

데이터구조 레포트 1

학년	2학년	학번	20211379
이름	장원영	제출일	2024.04.14

일체의 부정 행위 없이 스스로의 노력으로 작성한 레포트임을 확인합니다.

문제 1

문제 1.(1)

1.(1) 코드 및 주석

```
#include <stdio.h>

int main(){
    //문제의 조건인 12 * int자료형의 크기를 가지는 정수형 배열을 생성하고 문제의 조건값을
    넣어서 초기화
    int day[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    //for문을 이용하여 1월부터 12월까지 출력
    for(int i = 0; i < 12; i++){
        printf("%d월은 %d일까지 있습니다.\n", i+1, day[i]);
    }

    // int main이므로 0을 반환하며 종료
    return 0;
}
```

1.(1) 출력 화면 및 설명

```

cd "/Users/zwonyoung/Desktop/programing/datastrutu
zwonyoung@jang-won-yeong-ui-MacBookAir programing
_1
1월은 31일까지 있습니다.
2월은 29일까지 있습니다.
3월은 31일까지 있습니다.
4월은 30일까지 있습니다.
5월은 31일까지 있습니다.
6월은 30일까지 있습니다.
7월은 31일까지 있습니다.
8월은 31일까지 있습니다.
9월은 30일까지 있습니다.
10월은 31일까지 있습니다.
11월은 30일까지 있습니다.
12월은 31일까지 있습니다.
zwonyoung@jang-won-yeong-ui-MacBookAir Q1 %

```

위 코드를 보면 알수 있듯이, 배열의 요소는 '배열명[i]' 를 사용하여 i번째 요소를 출력할 수 있음을 알 수 있으며, 변수 i를 통한 반복문을 통해 '월'을 배열 요소의 '인덱스'로 1~12번째 요소까지, 그리고 '일'을 배열 요소의 '값'으로 접근하여 풀 것을 알 수 있습니다.

문제 1.(2)

1.(2) 코드 및 주석

```

#include <stdio.h>

//아래 main 단에서 넘겨받은 주소값을
//포인터 변수 a, b, c에 각각 저장합니다.
//이로서 포인터를 통한 배열 요소의 접근이 가능해집니다.
void diff(int *a, int *b, int *c){
    // 각각의 i번째 배열 요소끼리 문제 조건에 맞게 뺄셈 후
    //c의 i번째 요소에 값을 넣어주었습니다.
    for(int i = 0; i < 10; i++){
        c[i] = a[i] - b[i];
    }
}

int main(){
    //문제에서 주어진 크기만큼의 배열을 선언하고 값은 제가 임의로 집어넣었습니다.
    int a[10] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int b[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    //문제 조건인 입력받을 배열입니다.
    int c[10];

```

```

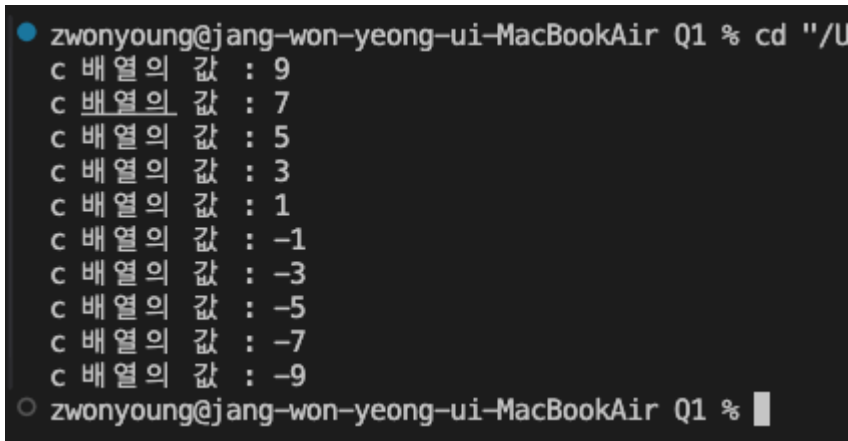
//배열 a 와 b의 차를 구해 배열 c에 저장하는 함수 호출
//이때 배열의 처음 시작 주소를 넘겨 간접 참조를 진행하도록 하였습니다.
//배열의 이름은 그 자체로 배열의 처음 시작 주소를 의미하기도 합니다.
diff(a, b, c);

for(int i =0; i < 10; i++){
    printf("c 배열의 값 : %d\n", c[i]);
    // c배열의 값을 배열의 크기만큼 순차적으로 순회하면서 출력
}

// 0을 반환하며 종료
return 0;
}

```

1.(2) 출력 화면 및 설명



```

zwonyoung@jang-won-yeong-ui-MacBookAir Q1 % cd "/U
c 배열의 값 : 9
c 배열의 값 : 7
c 배열의 값 : 5
c 배열의 값 : 3
c 배열의 값 : 1
c 배열의 값 : -1
c 배열의 값 : -3
c 배열의 값 : -5
c 배열의 값 : -7
c 배열의 값 : -9
zwonyoung@jang-won-yeong-ui-MacBookAir Q1 %

```

위 코드를 보면 알 수 있듯이 문제의 조건인 임의의 값으로 초기화한 int 10개만큼의 크기를 가지는 배열 a, b와 그 값을 받을 같은 크기의 배열 c를 선언하였고, 문제의 연산을 진행하기 위해 이 배열의 이름들을 넘긴 것을 확인할 수 있습니다.

이때, 배열을 넘길때 배열의 '주소'를 넘겨서 포인터를 통한 값의 조작을 할 수 있음을 알 수 있습니다.

그리고 함수의 인수로 a, b, c와 같이 배열의 이름을 넘겨주었는데 배열의 이름은 그 자체로 배열을 의미하기도 하지만 기본적으로는 배열의 처음 시작 주소를 가지고 있기에 배열의 이름을 파라미터에 인수로 넘기고 주소를 받았으므로 함수 diff의 파라미터도 int* 즉 '정수형 포인터'로 선언하여 포인터 값을 받은 것을 알 수 있습니다. 이후 이 함수는 그 포인터를 이용해 '파라미터명[i]' 문법을 통해 해당 배열의 i번째 값을 지정하여 읽어들이고,

$c[i] = a[i] - b[i]$; 와 같은 연산을 for문을 통해 10 번 반복하여 함수가 직접 c배열을 수정하고 있는 모습을 볼 수 있습니다. 직접 수정하므로 값을 반환할 필요가 없어 void로 선언해준 모습입니다.

