

HW3_김기남_2022299002



ap1.c

C:\> 명령 프롬프트

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc ap1.c
```

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
```

```
[----- [김 기 남] [2022299002] -----]  
value of list[0] = 1  
address of list[0] = 000000000061FE00  
value of list = 000000000061FE00  
address of list (&list) = 000000000061FE00  
-----
```

```
value of list[1] = 100  
address of list[1] = 000000000061FE04  
value of *(list+1) = 100  
address of list+1 = 000000000061FE04  
-----
```

```
value of *plist[0] = 200  
&plist[0] = 000000000061FDD0  
&plist = 000000000061FDD0  
plist = 000000000061FDD0  
plist[0] = 00000000006C6AA0  
plist[1] = 0000000000000000  
plist[2] = 0000000000000000  
plist[3] = 0000000000000000  
plist[4] = 0000000000000000
```

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ 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, 동적메모리할당이기에 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

```
CA 명령 프롬프트
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc ap2.c
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
[----- [김 기 남] [2022299002] -----]
list[0] = 10
address of list = 000000000061FE00
address of list[0] = 000000000061FE00
address of list + 0 = 000000000061FE00
address of list + 1 = 000000000061FE04
address of list + 2 = 000000000061FE08
address of list + 3 = 000000000061FE0C
address of list + 4 = 000000000061FE10
address of list[4] = 000000000061FE10
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ 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 3개 크기인 12bytes의 메모리 주소가 이동한다.
- list +4를 하게 되면 int 타입 4개 크기인 16bytes의 메모리 주소가 이동한다.

p2-1.c

CA. 명령 프롬프트

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc p2-1.c
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
[----- [김 기 남] [2022299002] -----]
address of input = 0000000000407980
value of list = 0000000000407980
address of list = 000000000061FE00

The sum is: 4950.000000

C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ 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

C:\ 명령 프롬프트

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc p2-2.c
```

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
```

```
[----- [김 기 남] [2022299002] -----]
```

```
one = 00000000061FE00
```

```
&one = 00000000061FE00
```

```
&one[0] = 00000000061FE00
```

Address	Contents
00000000061FE00	0
00000000061FE04	1
00000000061FE08	2
00000000061FE0C	3
00000000061FE10	4

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ 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

명령 프롬프트

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc padding.c
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
[----- [김 기 남] [2022299002] -----]
size of student = 24
size of int = 4
size of short = 2
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ Section 1

- lastName[13]은 char의 배열, 즉 string이기 때문에 마지막 '\0'을 포함해서 14bytes이며 int는 4bytes, short은 2byte이므로 총 20bytes이다.
- 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

명령 프롬프트

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc size.c
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
[----- [김 기 남] [2022299002] -----]
sizeof(x) = 8
sizeof(*x) = 8
sizeof(**x) = 4
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ Section 1

- 포인터는 할당된 값으로 메모리주소 (ex '6422040W0')
으로 가지고 되며 이러한 메모리주소를 가지게 되기
때문에 포인터 x와 *x의 사이즈는 8bytes이다.
- 최종 참조 값은 int형이기 때문에 4bytes를 가지게 된
다.

struct.c

CA 명령 프롬프트

```
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>gcc struct.c
C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>a.exe
[----- [김 기 남] [2022299002] -----]
st1.lastName = A
st1.studentId = 100
st1.grade = A

st2.lastName = B
st2.studentId = 200
st2.grade = B

st3.lastName = B
st3.studentId = 200
st3.grade = B

C:\Users\kgn41\Desktop\Lecture\2022_1\자료구조\hw3>
```

❖ Section 1

- struct "struct name" {}과 typedef struct {} "struct name" 둘 중 어느것으로도 struct를 만들 수 있다.
- 만들어진 struct를 이용하여 할당할 때는 struct "struct 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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      printf("[----- [김 기 남]    [2022299002] -----]\n");
7
8      int list[5];
9      int *plist[5] = {NULL,};
10     plist[0] = (int *)malloc(sizeof(int));
11     list[0] = 1;
12     list[1] = 100;
13     *plist[0] = 200;
14
15     /* Array의 주소는 0번째 인덱스를 가르키며
16     Array에 할당된 값은 0번째 인덱스 주소이다.*/
17     printf("value of list[0] = %d\n", list[0]);
18     printf("address of list[0] = %p\n", &list[0]);
19     printf("value of list = %p\n", list);
20     printf("address of list (&list) = %p\n", &list);
21     printf("-----\n\n");
22
23     /* Array에 +1, +2 와 같이 더하는 것은 해당 Array의 인덱스를 이동하는 방법이다.
24     Array가 int type으로 Initialize되었고, 해당 리스트에 +1을 하게 되면
25     해당리스트의 현재 주소를 int타입의 크기인 4bytes의 메모리 주소를 이동하는 것이다.*/
26     printf("value of list[1] = %d\n", list[1]);
27     printf("address of list[1] = %p\n", &list[1]);
28     printf("value of *(list+1) = %d\n", *(list + 1));
29     printf("address of list+1 = %p\n", list+1);
30
31     /* plist는 5개의 Element들이 들어갈 수 있게끔 Initialize되었지만,
32     동적메모리할당이기에 때문에, 값이 할당되지 않은 인덱스들은
33     값이 없으므로, 메모리 주소가 할당되지 않는다.
34     또한 마지막에 free함수를 사용하여 plist[0]에 할당된 메모리를 해제하여
35     나중에 다시 사용할 수 있게끔 해준다.
36     */
37     printf("-----\n\n");
38     printf("value of *plist[0] = %d\n", *plist[0]);
39     printf("&plist[0] = %p\n", &plist[0]);
40     printf("&plist = %p\n", &plist);
41     printf("plist = %p\n", plist);
42     printf("plist[0] = %p\n", plist[0]);
43     printf("plist[1] = %p\n", plist[1]);
44     printf("plist[2] = %p\n", plist[2]);
45     printf("plist[3] = %p\n", plist[3]);
46     printf("plist[4] = %p\n", plist[4]);
47     free(plist[0]);
48 }
```

