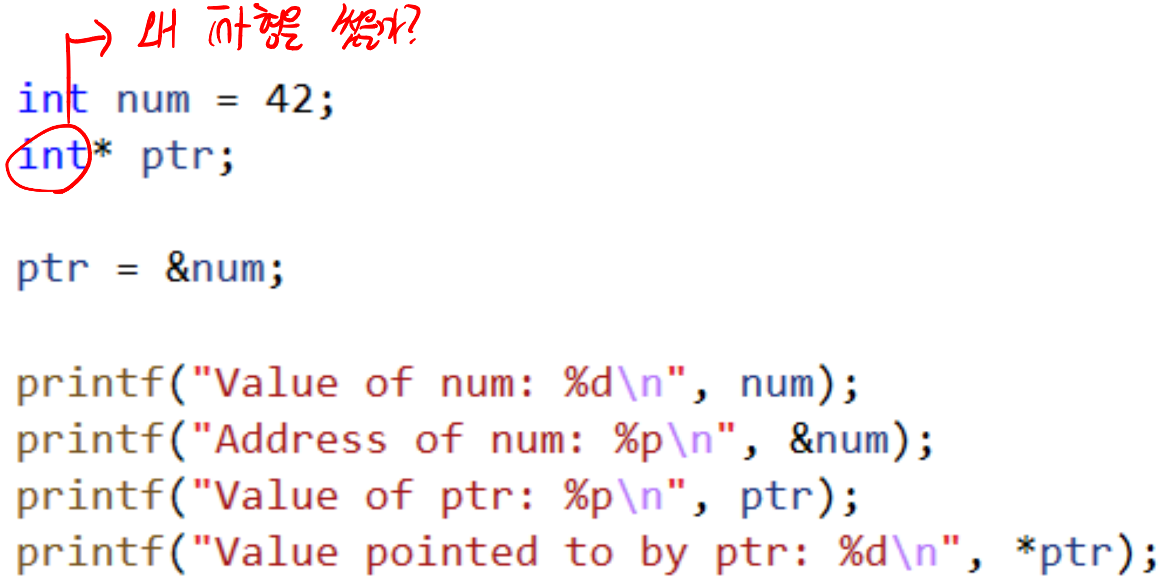

inputInt.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Ln 7, Col 2
100%
Windows (CRLF)
UTF-8

```
1
2
102
30
40
5
6
```



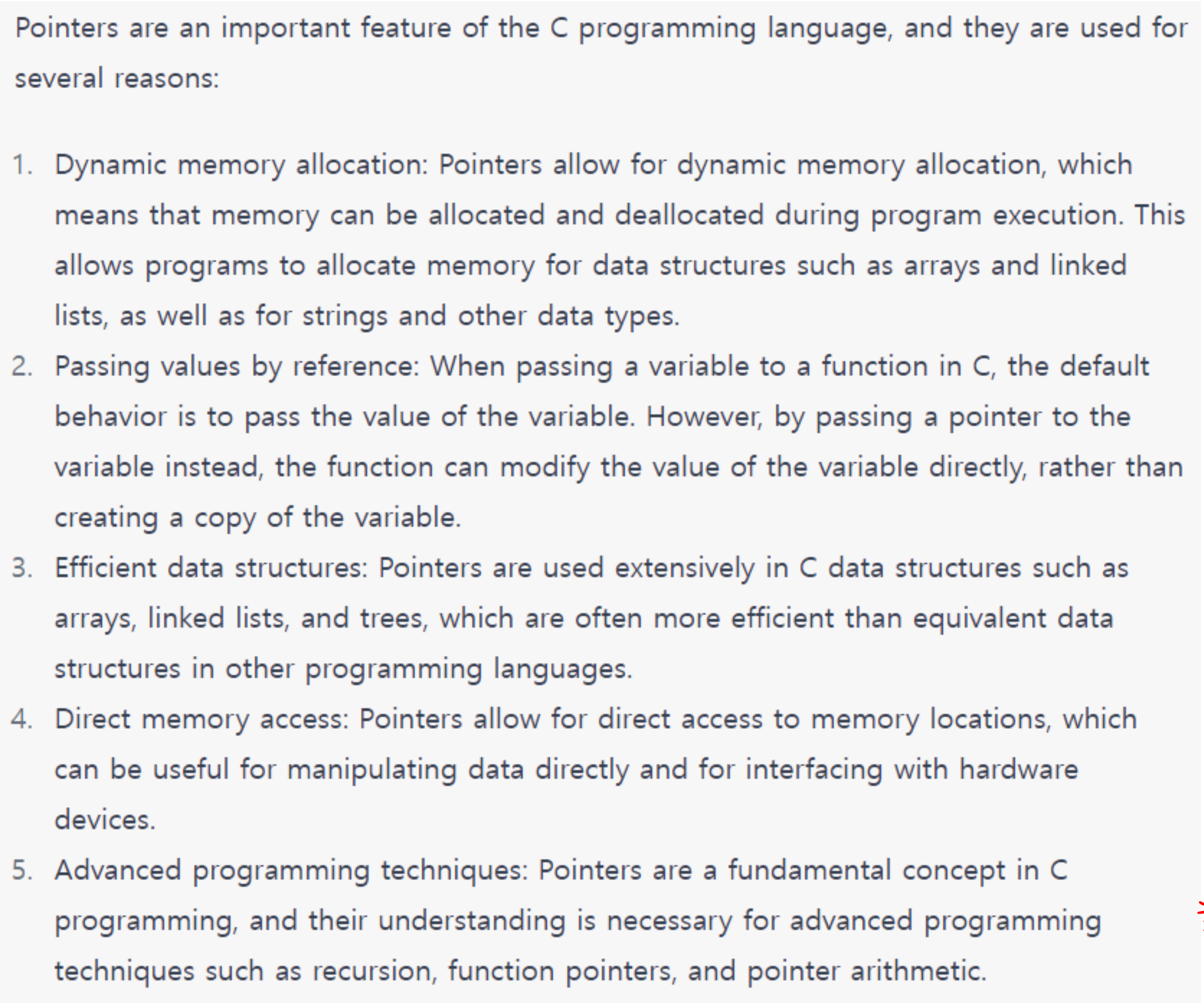
```
Read integer: 1
Read integer: 2
Read integer: 102
Read integer: 30
Read integer: 40
Read integer: 5
Read integer: 6

C:₩Users₩ekryu₩source₩repos₩cpro₩Debug₩lect.exe(프로세스 24332개)이(
가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

Pointer



→ 내 짝꿍 뭘까?

