C언어 기초 part. 2

Week 4 – 구조체와 메모리

QnA 메일 : edujongkook@gmail.com

Pdf 파일 : github.com/edujongkook
/pdf_sbs_c_weekend

©Saebyeol Yu. Saebyeol's PowerPoint

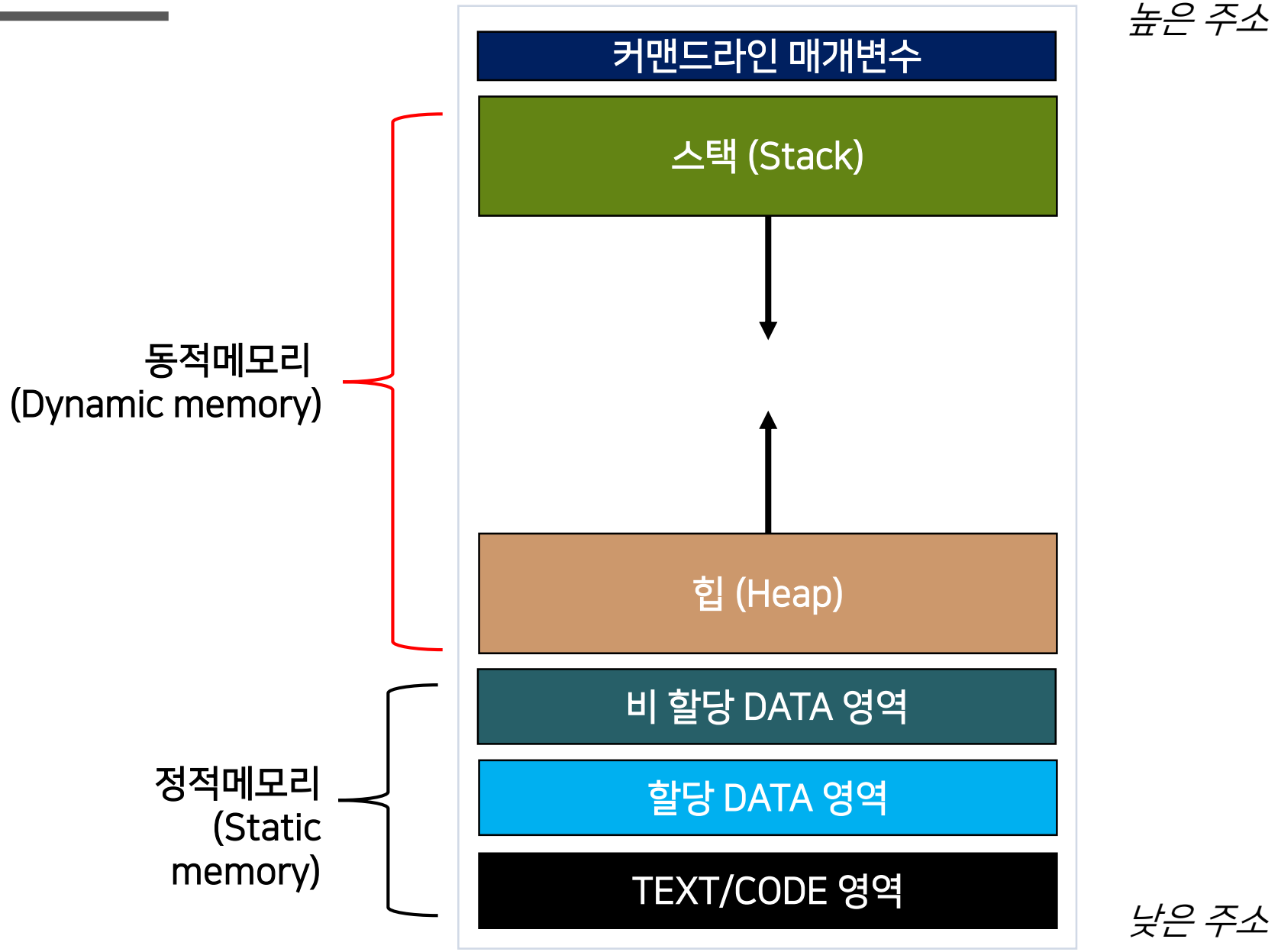
목차 A table of contents

©Saebyeol Yu. Saebyeol's PowerPoint

1.

동적 메모리 복습

1 메모리 영역

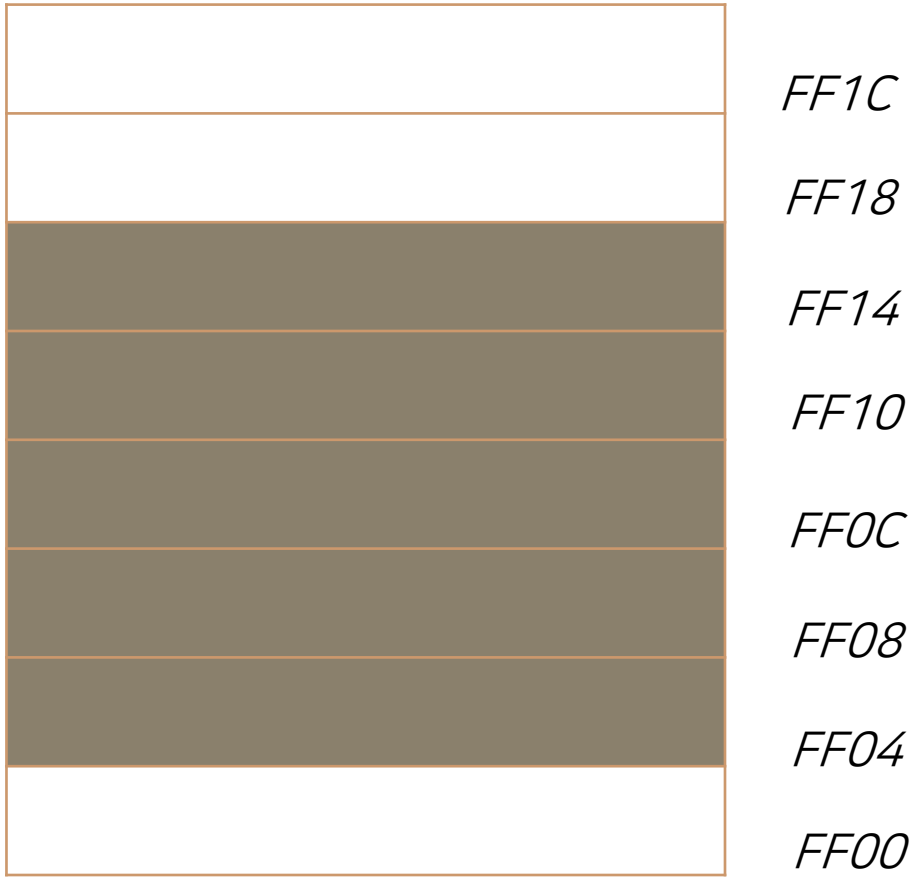

높은 주소
커맨드라인 매개변수
스택 (Stack)
동적메모리
(Dynamic memory)
힙 (Heap)
비 할당 DATA 영역
정적메모리
(Static
memory)
할당 DATA 영역
TEXT/CODE 영역
낮은 주소