ap2.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      printf("[----- [김 기 남]      [2022299002] -----]\n");
7
8      int list[5];
9      int *plist[5];
10     list[0] = 10;
11     list[1] = 11;
12     plist[0] = (int*)malloc(sizeof(int));
13
14     /* Array에 +1, +2 와 같이 더하는 것은 해당 Array의 인덱스를 이동하는 방법이다.
15     Array가 int type으로 Initialize되었고, list +1을 하게 되면
16     해당리스트의 현재 주소를 int타입의 크기인 4bytes의 메모리 주소를 이동하는 것이다.
17     list +0를 하게 되면 int 타입 0개 크기인 0byte의 메모리 주소가 이동한다.
18     따라서, 0byte의 메모리 주소가 이동되는 것은 현재 메모리 주소에 그대로 위치하는 것이다.
19     list +2를 하게 되면 int 타입 2개 크기인 8bytes의 메모리 주소가 이동한다.
20     list +3를 하게 되면 int 타입 3개 크기인 12bytes의 메모리 주소가 이동한다.
21     list +4를 하게 되면 int 타입 4개 크기인 16bytes의 메모리 주소가 이동한다.*/
22     printf("list[0] \t= %d\n", list[0]);
23     printf("address of list \t= %p\n", list);
24     printf("address of list[0] \t= %p\n", &list[0]);
25     printf("address of list + 0 \t= %p\n", list+0);
26     printf("address of list + 1 \t= %p\n", list+1);
27     printf("address of list + 2 \t= %p\n", list+2);
28     printf("address of list + 3 \t= %p\n", list+3);
29     printf("address of list + 4 \t= %p\n", list+4);
30     printf("address of list[4] \t= %p\n", &list[4]);
31     free(plist[0]);
32 }
```

p2-1.c

```
1  #include <stdio.h>
2  #define MAX_SIZE 100
3  float sum(float [], int);
4  float input[MAX_SIZE], answer;
5  int i;
6
7  void main(void)
8  {
9      printf("----- [김 기 남]    [2022299002] -----]\n");
10
11      /* 아래의 정의된 sum함수를 이용하여 list의 각 element들을 더한 값을 return 받을 수 있음.
12       아래의 반복문을 실행하여 input값에 Index 0~99까지에 0~99값을 할당함.
13       따라서 answer값은 0~99까지 합한 4,950이 할당됨.
14
15       아래 for 반복문은 main()에서 실행되는 것이고,
16       sum은 다른 위치에서 실행되는 것이기에,
17       main()와 sum()가 가르키는 주소는 서로 다르다.
18       sum에서 실행되는 주소값이 main 같으려면 포인터를 이용해야 한다.*/
19      for(i=0; i < MAX_SIZE; i++)
20          input[i] = i;
21      /* for checking call by reference */
22      printf("address of input = %p\n", input);
23      answer = sum(input, MAX_SIZE);
24      printf("The sum is: %f\n", answer);
25  }
26
27  float sum(float list[], int n)
28  {
29      printf("value of list = %p\n", list);
30      printf("address of list = %p\n\n", &list);
31      int i;
32      float tempsum = 0;
33      for(i = 0; i < n; i++)
34          tempsum += list[i];
35      return tempsum;
36  }
```

p2-2.c