```
int num = 42;
int* ptr;

ptr = &num;

printf("Value of num: %d\n", num);
printf("Address of num: %p\n", &num);
printf("Value of ptr: %p\n", ptr);
printf("Value pointed to by ptr: %d\n", *ptr);
```



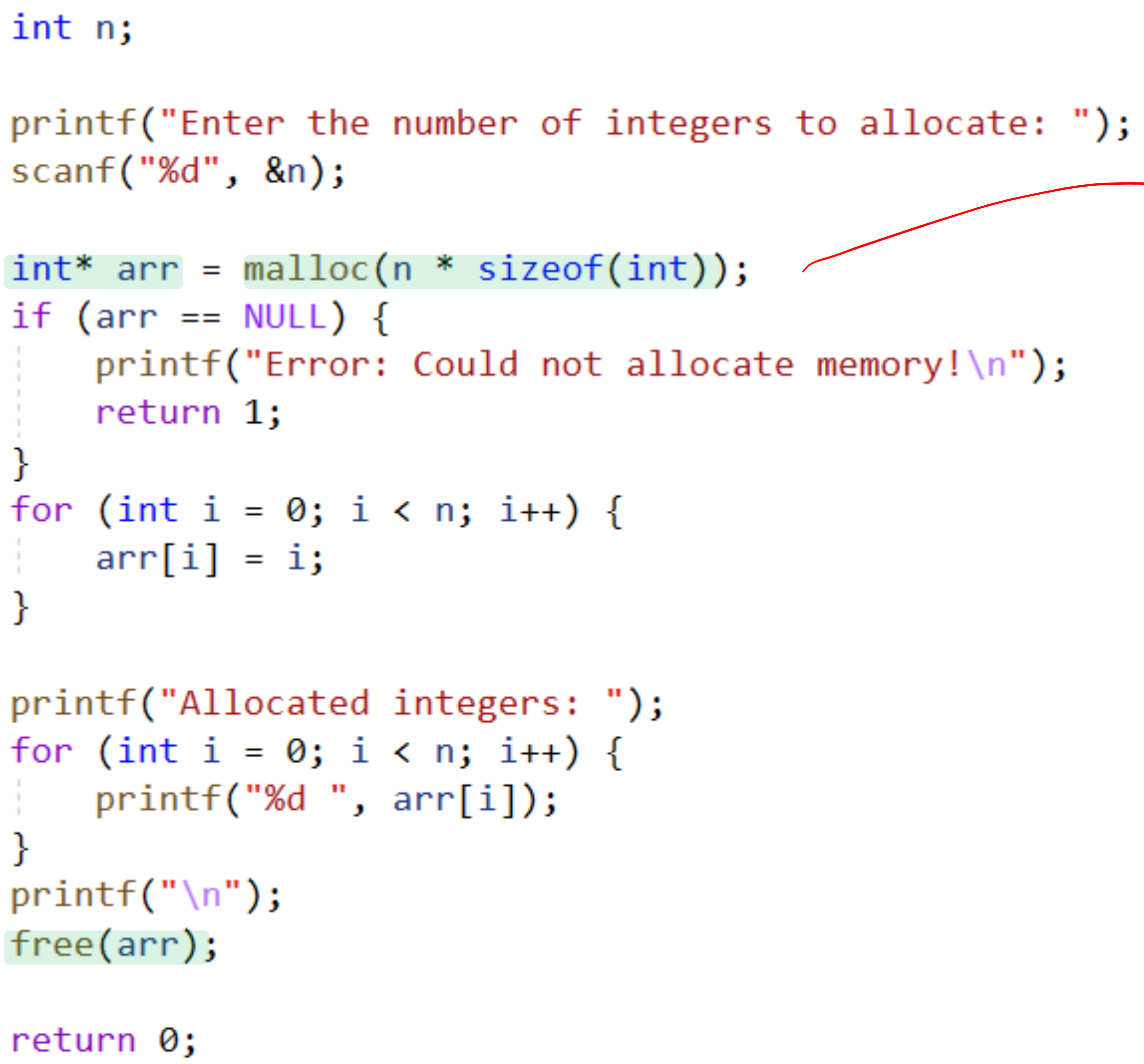
Pointers are an important feature of the C programming language, and they are used for several reasons:

1. Dynamic memory allocation: Pointers allow for dynamic memory allocation, which means that memory can be allocated and deallocated during program execution. This allows programs to allocate memory for data structures such as arrays and linked lists, as well as for strings and other data types.
2. Passing values by reference: When passing a variable to a function in C, the default behavior is to pass the value of the variable. However, by passing a pointer to the variable instead, the function can modify the value of the variable directly, rather than creating a copy of the variable.

