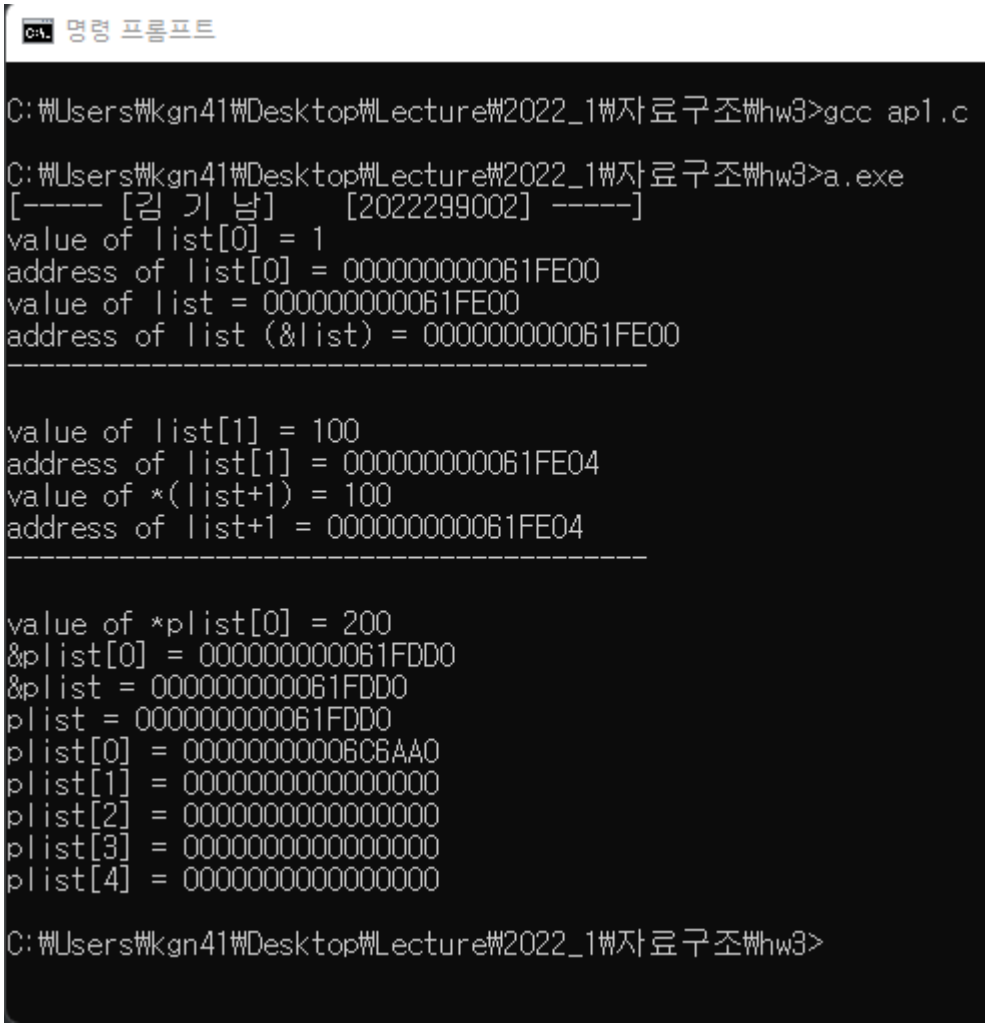
# HW3_김기남_20222299002

# ap1.c




## ❖ Section 1

- Array의 주소는 0번째 인덱스이며, Array에 할당된 값은 0번째 인덱스 주소이다.

