자료구조 HOMEWORK #3

학번: 2019015035
학과: 수학과, 소프트웨어학과(복수전공)
이름: 문혜준

1. Lab#3의 다음 프로그램을 작성하고 실행결과를 보고서로 제출하라.

ap1.c, ap2.c, p2-1.c, p2-2.c, size.c, struct.c, padding.c

1. ap1.c

```c
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int list[5]; // int형이고, 크기가 5인 배열 list를 선언한다.
    int *plist[5] = {
        // int 포인터형이고, 크기가 5이고, NULL로 초기화된 배열 plist를 선언한다.
        NULL,
    };
    plist[0] = (int *)malloc(sizeof(int)); // int형으로 동적할당 해준다. plist[0]은 동적할당된 heap 구역을 가리키게 된다.
    list[0] = 1;                           // list[0]은 1,
    list[1] = 100;                         // list[1]은 100,
    *plist[0] = 200;                       // 동적할당한 plist[0]이 가리키는 곳에 200을 저장한다.
    printf("[----- [Munhyaejun] [2019015035] -----]\n");
    printf("list[0] = %d\n", list[0]);   // list[0]에 1이 저장되어 있으므로 1이 출력될 것이다.
    printf("&list[0] = %p\n", &list[0]); // list[0]의 주소값을 출력할 것이다.
    printf("list = %p\n", list);         // (list) == (list[0]의 주소)이므로, 앞의 값과 같은 값이 출력될 것이다.
    printf("&list = %p\n", &list);       // list는 배열 list의 주소값을 의미하므로, &list와 같은 값이 된다. 따라서 앞의 값과 같은 값이 출력될 것이다.
    printf("----------------------------------------\n\n");
    printf("list[1] = %d\n", list[1]);       // list[1]에 100이 저장되어 있으므로 100이 출력될 것이다.
    printf("&list[1] = %p\n", &list[1]);     // list[1]의 주소값을 출력할 것이다. 배열이므로, list[0]의 값에서 4(int형의 크기)만큼 차이가 난다.
    printf("*(list+1) = %d\n", *(list + 1)); // (list + 1) == (list[1]의 주소) *(list + 1) 은 list[1]을 출력할 것이다. 따라서 앞의 값과 같은 값이 출력될 것이다.
    printf("list+1 = %p\n", list + 1);       // 위에서 말한 것과 같이, list[1]의 주소값이 출력될 것이다.
    printf("----------------------------------------\n\n");
    printf("*plist[0] = %d\n", *plist[0]); // 200을 저장했으므로 200이 출력될 것이다.
    printf("&plist[0] = %p\n", &plist[0]); // plist[0]의 주소값을 출력할 것이다.
    printf("&plist = %p\n", &plist);       // &plist는 plist 배열의 주소값을 의미하고, 그것은 plist[0]의 주소와 같으므로 위의 값과 같은 값을 출력할 것이다..
    printf("plist = %p\n", plist);         // plist는 plist 배열의 주소를 의미하므로, 위의 값과 같은 값을 출력할 것이다.
    printf("plist[0] = %p\n", plist[0]);   // plist[0]은 포인터이고, 동적할당하였으므로 포인터 값이 나온다.
    printf("plist[1] = %p\n", plist[1]);   // plist[1] ~ plist[4]는 처음에 NULL로 초기화하였으므로, 0이 나올 것이다.
    printf("plist[2] = %p\n", plist[2]);
    printf("plist[3] = %p\n", plist[3]);
    printf("plist[4] = %p\n", plist[4]);
    free(plist[0]); // 동적할당 해주었으므로, free로 풀어 주어야 한다.
}
```