1 동적 메모리 할당

원하는 크기만큼 메모리를 할당하고 시작 주소를 받을 수 있습니다.

malloc (확보할 데이터크기)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int* p;
    p = (int*)malloc(sizeof(int) * 5);

    free(p);
}
```

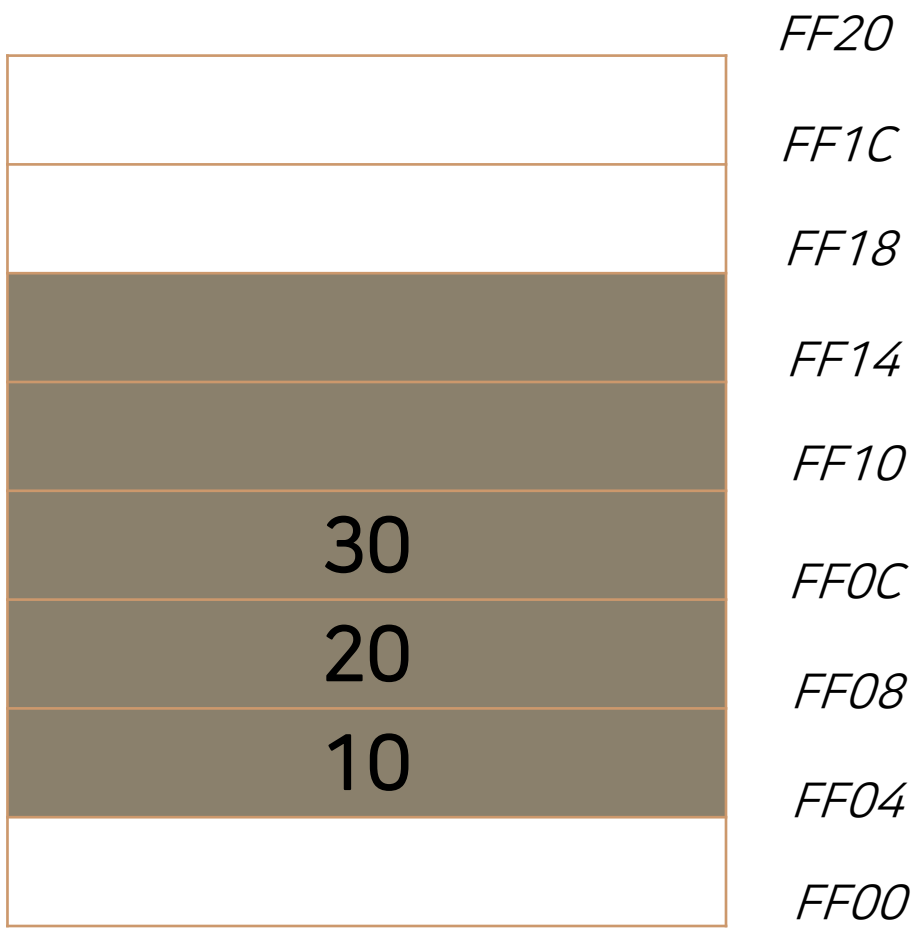


1 동적 메모리 할당

이렇게 만든 메모리 영역에 자유롭게 값을 읽고 쓸 수 있습니다.

사용이 끝난 메모리는 반드시 free 함수로 반환 해야합니다.

```
int main(void){
    int* p;
    p = (int*)malloc(sizeof(int) * 5);
    p[0] = 10;
    p[1] = 20;
    p[2] = 30;
    free(p);
}
```

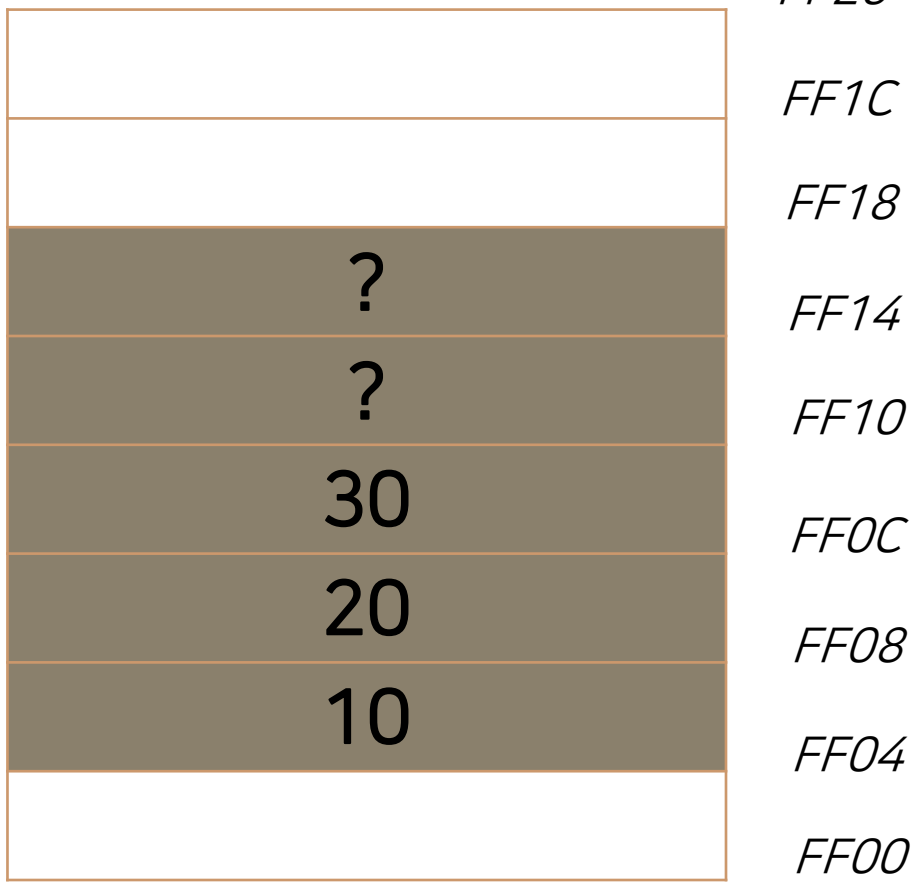


1 동적 메모리 할당

이 때 값을 넣지않은 메모리 안에 데이터는

어떤값이 될지 알 수 없습니다. (쓰레기값)

