

# # 튜터링 15주차

( TUTOR: 성열암 )

## 응용컴퓨터 프로그래밍

TUTORING ————— <https://github.com/developersung13/cbnu-tutoring>

2 ~ 10 주 차 학 습 내 용 복 습

# CONTENTS

## INDEX

---

**01** c언어 개념 복습

**02** 질의응답

# C언어 개념 복습

---



2~10주 차 튜터링 시간에 학습했던 C언어의 핵심 개념들을  
기말 시험에 앞서 다시 한번 복습해 보는 시간입니다.

01

## 02 포인터 (1/6)

### □ 포인터

```
#include <stdio.h>

int main() {
    int num = 10;
    int *pNum = &num;
    *pNum = *pNum + 7;
    printf("%d", num);
    return 0;
}
```

포인터 변수는 다른 변수의  
주솟값을 참조합니다.

### □ 2중 포인터

```
#include <stdio.h>

int main() {
    int num = 10;
    int *pNum = &num;
    int **dpNum = &pNum;
    *pNum = *pNum + 7;
    printf("%d\n", num);
    **dpNum = **dpNum - 5;
    printf("%d", num);
    return 0;
}
```

포인터 변수의 주솟값을  
참조하는 포인터 변수입니다.

### □ 3중 포인터

```
#include <stdio.h>

int main() {
    int num = 10;
    int *pNum = &num;
    int **dpNum = &pNum;
    int ***tpNum = &dpNum;

    *pNum = *pNum + 1;
    printf("%d\n", num);

    **dpNum = **dpNum + 1;
    printf("%d\n", num);

    ***tpNum = ***tpNum + 1;
    printf("%d\n", num);
    return 0;
}
```

포인터 변수의 주솟값을  
참조하는 포인터 변수의  
주솟값을 참조하는 포인터  
변수입니다.

## 02 포인터 (4/6)

### □ Call-by-value(값에 의한 참조)

```
#include <stdio.h>

int swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}

int main() {
    int a = 100, b = 200;
    printf("Before: %d, %d\n", a, b);
    swap(a, b);
    printf("After: %d, %d\n", a, b);
    return 0;
}
```

```
Before: 100, 200
After: 100, 200
```

인수값의 복사로  
데이터를 처리하는  
방법

## 02 포인터 (5/6)

### □ Call-by-reference(주소에 의한 참조)

```
#include <stdio.h>

int swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 100, b = 200;
    printf("Before: %d, %d\n", a, b);
    swap(&a, &b);
    printf("After: %d, %d\n", a, b);
    return 0;
}
```

Before: 100, 200  
After: 200, 100

인수값이 저장되어  
있는 주소값 자체를  
참조하여 데이터를  
처리하는 방법



## 02 포인터 (6/6)

### □ 배열과 포인터의 관계

```
#include <stdio.h>

int main() {
    int a[5] = { 10, 3, 1, 2, 4 };

    printf("a의 address: &a[0] = %p, a = %p \n", &a[0], a);
    printf("a의 value: %d\n", *a);
    for (int k = 0; k < 5; k++)
        printf("\t address: %p, a[%d]: %d, *(a+%d): %d\n", (a+k), k, a[k], k, *(a+k));
}
```



```
a의 address: &a[0] = 0x7fffd7b4c3d0, a = 0x7fffd7b4c3d0
a의 value: 10
\t address: 0x7fffd7b4c3d0, a[0]: 10, *(a+0): 10
\t address: 0x7fffd7b4c3d4, a[1]: 3, *(a+1): 3
\t address: 0x7fffd7b4c3d8, a[2]: 1, *(a+2): 1
\t address: 0x7fffd7b4c3dc, a[3]: 2, *(a+3): 2
\t address: 0x7fffd7b4c3e0, a[4]: 4, *(a+4): 4
```

## 01 문자와 문자열 (1/3)

### □ 문자

```
#include <stdio.h>

int main() {
    char c = 'A';
    printf("%c", c);
}
```

Compiled Successfully.

A

문자형은 1byte 크기의  
데이터를 저장할 때 사용하는  
자료형입니다.