문제 1.(3)

1.(3) 코드 및 주석

```
#include<stdio.h>
#include<stdlib.h> //rand 함수 사용을 위한 헤더파일

int main(){
    // random 숫자를 채워넣을 정수 배열 선언
    int input[10];

    // 배열의 첫번째부터 열번째 요소까지 i번째 인덱스 안의 값에
    // rand함수를 통해 임의로 생성된 무작위 수를 넣어줍니다.
    for(int i = 0; i < 10; i++){
        input[i] = rand();
        //값이 제대로 할당되었는지 확인하는 용도의 출력입니다.
        printf("%d\n", input[i]);
    }
    // 배열 내에서의 비교를 위해 임의의 값을 주는데
    // 배열 '내'의 아무 값을 주어 대소비교가 제대로 이루어지게 하였습니다.
    int max = input[0];
    int min = input[0];

    for(int i = 0; i < 10; i++){
        //max와 min을 찾는 반복문 각 배열 요소의 값을 if문을 통해서
        //max로서 한번 min으로서 한번 비교하고 해당 값이
        //1. min보다 작으면
        //2. max보다 크면
        //해당 변수값에 현재 배열 요소값을 집어넣는 알고리즘입니다.

        int value = input[i];// 굳이 value로 값을 넣어준 이유는 중복 연산을 줄
        //이기 위함입니다.

        if(min > value){
            min = value;
        }
        if(max < value){
            max = value;
        }
    }
    // 알고리즘 종료 후의 max값과 min 값을 출력합니다.
    printf("max : %d, min : %d\n", max, min);

    // 0을 반환하며 종료
    return 0;
}
```

1.(3) 출력 화면 및 설명

```
● zwonyoung@jang-won-yeong-ui-MacBookAir Q1 % cd "/U
16807
282475249
1622650073
984943658
1144108930
470211272
101027544
1457850878
1458777923
2007237709
max : 2007237709, min : 16807
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q1 % □
```

문제 조건에 맞게 크기가 10인 배열을 선언해주고 반복문을 통해 1~10번째까지의 배열의 '인덱스'가 가리키는 '값'에 rand함수(반환값은 랜덤한 정수)를 통한 랜덤값을 넣어주었고 그 값이 제대로 할당되었는지 매 분기마다 출력해 주었습니다.

이후 최댓값을 저장할 max, 최솟값을 저장할 min을 선언하고 각각에 배열의 첫번째 요소를 할당해 주었는데, 이렇게 해준 이유로는,

다른 임의의 값을 할당해 버리면 실제 랜덤한 값의 최솟값보다 작은, 최댓값보다 큰 수가 할당될 가능성이 있으므로 배열의 임의의 요소를 할당함으로서 이러한 사고를 방지하고자 필자는 배열의 첫번째 요소를 할당하였습니다.

이후 if문을 2번 사용하여 비교하여

1. 그 인덱스의 랜덤값이 min보다 작으면 현재값을 min에 집어넣고
2. 그 인덱스의 랜덤 값이 max보다 크면 현재값을 max에 집어넣고

이 로직을 배열의 길이인 10번동안 반복하여 그때마다의 i값을 배열의 인덱스로 넣어주었고, 마지막으로 max와 min의 값을 출력하여 제대로 작동되었음을 확인하였습니다.

문제 1.(4)

1.(4) 코드 및 주석

```
#include <stdio.h>

// 순서대로 파라미터를 이렇게 받은 이유를 설명하자면
// 1. 배열의 이름(배열의 시작 주소)를 받았으므로 int *형 변수로 받아 포인터로 배열 요소 참조를
//    가능하게 한다.
// 2. 사용자의 입력 값은 그냥 정수형 '값'을 받았습니다.
// 3. 2중 포인터로 받은 이유는 wantp가 포인터 변수인데 이 값을 이 함수가 '직접' 바꿔주어야 하므로
```

포인터 변수를 참조하는 포인터 변수의 의미로서 2중 포인터를 사용하였습니다.

```
void search(int *arr, int input, int** wantp){
    // 배열의 1~10번째 요소까지 스캔
    for(int i = 0; i < 10; i++){
        // 배열의 값이 사용자가 입력한 값과 같으면
        if(arr[i] == input){
            // 해당 인덱스의 주소를 기억할 wantp를 그 인덱스의 주소값으로 변환
            *wantp = &arr[i];
            // 반환값 없이 함수 종료
            return;
        }
    }
}

int main(){
    // 문제에서 주어진 배열
    int iArr[10] = {23, 45, 12, 34, 65, 25, 89, 61, 26, 11};
    // 사용자가 찾고자 하는 값 입력받기
    int input;
    scanf("%d",&input);
    // 위치를 찾았을때 반환할 변수 output 선언
    int output;
    // 찾고자 하는 배열의 위치를 저장하기 위한 포인터 변수
    int* wantp = NULL;
    // 사용자가 입력한 값이 배열내의 몇번째 요소로서 존재하는지 반환할 함수 호출
    // 파라미터로는 배열의 주소, 입력값, 배열의 위치를 저장하는 포인터 변수를 전달
    search(iArr, input, &wantp);
    if(wantp == NULL){
        // 반환한 값이 없으면 wantp가 null이므로 찾는 값이 없다고 출력
        printf("찾는 값이 없습니다...");
    }
    else{
        // wantp가 null값이 아니면 일치하는 값이 있는 것이므로
        // 배열의 처음 시작 주소인 iArr을 빼주어 ?번째 인덱스임을 정수로 표현
        output = wantp - iArr;
        // 이후 그 값을 출력
        printf("위치는 : %d입니다.", output);
    }
    return 0;
}
```

1.(4) 출력 화면 및 설명

```
cd "/Users/zwonyoung/Desktop/programing/datastruture/
e/code/Q1/"q1_4
```

```
● zwonyoung@jang-won-yeong-ui-MacBookAir programing %  
&& "/Users/zwonyoung/Desktop/programing/datastruture/
45
```

```
위 치 는 : 1입 니 다 .%
```

```
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q1 % █
```

```
● zwonyoung@jang-won-yeong-ui-MacBookAir Q1 % cd "/Use
ers/zwonyoung/Desktop/programing/datastruture/code/Q
17
```

```
찾 는 값 이 없 습 니 다 ...%
```

```
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q1 % █
```

문제의 조건에 맞게 주어진 요소를 저장할 배열 iArr을 선언하였고, 사용자의 입력값을 받을 정수형 변수 input을 선언하여 input에 사용자의 값을 입력받았습니다.

이후 위치를 찾았을때 반환할 변수 output과 찾고자 하는 배열의 위치를 저장할 '포인터' 변수 wantp를 선언하였고 당장은 아무것도 가리키지 않을 것이므로 안전하게 null 값을 넣어주었습니다.