```c
int main(void){
    int* p;
    p = (int*)malloc(sizeof(int) * 5);
    p[0] = 10;
    p[1] = 20;
    p[2] = 30;
    printf("%d", p[3]);
    free(p);
}
```

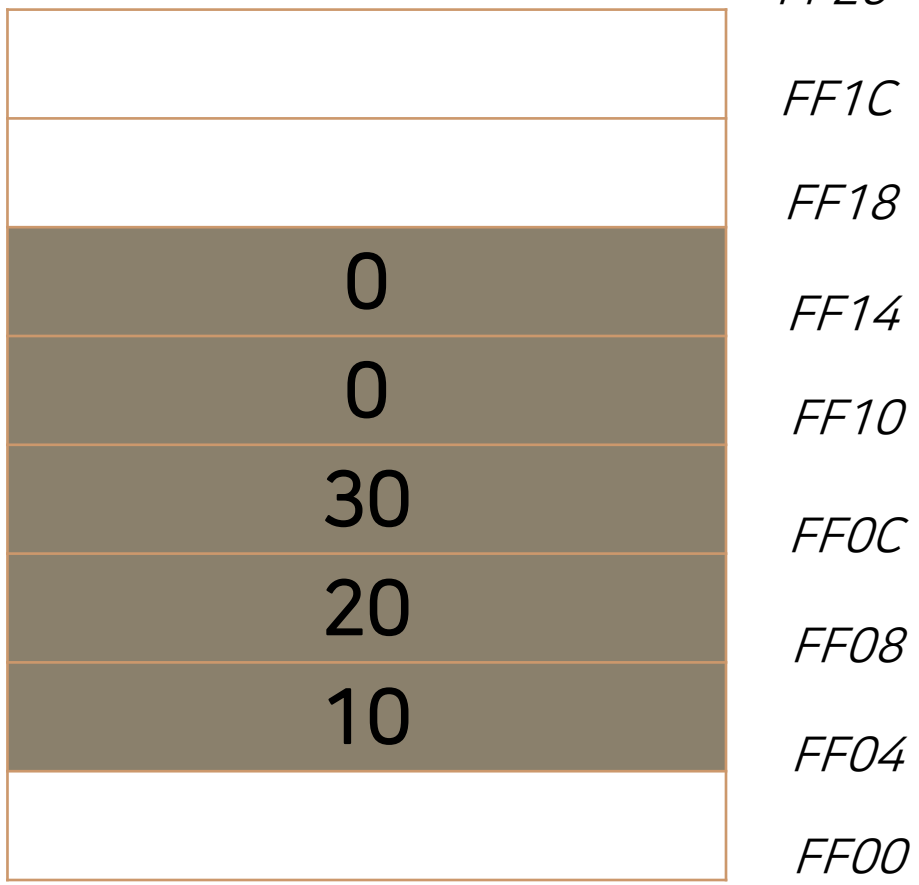


1 동적 메모리 할당

메모리를 할당해 줄 때 0으로 초기화 시켜주는 calloc 함수도 있습니다.

calloc (데이터개수, 데이터 하나의 크기)

```c
int main(void){
    int* p;
    p = (int*)calloc(5, sizeof(int));
    p[0] = 10;
    p[1] = 20;
    p[2] = 30;
    printf("%d", p[3]);
    free(p);
}
```

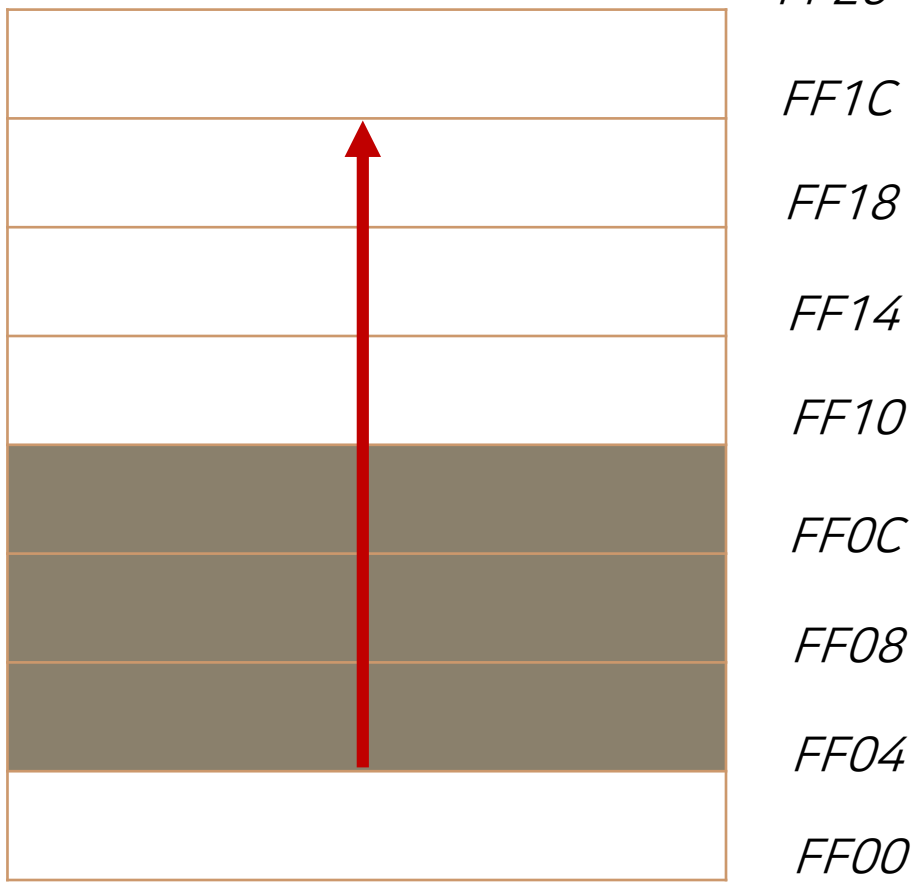


1 동적 메모리 할당

메모리의 크기를 재조정하는 realloc 함수도 있습니다 - 기존 데이터 유지

realloc (메모리 크기를 조정할 영역 시작주소, 재할당할 크기)