## 01 문자와 문자열 (2/3)

### □ 문자열

```
#include <stdio.h>

int main() {
    char c[] = "ABC DEF";
    printf("%s", c);
}
```

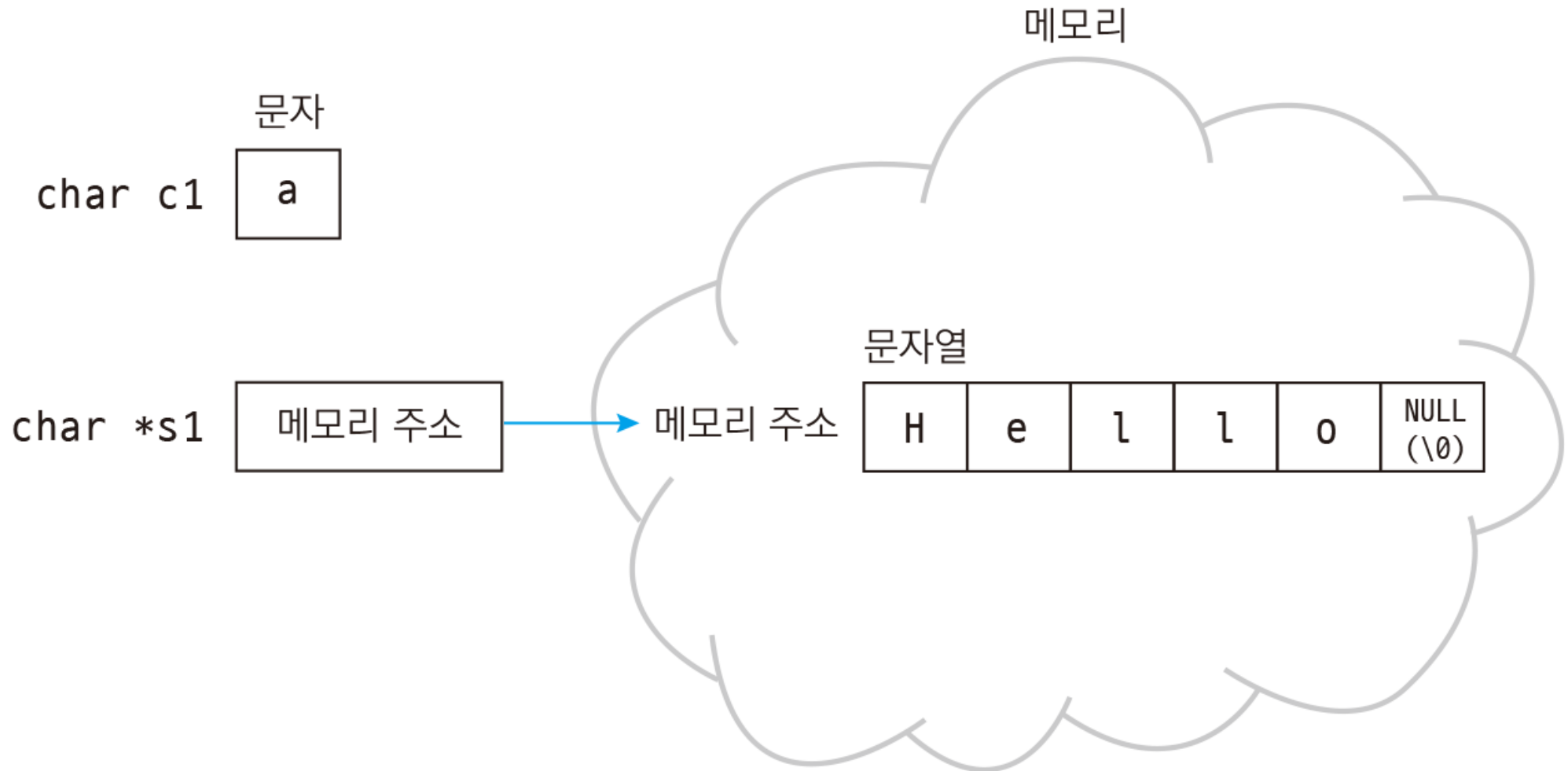
Compiled Successfully.

ABC DEF

1개 이상의 문자들을 하나의 변수에 저장하고자 할 때 사용하는 문자의 배열입니다.

# 01 문자와 문자열 (3/3)

## □ 구조



### □ 구조체

```
키워드  구조체 이름
↓      ↓
struct book
{
    char title[30];
    char author[30];
    int price;
};
구조체의 멤버 변수
↑
세미 콜론
```

The diagram illustrates the syntax of a C struct definition. It shows the code: `struct book { char title[30]; char author[30]; int price; };`. Annotations include: '키워드' (keyword) pointing to 'struct', '구조체 이름' (struct name) pointing to 'book', '구조체의 멤버 변수' (struct member variables) pointing to the list of variables inside the braces, and '세미 콜론' (semicolon) pointing to the closing brace and semicolon.

배열이 같은 타입의 변수 집합이라고 한다면, 구조체는 다양한 타입의 변수 집합을 하나의 타입으로 나타낸 것입니다.

## 01 구조체 [2/3]

### □ 구조체 변수 선언

```
#include <stdio.h>

struct book {
    char title[30];
    char author[30];
    int price;
};

int main() {
    struct book myBook = { "제목", "작가", 5000 };
    printf("%s\n%s\n%d", myBook.title, myBook.author, myBook.price);
}
```

# 01 구조체 (1/3)

## □ typedef

```
1  #include <stdio.h>
2
3  typedef struct {
4      int age;
5      char phone_number[14];
6  } Student;
7
8  int main(){
9      Student goorm;
10
11     printf("나이 : ");
12     scanf("%d", &goorm.age);
13     printf("번호 : ");
14     scanf("%s", goorm.phone_number);
15
16     printf("----\n나이 : %d\n번호 : %s\n----", goorm.age, goorm.phone_number);
17
18     return 0;
19 }
20
```

### □ 장점

데이터를 저장할 공간이 필요할 때마다 동적으로  
공간을 만들어서 쉽게 추가할 수 있다는 것.  
이것은 순차적인 표현 방법은 배열에 비하여  
상당한 장점.