이후 사용자가 입력한 값이 iArr에 들어있는 값인지, 있다면 그 배열의 인덱스를 반환하는 함수 search에 문제의 조건에 맞게 배열의 시작 주소(iArr), 사용자가 입력한 정수 값(input), 인덱스 위치 주소를 저장할 포인터 변수의 '주소' (&wantp)를 넘겨주었습니다.

이후 받은 파라미터 변수의 값에 대해서 설명하자면

1. 넘겨준 값이 배열의 시작 주소, 즉 주소값이므로 포인터 변수 arr로 받아 배열의 값에 접근할 수 있게 하였고
2. 사용자가 입력한 정수값은 그냥 값 자체를 넘겨주었으며
3. 여기가 매우 중요한데 넘겨받은 변수 wantp는 해당하는 배열 요소의 '주소' 값에 접근하는 '포인터' 변수입니다.

이 값을 직접 수정할 것이므로 이 '포인터' 변수의 '주소'를 넘겨주었는데 일단은 '주소'로 받았으므로 *로 받고 포인터 변수를 받았으므로 *를 하나 더 써서 이중 포인터로서 값을 받은 모습입니다.

'포인터 변수'의 주소를 가리키는 포인터 이므로 이중 포인터인것입니다.

이런 식으로 받은 값을 통해 배열의 1번째 요소부터 10번째 요소까지 비교하기 위한 반복문을 돌리고 이때의 i값을 인덱스에 넣어 'arr[i]가 == input'이라면

즉, 배열의 요소가 사용자가 입력한 값과 같다면 그 배열 요소의 주소를 의미하는 &arr[i] 을 포인터 변수 wantp에 저장하는데 *와 같이 사용했으므로 파라미터로 받은 '포인터 배열'의 '주소값'을 참조하여 그 주소값이 가리키는 포인터 변수의 값으로서 집어넣습니다.

이러면 더 탐색할 필요가 없으므로 그냥 return을 하여 함수를 종료시키고 이때의 정상적으로 반영된 wantp는 주소값을 가지고 있게 되는데, if문을 통해 wantp가 여전히 null값이라면 해당 배열에 사용자가 입력한 값이 존재하지 않는 것이므로 그대로 찾는 값이 없다는 내용을 printf를 통해 출력합니다.

만약에 있다면 이것을 그냥 그대로 출력하면 주소값을 얻는 것이지 몇번째 요소인지는 알기 힘들기 때문에 iArr이 배열의 처음 시작 주소로 해석이 가능하므로 wantp(iArr + ?)로 해석이 가능합니다.

따라서 wantp - iArr을 해주면 배열의 인덱스를 얻을 수 있고 이것을 정수형 변수 output에 집어넣은 후 printf를 통해 그 배열의 인덱스를 출력해주어 해결이 가능한 모습을 볼 수 있습니다.

문제 1.(5)

1.(5) 코드 및 주석

```
#include <stdio.h>

#define LIST_SIZE 10
int list[LIST_SIZE] = { 23, 45, 12, 34, 65, 25, 89, 61, 26, 11 };

int main(){
    int i; // 반복문을 위한 변수 i 선언
    //0-9까지의 값을 집어넣어 10번 반복문을 돌리고 있다
    //해당 값의 주소를 반환한다.
    for(i = 0; i < LIST_SIZE; i++){
        printf("list[%d]의 주소 = %p\n", i, &list[i]);
    }
}

// int 형의 크기가 4이므로 주소도 4칸씩 띄어서 읽는다.
```

1.(5) 출력 화면 및 설명


```
ers/zwonyoung/Desktop/programing/datastruture/c
list[0]의 주소 = 0x1028f4000
list[1]의 주소 = 0x1028f4004
list[2]의 주소 = 0x1028f4008
list[3]의 주소 = 0x1028f400c
list[4]의 주소 = 0x1028f4010
list[5]의 주소 = 0x1028f4014
list[6]의 주소 = 0x1028f4018
list[7]의 주소 = 0x1028f401c
list[8]의 주소 = 0x1028f4020
list[9]의 주소 = 0x1028f4024
zwonyoung@jang-won-yeong-ui-MacBookAir Q1 %
```

이 문제는 포인터와 데이터형 메모리 주소의 관계를 알 수 있는 문제입니다.

로직 자체는 매우 별볼일 없는데, 그냥 주어진 배열을 for문을 통해 list[i]와 같이 순회하면서 그 요소의 주소값을 반환하여 출력하는 로직입니다.

메모리의 주소는 기본적으로 16진수로 저장됩니다. 0 1 2 3 4 5 6 7 8 9 a b c d e f 로 1자리수를 표현합니다.

이때 메모리의 끝 자리수가 4씩 떨어져 있는 것을 확인할 수 있는데, 배열은 순차적으로 이어져 있는 데이터의 집합이며, 정수형 int 데이터 1개의 크기가 4바이트인 점을 감안했을때,

메모리 주소 1개가 1바이트를 저장할 수 있고 4바이트짜리 데이터를 10개 쪽 이어놓았기에 4씩 규칙적으로 올라가는 것을 확인할 수 있고

실제 주소값은 시작 주소값 + (int데이터의 크기 * 찾고자 하는 인덱스 순서)임을 알 수 있습니다.

문제 1.(6)

1.(6) 코드 및 주석

```
#include <stdio.h>

void printArrayAddress(int [][][3]); // 사용하기 전 입력받을 배열의 형태를 지정해준다
void main(void){

    //2행 3열의 2차원 배열 array를 선언하고 1행에는 1, 2, 3 2행에는 4, 5, 6을 할당하
```

였습니다.

```
int array[2][3] = {{1, 2, 3}, {4, 5, 6}};
printf("배열 array의 출력 : \n");
// 위에서 정의해준
printArrayAddress(array);
}

void printArrayAddress(int a[][3]){
    int i, j;

    for(i = 0; i <= 1; i++) {
        for(j = 0; j <= 2; j++){
            printf("%p ", &a[i][j]);
        }
        printf("\n");
    }
}
```

1.(6) 출력 화면 및 설명

```
q1_6.c:5:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(void){
^
q1_6.c:5:1: note: change return type to 'int'
void main(void){
^~~~~
int
1 warning generated.
배열 array의 출력 :
0x16d75f300 0x16d75f304 0x16d75f308
0x16d75f30c 0x16d75f310 0x16d75f314
zwonyoung@jang-won-yeong-ui-MacBookAir Q1 %
```

~~void 쓰지말라는 양증맞은 경고창은 무시해주시면 감사하겠습니다...~~

답부터 말해보자면 행 우선입니다.. 그 이유를 설명하려면 다음의 로직을 살펴보아야만 합니다.