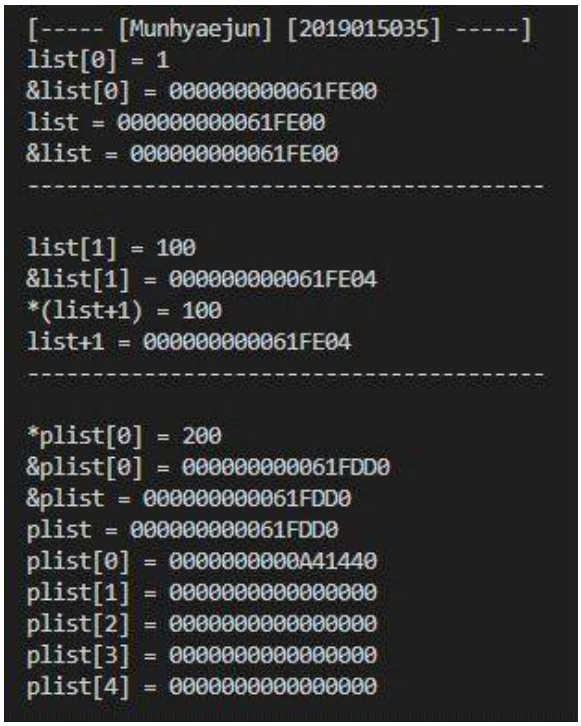
커밋 명령어:

```
PS C:\Users\gpwns\OneDrive\Desktop\3학년_자료모음\자료구조\3주차\homework3> git add ap1.c
PS C:\Users\gpwns\OneDrive\Desktop\3학년_자료모음\자료구조\3주차\homework3> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   ap1.c

PS C:\Users\gpwns\OneDrive\Desktop\3학년_자료모음\자료구조\3주차\homework3> git commit -m "ap1.c 생성"
[main 22c93b4] ap1.c 생성
 1 file changed, 22 insertions(+), 20 deletions(-)
PS C:\Users\gpwns\OneDrive\Desktop\3학년_자료모음\자료구조\3주차\homework3>
```

결과:



설명: list 배열은 int형 배열이므로, 각각 4byte를 차지한다.

list[0]=1, list[1]=100이라고 하였으므로, 이들은 각각 이 값을 가진다.

list: 배열의 주소를 의미한다. &list[0], &list 또한 주소를 의미한다.

list+1: 배열의 주소 +1칸(int형이 4byte이므로 주소는 4 변화)이동했음을 의미한다. 따라서 &list[1]을 의미한다.

plist 배열은 int형을 가리키는 포인터 배열이므로, 각각 8byte를 차지한다.

*plist[0]=200 하였으므로, 이 값을 가진다. plist는 동적할당하였으므로, heap 영역에 저장하고 plist[0]은 그 위치를 가리키게 된다.

plist: 배열의 주소를 의미한다. &plist[0], &plist 또한 주소를 의미한다.

plist 배열은 처음에 NULL로 초기화 되었으므로, 값을 정해준 plist[0] 외엔 모두 0을 가진다.

2. ap2.c

```c
C ap2.c > main()
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int list[5];   // int형이고, 크기가 5인 배열 list를 선언한다.
    int *plist[5]; // int 포인터형이고, 크기가 5인 배열 plist를 선언한다.
    list[0] = 10;  // list[0]에 10을, list[1]에 11을 저장한다.
    list[1] = 11;
    plist[0] = (int *)malloc(sizeof(int)); // plist[0]에 int형 변수를 동적으로 할당한다.
    printf("[----- [hyaejun-mun] [2019015035] -----]\n");
    printf("list[0] \t= %d\n", list[0]);   // list[0] = 10이므로, 10이 출력될 것이다.
    printf("list \t\t= %p\n", list);       // list는 배열의 주소를 의미하므로, 배열의 주소가 출력될 것이다.
    printf("&list[0] \t= %p\n", &list[0]); // list[0]의 주소는 배열의 주소 + 0과 같으므로, 위와 같은 값이 출력될 것이다.
    printf("list + 0 \t= %p\n", list + 0); // list + 0은 배열의 주소 + 0과 같으므로 위와 같은 값이 출력될 것이다.
    printf("list + 1 \t= %p\n", list + 1); // list + 1은 배열의 주소 + 4(int형의 크기 * 1)이와 같이 출력될 것이다.
    printf("list + 2 \t= %p\n", list + 2); // list + 2은 배열의 주소 + 8(int형의 크기 * 2)이와 같이 출력될 것이다.
    printf("list + 3 \t= %p\n", list + 3); // list + 3은 배열의 주소 + 12(int형의 크기 * 3)이와 같이 출력될 것이다.
    printf("list + 4 \t= %p\n", list + 4); // list + 4은 배열의 주소 + 16(int형의 크기 * 4)이와 같이 출력될 것이다.
    printf("&list[4] \t= %p\n", &list[4]); // list[4]의 주소는 list에서 (int형 기준으로)4칸 떨어진 곳이므로,
                                           // 배열의 주소 + 16이 출력되어 위와 같은 값이 출력될 것이다.
    free(plist[0]);                        // 동적할당한 plist[0]을 해제해 주어야 한다.
}
```