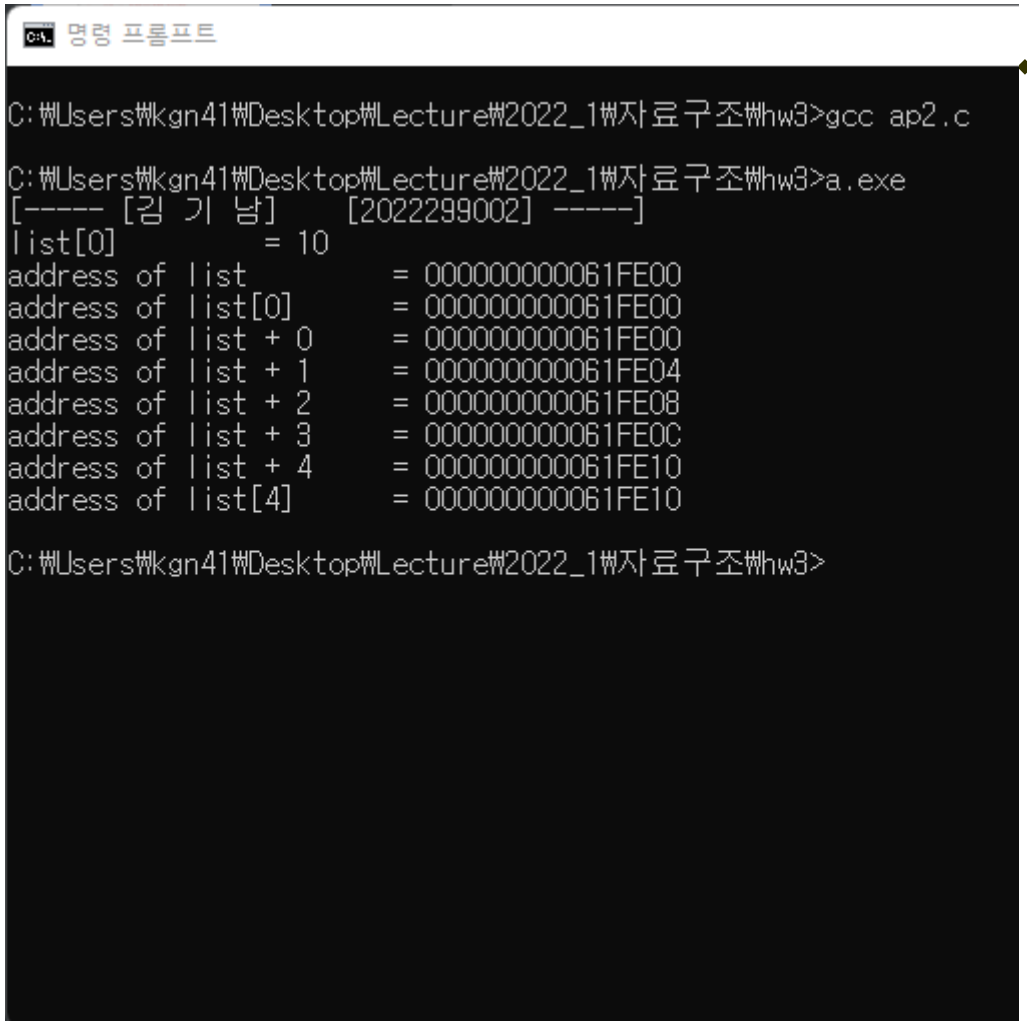
## ❖ Section 2

- Array에 +1, +2 와 같이 더하는 것은 해당 Array의 인덱스를 이동하는 방법이다.
- Array가 int type으로 Initialize되었고, 해당 리스트에 +1을 하게 되면 해당리스트의 현재 주소를 int타입의 크기인 4bytes의 메모리 주소를 이동하는 것이다.

## ❖ Section 3

- plist는 5개의 Element들이 들어갈 수 있게끔 Initialize되었지만, 동적메모리할당이기 때문에, 값이 할당되지 않은 인덱스들은 값이 없으므로, 메모리 주소가 할당되지 않는다.
- 또한 마지막에 free함수를 사용하여 plist[0]에 할당된 메모리를 해제하여 나중에 다시 사용될 수 있게끔 해준다.

# ap2.c




## ❖ Section 1

- Array에 +1, +2 와 같이 더하는 것은 해당 Array의 인덱스를 이동하는 방법이다.
- Array가 int type으로 Initialize되었고, list +1을 하게 되면 해당 리스트의 현재 주소를 int타입의 크기인 4bytes의 메모리 주소를 이동하는 것이다.
- list +0를 하게 되면 int 타입 0개 크기인 0byte의 메모리 주소가 이동한다.
- 따라서, 0byte의 메모리 주소가 이동되는 것은 현재 메모리 주소에 그대로 위치하는 것이다.
- list +2를 하게 되면 int 타입 2개 크기인 8bytes의 메모리 주소가 이동한다.
- list +3를 하게 되면 int 타입 2개 크기인 12bytes의 메모리 주소가 이동한다.
- list +4를 하게 되면 int 타입 2개 크기인 16bytes의 메모리 주소가 이동한다.