로직 자체는 매우 간단한 편입니다. 순서대로 설명해보자면, 행이 2개 열이 3개인 array 변수를 선언하고 3개의 원소를 가지는 1차원 배열을 2개 저장한 모습입니다.

우리가 2차원 배열을 인식할 때는 다음의 표와 같이 인식하는데.

증가값(주소)	+4	+4	+4
주소값	0x16d75f300	0x16d75f304	0x16d75f308
값	1	2	3
증가값(주소)	+4	+4	+4

증가값(주소)	+4	+4	+4
주소값	0x16d75f30c	0x16d75f310	0x16d75f314
값	4	5	6

이때의 배열 한칸의 규칙은 문제 1.(5)에서 설명한 것처럼 int형 값을 가지고 있으므로 메모리 주소는 4칸씩 끊어(해당 인덱스의 주소값은 +4씩 증가하며) 배정된다.

이때 메모리의 값은 16진수로 표현하는 것을 감안한다.(2진수 변환이 쉽고... 등등 많은 이점이 있다.)

그러나 실제 메모리는 2차원으로 인식하지 않고 다음과 같이 저장됩니다.

증가 값(주소)	+4	+4	+4	+4	+4	+4
주소 값	0x16d75f300	0x16d75f304	0x16d75f308	0x16d75f30c	0x16d75f310	0x16d75f314
값	1	2	3	4	5	6

즉 좀 복잡하게 보일 수 있지만 본질을 따져보면 다음과 같습니다

array[2][3] 일때 [3]부분을 한칸씩 건드리면 '원소 1개'씩 움직일 수 있고(sizeof(int)) [2]부분을 건드리면 '한 줄'씩(sizeof(int) * 3) 움직일 수 있음을 알 수 있습니다. 이를 감안하여 해석해 보겠습니다.

우선 배열이 어떻게 출력되는지 사용자에게 알려주는 print문은 중요하지 않으니 넘어가고 중요한 함수 호출 부분을 살펴보겠습니다.

이러면 아까 설명한 원리에 의해, 파라미터를 int a[][3]과 같은 형태로 받는 이유도 알 수 있는데, 배열의 이름은 2가지의 의미를 갖는다는 것을 저번 시간에 배웠습니다.

1. 배열 그 자체
2. 배열이 시작하는 곳의 주소
3. 그러나 우리는 배열을 다루고 조작하고 관찰할때 2번의 의미로 많이 사용하였습니다.

int a[][3] == int(*a)[3]			
증가값(주소)	+4	+4	+4
주소값	0x16d75f300	0x16d75f304	0x16d75f308
값	1	2	3
int*(a + 1)[3]			
증가값(주소)	+4	+4	+4

<code>int a[][3] == int(*a)[3]</code>			
주소값	0x16d75f30c	0x16d75f310	0x16d75f314
값	4	5	6

즉 해석해 보면 3개씩 1줄로 읽어들이는 2차원 배열이라는 정보를 넘겨주고 있다는 것을 알 수 있고(1차원 배열은 그냥 배열명으로만 다루기도 하므로)

아까 우리는 2차원 배열이 1차원 데이터를 몇칸씩 읽는지에 따라 2차원 배열을 정의하였으므로 이렇게 넘겨주어도 값을 제대로 읽어들이 수 있음을 알 수 있습니다.

(처음 시작 포인터를 제대로 넘겨 주었고, 몇칸씩 끊어서 줄로 인식할 것인지 정보를 넘겨 주었으므로)

따라서, '배열 전체의 위치 정보를 넘겨 준것'으로 해석해도 무방한 것을 알수 있고 2중 for문을 통해 줄 요소의 접근은 i로, 줄별 원소의 접근은 j로 해 줌으로서 %p 연산자와 &연산자로 주소를 넘겨주고 각 요소의 주소값을 성공적으로 출력하고 있는 모습을 볼 수 있고,

지금까지의 내용으로 보아할 때 요소를 포함하고 있는 '행'을 읽어들이고 그 행을 '몇 개'표시할 것인지를 정의하는 구조이므로 행 우선이라고 보는게 맞다고 필자는 판단합니다.

문제 2

완성된 전체 코드

```
#include<stdio.h> // 기본이라 넘어가겠습니다.
#include<string.h> //입력값이 stop일때의 문자열 비교를 위한 헤더파일
#include<math.h> // 제곱과 루트를 사용하기 위한 헤더파일

//문제의 조건에 맞는 구조체 생성
typedef struct student {
    char name[100];
    int student_number;
    int score;
} student;

int get_mean(student *students, int counter){
    // 점수의 합을 더할 float형 변수 total 선언
    float total = 0.0;

    // 학생의 수 만큼 반복
    for(int i = 0; i < counter; i++){
```

```

        // i번째의 요소의 score항목을 total에 더하기
        total += students[i].score;
    }
    // 평균값을 구해야 하므로 total을 학생의 수로 나누고 mean에 그 값을 저장
    float mean = total / counter;
    // mean 반환
    return mean;
}

```

```

// 학생 정보가 저장된 student형 배열의 포인터, 학생의 수의 값을 받는다
float get_deviation(student *students, int counter){
    // 분산의 합을 더할 float형 변수 total 선언
    float total = 0;
    // get_mean 함수를 호출하여 평균값을 구하고 mean에 그 값을 저장
    float mean = get_mean(students, counter);
    // 학생의 수만큼 반복
    for(int i = 0; i < counter; i++){
        // 각 학생의 점수에 평균값을 빼고 제곱을 취해줘 분산으로 만든 다음
        // 그 분산의 값을 total에 더해준다
        total += pow(students[i].score - mean, 2);
    }

    // 표준편차 계산을 위해 지금까지 나온 분산의 합을 학생의 수로 나누어주고
    // 그 값에 루트 취하고 deviation변수에 표준편차 반환
    float deviation = sqrt(total / counter);
    // 표준 편차 반환
    return deviation;
}

```

```

int main(){
    //문제의 조건에 맞는 구조체 배열 생성
    student students[1000] = {'\0'};
    // 간단한 안내 멘트
    printf("값을 입력해주세요... stop을 입력하면 입력받기가 종료됩니다.\n");
    // 몇명의 학생이 입력되었는지 측정할 카운터
    int counter = 0;
    // counter의 크기가 1000을 넘어가지 않도록 계속 입력 받으나
    for (; counter < 1000; counter++) {
        // 1. 학생의 이름 입력을 받기 2. stop을 입력받고 반복문 탈출하기를 위해서 먼저 따로 입
        력받기

        scanf("%s", students[counter].name);
        // 개행 문자가 들어있는 버퍼 메모리 초기화를 위한 getchar()
        getchar();
        // 입력받은 값이 stop이면 반복문 탈출
        if (strcmp(students[counter].name, "stop") == 0)
            break;
    }
}