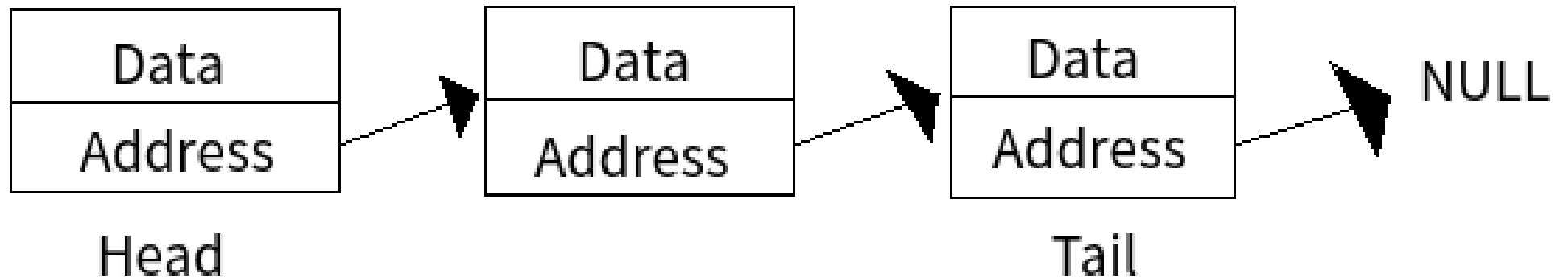


### □ 단점

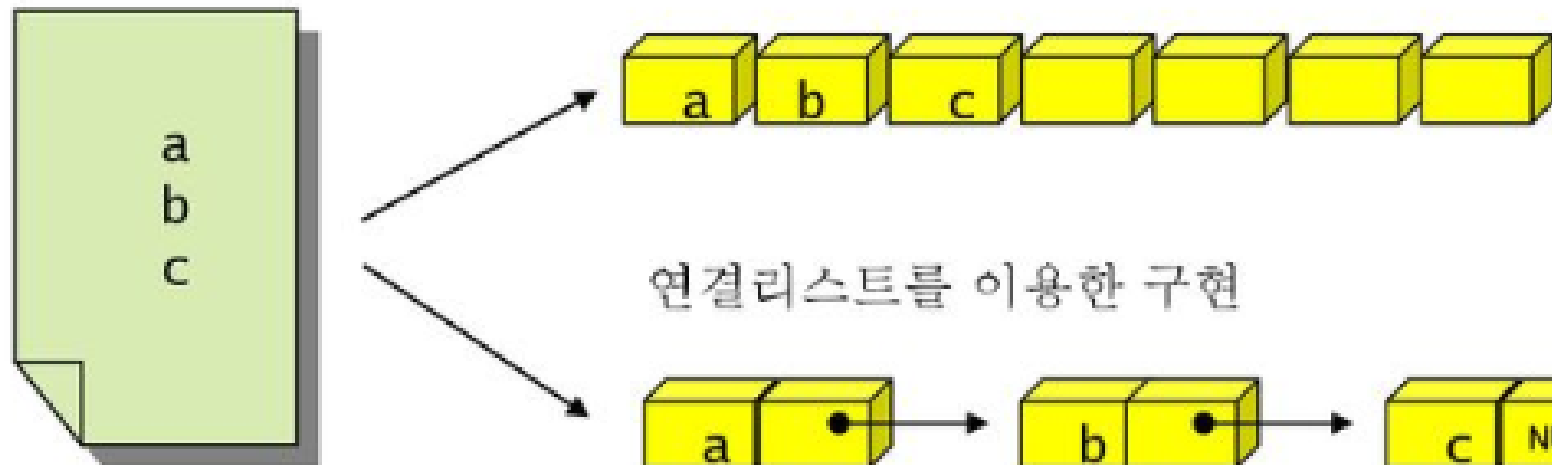
배열에 비하여 상대적으로 구현이 어렵고 오류가 발생하기쉬움 또한 데이터 뿐만 아니라 포인터도 저장해야 하므로 메모리 공간을 많이 사용. 또  $i$  번째 데이터를 찾으려면 앞에서부터 순차적으로 접근해야함.

## 02 연결 리스트 (3/5)

### □ 구조



배열을 이용한 구현



연결리스트를 이용한 구현

## 02 연결 리스트 (4/5)

### □ 종류

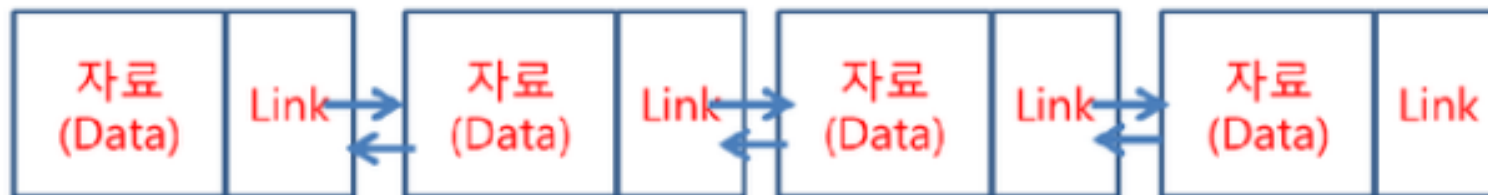
단순 연결 리스트



원형 연결 리스트



이중 연결 리스트



### □ 적용 코드

[shorturl.at/bGJLS](https://shorturl.at/bGJLS)