```
int main(void){
    int* p;
    p = (int*)malloc(sizeof(int) * 3);
    p[0] = 10;
    p = (int*)realloc(p, sizeof(int) * 6);
    printf("%d₩n", p[0]);
    p = (int*)malloc(6, sizeof(int));
    printf("%d₩n", p[0]);
}
```

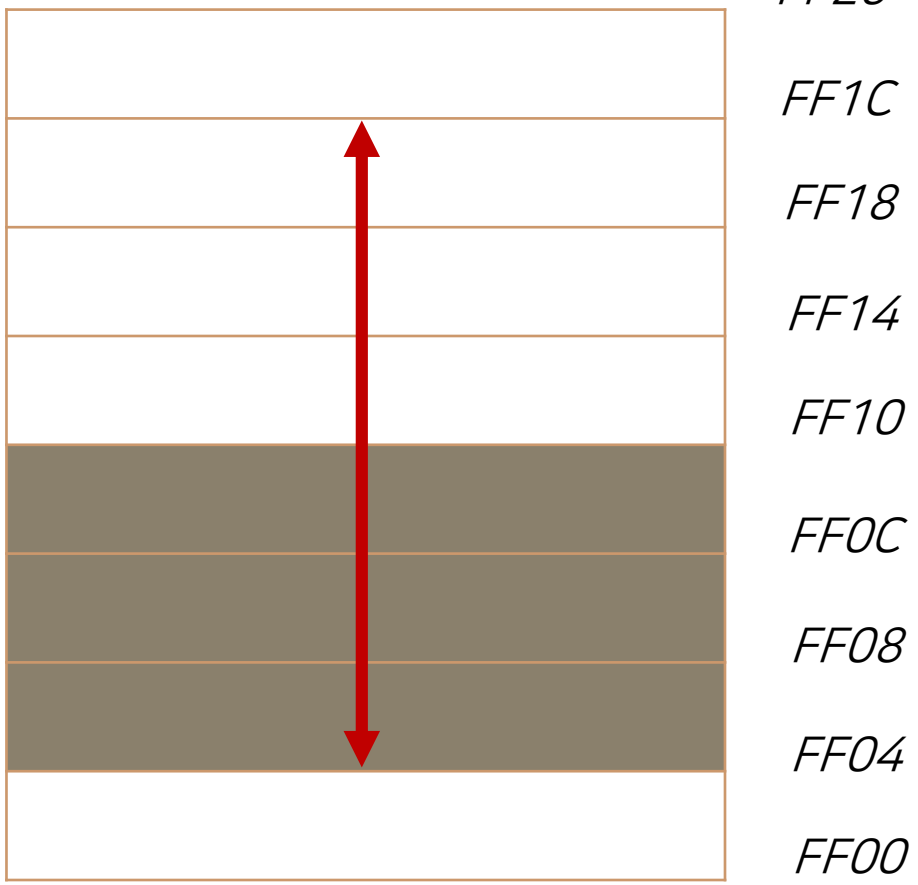


1 동적 메모리 할당

이렇게 배열은 크기를 코드에 직접 적어야하는

반면에 동적메모리는 자유롭게 크기를 언제든 조정할 수 있습니다.

```c
int main(void){
    int* p, size, i;
    puts("저장할 숫자 개수를 입력:");
    scanf_s("%d", &size);
    p = (int*)malloc(sizeof(int) * size);

    for (i = 0; i < size; i++) {
        printf("%d번째 숫자입력:", i+1);
        scanf_s("%d", &p[i]);
    }
```



1 문자열 동적할당

문자열을 scanf_s 로 저장하는 코드를 작성해 보겠습니다.

```
int main(void){
    char name[10];
    scanf_s("%s", name, 10);
    printf("%s₩n", name);
}
```

k	i	m	₩0						

1 문자열 동적할당

문자열의 글자 크기를 확인하는 strlen 함수입니다. (₩0 제외)

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

int main(void){
    char name[10];
    scanf_s("%s", name, 10);
    int size = strlen(name);
    printf("%d₩n", size);
}
```

1 문자열 동적할당

딱 필요한 크기만큼 메모리를 할당해 보겠습니다.

```c
#include <stdio.h>
#include <string.h> // strlen()
#include <stdlib.h> // malloc()
int main(void){
    char name[10];
    scanf_s("%s", name, 10);
    int size = strlen(name);
    char* c = (char*)malloc(sizeof(char) * (size + 1));
}
```


1 문자열 동적할당

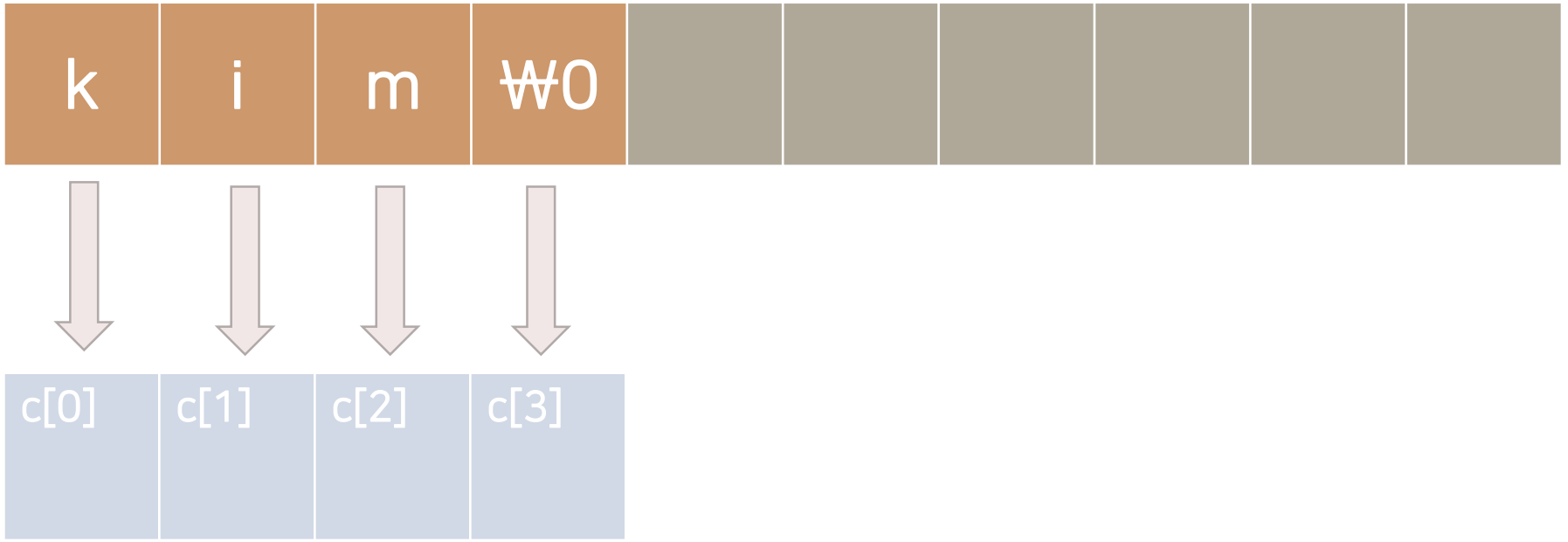
만들어진 메모리에 수동으로 테이터를 복사 합니다.

```
int main(void){
    char name[10];
    scanf_s("%s", name, 10);
    int size = strlen(name);
    char* c = (char*)malloc(sizeof(char) * (size + 1));
    for (int i = 0; i < size + 1; i++) {
        c[i] = name[i];
    }
    printf("%s₩n", c);
    free(c);
}
```

1 문자열 동적할당

만들어진 메모리에 수동으로 테이터를 복사 합니다.

```
char name[10];
```




```
char* c = (char*)malloc(sizeof(char) * (size + 1));
```

1 문자열 동적할당

```c
#include <stdio.h>
#include <string.h> // strlen(), strcpy_s()
#include <stdlib.h> // malloc()
int main(void){
    char name[10];
    scanf_s("%s", name, 10);
    int size = strlen(name);
    char* c = (char*)malloc(sizeof(char) * (size + 1));
    strcpy_s(c, size + 1, name);
    printf("%s", c);
}
```