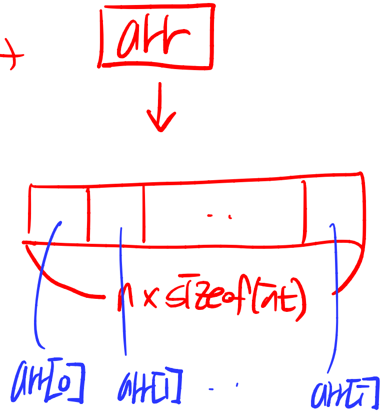
함수에 포인터를 전달

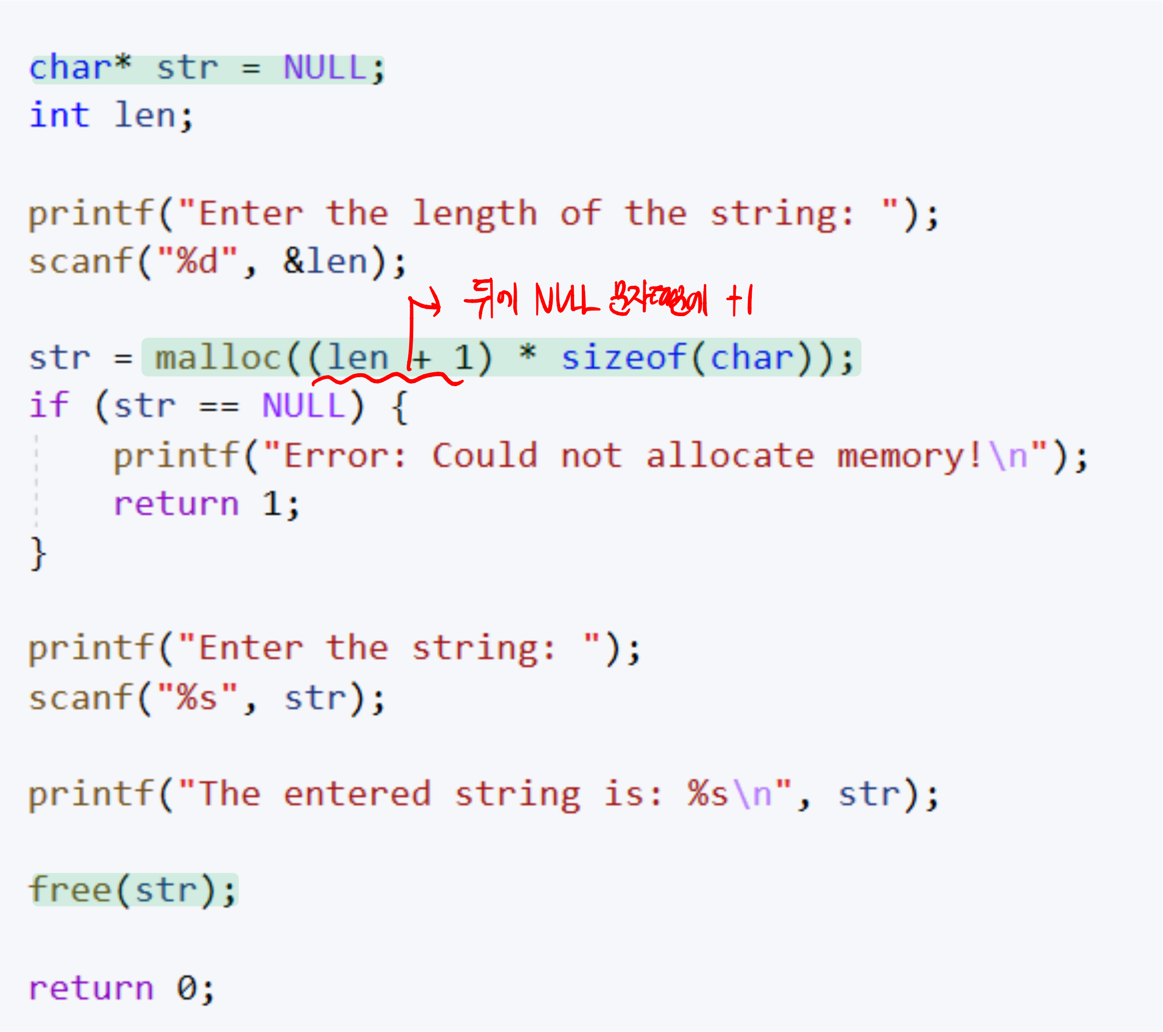
3. Efficient data structures: Pointers are used extensively in C data structures such as arrays, linked lists, and trees, which are often more efficient than equivalent data structures in other programming languages.
4. Direct memory access: Pointers allow for direct access to memory locations, which can be useful for manipulating data directly and for interfacing with hardware devices.
5. Advanced programming techniques: Pointers are a fundamental concept in C programming, and their understanding is necessary for advanced programming techniques such as recursion, function pointers, and pointer arithmetic.

함수 포인터



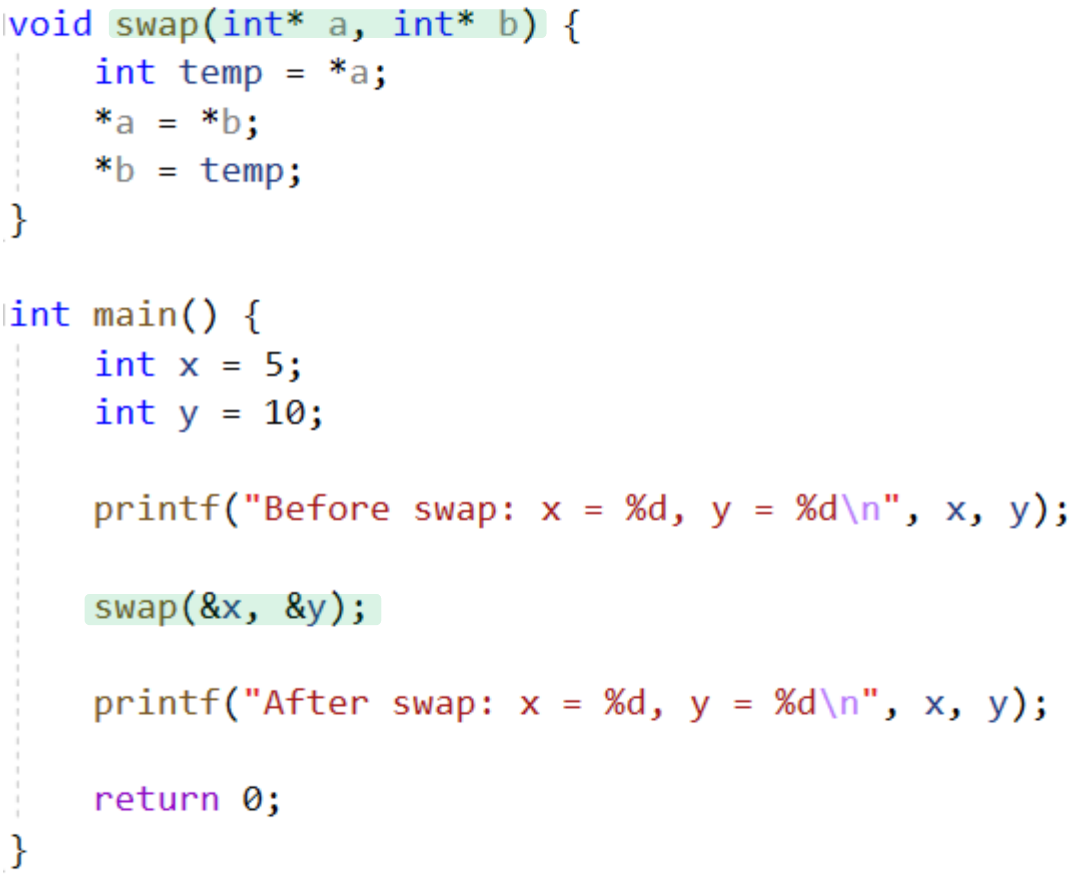
```
int n;