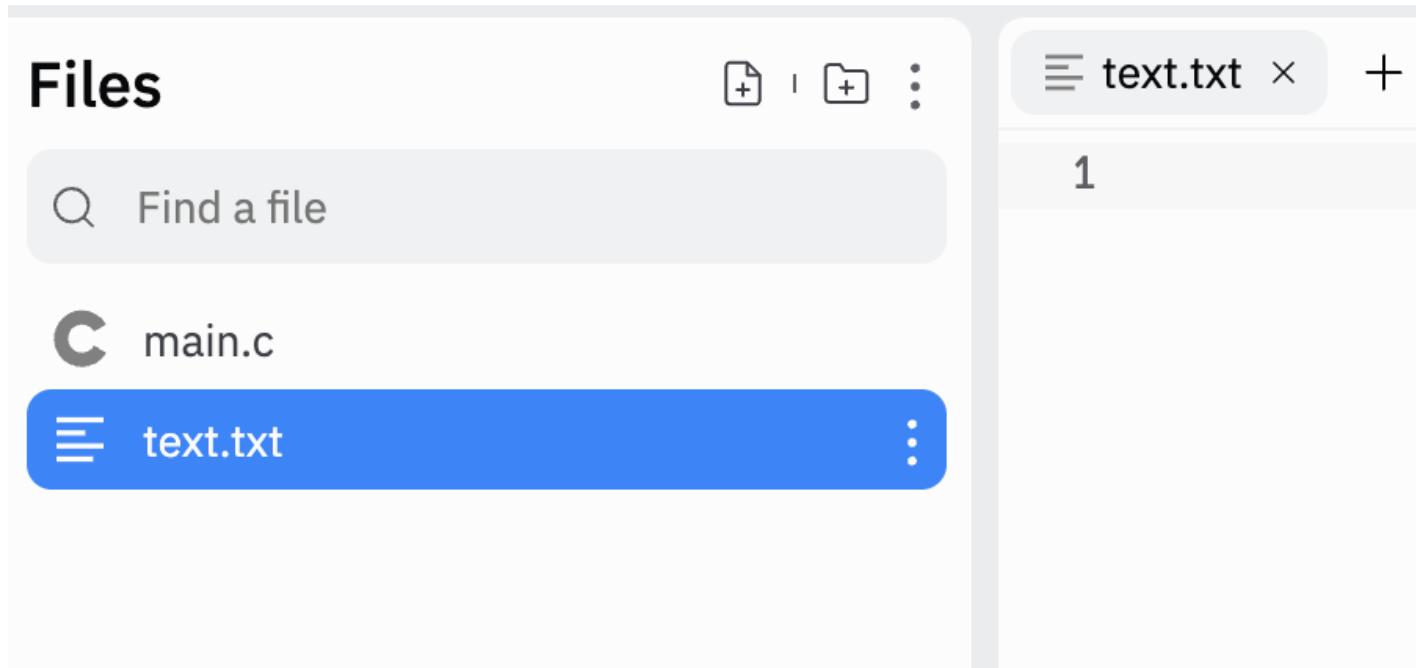
```
INSERT [10]
INSERT [30]
INSERT [20]
INSERT [50]
HEAD > 10 30 20 50 END.
DELETE [30]
DELETE [10]
HEAD > 20 50 END.
DELETE [15]
Can't find the key!
```

### □ 파일 생성

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      FILE* fp;
5      fp = fopen("text.txt", "w");
6      fclose(fp);
7  }
```

fopen함수는 주어진 파일명으로 파일을 생성하여 FILE 포인터를 반환한다.

### □ 파일 생성



"text.txt"라는 이름으로 C코드가  
위치한 경로에 파일이 생성됨.

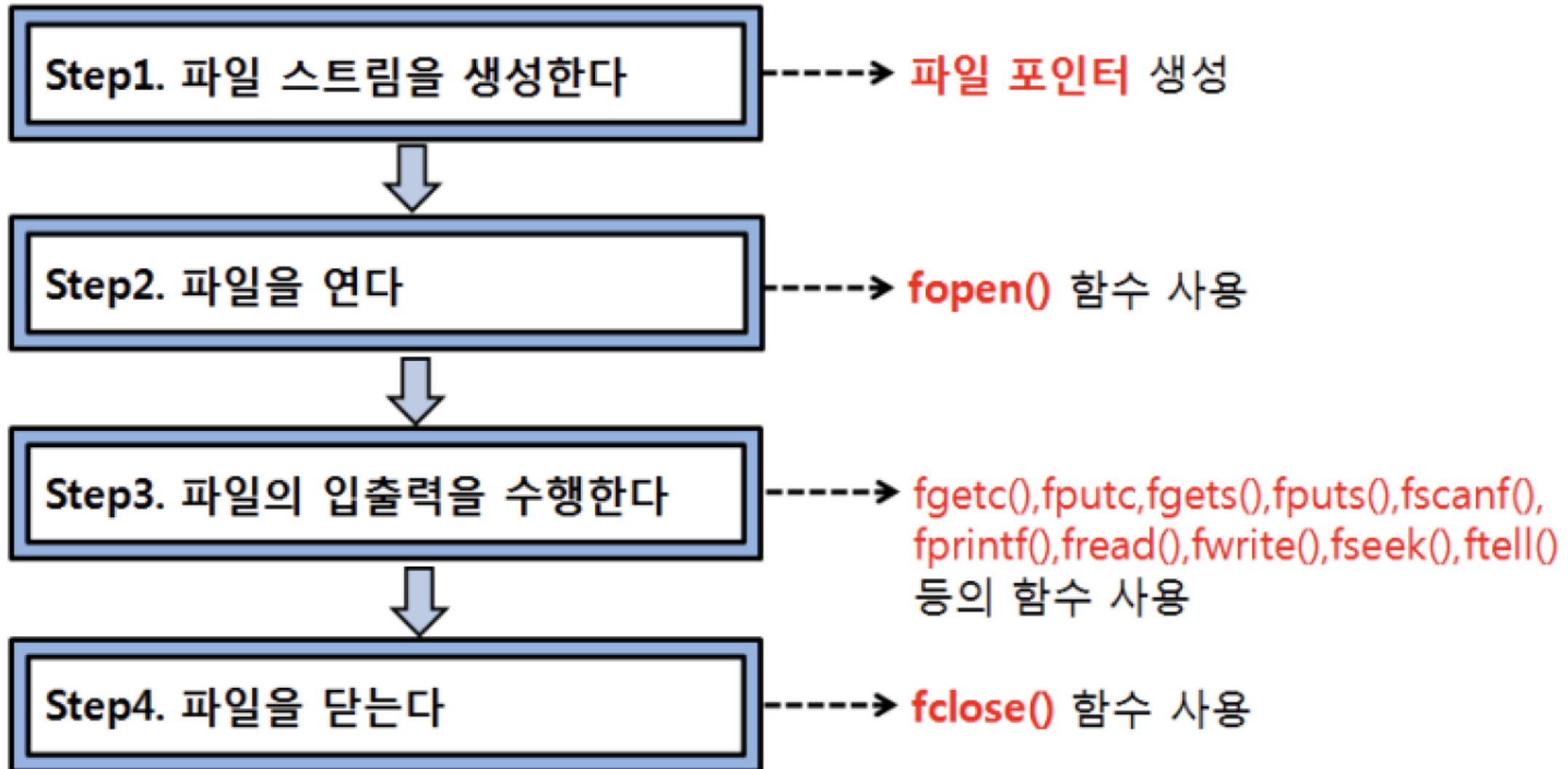
## □ 파일 모드

모드	설명
"r"	읽기 모드로 파일을 연다. 만약 파일이 존재하지 않으면 오류가 발생한다.
"w"	쓰기 모드로 새로운 파일을 생성한다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a"	추가 모드로 파일을 연다. 만약 기존의 파일의 있으면 데이터가 파일의 끝에 추가된다. 파일이 없으면 새로운 파일을 만든다.
"r+"	읽기 모드로 파일을 연다. 쓰기 모드로 전환할 수 있다. 파일이 반드시 존재하여야 한다.
"w+"	쓰기 모드로 새로운 파일을 생성한다. 읽기 모드로 전환할 수 있다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a+"	추가 모드로 파일을 연다. 읽기 모드로 전환할 수 있다. 데이터를 추가하면 EOF 마커를 추가된 데이터의 뒤로 이동한다. 파일이 없으면 새로운 파일을 만든다.
"t"	텍스트 파일 모드로 파일을 연다.
"b"	이진 파일 모드로 파일을 연다.

**파일의 조작 방식에 따라 모드를 선택한다.**

(기본적인 파일 모드에 "t"나 "b"를 붙일 수 있다)

### □ 파일 입출력 과정





## 01 파일입출력 (5/12)

### □ 텍스트 파일 쓰기(문자)

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      FILE *fp = NULL;
5      fp = fopen("text.txt", "w");
6      if (fp) // if (fp) is not NULL?
7          printf("파일 열기 성공");
8      else
9          printf("파일 열기 실패");
10
11     fputc('A', fp);
12     fclose(fp);
13 }
```

```
> make -s
> ./main
파일 열기 성공 > □
```