커밋:

```
PS C:\Users\gpwns\OneDrive\Desktop\3학년_자료모음\자료구조\3주차\homework3> git add ap2.c
PS C:\Users\gpwns\OneDrive\Desktop\3학년_자료모음\자료구조\3주차\homework3> git commit -m "ap2.c 생성"
[main 64fa932] ap2.c 생성
 1 file changed, 21 insertions(+)
 create mode 100644 ap2.c
```

결과:

```
[----- [hyaejun-mun] [2019015035] -----]
list[0]         = 10
list            = 000000000061FE00
&list[0]        = 000000000061FE00
list + 0        = 000000000061FE00
list + 1        = 000000000061FE04
list + 2        = 000000000061FE08
list + 3        = 000000000061FE0C
list + 4        = 000000000061FE10
&list[4]        = 000000000061FE10
```

설명: list 배열은 int형 배열이므로, 각각 4byte를 차지한다.

list[0]=10, list[1]=11이라고 하였으므로, 이들은 각각 이 값을 가진다.

list: 배열의 주소를 의미한다. &list[0]또한 주소를 의미한다.

list+n: 배열의 주소 +n칸(int형이 4byte이므로 주소는 4*n 변화)이동했음을 의미한다. 따라서 배열의 주소 + 4*n을 의미한다.

&list[4]: list[4]의 주소를 의미한다. list[4]는 배열에서 4칸 떨어져 있으므로, 배열의 주소 + 4*4를 의미한다.

3. p2-1.c

```c
#include <stdio.h>
#define MAX_SIZE 100                                    // MAX_SIZE를 정의한다.
float sum1(float list[], int);                          // 함수 sum1, sum2, sum3을 선언한다.
float sum2(float *list, int);
float sum3(int n, float *list);
float input[MAX_SIZE], answer;                          // 전역 변수 input, answer을 정의한다.
int i;                                                  // for문 반복 시에 사용할 변수 i를 정의한다. 이들은 data 영역에 저장된다.
void main(void)
{
    for (i = 0; i < MAX_SIZE; i++)                      // input 배열에 1~100까지 순서대로 입력한다.
        input[i] = i;
    /* for checking call by reference */
    printf("[----- [hyaejun-mun] [2019015035] -----]\n");
    printf("----------------------------------------\n");
    printf(" sum1(input, MAX_SIZE) \n");
    printf("----------------------------------------\n");
    printf("input \t= %p\n", input);                    // input의 주소를 출력한다.

    answer = sum1(input, MAX_SIZE);                     // sum1로 계산한다.
    printf("The sum is: %f\n\n", answer);               // 1~100까지 더한 값인 4950이 출력된다.

    printf("----------------------------------------\n");
    printf(" sum2(input, MAX_SIZE) \n");
    printf("----------------------------------------\n");
    printf("input \t= %p\n", input);

    answer = sum2(input, MAX_SIZE);                     // sum2로 계산한다.
    printf("The sum is: %f\n\n", answer);               // 마찬가지로, 4950이 출력된다.

    printf("----------------------------------------\n");
    printf(" sum3(MAX_SIZE, input) \n");
    printf("----------------------------------------\n");
    printf("input \t= %p\n", input);

    answer = sum3(MAX_SIZE, input);                     // sum3으로 계산한다.
    printf("The sum is: %f\n\n", answer);               // 마찬가지로, 4950이 출력된다.
}
```

```c
float sum1(float list[], int n) // 배열을 매개 변수로 받았다.
{
    printf("list\t= %p\n", list);     // list는 전역 변수에 있는 input를 의미하므로, 위의 input 주소를 출력할 것이다.
    printf("&list\t= %p\n\n", &list); // &list는 매개 변수로 받아온 list 배열의 주소를 가리키는 포인터를 의미하므로, 다른 주소를 출력할 것이다.

    int i;
    float tempsum = 0;
    for (i = 0; i < n; i++) // tempsum에 모든 수를 더한다.
        tempsum += list[i]; // 배열을 매개 변수로 받았으므로, 배열을 이용한 연산(list[i]) 등이 가능하다.
    return tempsum;
}
float sum2(float *list, int n) // 포인터를 매개 변수로 받았다.
{
    printf("list\t= %p\n", list);     // list는 전역 변수에 있는 input를 의미하므로, 위의 input 주소를 출력할 것이다.
    printf("&list\t= %p\n\n", &list); // &list는 인자로 받아온 포인터의 주소를 가리키는 포인터를 의미하므로, 다른 주소를 출력할 것이다.
    int i;
    float tempsum = 0;
    for (i = 0; i < n; i++)     // tempsum에 모든 수를 더한다.
        tempsum += *(list + i); // 포인터를 매개 변수로 받았으므로, 배열을 이용한 연산(list[i]) 등이 불가능하다.
    return tempsum;
}
/* stack variable reuse test */
float sum3(int n, float *list) // sum2와 매개 변수 순서가 다르다.
{
    printf("list\t= %p\n", list);     // list는 전역 변수에 있는 input를 의미하므로, 위의 input 주소를 출력할 것이다.
    printf("&list\t= %p\n\n", &list); // &list는 인자로 받아온 포인터의 주소를 가리키는 포인터를 의미하므로, 다른 주소를 출력할 것이다.
                                      // 그런데, 매개 변수 순서가 다르므로, n과 *list의 정의 순서가 달라져 sum2와 다른 주소를 출력할 것이다.
    int i;
    float tempsum = 0;
    for (i = 0; i < n; i++)     // tempsum에 모든 수를 더한다.
        tempsum += *(list + i); // 포인터를 매개 변수로 받았으므로, 배열을 이용한 연산(list[i]) 등이 불가능하다.
    return tempsum;
}
```