printf("Enter the number of integers to allocate: ");
scanf("%d", &n);

int* arr = malloc(n * sizeof(int));
if (arr == NULL) {
    printf("Error: Could not allocate memory!\n");
    return 1;
}
for (int i = 0; i < n; i++) {
    arr[i] = i;
}

printf("Allocated integers: ");
for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");
free(arr);

return 0;
```



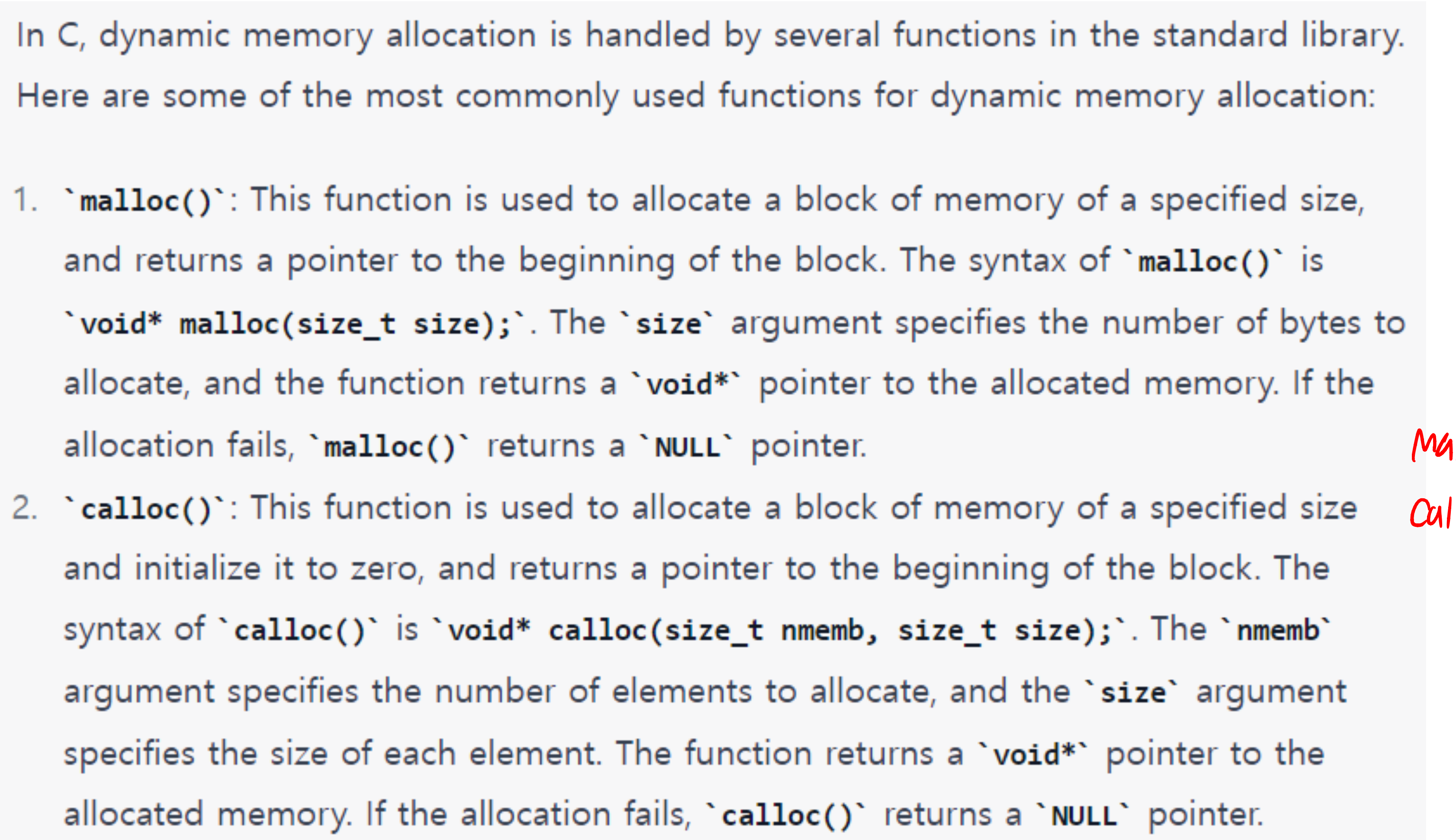


```c
char* str = NULL;
int len;

printf("Enter the length of the string: ");
scanf("%d", &len);

str = malloc((len + 1) * sizeof(char));
if (str == NULL) {
    printf("Error: Could not allocate memory!\n");
    return 1;
}

printf("Enter the string: ");
scanf("%s", str);

printf("The entered string is: %s\n", str);

free(str);

return 0;
```

뒤에 NULL 문자때문에 +1



```
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5;
    int y = 10;

    printf("Before swap: x = %d, y = %d\n", x, y);

    swap(&x, &y);

    printf("After swap: x = %d, y = %d\n", x, y);