# p2-1.c


## ❖ Section 1

- 아래의 정의된 sum함수를 이용하여 list의 각 element들을 더한 값을 return 받을 수 있음.
- 아래의 반복문을 실행하여 input값에 Index 0~99까지에 0~99 값을 할당함.
- 따라서 anser값은 0~99까지 합한 4,950이 할당됨.
- 아래 for 반복문은 main()에서 실행되는 것이고, sum은 다른 위치에서 실행되는 것이기에, main()와 sum()가 가르키는 주소는 서로 다르다.
- sum에서 실행되는 주소값이 main 같으려면 포인터를 이용해야 한다.

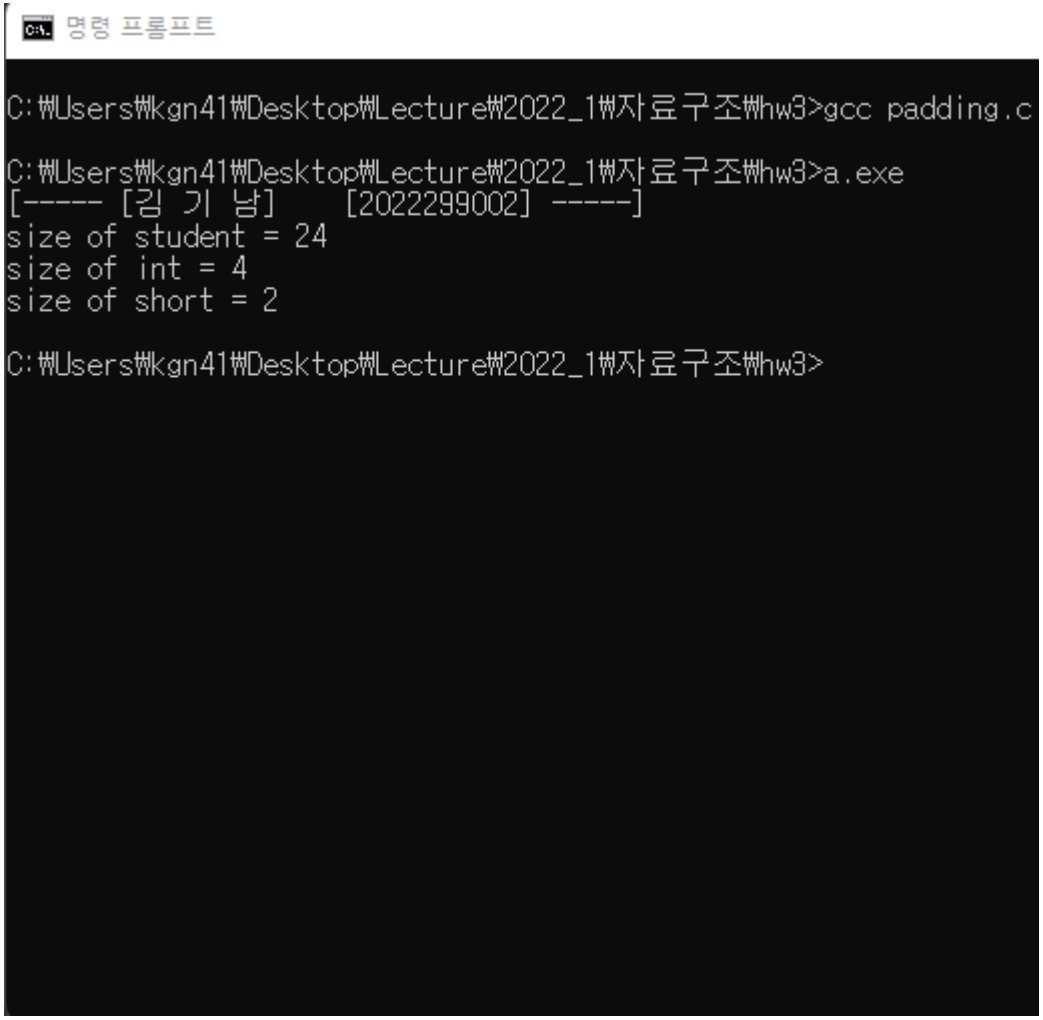
# p2-2.c


## ❖ Section 1

- Array의 주소는 Array의 0번째 인덱스의 주소를 가르키며, Array의 값은 0번째 인덱스의 주소를 가르킴.