커밋:

```
gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git add p2-1.c

gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git commit -m "p2-1.c 생성"
[main c6b3cc5] p2-1.c 생성
 1 file changed, 71 insertions(+)
 create mode 100644 p2-1.c
```

결과:

```
[----- [hyaejun-mun] [2019015035] -----]
---------------------------------------
 sum1(input, MAX_SIZE)
---------------------------------------
input   = 0000000000408980
list    = 0000000000408980
&list   = 000000000061FE00

The sum is: 4950.000000

---------------------------------------
 sum2(input, MAX_SIZE)
---------------------------------------
input   = 0000000000408980
list    = 0000000000408980
&list   = 000000000061FE00

The sum is: 4950.000000

---------------------------------------
 sum3(MAX_SIZE, input)
---------------------------------------
input   = 0000000000408980
list    = 0000000000408980
&list   = 000000000061FE08

The sum is: 4950.000000
```

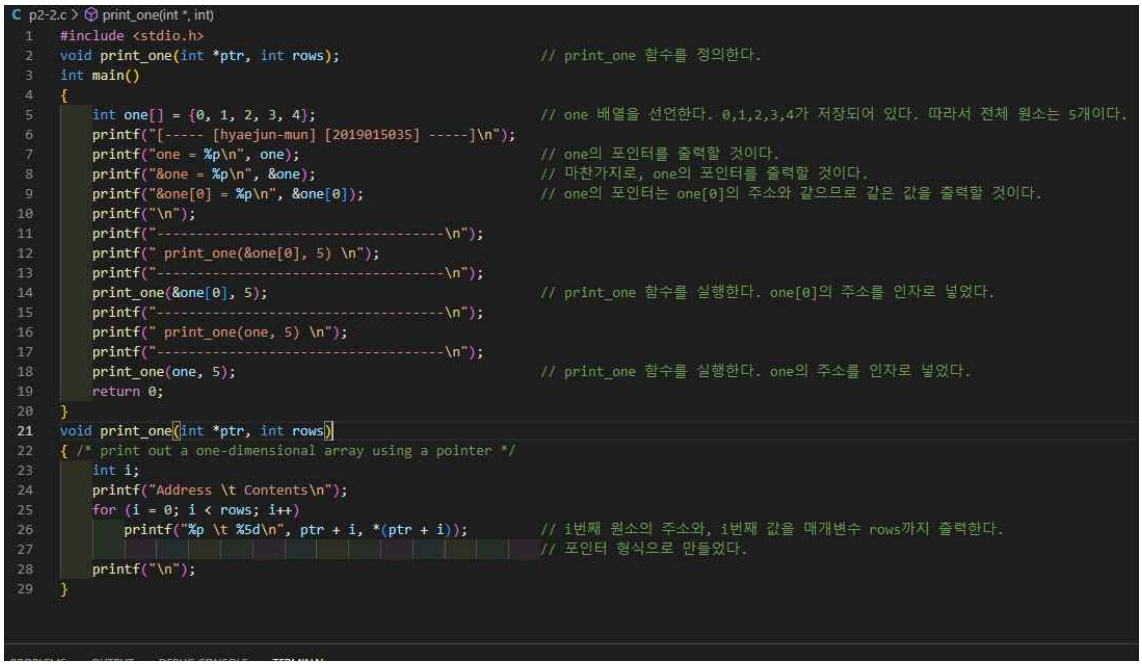
설명: sum1, sum2, sum3은 1~100까지 더하는 함수이다.
list : list는 매개변수로 받아온 배열 input의 주소를 의미한다.
&list: &list는 list의 주소를 의미하므로, 매개 변수의 주소를 의미한다.

sum1은 배열을 매개변수로 받게 되었다.

sum2는 포인터를 매개변수로 받게 되었다. sum1과 sum2는 아무런 실행 과정에서 차이를 보이지 않는다.