text.txt 파일이  
어떻게 변화  
하였나요?

## 01 파일입출력 (6/12)

### □ 텍스트 파일 쓰기(문자)

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      FILE *fp = NULL;
5      fp = fopen("text.txt", "w");
6      if (fp) // if (fp) is not NULL?
7          printf("파일 열기 성공");
8      else
9          printf("파일 열기 실패");
10
11     fputc('A', fp);
12     fclose(fp);
13 }
```

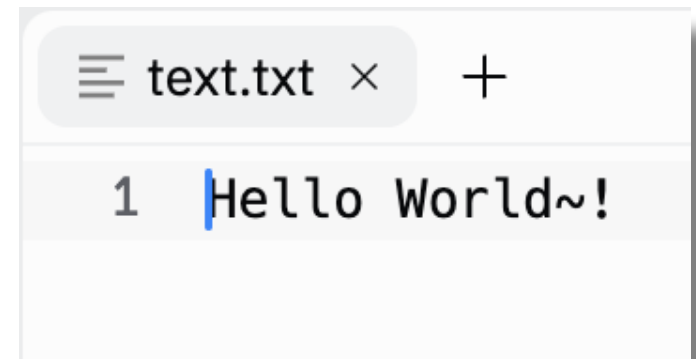
What's wrong?

## 01 파일입출력 (7/12)

### □ 텍스트 파일 쓰기(문자열)

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      FILE *fp = NULL;
5      fp = fopen("text.txt", "w");
6      if (fp) // if (fp) is not NULL?
7          printf("파일 열기 성공");
8      else
9          printf("파일 열기 실패");
10
11     fputs("Hello World~!", fp);
12     fclose(fp);
13 }
```

```
> make -s
> ./main
파일 열기 성공 □
```



The screenshot shows a text editor window with a tab labeled 'text.txt'. The editor contains a single line of text: '1 Hello World~!'. The cursor is positioned at the end of the line.

## 01 파일입출력 (8/12)

### □ 텍스트 파일 읽기(문자 - 1)

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      char c;
5      FILE *fp = NULL;
6      fp = fopen("text.txt", "r");
7      if (fp) // if (fp) is not NULL?
8          printf("파일 열기 성공\n");
9      else
10         printf("파일 열기 실패\n");
11
12     c = fgetc(fp);
13     printf("%c\n", c);
14     fclose(fp);
15 }
```

```
> make -s
> ./main
파일 열기 성공
H
> □
```

모든 문자를 받아  
오려면 어떻게  
작성하면 될까요?

## 01 파일입출력 (9/12)

### □ 텍스트 파일 읽기(문자 - 2)

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      char c;
5      FILE *fp = NULL;
6      fp = fopen("text.txt", "r");
7      if (fp) // if (fp) is not NULL?
8          printf("파일 열기 성공\n");
9      else
10         printf("파일 열기 실패\n");
11
12  ▼ while((c = fgetc(fp)) != EOF) {
13      putchar(c);
14  }
15      fclose(fp);
16  }
```