- 'one'은 int의 배열로 할당되었기 때문에, 'one' Array를 +1,+2 .. +5 를 할 경우 int의 메모리 크기만큼 주소를 이동한다.
- 아래의 print1 함수를 사용해서 0~5번째 Index까지 이동하며 주소와 할당된 값을 출력한다.
- list +0를 하게 되면 int 타입 0개 크기인 0byte의 메모리 주소가 이동한다.
- 따라서, 0byte의 메모리 주소가 이동되는 것은 현재 메모리 주소에 그대로 위치하는 것이다.
- list +1를 하게 되면 int 타입 2개 크기인 4bytes의 메모리 주소가 이동한다.
- list +2를 하게 되면 int 타입 2개 크기인 8bytes의 메모리 주소가 이동한다.
- list +3를 하게 되면 int 타입 2개 크기인 12bytes의 메모리 주소가 이동한다.
- list +4를 하게 되면 int 타입 2개 크기인 16bytes의 메모리 주소가 이동한다.

# padding.c




## ❖ Section 1

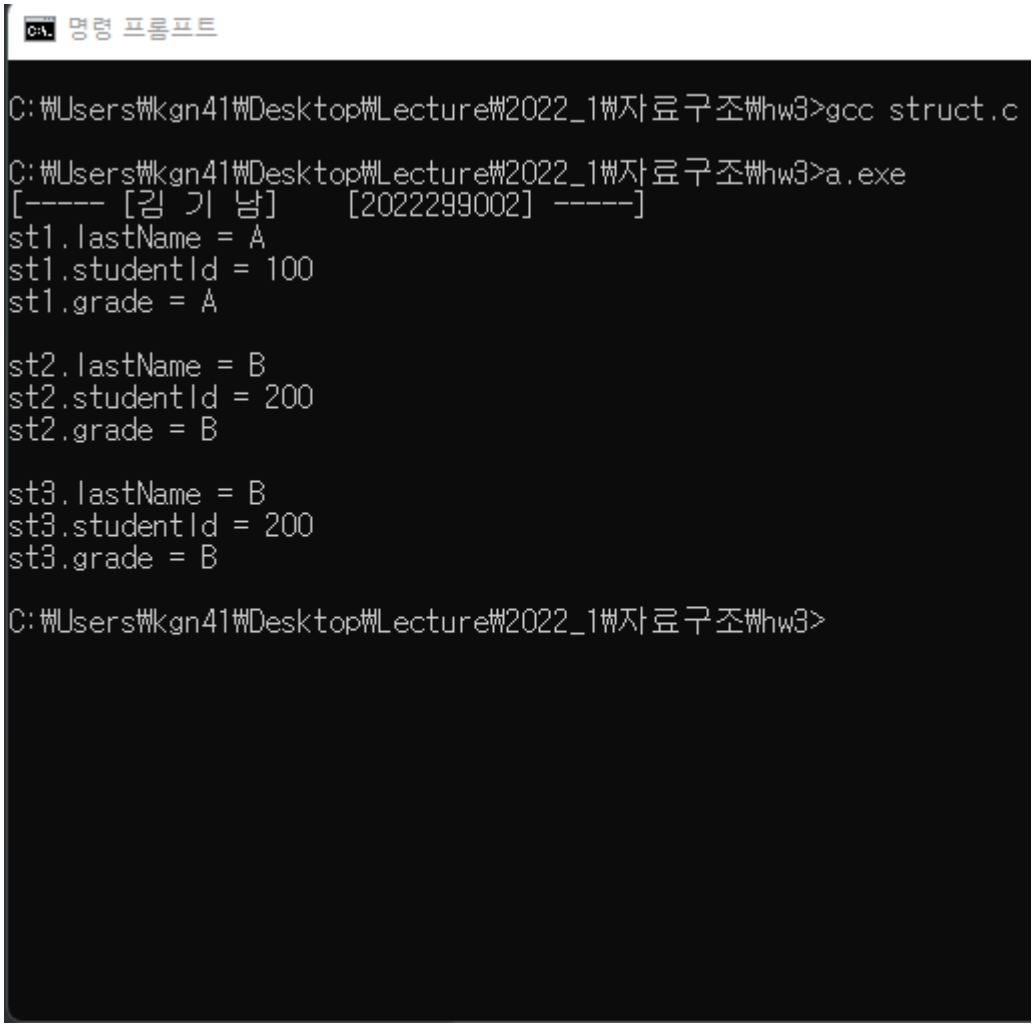
- lastName[13]은 char의 배열, 즉 string이기 때문에 마지막 '₩0'을 포함해서 14bytes이며 int는 4bytes, short은 2byte이므로 총 20btyes이다.
- 하지만 구조체의 특성상 해당 구조체의 Element중에서 가장 큰 자료형의 배수로 정렬되므로, int 자료형의 배수로 결정된다. 따라서 20보다 큰 4의 다음 배수는 24이므로 student 구조체에는 24bytes를 할당한다.