```

```

        // 이후 나머지 정보를 받고
        scanf("%d %d", &students[counter].student_number,
&students[counter].score);
        // 제대로 입력되었는지 확인할 printf함수 출력
        printf("이름 : %s 학번 : %d 성적 : %d\n", students[counter].name,
students[counter].student_number, students[counter].score);
    }
    // 문제의 요구 사항에 맞게 소수점 연산자 %f로 연산을 진행하였고
    // 그 안에 들어갈 값은 get_deviation함수를 사용하여 표준 편차를 구해주어
    // 그 값을 %f에 집어넣어 주었습니다.
    printf("%f", get_deviation(students, counter));
    return 0;
}
// p.s) 표본 수가 작을때를 대비해 n-1로 접근하는 것이 정확하나
// 문제에서 그것을 요구하지 않았으므로 일부러 사용하지 않았습니다...

```

문제 2.(1)

2.(1) 코드 및 주석

```

#include<stdio.h> // 기본이라 넘어가겠습니다.
#include<string.h> //입력값이 stop일때의 문자열 비교를 위한 헤더파일
#include<math.h> // 제곱과 루트를 사용하기 위한 헤더파일

//문제의 조건에 맞는 구조체 생성
typedef struct student {
    char name[100];
    int student_number;
    int score;
} student;

int main(){
    // 제 정보입니다 :) student형 students를 선언하고 구조체 요소에 각각의 자료형에 맞
    게 값을 집어넣어 주었습니다.
    student students = {"장원영", 20211379, 100};

    // 구조체의 각 요소에 접근하여 출력하고 있습니다.
    printf("%s %d %d", students.name, students.student_number,
students.score);
    return 0;
}

```

2.(1) 출력 화면 및 설명

```
● zwonyoung@jang-won-yeong-ui-MacBookAir Q2 % cd "/Users/zwonyoung/Desktop/prog  
p/programing/datastruture/code/Q2/"q2_1  
장 원 영 20211379 100%  
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q2 % []
```

문제의 조건에 맞는 100의 크기를 가지는 문자열 변수 name, int형 학번, int형 점수 를 각각 구조체로 묶어
서 선언하고 typedef를 통해 student형 자료로 만들어 주었습니다.

typedef를 해주는 이유는 typedef는 typedef뒤에 붙는 변수 이름을 지정해주면 그 타입을 부를때 대신 지정
해준 이름으로 부를 수 있습니다.

원래는 student형 구조체를 호출할때

```
struct student students
```

와 같이 선언해야 하는데 typedef를 통해 이 구문을 단순하게

```
student students
```

와 같이 만들어 줄 수 있습니다.

그리고 이때 주의할 점은 구조체를 만든 것은 데이터 구조의 형태를 만들어 준 것이지 실제 변수를 선언한 것
이 아닙니다.

따라서 main 단에서 student형 students 변수를 선언해주고 구조체의 각 요소(이름, 학번, 점수)에 값을 순
서대로 넣어주어 값을 할당한 다음, printf로 출력하였습니다.

이때 유의할 점은 구조체는 구조체 자체로 절대 값을 비교하거나, 구조체 통째로 집어넣거나 하는것은 불가능
합니다. 반드시 구조체의 요소 개별로 접근하여 처리해야 합니다.

이를 반영하여 printf문도 생성된구조체.요소 와 같이 개별로 접근하여 출력하고 있는 모습을 확인할 수 있습
니다.

문제 2.(2)

2.(2) 코드 및 주석

```
#include<stdio.h> // 기본이라 넘어가겠습니다.
#include<string.h> //입력값이 stop일때의 문자열 비교를 위한 헤더파일

//문제의 조건에 맞는 구조체 생성
typedef struct student {
    char name[100];
    int student_number;
    int score;
} student;

int main(){
    //문제의 조건에 맞는 구조체 배열 생성
    student students[1000] = {'\0'};
    // 간단한 안내 멘트
    printf("값을 입력해주세요... stop을 입력하면 입력받기가 종료됩니다.\n");
    // 몇명의 학생이 입력되었는지 측정할 카운터
    int counter = 0;
    // counter의 크기가 1000을 넘어가지 않도록 계속 입력 받으나
    for (; counter < 1000; counter++) {
        // 1. 학생의 이름 입력을 받기 2. stop을 입력받고 반복문 탈출하기를 위해서 먼저 따로 입
        력받기
        scanf("%s", students[counter].name);
        // 개행 문자가 들어있는 버퍼 메모리 초기화를 위한 getchar()
        getchar();
        // 입력받은 값이 stop이면 반복문 탈출
        if (strcmp(students[counter].name, "stop") == 0)
            break;
        // 이후 나머지 정보를 받고
        scanf("%d %d", &students[counter].student_number,
            &students[counter].score);
        // 제대로 입력되었는지 확인할 printf함수 출력
        printf("이름 : %s 학번 : %d 성적 : %d\n", students[counter].name,
            students[counter].student_number, students[counter].score);
    }
    // 학생 수를 제대로 입력받았는지 확인하는 print문
    printf("%d", counter);
    return 0;
}
```

2.(2) 출력 화면 및 설명


```

● zwonyoung@jang-won-yeong-ui-MacBookAir Q2 % cd "/Users/zwonyoung/Desktop/prog
p/programing/datastruture/code/Q2/"q2_2
값을 입력해주세요... stop을 입력하면 입력받기가 종료됩니다.
장원영 20211379 100
이름 : 장원영 학번 : 20211379 성적 : 100
홍길동 20211378 50
이름 : 홍길동 학번 : 20211378 성적 : 50
김철수 20201380 67
이름 : 김철수 학번 : 20201380 성적 : 67
stop
3%
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q2 %

```

로직을 설명하자면 다음과 같습니다.

student students[1000] = {"\0"};로 선언해 주었는데 하나의 요소가 student형 만큼의 크기를 가지는 구조체 배열 students를 선언하고 그 배열의 요소들을 전부 초기화 해준 모습입니다.

초기화를 해준 이유는 다음과 같은 이유가 있습니다.