>\_ Console × +

```
> make -s
> ./main
파일 열기 성공
Hello World~! □
```

EOF는 파일의 끝을  
표현하기 위해 사용  
하는 상수 -1

## 01 파일입출력 (10/12)

### □ 텍스트 파일 읽기(문자열)

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      char str[100];
5      FILE *fp = NULL;
6      fp = fopen("text.txt", "w");
7      if (fp) // if (fp) is not NULL?
8          printf("파일 열기 성공\n");
9      else
10         printf("파일 열기 실패\n");
11
12  ▼ while(fgets(str, 100, fp)) {
13      printf("%s", str);
14  }
15  fclose(fp);
16 }
```


```
➤ make -s
➤ ./main
파일 열기 성공
Hello World~!
Hello World~!
Hello World~! ➤ □
```

fgets함수는 파일의  
끝을 만나면 NULL  
값을 반환함

파일에 숫자와 문자열을 섞어서  
입출력하려면 어떻게 해야 할까요?

# 01 파일입출력 (11/12) 형식화된 파일 내 데이터 쓰기

```
1 #include <stdio.h>
2
3 ▼ int main() {
4     char stuName[50];
5     int stuNum, engScore, mathScore;
6     FILE *fp = NULL;
7     fp = fopen("text.txt", "w");
8     if (fp) // if (fp) is not NULL?
9         printf("파일 열기 성공\n");
10 ▼ else {
11     printf("파일 열기 실패\n");
12     exit(1);
13 }
14
15 printf("학생 수를 입력해 주세요: ");
16 scanf("%d", &stuNum);
17
18 ▼ for (int k=1; k <= stuNum; k++) {
19     printf("%d번 학생의 이름을 입력하세요: ", k);
20     scanf("%s", &stuName);
21     printf("학생의 영어 점수를 입력하세요: ");
22     scanf("%d", &engScore);
23     printf("학생의 수학 점수를 입력하세요: ");
24     scanf("%d", &mathScore);
25
26     fprintf(fp, "%s %d %d\n", stuName, engScore, mathScore);
27 }
28
29 fclose(fp);
30 }
```

```
> make -s
> ./main
파일 열기 성공
학생 수를 입력해 주세요: 2
1번 학생의 이름을 입력하세요: 홍길동
학생의 영어 점수를 입력하세요: 100
학생의 수학 점수를 입력하세요: 90
2번 학생의 이름을 입력하세요: 이순신
학생의 영어 점수를 입력하세요: 90
학생의 수학 점수를 입력하세요: 80
> 
```


≡ text.txt × +

1	홍길동	100	90
2	이순신	90	80
3			



# 01 파일입출력 (12/12) 형식화된 파일 내 데이터 읽기

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      char stuName[50];
5      int stuNum, engScore, mathScore;
6      FILE *fp = NULL;
7
8  ▼  if (!(fp = fopen("text.txt", "r"))) {
9      printf("파일을 열 수 없습니다.");
10     exit(1);
11 }
12
13 ▼  while(!feof(fp)) {
14     fscanf(fp, "%s %d %d\n", stuName, &engScore,
15           &mathScore);
16     printf("<%s 학생의 성적 정보>\n", stuName);
17     printf("영어 점수: %d\n", engScore);
18     printf("수학 점수: %d\n", mathScore);
19 }
20 }
```

```
> make -s
> ./main
<홍길동 학생의 성적 정보>
영어 점수 : 100
수학 점수 : 90
<이순신 학생의 성적 정보>
영어 점수 : 90
수학 점수 : 80
> 
```

## 01 동적메모리 (1/12)

### □ 동적할당 메모리의 개념

C언어 프로그램은 **정적**(static), **동적**(dynamic)으로 메모리를 할당받을 수 있습니다.

메모리도 필요할 때마다  
요청해서 사용하면 좋은데...



# 01 동적메모리 (2/12)

## □ 정적 메모리 할당

프로그램이 시작되기 전에 미리 정해진 크기의 메모리를 할당받는다. 메모리의 크기는 프로그램이 시작되기 전에 결정된다.

```
1  #include <stdio.h>
2
3  ▼ int main() {
4      int numArr[5] = { 1, 2, 3, 4, 5 };
5      for (int k=0; k < 5; k++)
6          printf("%d ", numArr[k]);
7      return 0;
8  }
```

## 01 동적메모리 (3/12)

### □ 동적 메모리 할당

프로그램 실행 도중에 동적으로 메모리를 할당받는 방법이다. 사용이 끝나면 시스템에 메모리를 반납한다.

```
scanf( "%d", &n );  
p = ( int* ) malloc( sizeof( int ) * n );
```

# 01 동적메모리 (4/12)

## □ 동적 메모리 사용

1. 포인터를 통한 사용

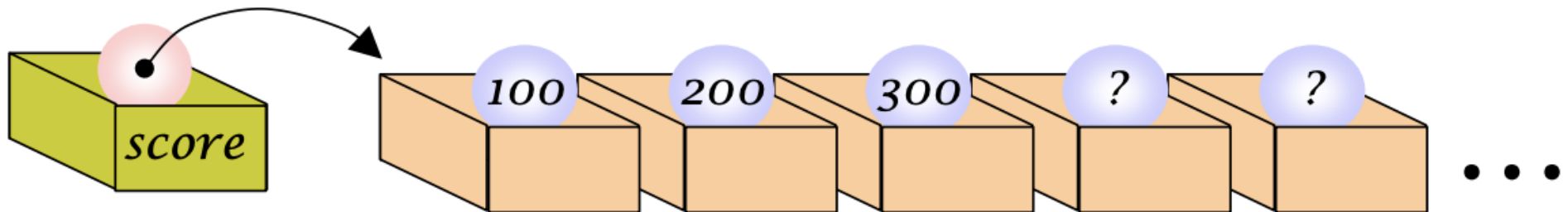
```
*score = 100;  
*(score+1) = 200;  
*(score+2) = 300;
```

...