# size.c


## ❖ Section 1

- 포인터는 할당된 값으로 메모리주소 (ex '6422040₩0') 으로 가지고 되며 이러한 메모리주소를 가지게 되기 때문에 포인터 x와 *x의 사이즈는 8bytes이다.
- 최종 참조 값은 int형이기 때문에 4bytes를 가지게 된다.

# struct.c




## ❖ Section 1

- struct "struct name" {}과 typedef struct {} "strcut name" 둘 중 어느것으로도 struct를 만들 수 있다.
- 만들어진 struct를 이용하여 할당할 때는 struct "sturct name" "allocation name" 을 사용하여 할당하거나 "struct name" "allocation name"을 사용하여 할당할 수 있다.
- struct의 Element의 값을 할당할 때는 {} 안에 선언한 Element의 순서대로 작성한다.
- 구조체의 Element를 접근할 때는 "struct name"."Element"로 접근할 수 있다.

# ap1.c

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

void main()
{
    printf("[----- [김 기 남]    [2022299002] -----]\n");

    int list[5];
    int *plist[5] = {NULL,};
    plist[0] = (int *)malloc(sizeof(int));
    list[0] = 1;
    list[1] = 100;
    *plist[0] = 200;

    /* Array의 주소는 0번째 인덱스를 가르키며
    Array에 할당된 값은 0번째 인덱스 주소이다.*/
    printf("value of list[0] = %d\n", list[0]);
    printf("address of list[0] = %p\n", &list[0]);
    printf("value of list = %p\n", list);
    printf("address of list (&list) = %p\n", &list);
    printf("----------------------------------------\n\n");

    /* Array에 +1, +2 와 같이 더하는 것은 해당 Array의 인덱스를 이동하는 방법이다.
    Array가 int type으로 Initialize되었고, 해당 리스트에 +1을 하게 되면
    해당리스트의 현재 주소를 int타입의 크기인 4bytes의 메모리 주소를 이동하는 것이다.*/
    printf("value of list[1] = %d\n", list[1]);
    printf("address of list[1] = %p\n", &list[1]);
    printf("value of *(list+1) = %d\n", *(list + 1));
    printf("address of list+1 = %p\n", list+1);