```
1  #include <stdio.h>
2  void print1 (int *ptr, int rows);
3  int main()
4  {
5      printf("----- [김 기 남]      [2022299002] -----]\n");
6
7      /* Array의 주소는 Array의 0번째 인덱스의 주소를 가르키며,
8      Array의 값은 0번째 인덱스의 주소를 가르킴.
9
10     'one'은 int의 배열로 할당되었기 때문에,
11     'one' Array를 +1,+2 .. +5 를 할 경우 int의 메모리 크기만큼 주소를 이동한다.
12     아래의 print1 함수를 사용해서 0~5번째 Index까지 이동하며 주소와 할당된 값을 출력한다.
13     list +0를 하게 되면 int 타입 0개 크기인 0byte의 메모리 주소가 이동한다.
14     따라서, 0byte의 메모리 주소가 이동되는 것은 현재 메모리 주소에 그대로 위치하는 것이다.
15     list +1를 하게 되면 int 타입 2개 크기인 4bytes의 메모리 주소가 이동한다.
16     list +2를 하게 되면 int 타입 2개 크기인 8bytes의 메모리 주소가 이동한다.
17     list +3를 하게 되면 int 타입 2개 크기인 12bytes의 메모리 주소가 이동한다.
18     list +4를 하게 되면 int 타입 2개 크기인 16bytes의 메모리 주소가 이동한다.*/
19     int one[] = {0, 1, 2, 3, 4};
20     printf("one = %p\n", one);
21     printf("&one = %p\n", &one);
22     printf("&one[0] = %p\n", &one[0]);
23     printf("\n");
24     print1(&one[0], 5);
25     return 0;
26 }
27
28 void print1 (int *ptr, int rows)
29 { /* print out a one-dimensional array using a pointer */
30     int i;
31     printf ("Address \t Contents\n");
32     for (i = 0; i < rows; i++)
33         printf("%p \t %5d\n", ptr + i, *(ptr + i));
34     printf("\n");
35 }
```

padding.c

```
1  #include <stdio.h>
2  struct student {
3      char lastName[13]; /* 13 bytes */
4      int studentId; /* 4 bytes */
5      short grade; /* 2 bytes */
6  };
7  int main()
8  {
9      printf("[----- [김 기 남]      [2022299002] -----]\n");
10     /* lastName[13]은 char의 배열, 즉 string이기 때문에 마지막 '\0'을 포함해서 14bytes이며
11     int는 4bytes, short은 2byte이므로 총 20bytes이다.
12     하지만 구조체의 특성상 해당 구조체의 element중에서 가장 큰 자료형의 배수로 정렬되므로
13     int 자료형의 배수로 결정된다. 따라서 20보다 큰 4의 다음 배수는 24이므로
14     student 구조체에는 24bytes를 할당한다. */
15     struct student pst;
16     printf("size of student = %ld\n", sizeof(struct student));
17     printf("size of int = %ld\n", sizeof(int));
18     printf("size of short = %ld\n", sizeof(short));
19     return 0;
20 }
```

size.c

C size.c

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  void main()
4  {
5      printf("[----- [김 기 남]      [2022299002] -----]\n");
6      /*포인터는 할당된 값으로 메모리주소 (ex '6422040\0')으로 가지고 되며
7      이러한 메모리주소를 가지게 되기 때문에 포인터 x와 *x의 사이즈는 8bytes이다.
8      최종 참조 값은 int형이기 때문에 4bytes를 가지게 된다.*/
9
10     int **x;
11     printf("sizeof(x) = %lu\n", sizeof(x));
12     printf("sizeof(*x) = %lu\n", sizeof(*x));
13     printf("sizeof(**x) = %lu\n", sizeof(**x));
14     printf("%d",&x);
15
16 }
```

struct.c

```
1 #include <stdio.h>
2 struct student1 {
3     char lastName;
4     int studentId;
5     char grade;
6 };
7 typedef struct {
8     char lastName;
9     int studentId;
10    char grade;
11 } student2;
12
13 int main() {
14     printf("[----- [김 기 남]    [2022299002] -----]\n");
15
16     struct student1 st1 = {'A', 100, 'A'};
17
18     /* struct "struct name" {}과
19     typedef struct {} "strcut name" 둘 중 어느것으로 struct를 만들 수 있다.
20
21     만들어진 struct를 이용하여 할당할때는
22     struct "sturct name" "allocation name" 을 사용하여 할당하거나
23     "struct name" "allocation name"을 사용하여 할당할 수 있다.
24
25     struct의 Element의 값을 할당할때는 {} 안에 선언한 Element의 순서대로 작성한다.
26
27     구조체의 Element를 접근할때는 "struct name"."Element"로 접근할 수 있다.*/
28     printf("st1.lastName = %c\n", st1.lastName);
29     printf("st1.studentId = %d\n", st1.studentId);
30     printf("st1.grade = %c\n", st1.grade);
31
32
33     student2 st2 = {'B', 200, 'B'};
34
35     printf("\nst2.lastName = %c\n", st2.lastName);
36     printf("st2.studentId = %d\n", st2.studentId);
37     printf("st2.grade = %c\n", st2.grade);
38
39     /* strcut A를 B에 할당할 수 있지만,
40     struct끼리의 연산자(==, +, -, *, / 등) 사용은 불가능하다.*/
41     student2 st3;
42
43     st3 = st2;
44
45     printf("\nst3.lastName = %c\n", st3.lastName);
46     printf("st3.studentId = %d\n", st3.studentId);
47     printf("st3.grade = %c\n", st3.grade);
48
49     // /* equality test */
50     // if(st3 == st2) /* not working */
51     //     printf("equal\n");
52     // else
53     //     printf("not equal\n");
54     return 0;
55 }
```