2. 배열과 같이 취급하여 사용

```
score[0] = 100;  
score[1] = 200;  
score[2] = 300;
```

...



# 01 동적메모리 (5/12)

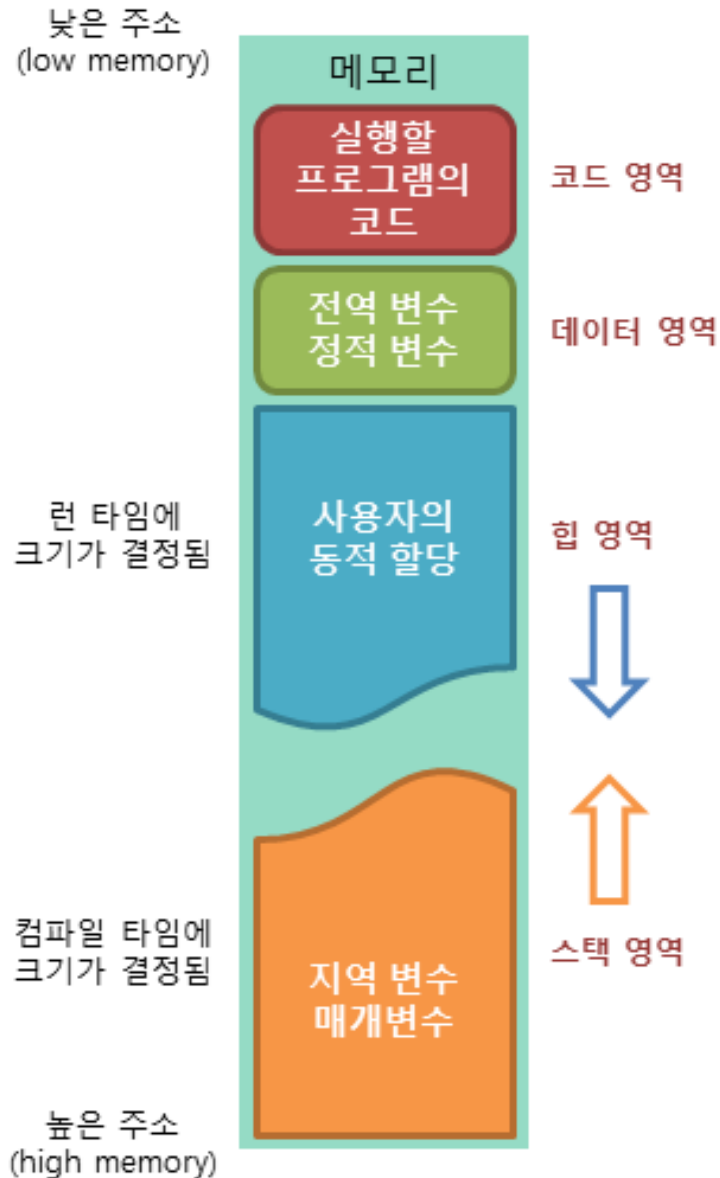
## □ 동적 메모리 반납

Syntax: 동적메모리해제

```
예  score = (int *)malloc(100*sizeof(int));  
    ...  
    free(score);
```

score가 가리키는 동적 메모리를 반납한다.

### □ 메모리 누수



동적으로 메모리를 할당하면  
힙 메모리에 공간이 생성되는데,  
이는 프로그램이 종료되기 전까지  
존재하여 메모리의 낭비를 초래해 성능  
부하를 일으킬 수 있습니다.

# 01 동적메모리 (7/12)

## □ malloc 함수 예제

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  ▼ int main() {
5      int n, k;
6      int *p;
7
8      scanf("%d", &n);
9      p = (int*)malloc(sizeof(int) * n);
10
11     printf("%d개의 정수를 입력해 주세요: ", n);
12     for (k = 0; k < n; k++)
13         scanf("%d", &p[k]);
14
15     printf("reversed: ");
16     for (k=n-1; k >=0; k--)
17         printf("%d ", p[k]);
18     return 0;
19 }
```

포인터 변수와 동적할당을  
함께 사용함으로써 런타임  
중에 동적으로 메모리 공간을  
생성할 수 있습니다.

```
➤ make -s
➤ ./main
5
5개의 정수를 입력해 주세요: 1 2 3 4 5
➤ □
```



# 01 동적메모리 (8/12)

## ❑ malloc 함수 예제

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  ▼ int main() {
5      int n, k;
6      int *p;
7
8      scanf("%d", &n);
9      p = (int*)malloc(sizeof(int) * n);
10
11     printf("%d개의 정수를 입력해 주세요: ", n);
12     for (k = 0; k < n; k++)
13         scanf("%d", &p[k]);
14
15     printf("reversed: ");
16     for (k=n-1; k >=0; k--)
17         printf("%d ", p[k]);
18     return 0;
19 }
```

What's wrong?