    /* plist는 5개의 Element들이 들어갈 수 있게끔 Initialize되었지만,
    동적메모리할당이기 때문에, 값이 할당되지 않은 인덱스들은
    값이 없으므로, 메모리 주소가 할당되지 않는다.
    또한 마지막에 free함수를 사용하여 plist[0]에 할당된 메모리를 해제하여
    나중에 다시 사용될 수 있게끔 해준다.
    */
    printf("----------------------------------------\n\n");
    printf("value of *plist[0] = %d\n", *plist[0]);
    printf("&plist[0] = %p\n", &plist[0]);
    printf("&plist = %p\n", &plist);
    printf("plist = %p\n", plist);
    printf("plist[0] = %p\n", plist[0]);
    printf("plist[1] = %p\n", plist[1]);
    printf("plist[2] = %p\n", plist[2]);
    printf("plist[3] = %p\n", plist[3]);
    printf("plist[4] = %p\n", plist[4]);
    free(plist[0]);
}
```

# ap2.c

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

void main()
{
    printf("[----- [김 기 남]    [2022299002] -----]\n");

    int list[5];
    int *plist[5];
    list[0] = 10;
    list[1] = 11;
    plist[0] = (int*)malloc(sizeof(int));

    /* Array에 +1, +2 와 같이 더하는 것은 해당 Array의 인덱스를 이동하는 방법이다.
    Array가 int type으로 Initialize되었고, list +1을 하게 되면
    해당리스트의 현재 주소를 int타입의 크기인 4bytes의 메모리 주소를 이동하는 것이다.
    list +0를 하게 되면 int 타입 0개 크기인 0byte의 메모리 주소가 이동한다.
    따라서, 0byte의 메모리 주소가 이동되는 것은 현재 메모리 주소에 그대로 위치하는 것이다.
    list +2를 하게 되면 int 타입 2개 크기인 8bytes의 메모리 주소가 이동한다.
    list +3를 하게 되면 int 타입 2개 크기인 12bytes의 메모리 주소가 이동한다.
    list +4를 하게 되면 int 타입 2개 크기인 16bytes의 메모리 주소가 이동한다.*/
    printf("list[0] \t= %d\n", list[0]);
    printf("address of list \t= %p\n", list);
    printf("address of list[0] \t= %p\n", &list[0]);
    printf("address of list + 0 \t= %p\n", list+0);
    printf("address of list + 1 \t= %p\n", list+1);
    printf("address of list + 2 \t= %p\n", list+2);
    printf("address of list + 3 \t= %p\n", list+3);
    printf("address of list + 4 \t= %p\n", list+4);
    printf("address of list[4] \t= %p\n", &list[4]);
    free(plist[0]);
}
```

# p2-1.c

```c
#include <stdio.h>
#define MAX_SIZE 100
float sum(float [], int);
float input[MAX_SIZE], answer;
int i;

void main(void)
{
    printf("[----- [김 기 남]    [2022299002] -----]\n");

    /* 아래의 정의된 sum함수를 이용하여 list의 각 element들을 더한 값을 return 받을 수 있음.
       아래의 반복문을 실행하여 input값에 Index 0~99까지에 0~99값을 할당함.
       따라서 anser값은 0~99까지 합한 4,950이 할당됨.

       아래 for 반복문은 main()에서 실행되는 것이고,
       sum은 다른 위치에서 실행되는 것이기에,
       main()와 sum()가 가르키는 주소는 서로 다르다.
       sum에서 실행되는 주소값이 main 같으려면 포인터를 이용해야 한다.*/
    for(i=0; i < MAX_SIZE; i++)
        input[i] = i;
    /* for checking call by reference */
    printf("address of input = %p\n", input);
    answer = sum(input, MAX_SIZE);
    printf("The sum is: %f\n", answer);
}

float sum(float list[], int n)
{
    printf("value of list = %p\n", list);
    printf("address of list = %p\n\n", &list);
    int i;
    float tempsum = 0;
    for(i = 0; i < n; i++)
        tempsum += list[i];
    return tempsum;
}
```

# p2-2.c

```c
#include <stdio.h>
void print1 (int *ptr, int rows);
int main()
{
    printf("[----- [김 기 남]    [2022299002] -----]\n");

    /* Array의 주소는 Array의 0번째 인덱스의 주소를 가르키며,
    Array의 값은 0번째 인덱스의 주소를 가르킴.

    'one'은 int의 배열로 할당되었기 때문에,
    'one' Array를 +1,+2 .. +5 를 할 경우 int의 메모리 크기만큼 주소를 이동한다.
    아래의 print1 함수를 사용해서 0~5번째 Index까지 이동하며 주소와 할당된 값을 출력한다.
    list +0를 하게 되면 int 타입 0개 크기인 0byte의 메모리 주소가 이동한다.
    따라서, 0byte의 메모리 주소가 이동되는 것은 현재 메모리 주소에 그대로 위치하는 것이다.
    list +1를 하게 되면 int 타입 2개 크기인 4bytes의 메모리 주소가 이동한다.
    list +2를 하게 되면 int 타입 2개 크기인 8bytes의 메모리 주소가 이동한다.
    list +3를 하게 되면 int 타입 2개 크기인 12bytes의 메모리 주소가 이동한다.
    list +4를 하게 되면 int 타입 2개 크기인 16bytes의 메모리 주소가 이동한다.*/
    int one[] = {0, 1, 2, 3, 4};
    printf("one = %p\n", one);
    printf("&one = %p\n", &one);
    printf("&one[0] = %p\n", &one[0]);
    printf("\n");
    print1(&one[0], 5);
    return 0;

}
void print1 (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));
    printf("\n");
}
```