1. 초기화를 통해 루프문을 빠져나갈 조건을 충족했을 경우 점수 부분을 전부 더해야 하는데 이상한 값을 집어넣지 않기 위한 안전 장치

그리고 입력받은 학생의 수를 측정할 정수형 변수 counter를 선언하였고 for문의 선언부는 필요 없으므로 생략, 조건부에 counter를 집어넣고 배열의 최대 크기인 1000만큼 반복하도록 설정해 주었고 반복문이 실행될 때마다 counter의 값이 증가하게 설계하였습니다.

이후 counter번째의 인덱스의 name부분에 구조체 문법으로 접근해 주었는데, 이름, 학번, 점수를 한번에 묶어서 받지 않고 따로 받는 이유는 다음과 같은 2가지의 이유가 있습니다.

1. name을 입력받는 목적
2. 뒤에 나와있는 stop이 입력되었을 때 바로 다른 구조체 요소를 입력받지 않고 break문으로 종료할 수 있게 하기 위해서

이런 이유로 scanf를 따로 받았는데, 여기서 scanf를 연속으로 받을 경우 개행 문자가 입력 버퍼에 남아있어 그대로 다음 scanf를 제대로 받을 수 없게 됩니다.

따라서 getchar();와 같이 활용하여 입력 버퍼를 초기화해주었습니다.

getchar는 키보드의 입력값을 문자형으로 한글자씩 받아오는데 개행 문자를 getchar가 받을 수 있습니다.

그리고 이렇게 getchar는 개행 문자를 받고 그 개행 문자를 반환했지만 받을 곳이 없으므로 그대로 개행 문자를 소비할 수 있습니다.

그 다음으로 조건문을 통해 조건식에 students[counter].name 즉 해당 배열 요소의 name 구조체 요소에 stop이 입력되면 입력을 받는 for문을 빠져나와 입력받기를 종료합니다.

문자열끼리의 비교이므로 strcmp함수를 이용하여 참 거짓을 반환하여 처리하였습니다.

이후 반복문이 계속된다면 나머지 학번, 성적을 순서대로 입력받고 다시 루프를 진행합니다.

이때 성공적으로 종료되었다면, 입력한 횟수만큼 counter에 저장 된 것을 확인할 수 있습니다.

문제 2.(3)

2.(3) 코드 및 주석

```
#include<stdio.h> // 기본이라 넘어가겠습니다.
#include<string.h> //입력값이 stop일때의 문자열 비교를 위한 헤더파일
#include<math.h> // 제곱과 루트를 사용하기 위한 헤더파일

//문제의 조건에 맞는 구조체 생성
typedef struct student {
    char name[100];
    int student_number;
    int score;
} student;

int get_mean(student *students, int counter){
    // 점수의 합을 더할 float형 변수 total 선언
    float total = 0.0;

    // 학생의 수 만큼 반복
    for(int i = 0; i < counter; i++){
        // i번째의 요소의 score항목을 total에 더하기
        total += students[i].score;
    }
    // 평균값을 구해야 하므로 total을 학생의 수로 나누고 mean에 그 값을 저장
    float mean = total / counter;
    // mean 반환
    return mean;
}

// 학생 정보가 저장된 student형 배열의 포인터, 학생의 수의 값을 받는다
float get_deviation(student *students, int counter){
    // 분산의 합을 더할 float형 변수 total 선언
    float total = 0.0;
    // get_mean 함수를 호출하여 평균값을 구하고 mean에 그 값을 저장
    float mean = get_mean(students, counter);
    // 학생의 수만큼 반복
    for(int i = 0; i < counter; i++){
        // 각 학생의 점수에 평균값을 빼고 제곱을 취해줘 분산으로 만든 다음
```

```

        // 그 분산의 값을 total에 더해준다
        total += pow(students[i].score - mean, 2);
    }

    // 표준편차 계산을 위해 지금까지 나온 분산의 합을 학생의 수로 나누어주고
    // 그 값에 루트 취하고 deviation변수에 표준편차 반환
    float deviation = sqrt(total / counter);
    // 표준 편차 반환
    return deviation;
}

int main(){
    //문제의 조건에 맞는 구조체 배열 생성
    student students[1000] = {'\0'};
    // 간단한 안내 멘트
    printf("값을 입력해주세요... stop을 입력하면 입력받기가 종료됩니다.\n");
    // 몇명의 학생이 입력되었는지 측정할 카운터
    int counter = 0;
    // counter의 크기가 1000을 넘어가지 않도록 계속 입력 받으나
    for (; counter < 1000; counter++) {
        // 1. 학생의 이름 입력을 받기 2. stop을 입력받고 반복문 탈출하기를 위해서 먼저 따로 입
        력받기

        scanf("%s", students[counter].name);
        // 개행 문자가 들어있는 버퍼 메모리 초기화를 위한 getchar()
        getchar();
        // 입력받은 값이 stop이면 반복문 탈출
        if (strcmp(students[counter].name, "stop") == 0)
            break;
        // 이후 나머지 정보를 받고
        scanf("%d %d", &students[counter].student_number,
            &students[counter].score);
        // 제대로 입력되었는지 확인할 printf함수 출력
        printf("이름 : %s 학번 : %d 성적 : %d\n", students[counter].name,
            students[counter].student_number, students[counter].score);
    }
    // 문제의 요구 사항에 맞게 소수점 연산자 %f로 연산을 진행하였고
    // 그 안에 들어갈 값은 get_deviation함수를 사용하여 표준 편차를 구해주어
    // 그 값을 %f에 집어넣어 주었습니다.
    printf("%f", get_deviation(students, counter));
    return 0;
}

// p.s) 표본 수가 작을때를 대비해 n-1로 접근하는 것이 정확하나
// 문제에서 그것을 요구하지 않았으므로 일부러 사용하지 않았습니다...