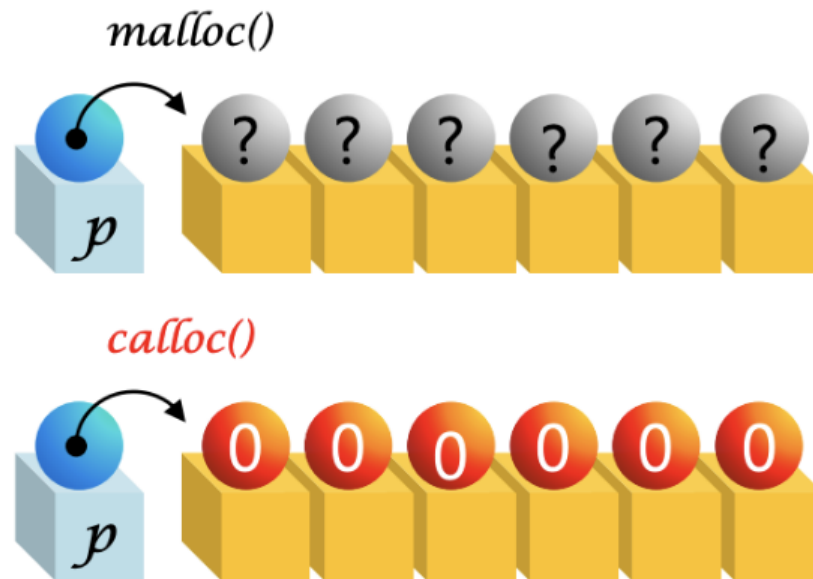
```
> make -s
> ./main
5
5개의 정수를 입력해 주세요: 1 2 3 4 5
> □
```

## 01 동적메모리 (9/12)

### □ calloc 함수

calloc 함수는 0으로 초기화된 메모리를 할당한다.

```
int *p;  
p = (int *)calloc(5, sizeof(int));
```



# 01 동적메모리 (10/12)

## □ calloc 함수 예제

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 7
5
6  int main() {
7      int n, k;
8      int *p;
9
10     scanf("%d", &n);
11     p = (int*)calloc(MAX, sizeof(int));
12
13     printf("%d개의 정수를 입력해 주세요: ", n);
14     for (k = 0; k < n; k++)
15         scanf("%d", &p[k]);
16
17     for (k=0; k < MAX; k++)
18         printf("%d ", p[k]);
19     free(p);
20     return 0;
21 }
```

메모리 할당 시 각 항목 단위로  
전부 0의 값이 할당된다.

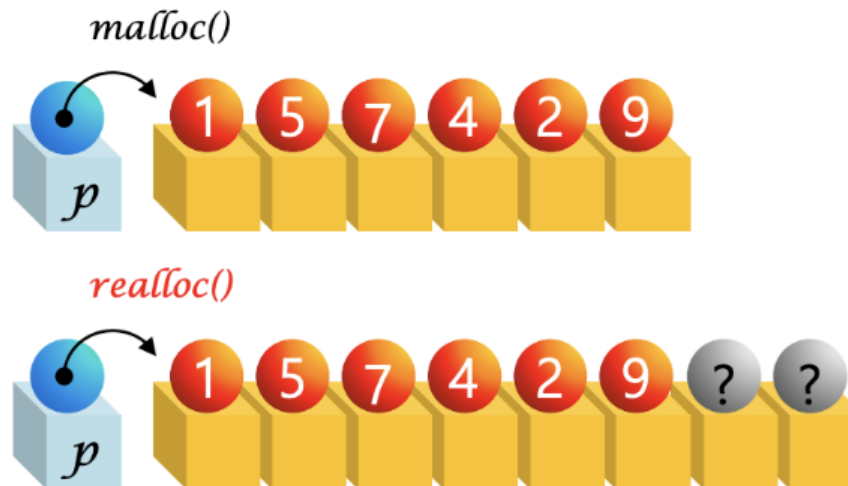
```
➤ make -s
➤ ./main
5
5개의 정수를 입력해 주세요: 1 2 3 4 5
1 2 3 4 5 0 0 ➤ □
```

## 01 동적메모리 (11/12)

### □ realloc 함수

realloc 함수는 할당되었던 메모리 블록의 크기를 변경합니다.

```
int *p;  
p = (int *)malloc(5 * sizeof(int));  
p = realloc(p, 7 * sizeof(int));
```



# 01 동적메모리 (12/12)

## □ realloc 함수 예제

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  ▼ int main() {
5      printf("정수 2개를 저장할 공간이 필요 \n");
6      int *list = (int *)malloc(sizeof(int) * 2);
7      int i;
8      int *list_new;
9      list[0] = 10;
10     list[1] = 20;
11
12     printf("정수 3개를 저장할 공간으로 확장 \n");
13     list_new = (int *)realloc(list, sizeof(int) * 3);
14     list_new[2] = 30;
15
16     for (i = 0; i < 3; i++)
17         printf("%d ", list_new[i]);
18     printf("\n");
19
20     free(list);
21     return 0;
22 }
```

기존에 할당된 메모리 공간의 크기를 변경하여 다시 할당할 수 있다.

```
> make -s
> ./main
정수 2개를 저장할 공간이 필요
정수 3개를 저장할 공간으로 확장
10 20 30
> □
```



# 질의응답

---

금일 튜터링을 진행하며 이해가 어려운 부분이 있었거나,  
교과목과 관련하여 궁금한 내용을 질문하고 답변드리는  
시간입니다.

02

THANKYOU

TUTORING

---

*<https://github.com/developersung13/cbnu-tutoring>*