문자열을 자동으로 복사해 주는

strcpy_s 함수가

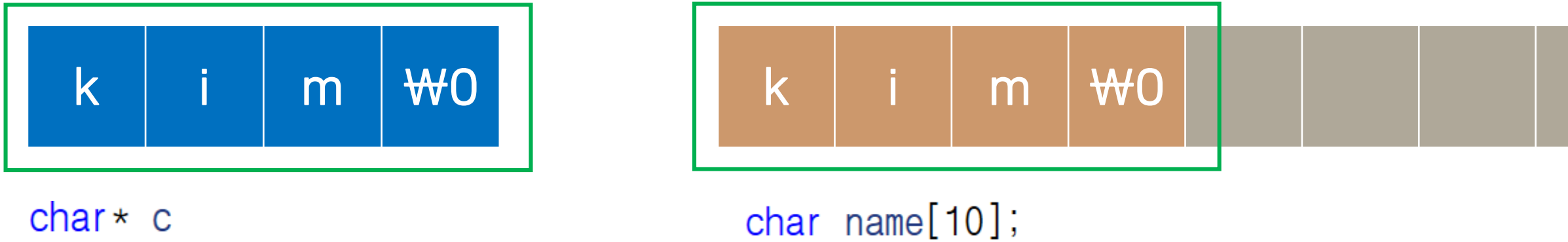
string.h 에 들어있습니다.

1 문자열 동적할당

문자열을 자동으로 복사해 주는 strcpy_s 함수가 string.h 에 들어있습니다.

c로 name 안의 문자를 size + 1 만큼 (문자 개수 + null문자)

```
strcpy_s(c, size + 1, name);
```




```
char* c
```

```
char name[10];
```

1 문자열 동적할당 strdup() 함수

한번에 malloc 과 strcpy_s 를 동시에 해주는 함수가 있는데 바로 _strdup()함수 입니다.

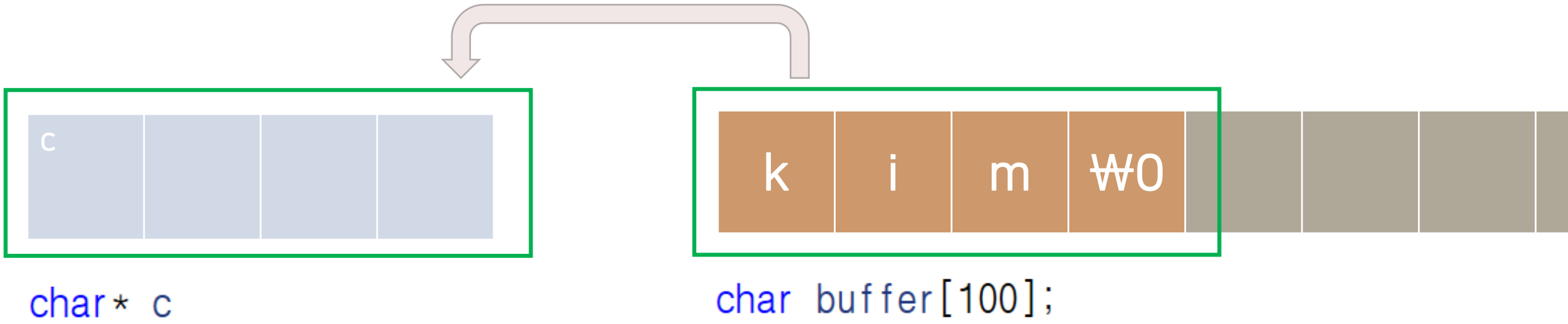
```c
#include <string.h> // _strdup()

int main(void){
    char buffer[100];
    scanf_s("%s", buffer, 100);
    char* c = _strdup(buffer);
    printf("%s", c);
    free(c);
}
```

1 문자열 동적할당 strdup() 함수

한번에 malloc 과 strcpy_s 를 동시에 해주는 함수가 있는데 바로 _strdup()함수 입니다.

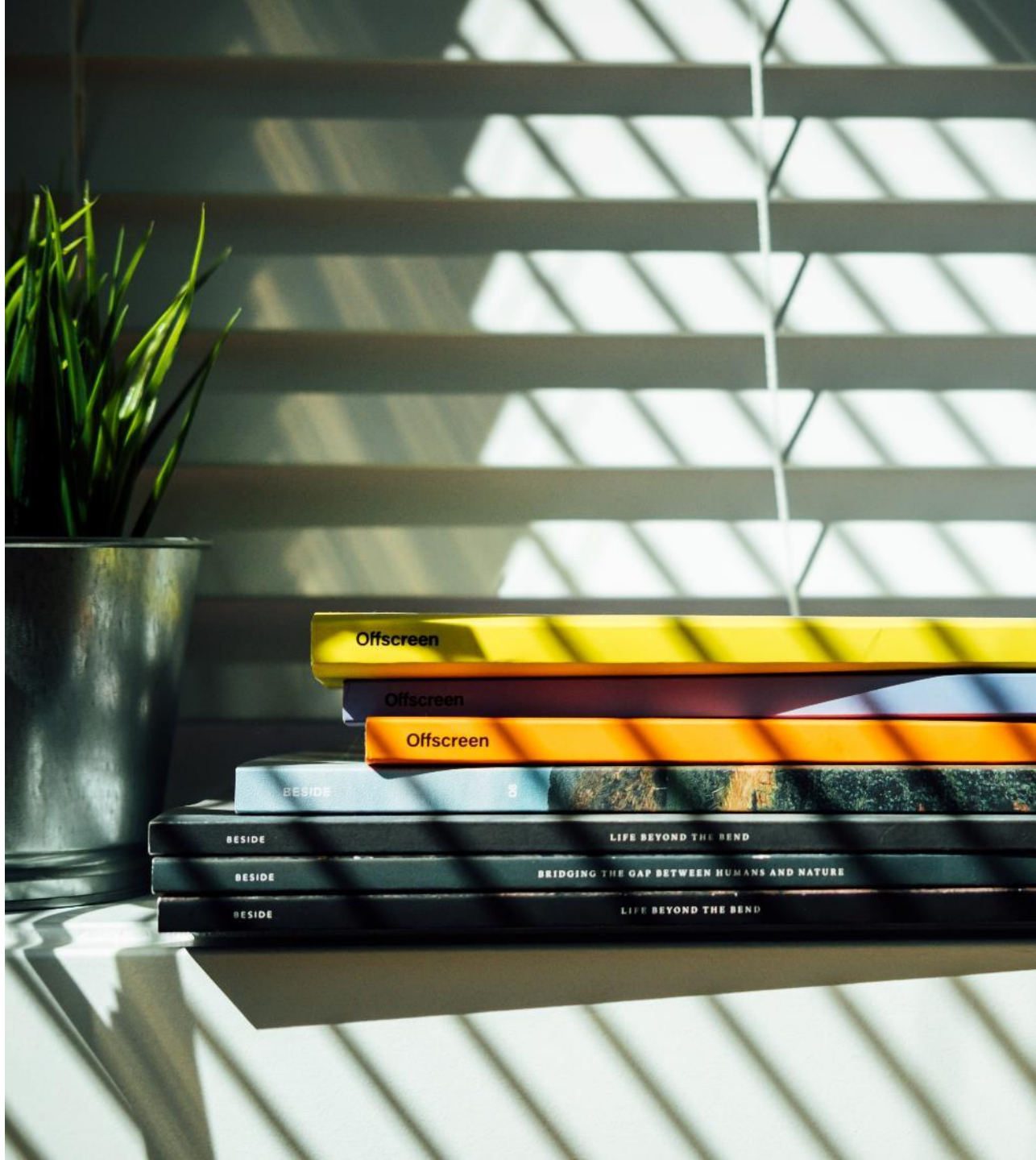
c는 buffer 안의 문자 + null문자 크기만큼 메모리를 할당하고 복사

```
char* c = _strdup(buffer);
```



2.