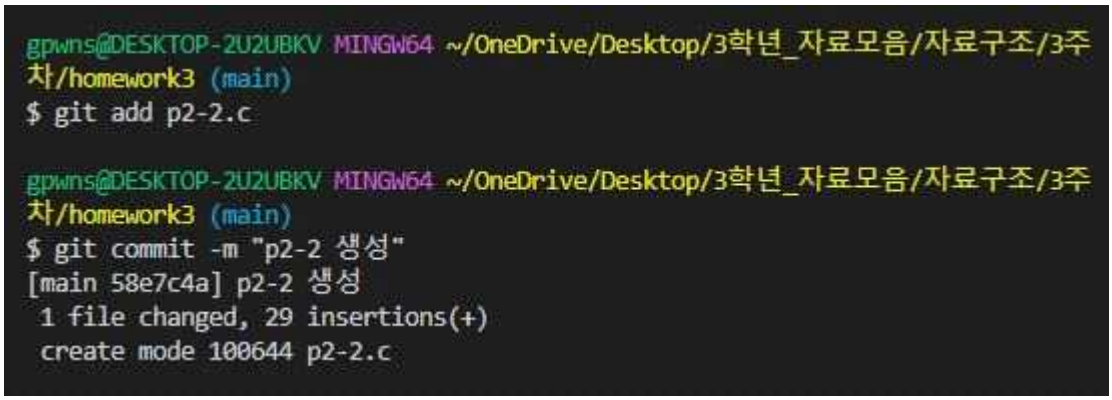
sum3은 sum2와 같지만, 매개변수 순서가 다르다. 그 결과, &list를 출력할 때, 매개 변수 위치가 달라져 다른 sum2의 &list와 다른 값을 출력하게 되었다.

4. p2-2.c



```c
#include <stdio.h>
void print_one(int *ptr, int rows);                     // print_one 함수를 정의한다.
int main()
{
    int one[] = {0, 1, 2, 3, 4};                        // one 배열을 선언한다. 0,1,2,3,4가 저장되어 있다. 따라서 전체 원소는 5개이다.
    printf("[----- [hyaejun-mun] [2019015035] -----]\n");
    printf("one = %p\n", one);                          // one의 포인터를 출력할 것이다.
    printf("&one = %p\n", &one);                        // 마찬가지로, one의 포인터를 출력할 것이다.
    printf("&one[0] = %p\n", &one[0]);                  // one의 포인터는 one[0]의 주소와 같으므로 같은 값을 출력할 것이다.
    printf("\n");
    printf("-------------------------------------\n");
    printf(" print_one(&one[0], 5) \n");
    printf("-------------------------------------\n");
    print_one(&one[0], 5);                              // print_one 함수를 실행한다. one[0]의 주소를 인자로 넣었다.
    printf("-------------------------------------\n");
    printf(" print_one(one, 5) \n");
    printf("-------------------------------------\n");
    print_one(one, 5);                                  // print_one 함수를 실행한다. one의 주소를 인자로 넣었다.
    return 0;
}
void print_one(int *ptr, int rows)
{ /* print out a one-dimensional array using a pointer */
    int i;
    printf("Address \t Contents\n");
    for (i = 0; i < rows; i++)
        printf("%p \t %5d\n", ptr + i, *(ptr + i));     // i번째 원소의 주소와, i번째 값을 매개변수 rows까지 출력한다.
                                                        // 포인터 형식으로 만들었다.
    printf("\n");
}
```