    return 0;
}
```

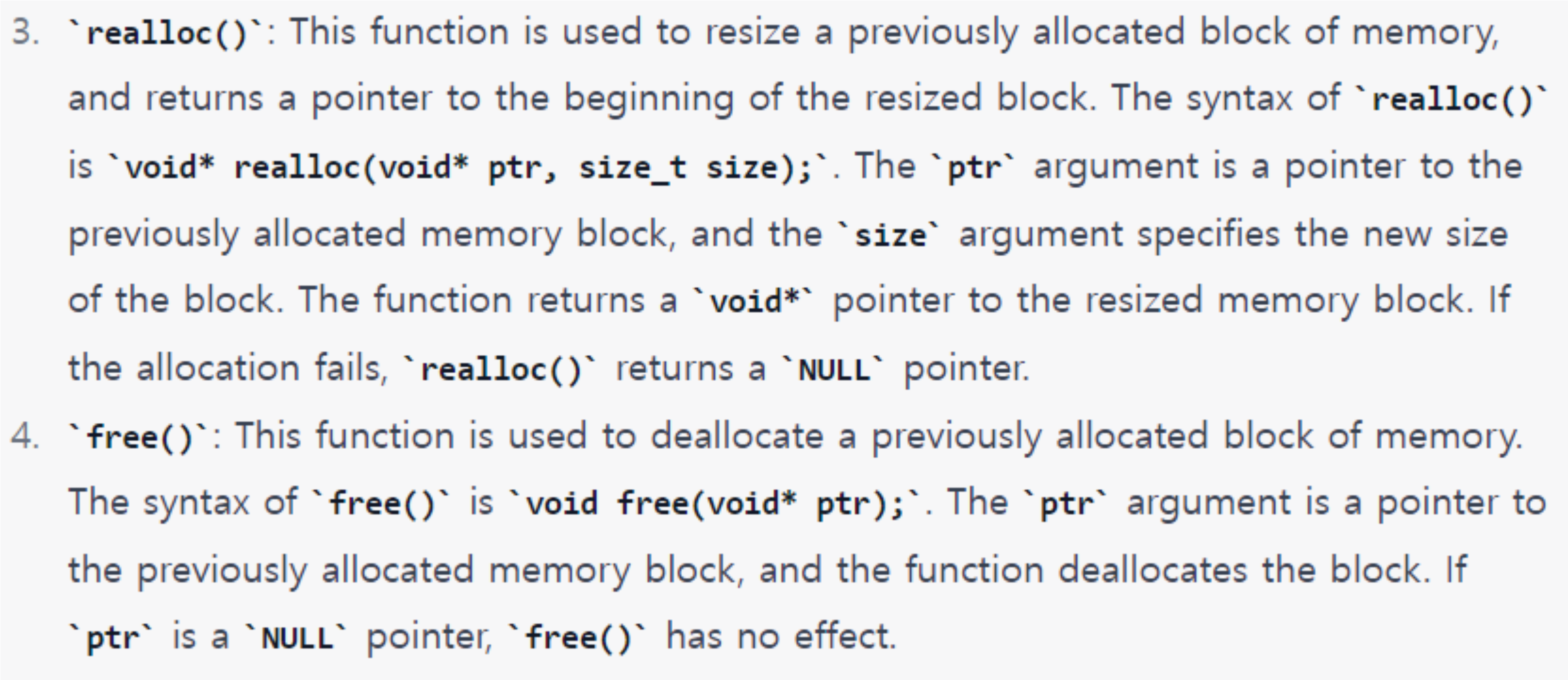


In C, dynamic memory allocation is handled by several functions in the standard library. Here are some of the most commonly used functions for dynamic memory allocation:

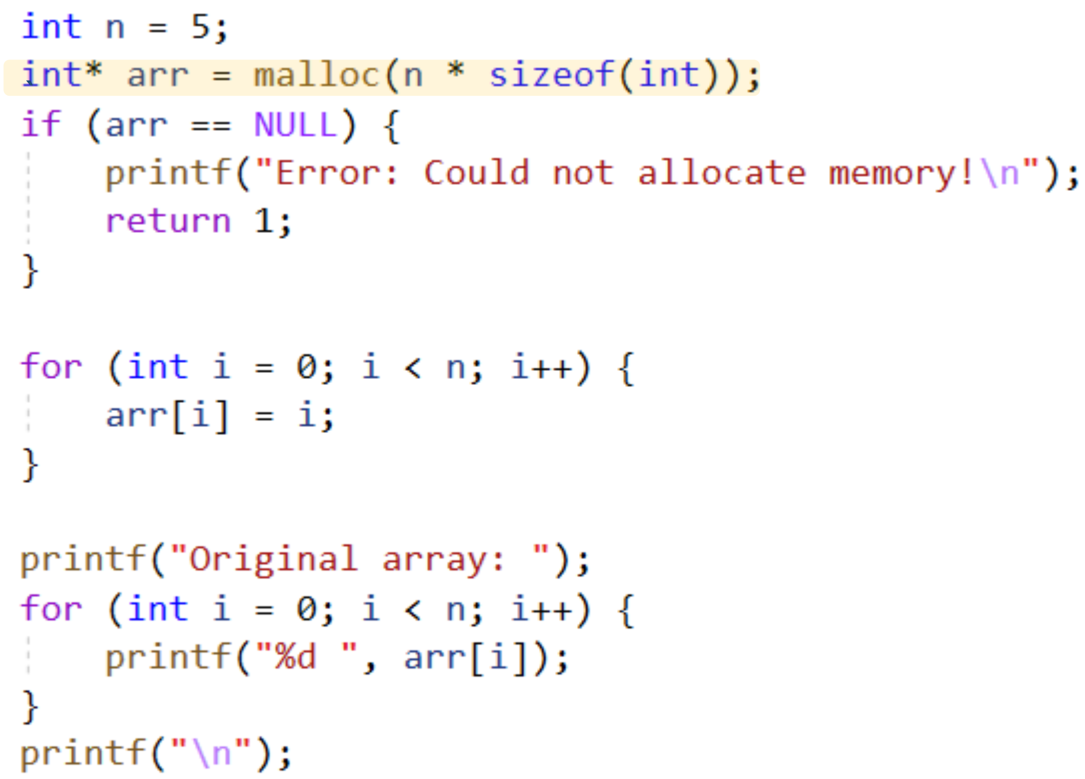
1. `malloc()`: This function is used to allocate a block of memory of a specified size, and returns a pointer to the beginning of the block. The syntax of `malloc()` is `void* malloc(size_t size);`. The `size` argument specifies the number of bytes to allocate, and the function returns a `void*` pointer to the allocated memory. If the allocation fails, `malloc()` returns a `NULL` pointer.