구조체 , typedef



구조체

기존에 배운 자료형이나 배열은 같은 자료형의 연속 이었습니다.

```
int main(void) {

    char names[100][6];
    int scores[100][3];
```

K	I	M	₩0		
P	A	R	K	₩0	
S	O	N	G	₩0	
…					

char형 배열 names

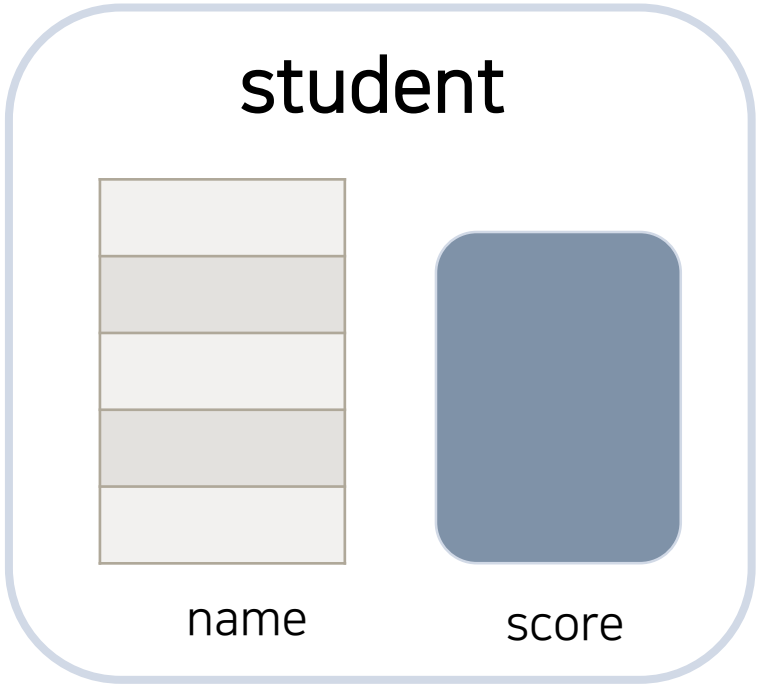
20	40	55
44	95	99
39	89	12
…		

int형 배열 scores

2 구조체 만들기

서로 다른 자료형을 모아서 만들 수 있는데 이를 구조체 라고 합니다.

```
int main(void) {

    struct student {
        char name[20];
        int score;
    };

    struct student st1;
    struct student st2;
```



2 구조체 초기화

배열처럼 선언과 동시에 초기화 할 수 있습니다.

```
int main(void) {
    struct student {
        char name[20];
        int score;
    };

    struct student st1 = { "김철수", 80 };
```

2 구조체 맴버변수

구초체에 속하는 각 변수를 맴버변수라고 하고 아래처럼 . 을 이용해 접근할 수 있습니다.

```
int main(void) {
    struct student {
        char name[20];
        int score;
    };

    struct student st1 = { "김철수", 80 };
    printf("이름 : %s₩n", st1.name);
    printf("점수 : %d₩n", st1.score);
}
```

구조체 맴버변수

물론 먼저 선언만 하고 나중에 값을 넣을 수도 있습니다.

```
int main(void) {
    struct student {
        char name[20];
        int score;
    };

    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    printf("이름 : %s₩n", st2.name);
    printf("점수 : %d₩n", st2.score);
}
```

맴버변수 name은 배열이기 때문에 바로 값을 대입할 수는 없습니다. (strcpy_s 함수 사용)

구조체 맴버변수

물론 먼저 선언만 하고 나중에 값을 넣을 수도 있습니다.

```
int main(void) {
    struct student {
        char* name;
        int score;
    };

    struct student st1 = { "김철수", 80 };
    struct student st2;
    st2.name = "박지성";
    st2.score = 100;
    printf("이름 : %s₩n", st2.name);
    printf("점수 : %d₩n", st2.score);
}
```

맴버변수 name 을
포인터로 만들면
직접 문자열(주소)를
대입할 수 있습니다.

2 구조체를 함수의 매개변수로

보통 main함수 바깥에서 선언을 합니다. (소스코드 어디든지 사용가능하도록)

```
struct student {
    char name[20];
    int score;
};

int main(void) {
    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    print_st(st2);
}
```

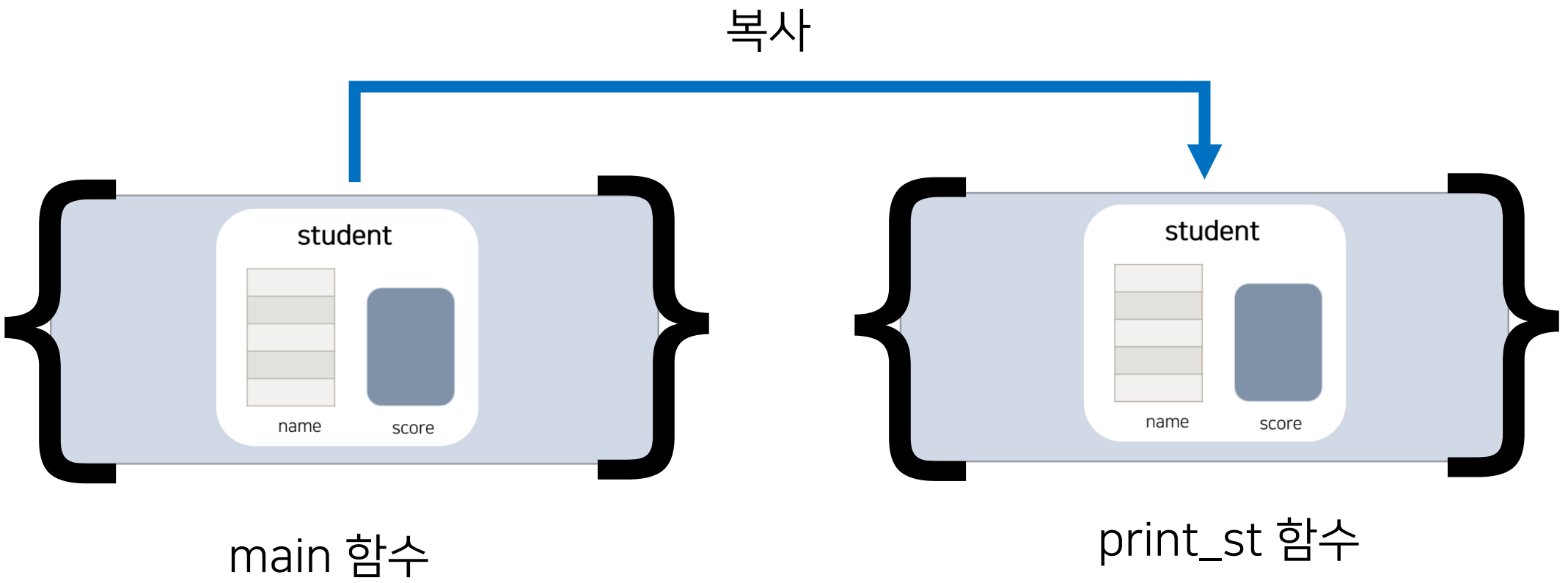
2 구조체를 함수의 매개변수로

구조체를 함수의 매개변수로 전달하는 방법을 알아보겠습니다.

```
int main(void) {
    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    print_st(st2);
}
void print_st(struct student st) {
    printf("이름 : %s₩n", st.name);
    printf("점수 : %d₩n", st.score);
}
```