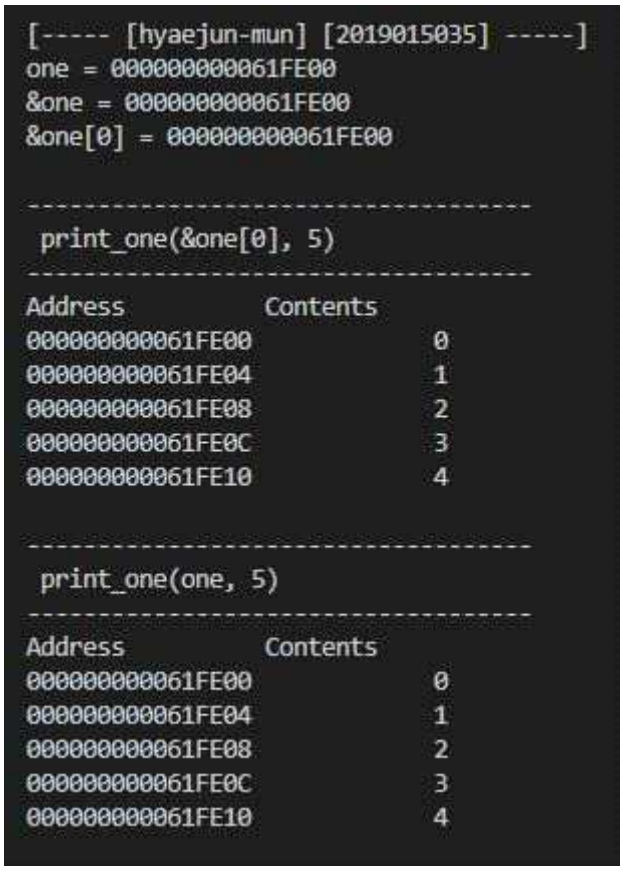
커밋:



```
gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git add p2-2.c

gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git commit -m "p2-2 생성"
[main 58e7c4a] p2-2 생성
 1 file changed, 29 insertions(+)
 create mode 100644 p2-2.c
```

결과:



설명: 인자로 포인터 형식을 취하면 어떤 상황이 가능한지 보여준다.

print_one(&one[0], 5): 포인터로 인자를 입력했다.

print_one(one, 5): 배열로 인자를 입력했다.(위에서 보듯, one == &one[0]이므로 문제없다.)

이처럼 매개 변수를 포인터로 설정해도, 함수는 배열과 포인터 인자 모두 받을 수 있다.

5. size.c

```c
C size.c > main()
#include <stdio.h>
#include <stdlib.h>
void main()
{
    printf("[----- [hyaejun-mun] [2019015035] -----]\n");
    int **x;                                    // 이중 포인터 x를 선언한다.
    printf("sizeof(x) = %lu\n", sizeof(x));     // x의 size를 출력한다. 이중 포인터이므로 8byte이다.
    printf("sizeof(*x) = %lu\n", sizeof(*x));   // *x의 size를 출력한다. 포인터이므로 8byte이다.
    printf("sizeof(**x) = %lu\n", sizeof(**x)); // **x의 size를 출력한다. int형 변수이므로 4byte이다.
}
```

커밋:

```
gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git add size.c

gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git commit -m "size.c 생성"
[main 27a2ddb] size.c 생성
 1 file changed, 10 insertions(+)
 create mode 100644 size.c
```

결과:

```
[----- [hyaejun-mun] [2019015035] -----]
sizeof(x) = 8
sizeof(*x) = 8
sizeof(**x) = 4
```

설명: 포인터와 이중 포인터에 대해 설명한다.

sizeof(x): x는 이중 포인터이므로 포인터 변수의 크기인 8byte를 가진다.

sizeof(*x): *x는 이중 포인터에서 한 번 포인트했으므로 포인터이다. 따라서 포인터 변수의 크기인 8byte를 가진다.

sizeof(**x): **x는 이중 포인터에서 두 번 포인트했으므로 int형 변수이다. 따라서 int형 변수의 크기인 4byte를 가진다.

6. struct.c

struct.c > main()