```

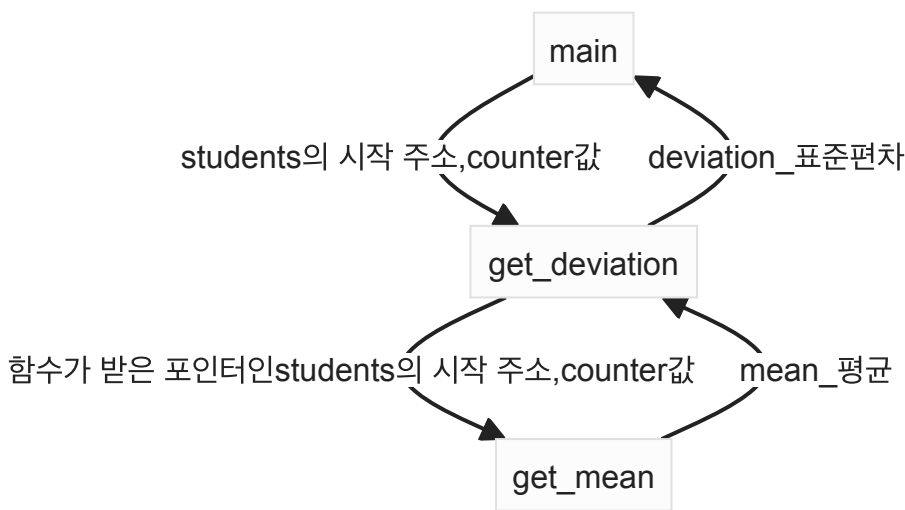
2.(3) 출력 화면 및 설명

```

● zwonyoung@jang-won-yeong-ui-MacBookAir Q2 % cd "/Users/zwonyoung/Desktop/prog
p/programing/datastruture/code/Q2/"q2_3
값을 입력해 주세요 ... stop을 입력하면 입력받기가 종료됩니다 .
장원영 20211379 100
이름 : 장원영 학번 : 20211379 성적 : 100
홍길동 20211380 50
이름 : 홍길동 학번 : 20211380 성적 : 50
stop
25.000000%
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q2 % █

```

아까의 입력 로직에 get_mean, get_deviation 함수를 추가하고 printf값을 get_deviation으로 주고 있는 모습입니다. 참조 관계는 다음과 같습니다.



get_deviation 함수를 정의하고 이 함수에 student의 시작 주소, 입력받은 학생의 수인 counter의 값을 파라미터로 넘겨주었고, 함수도 이에 걸맞게 각각 student형 포인터, 정수형으로 받고 있는 모습을 보이고 있습니다.

그리고 분산의 합을 저장할 total 변수를 선언해주고 분산을 구하기 위해 아까 받은 포인터 변수값과 정수값을 다시 같은 형식으로 get_mean함수에 파라미터로 넘겨주었으며, 아까 받은 counter 요소, 즉 각 학생의 수만큼 인덱스 문법으로 접근하여 구조체 요소중 성적 요소에 접근하였고 이 정수값을 전부 total에 더해주었습니다.

평균을 구해주어야 하므로 반복문이 완료된 후, total을 counter 즉 학생의 수로 나누어 mean 변수에 할당하였고 이 mean을 리턴하였습니다.

이러고 다시 get_deviation으로 넘어가서 이렇게 받은 평균값으로 get_deviation안에 선언한 mean 변수에 이 값을 넣어주고 학생 수만큼의 반복문을 돌려 역시 get_mean 함수에서 했던 방식과 동일하게 성적 요소에 접근, 각 분산들의 합을 인덱스 문법으로 접근한 개별 값 요소에 아까 구한 mean을 빼주고, 이 값에 pow 함수를 통해 제공해둔 다음 이 값을 total 변수에 계속해서 더하여 구해주었습니다.

이후 이 값을 학생의 수로 나누어주어 평균 분산을 구하고 이 값에 루트를 취해주기 위한 sqrt 함수의 파라미터로 집어넣어 루트값을 반환하고 이 값을 deviation 변수에 할당, 이후 이 값을 리턴해 주었습니다.

이렇게 리턴받은 값을 printf문으로 출력하여 문제 풀이를 완료하였습니다.

p.s) 표본 수가 작을때를 대비해 통계학적 정석은 $n-1$ 로 접근하는 것이 정확하나... 문제에서 그것을 요구하지 않았으므로 일부러 사용하지 않았습니다...

문제 2.(4)

2.(4) 코드 및 주석

```
#include<stdio.h> // 기본이라 넘어가겠습니다.
#include<string.h> //입력값이 stop일때의 문자열 비교를 위한 헤더파일
#include<math.h> // 제곱과 루트를 사용하기 위한 헤더파일

//문제의 조건에 맞는 구조체 생성
typedef struct student{
    char name[100];
    int student_number;
    int score;
} student;

int get_mean(student *ptr, int counter){
    // 점수의 합을 더할 float형 변수 total 선언
    float total = 0.0;
    // 학생의 수 만큼 반복
    for(int i = 0; i < counter; i++){
        // i번째의 요소의 score항목을 total에 더하기
        total += ptr[i].score;
    }
    // 평균값을 구해야 하므로 total을 학생의 수로 나누고 mean에 그 값을 저장
    float mean = total / counter;
    // mean 반환
    return mean;
}

// 학생 정보가 저장된 student형 배열의 포인터, 학생의 수의 값을 받는다
float get_deviation(student *ptr, int counter){
    // 분산의 합을 더할 float형 변수 total 선언
    float total = 0;
    // get_mean 함수를 호출하여 평균값을 구하고 mean에 그 값을 저장
    float mean = get_mean(ptr, counter);
```

```

// 학생의 수만큼 반복
for(int i = 0; i < counter; i++){
// 각 학생의 점수에 평균값을 빼고 제곱을 취해줘 분산으로 만든 다음
// 그 분산의 값을 total에 더해준다
total += pow(ptr[i].score - mean, 2);
}
// 표준편차 계산을 위해 지금까지 나온 분산의 합을 학생의 수로 나누어주고
// 그 값에 루트 취하고 deviation변수에 표준편차 반환
float deviation = sqrt(total / counter);
// 표준 편차 반환
return deviation;
}

int main(){
//문제의 조건에 맞는 구조체 배열 생성
student students[1000] = {'\0'};
// 문제 조건에 맞는 포인터 배정
student* ptr = students;
// 간단한 안내 멘트
printf("값을 입력해주세요... stop을 입력하면 입력받기가 종료됩니다.\n");
// 몇명의 학생이 입력되었는지 측정할 카운터
int counter = 0;
// counter의 크기가 1000을 넘어가지 않도록 계속 입력 받으나
for (; counter < 1000; counter++) {
// 1. 학생의 이름 입력을 받기 2. stop을 입력받고 반복문 탈출하기를 위해서 먼저 따로 입력받기
scanf("%s", students[counter].name);
// 개행 문자가 들어있는 버퍼 메모리 초기화를 위한 getchar()
getchar();
// 입력받은 값이 stop이면 반복문 탈출
if (strcmp(students[counter].name, "stop") == 0)
break;
// 이후 나머지 정보를 받고
scanf("%d %d", &students[counter].student_number, &students[counter].score);
// 제대로 입력되었는지 확인할 printf함수 출력
printf("이름 : %s 학번 : %d 성적 : %d\n", students[counter].name,
students[counter].student_number, students[counter].score);
}

// 문제의 요구 사항에 맞게 소수점 연산자 %f로 연산을 진행하였고
// 그 안에 들어갈 값은 get_deviation함수를 사용하여 표준 편차를 구해주어
// 그 값을 %f에 집어넣어 주었습니다.
printf("%f", get_deviation(ptr, counter));
return 0;
}

// p.s) 표본 수가 작을때를 대비해 n-1로 접근하는 것이 정확하나