2. `calloc()`: This function is used to allocate a block of memory of a specified size and initialize it to zero, and returns a pointer to the beginning of the block. The syntax of `calloc()` is `void* calloc(size_t nmemb, size_t size);`. The `nmemb` argument specifies the number of elements to allocate, and the `size` argument specifies the size of each element. The function returns a `void*` pointer to the allocated memory. If the allocation fails, `calloc()` returns a `NULL` pointer.

malloc (n x sizeof(int))

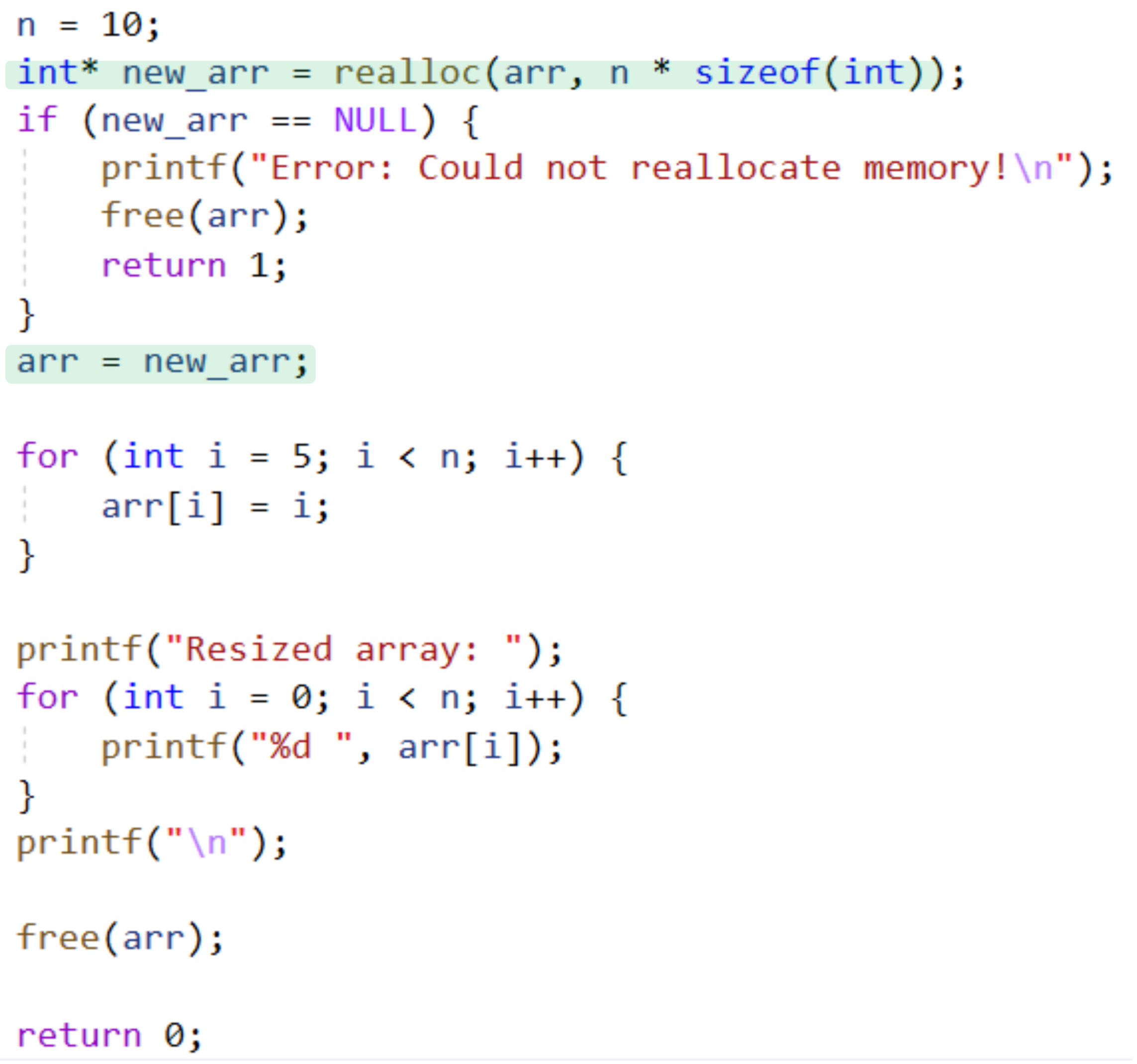
calloc (n, sizeof(int))



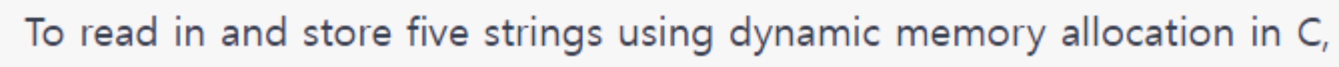
3. `realloc()`: This function is used to resize a previously allocated block of memory, and returns a pointer to the beginning of the resized block. The syntax of `realloc()` is `void* realloc(void* ptr, size_t size);`. The `ptr` argument is a pointer to the previously allocated memory block, and the `size` argument specifies the new size of the block. The function returns a `void*` pointer to the resized memory block. If the allocation fails, `realloc()` returns a `NULL` pointer.
4. `free()`: This function is used to deallocate a previously allocated block of memory. The syntax of `free()` is `void free(void* ptr);`. The `ptr` argument is a pointer to the previously allocated memory block, and the function deallocates the block. If `ptr` is a `NULL` pointer, `free()` has no effect.



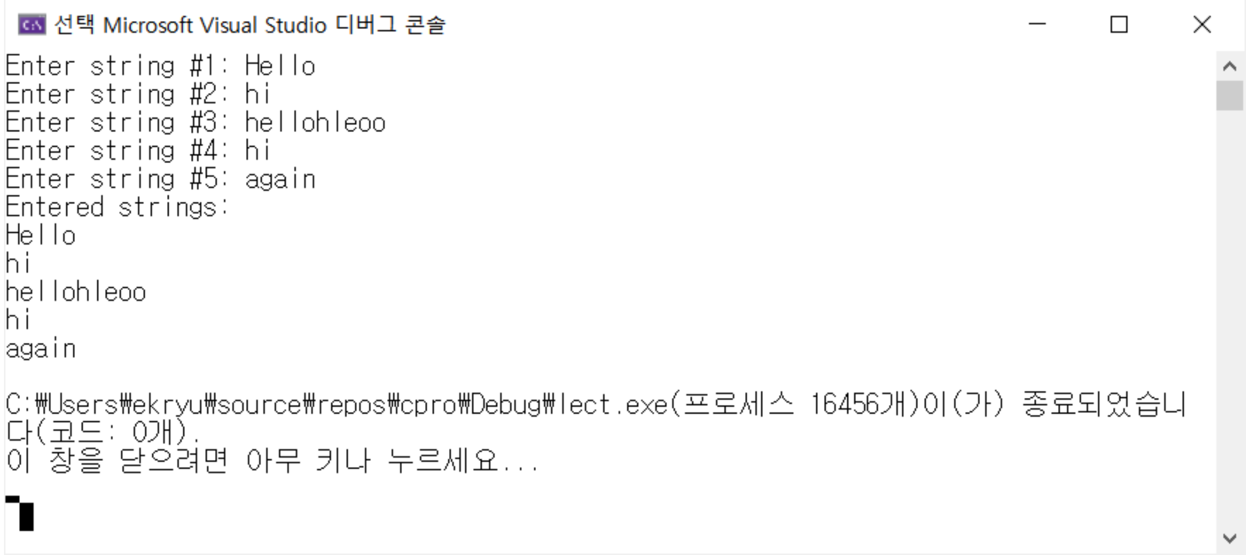
```c
int n = 5;
int* arr = malloc(n * sizeof(int));
if (arr == NULL) {
    printf("Error: Could not allocate memory!\n");
    return 1;
}