2 구조체를 함수의 매개변수로

일반적인 자료형을 전달하듯이 매개변수를 지정해 주면 됩니다.(값의 복사가 일어남)



2 구조체를 함수의 매개변수로

구조체가 복사되기 때문에 크기를 알 수 있습니다.

배열과 같이 구조체의 이름에는 크기값이 있습니다.

```
void print_st(struct student st) {
    printf("이름 : %s₩n", st.name);
    printf("점수 : %d₩n", st.score);
    printf("st의 크기: %d₩n", sizeof(st));
}
```

```
struct student {
    char name[20];
    int score;
};
```

20 byte
+
4 byte

크기 = 24 byte

2 구조체의 포인터

배열과 마찬가지로 구조체를 포인터를 이용하여 다룰 수도 있습니다.

```
int main(void) {
    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    struct student* st_p = &st2;
    printf("이름 : %s₩n", (*st_p).name);
    printf("점수 : %d₩n", (*st_p).score);
    //print_st(st2);
}
```

2 구조체의 포인터

구조체 포인터에 접근하여 맴버변수를 사용하는 쉬운 방식이 있습니다.

(*st_p).name 대신 st_p ->name

```
int main(void) {
    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    struct student* st_p = &st2;
    printf("이름 : %s₩n", st_p->name);
    printf("점수 : %d₩n", st_p->score);
    //print_st(st2);
}
```

2 구조체의 포인터

print_st 함수가 구조체 포인터를 매개변수로 받도록 수정해 보겠습니다.

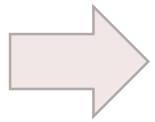
```
int main(void) {
    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    struct student* st_p = &st2;
    print_st(st_p);
}

void print_st(struct student* st) {
    printf("이름 : %s₩n", st->name);
    printf("점수 : %d₩n", st->score);
    printf("st의 크기: %d₩n", sizeof(st));
    printf("*st의 크기: %d₩n", sizeof(*st));
}
```

2 typedef

지금까지 struct student st1 처럼 struct 구조체이름 을 붙여왔는데

이를 간단하게 typedef 를 사용하면 하나의 이름으로 압축할 수 있습니다.

```c
struct student {
    char name[20];
    int score;
};

int main(void) {
    struct student st1 = { "김철수", 80 };
    struct student st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    print_st(st2);
}
```



```c
typedef struct{
    char name[20];
    int score;
}st;

int main(void) {
    st st1 = { "김철수", 80 };
    st st2;
    strcpy_s(st2.name, 20, "박지성");
    st2.score = 100;
    print_st(st2);
}
```

2 typedef

typedef 는 구조체 뿐만 아니라 내가 원하는 자료형을 자유롭게 만들 수 있습니다.

아래는 #include <stdint.h> 의 일부

```
typedef signed char          int8_t;
typedef short                int16_t;
typedef int                  int32_t;
typedef long long            int64_t;
typedef unsigned char        uint8_t;
typedef unsigned short       uint16_t;
typedef unsigned int         uint32_t;
typedef unsigned long long uint64_t;
```

2 구조체가 담긴 배열

간단하게 3개의 구조체가 들어있는 배열을 만들고

전체를 출력하는 함수를 작성해 보겠습니다.

```
int main(void) {
    st students[3];
    for (int i = 0; i < 3; i++) {
        printf("%d번 이름을 입력하세요: ", i + 1);
        scanf_s("%s", students[i].name, 20);
        printf("%d번 점수를 입력하세요: ", i + 1);
        scanf_s("%d", &students[i].score);
        rewind(stdin);
    }
    print_st(students);
}
```

2 구조체가 담긴 배열

간단하게 3개의 구조체가 들어있는 배열을 만들고

전체를 출력하는 함수를 작성해 보겠습니다.

```
void print_st(st* st) {
    for (int i = 0; i < 3; i++) {
        printf("%d번 이름 : %s₩n", i+1, st[i].name);
        printf("%d번 점수 : %d₩n", i+1, st[i].score);
    }
}
```

2 구조체 동적할당

배열이 아닌 동적할당으로 원하는 만큼의 학생수를 입력받도록 수정해 보겠습니다.

```
int main(void) {
    st* students;
    int size;
    puts("입력할 학생 수를 적어주세요");
    scanf_s("%d", &size);
    students = (st*)malloc(size * sizeof(st));
    for (int i = 0; i < size; i++) {
        printf("%d번 이름을 입력하세요: ", i + 1);
        scanf_s("%s", students[i].name, 20);
        printf("%d번 점수를 입력하세요: ", i + 1);
        scanf_s("%d", &students[i].score);
        rewind(stdin);
    }
    print_st(students, size);
}
```

2 구조체 동적할당

print_st 함수도 입력된 학생수만큼 반복될 수 있도록 매개변수를 수정합니다.

```
void print_st(st* st, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d번 이름 : %s₩n", i+1, st[i].name);
        printf("%d번 점수 : %d₩n", i+1, st[i].score);
    }
}
```

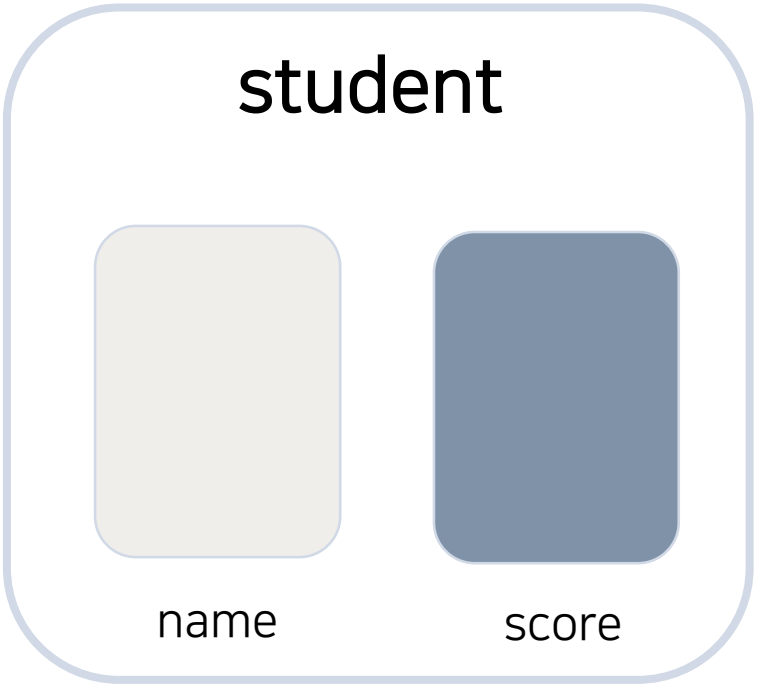
3.