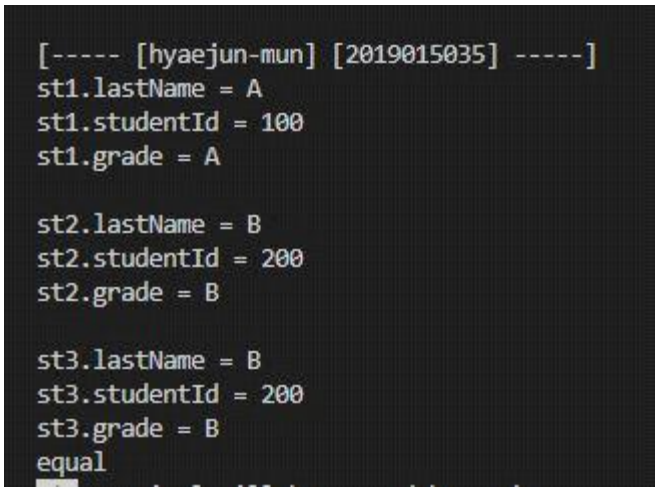
```c
#include <stdio.h>
struct student1                                 //student1 구조체를 만들었다.
{
    char lastName;                              // 마지막 이름, 학번, 점수를 저장한다.
    int studentId;
    char grade;
};
typedef struct                                  // student2 구조체를 만들었다.
{                                               // 앞에 typedef를 붙여서 정의 선언 시 strut를 붙이지 않아도 된다.
    char lastName;                              // student1과 같이 이름, 학번, 점수를 저장한다.
    int studentId;
    char grade;
} student2;
int main()
{
    struct student1 st1 = {'A', 100, 'A'};      // st1을 선언했다.
    printf("st1.lastName = %c\n", st1.lastName);    // 구조체는 .을 통해 내부 원소에 접근할 수 있다.
    printf("st1.studentId = %d\n", st1.studentId);
    printf("st1.grade = %c\n", st1.grade);
    student2 st2 = {'B', 200, 'B'};             // student2는 만들 때 앞에 typedef를 붙였으므로
                                                // 선언 시 struct를 말하지 않아도 된다.
    printf("\nst2.lastName = %c\n", st2.lastName);
    printf("st2.studentId = %d\n", st2.studentId);
    printf("st2.grade = %c\n", st2.grade);
    student2 st3;                               // 빈 구조체 st3을 선언했다.
    st3 = st2;                                  // 같은 구조체이므로, = 를 이용해 값들을 한 번에 옮길 수 없다.
    printf("\nst3.lastName = %c\n", st3.lastName);
    printf("st3.studentId = %d\n", st3.studentId);
    printf("st3.grade = %c\n", st3.grade);
    /* equality test */                         // 하지만, 같은지 확인할 때는 하나하나 체크해 주어야 한다.
    if (st3.lastName == st2.lastName)           // 이름을 비교한다.
    {
        if (st3.studentId == st2.studentId)     // 학번을 비교한다.
            if (st3.grade == st2.grade)         // 점수를 비교한다.
                printf("equal\n");              // 셋 모두 같아야 같다고 한다.
    }
    else
        printf("not equal\n");                  // 그렇지 않으면 다르다고 해야 한다.
    return 0;
}
```

커밋:

```
gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git add struct.c

gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git commit -m "struct.c 생성"
[main b2f2ad9] struct.c 생성
 1 file changed, 41 insertions(+)
 create mode 100644 struct.c
```

결과:



설명: 구조체에 관한 설명이다.

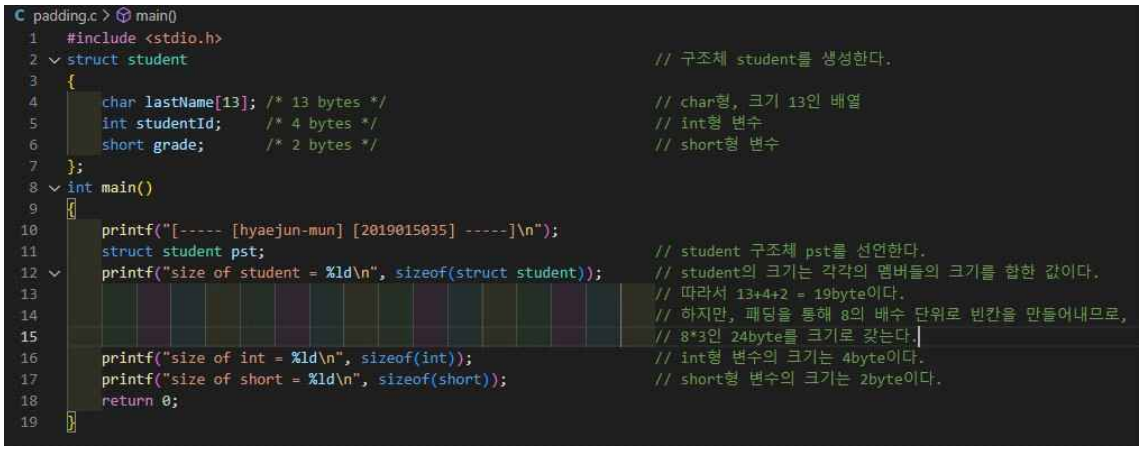
구조체 앞에 typedef를 붙이면, 구조체 정의 시 일일이 앞에 typedef를 붙여 줄 필요가 없다.