# padding.c

```c
#include <stdio.h>
struct student {
    char lastName[13]; /* 13 bytes */
    int studentId; /* 4 bytes */
    short grade; /* 2 bytes */
    };
int main()
{
    printf("[----- [김 기 남]    [2022299002] -----]\n");
    /* lastName[13]은 char의 배열, 즉 string이기 때문에 마지막 '\0'을 포함해서 14bytes이며
    int는 4bytes, short은 2byte이므로 총 20btyes이다.
    하지만 구조체의 특성상 해당 구조체의 Element중에서 가장 큰 자료형의 배수로 정렬되므로
    int 자료형의 배수로 결정된다. 따라서 20보다 큰 4의 다음 배수는 24이므로
    student 구조체에는 24bytes를 할당한다. */
    struct student pst;
    printf("size of student = %ld\n", sizeof(struct student));
    printf("size of int = %ld\n", sizeof(int));
    printf("size of short = %ld\n", sizeof(short));
    return 0;
}
```

# size.c

C size.c

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
    printf("[----- [김 기 남]    [2022299002] -----]\n");
    /*포인터는 할당된 값으로 메모리주소 (ex '6422040\0')으로 가지고 되며
    이러한 메모리주소를 가지게 되기 때문에 포인터 x와 *x의 사이즈는 8bytes이다.
    최종 참조 값은 int형이기 때문에 4bytes를 가지게 된다.*/

    int **x;
    printf("sizeof(x) = %lu\n", sizeof(x));
    printf("sizeof(*x) = %lu\n", sizeof(*x));
    printf("sizeof(**x) = %lu\n", sizeof(**x));
    printf("%d",&x);

}
```

# struct.c

```c
#include <stdio.h>
struct student1 {
    char lastName;
    int studentId;
    char grade;
};
typedef struct {
    char lastName;
    int studentId;
    char grade;
} student2;

int main() {
    printf("[----- [김 기 남]    [2022299002] -----]\n");

    struct student1 st1 = {'A', 100, 'A'};

    /* struct "struct name" {}과
    typedef struct {} "strcut name" 둘 중 어느것으로 struct를 만들 수 있다.

    만들어진 struct를 이용하여 할당할때는
    struct "sturct name" "allocation name" 을 사용하여 할당하거나
    "struct name" "allocation name"을 사용하여 할당할 수 있다.

    struct의 Element의 값을 할당할때는 {} 안에 선언한 Element의 순서대로 작성한다.

    구조체의 Element를 접근할때는 "struct name"."Element"로 접근할 수 있다.*/
    printf("st1.lastName = %c\n", st1.lastName);
    printf("st1.studentId = %d\n", st1.studentId);
    printf("st1.grade = %c\n", st1.grade);


    student2 st2 = {'B', 200, 'B'};

    printf("\nst2.lastName = %c\n", st2.lastName);
    printf("st2.studentId = %d\n", st2.studentId);
    printf("st2.grade = %c\n", st2.grade);

    /* strcut A를 B에 할당할 수 있지만,
    struct끼리의 연산자(==, +, -, *, / 등) 사용은 불가능하다.*/
    student2 st3;

    st3 = st2;

    printf("\nst3.lastName = %c\n", st3.lastName);
    printf("st3.studentId = %d\n", st3.studentId);
    printf("st3.grade = %c\n", st3.grade);

    // /* equality test */
    // if(st3 == st2) /* not working */
    //     printf("equal\n");
    // else
    //     printf("not equal\n");
    return 0;
}
```