구조체 실습

3 구조체 실습

다음과 같은 구조체로 학생데이터를 입력하는 프로그램을 만들어 보겠습니다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    char* name;
    int score;
} st;
```

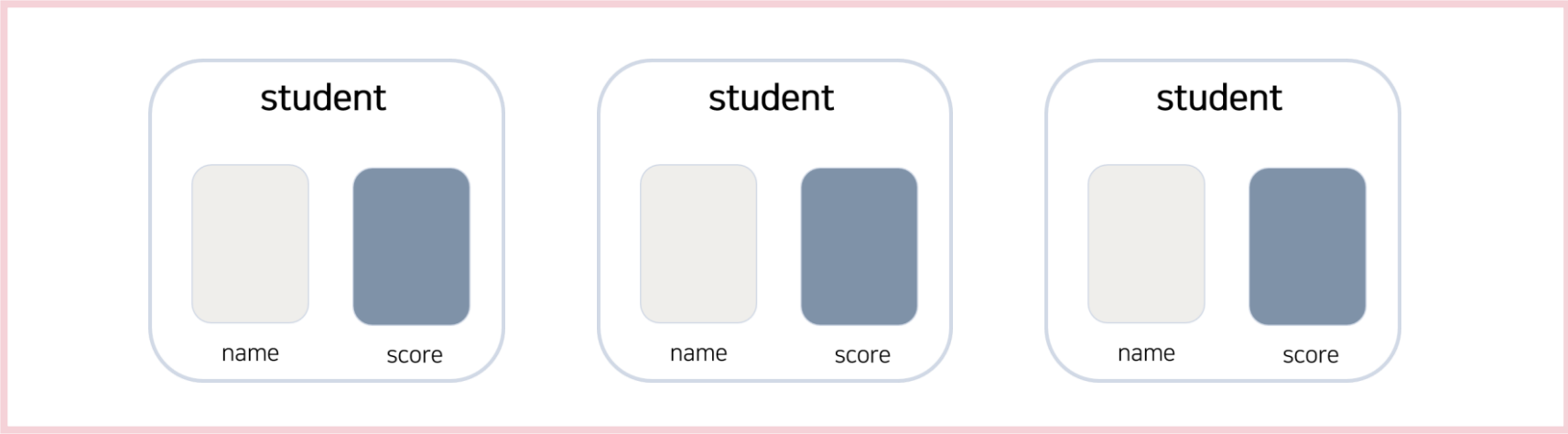


3 구조체 실습

무한반복 하면서 메뉴를 선택하면 switch를 이용하여 해당 작업을 처리합니다.

```
int main(void) {
    int cmd;
    int max_count = 3;
    int count = 0;
    st* students = (st*)malloc(max_count * sizeof(st));
    while (1) {
        puts("메뉴를 선택하세요 "
            "(0:종료, 1:학생정보 추가, 2:학생조회):");
        scanf_s("%d", &cmd);
        switch (cmd) {
```

3 구조체 실습




```
int main(void) {
    int cmd;
    int max_count = 3;
    int count = 0;
    st* students = (st*)malloc(max_count * sizeof(st));
    while (1) {
        puts("메뉴를 선택하세요 "
            "(0:종료, 1:학생정보 추가, 2:학생조회):");
        scanf_s("%d", &cmd);
        switch (cmd) {
```

처음에는 구조체 st 3개를 저장할 공간을 할당 받습니다.

구조체 실습

```
        switch (cmd) {
        case 0:
            puts("프로그램을 종료합니다.");
            free(students);
            return 0;
        case 1:
            puts("학생정보를 추가합니다.");
            students[count++] = make_st();
            if (count == max_count) {
                max_count += 3;
                students = (st*)realloc(students, sizeof(st) * max_count);
            }
            break;
        case 2:
            print_st(students, count);
            break;
        }
    }
}
```

구조체 실습

```
        switch (cmd) {
        case 0:
            puts("프로그램을 종료합니다.");
            free(students);
            return 0;
        case 1:
            puts("학생정보를 추가합니다.");
            students[count++] = make_st();
            if (count == max_count) {
                max_count += 3;
                students = (st*)realloc(students, sizeof(st) * max_count);
            }
            break;
        case 2:
            print_st(students, count);
            break;
        }
    }
}
```

0번 메뉴 (종료)를 선택하면
할당 받은 공간을 해제하고 return
으로 프로그램을 종료합니다.

구조체 실습

```
        switch (cmd) {
        case 0:
            puts("프로그램을 종료합니다.");
            free(students);
            return 0;
        case 1:
            puts("학생정보를 추가합니다.");
            students[count++] = make_st();
            if (count == max_count) {
                max_count += 3;
                students = (st*)realloc(students, sizeof(st) * max_count);
            }
            break;
        case 2:
            print_st(students, count);
            break;
        }
    }
}
```

1번 메뉴 (학생 추가)를 선택하면
학생의 이름과 성적을 입력받아
구조체로 돌려주는 함수
make_st() 를 호출합니다.

구조체 실습

```
        switch (cmd) {
        case 0:
            puts("프로그램을 종료합니다.");
            free(students);
            return 0;
        case 1:
            puts("학생정보를 추가합니다.");
            students[count++] = make_st();
            if (count == max_count) {
                max_count += 3;
                students = (st*)realloc(students, sizeof(st) * max_count);
            }
            break;
        case 2:
            print_st(students, count);
            break;
        }
    }
}
```

학생의 구조체를 할당 받은 공간에
추가하고 count값을 1 증가시키는데
초기의 할당 받은 3개가 꽉 차면
realloc으로 3개만큼의 공간을
확장해 줍니다.

3 구조체 실습

학생의 이름과 점수를 입력받아 구조체를 만들어 리턴하는 make_st 함수입니다.

```
st make_st(void) {
    char buffer[100];
    int score;
    rewind(stdin);
    puts("학생이름을 입력하세요.");
    scanf_s("%s", buffer, 100);
    puts("점수를 입력하세요.");
    scanf_s("%d", &score);
    st temp = { _strdup(buffer), score };
    return temp;
}
```

구조체 실습

입력된 모든 학생들의 정보를 출력하는 print_st 함수입니다.

```
void print_st(st* st, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d번 이름 : %s₩n", i + 1, st[i].name);
        printf("%d번 점수 : %d₩n", i + 1, st[i].score);
    }
}
```