구조체 선언 시 동시에 멤버들을 넣어 초기화해 줄 수 있다.

다른 구조체에 들어 있는 멤버들을 한 번에 =을 이용하여 복사할 수 있다.

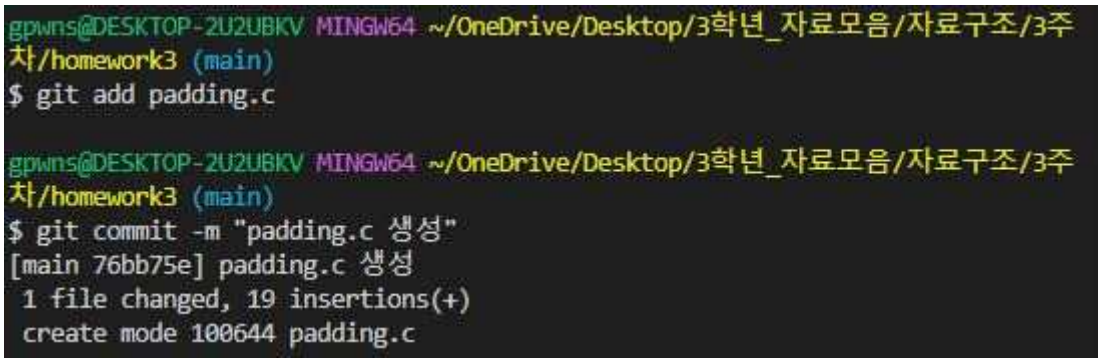
구조체의 비교는 한 번에 할 수 없고, 일일이 하나씩 멤버를 비교해 주어야 한다.

7. padding.c



```c
#include <stdio.h>
struct student                                  // 구조체 student를 생성한다.
{
    char lastName[13]; /* 13 bytes */          // char형, 크기 13인 배열
    int studentId;     /* 4 bytes */           // int형 변수
    short grade;       /* 2 bytes */           // short형 변수
};
int main()
{
    printf("[----- [hyaejun-mun] [2019015035] -----]\n");
    struct student pst;                         // student 구조체 pst를 선언한다.
    printf("size of student = %ld\n", sizeof(struct student));      // student의 크기는 각각의 멤버들의 크기를 합한 값이다.
                                                // 따라서 13+4+2 = 19byte이다.
                                                // 하지만, 패딩을 통해 8의 배수 단위로 빈칸을 만들어내므로,
                                                // 8*3인 24byte를 크기로 갖는다.
    printf("size of int = %ld\n", sizeof(int));         // int형 변수의 크기는 4byte이다.
    printf("size of short = %ld\n", sizeof(short));     // short형 변수의 크기는 2byte이다.
    return 0;
}
```

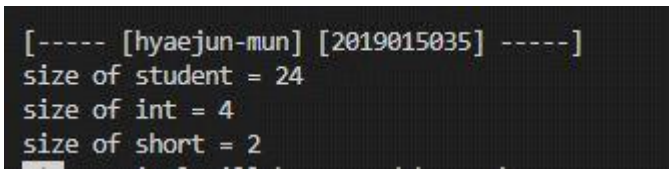
커밋:



```
gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git add padding.c

gpwns@DESKTOP-2U2UBKV MINGW64 ~/OneDrive/Desktop/3학년_자료모음/자료구조/3주
차/homework3 (main)
$ git commit -m "padding.c 생성"
[main 76bb75e] padding.c 생성
 1 file changed, 19 insertions(+)
 create mode 100644 padding.c
```

결과:



```
[----- [hyaejun-mun] [2019015035] -----]
size of student = 24
size of int = 4
size of short = 2
```

설명: 패딩에 관한 설명이다.

패딩이란, 구조체의 자료가 4*n 꼴 혹은 8*n 꼴이 아닌 경우(운영체제에 따라 다르다), 컴파일 시 자동으로 4*n 꼴 혹은 8*n 꼴로 만들어 주는 것을 의미한다. 이를 통해 거의 모든 자료들의 주소가 4의 배수, 8의 배수 꼴로 나타나게 할 수 있다.

위 프로그램에서도 원래 student의 크기는 19byte이지만, 패딩을 통해 5byte가 더해져서 24byte가 되었다.

깃허브 주소: https://github.com/hyaejun-mun/homework3