for (int i = 0; i < n; i++) {
    arr[i] = i;
}

printf("Original array: ");
for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");
```



```c
n = 10;
int* new_arr = realloc(arr, n * sizeof(int));
if (new_arr == NULL) {
    printf("Error: Could not reallocate memory!\n");
    free(arr);
    return 1;
}
arr = new_arr;

for (int i = 5; i < n; i++) {
    arr[i] = i;
}

printf("Resized array: ");
for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");

free(arr);

return 0;
```



To read in and store five strings using dynamic memory allocation in C,

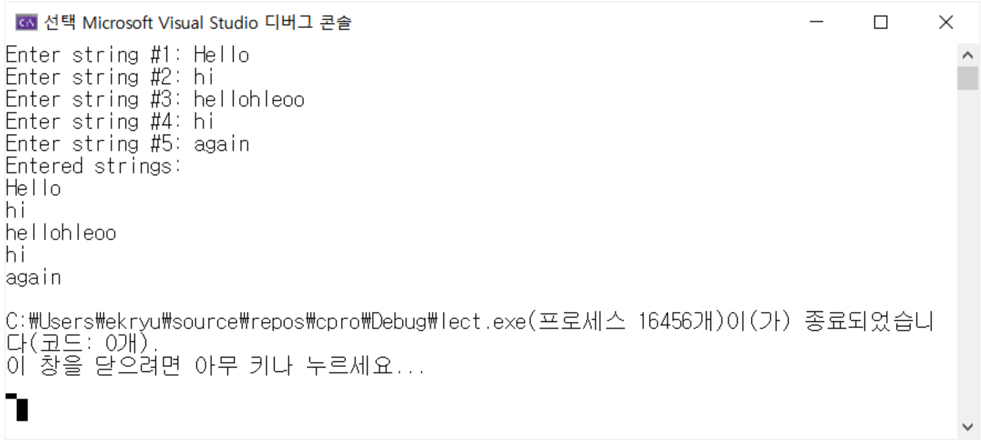


```
Enter string #1: Hello
Enter string #2: hi
Enter string #3: hellohleoo
Enter string #4: hi
Enter string #5: again
Entered strings:
Hello
hi
hellohleoo
hi
again

C:₩Users₩ekryu₩source₩repos₩cpro₩Debug₩lect.exe(프로세스 16456개)이(가) 종료되었습니
다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

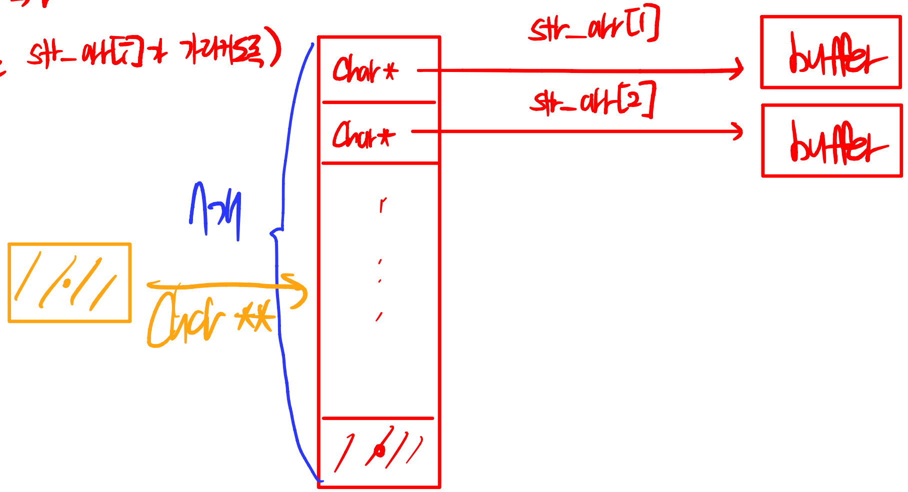
```c
int n = 5;
char** str_arr = malloc(n * sizeof(char*));
if (str_arr == NULL) {
    printf("Error: Could not allocate memory!\n");
    return 1;
}

for (int i = 0; i < n; i++) {
    char buffer[100];
    printf("Enter string #%d: ", i + 1);
    scanf_s("%s", buffer, 100);
    str_arr[i] = (char *) malloc((strlen(buffer) + 1) * sizeof(char));
    if (str_arr[i] == NULL) {
        printf("Error: Could not allocate memory!\n");
        return 1;
    }
    strcpy(str_arr[i], buffer);
}
printf("Entered strings:\n");
for (int i = 0; i < n; i++) {
    printf("%s\n", str_arr[i]);
}

for (int i = 0; i < n; i++) {
    free(str_arr[i]);
}
free(str_arr);

return 0;
```

buffer의 문자열을 str_arr[i]로 복사

(= buffer의 시작주소를 str_arr[i]가 가리키도록)



```
Enter string #1: Hello
Enter string #2: hi
Enter string #3: hellohleoo
Enter string #4: hi
Enter string #5: again
Entered strings:
Hello
hi
hellohleoo
hi
again

C:₩Users₩ekryu₩source₩repos₩cpro₩Debug₩lect.exe(프로세스 16456개)이(가) 종료되었습니
다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```



- File IO
- Pointer : Dynamic Memory Allocation