```

```
// 문제에서 그것을 요구하지 않았으므로 일부러 사용하지 않았습니다...
```

2.(4) 출력 화면 및 설명

```
● zwonyoung@jang-won-yeong-ui-MacBookAir Q2 % cd "/Users/zwonyoung/Desktop/prog  
p/programing/datastruture/code/Q2/"q2_3  
값을 입력해주세요... stop을 입력하면 입력받기가 종료됩니다.  
장원영 20211379 100  
이름 : 장원영 학번 : 20211379 성적 : 100  
홍길동 20211380 50  
이름 : 홍길동 학번 : 20211380 성적 : 50  
stop  
25.000000%  
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q2 %
```

아까의 3번 문제와 결과같은 다르지 않지만 `students`의 포인터를 `ptr`이라는 포인터 변수를 선언하고 그곳에
배정해 주었으며,

파라미터로 `ptr`을 넘겨주었습니다. 매개변수 변동 사항은 바뀌었음을 시각적으로 표현하였습니다.

문제 3

문제 3.(1)

3.(1) 코드 및 주석

```
#include <stdio.h>  
  
void main(){  
    char* pc; // char형 포인터 선언  
    int* pi; // int형 포인터 선언  
    float pf; // float형 포인터 선언  
    double pd; // double형 포인터 선언  
  
    char c; // char형 변수  
    int i; // int형 변수  
    float f; // float형 변수  
    double d; // double형 변수
```

```

        printf("pc = %lu\n", pc); // char형 포인터를 unsigned long 형 데이터로 출력
    }

```

3.(1) 출력 화면 및 설명

```

zwoyoung@jang-won-yeong-ui-MacBookAir Q2 % cd "/Users/zwoyoung/Desktop/programing/datastruture/code/Q3/" &
p/programing/datastruture/code/Q3/"q3_1
q3_1.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(){
^
q3_1.c:3:1: note: change return type to 'int'
void main(){
^~~~
int
q3_1.c:14:26: warning: format specifies type 'unsigned long' but the argument has type 'char *' [-Wformat]
    printf("pc = %lu\n", pc);
                   ~~~~   ^~
                   %s
2 warnings generated.
pc = 4341799824
zwoyoung@jang-won-yeong-ui-MacBookAir Q3 % 

```

보이는 코드와 같이 각각의 변수들은 순서대로

char형 포인터, int형 포인터, float형 포인터, double형 포인터,
char형 변수, int형 변수, float형 변수, double형 변수입니다.

이때 char형 포인터 pc는 아직 초기화가 되지 않은 상태이기에 쓰레기 값을 저장하고 있습니다. 따라서 저 상태로 출력하면 쓰레기 값을 내뱉습니다.

문제 3.(2)

3.(2) 코드 및 주석

```

#include <stdio.h>

void main(){
    char* pc; // char형 포인터 선언
    int* pi; // int형 포인터 선언
    float pf; // float형 포인터 선언
    double pd; // double형 포인터 선언

    char c; // char형 변수
    int i; // int형 변수
    float f; // float형 변수
    double d; // double형 변수
}

```



```

printf("pc = %lu\n", pc); // char형 포인터를 unsigned long 형 데이터로 출력
*pc = 100; // pc가 가리키고 있는 주소에 100값 할당
}

```

3.(2) 출력 화면 및 설명

```

ⓧ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 % cd "/Users/zwonyoung/Desktop/programing/datastruture/code/Q3/" &&
p/programing/datastruture/code/Q3/"q3_2
q3_2.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(){
^
q3_2.c:3:1: note: change return type to 'int'
void main(){
^~~~
int
q3_2.c:14:23: warning: format specifies type 'unsigned long' but the argument has type 'char *' [-Wformat]
printf("pc = %lu\n", pc); // char형 포인터를 unsigned long 형 데이터로 출력
                ^~      ^~
                %s
2 warnings generated.
pc = 4313144208
zsh: bus error  "/Users/zwonyoung/Desktop/programing/datastruture/code/Q3/"q3_2
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 %

```

보이는 에러 메시지와 같이 bus error를 일으키며 작동 자체가 막히는 모습을 보여줍니다.

이처럼 포인터는 시스템 메모리를 가리킬 수 있으며 잘못하면 시스템이나 환경, 취약점에 따라 스택 오버플로우, 해킹 취약점 등의 의도치 않은 동작이 발생할 수 있어 포인터를 사용할 때에는 주의해야 합니다.

문제 3.(3)

3.(3) 코드 및 주석

```

#include <stdio.h>

void main(){
    char* pc; // char형 포인터 선언
    int* pi; // int형 포인터 선언
    float pf; // float형 포인터 선언
    double pd; // double형 포인터 선언

    char c; // char형 변수
    int i; // int형 변수
    float f; // float형 변수
    double d; // double형 변수

    pc = NULL; // null로 초기화
    printf("pc = %lu\n", pc); // char형 포인터를 unsigned long 형 데이터로 출력
    *pc = 100; // pc가 가리키고 있는 주소에 100값 할당
}

```

```
}
```

3.(3) 출력 화면 및 설명

```
zwoyoung@jang-won-yeong-ui-MacBookAir Q3 % cd "/Users/zwoyoung/Desktop/programing/datastruture/code/Q3/" &&
p/programing/datastruture/code/Q3/"q3_3
q3_3.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(){
^
q3_3.c:3:1: note: change return type to 'int'
void main(){
~~~~~
int
q3_3.c:15:23: warning: format specifies type 'unsigned long' but the argument has type 'char *' [-Wformat]
    printf("pc = %lu\n", pc); // char형 포인터를 unsigned long 형 데이터로 출력
                   ~~~~      ^~
                           %s
2 warnings generated.
pc = 0
zsh: segmentation fault "/Users/zwoyoung/Desktop/programing/datastruture/code/Q3/"q3_3
zwoyoung@jang-won-yeong-ui-MacBookAir Q3 %
```

출력 결과에서 보아도 알 수 있듯이 segmentation fault 에러가 발생하면서 값이 제대로 할당되지 않는 것을 볼 수 있습니다.

이는 pc가 가리키는 메모리가 없음을 나타내는 null값으로 초기화를 해 주었기 때문에 값을 저장할 저장 공간 자체가 없어 오류가 발생합니다.

하지만 이로서 알 수 있는 것은 null값을 할당함으로서 갱글링 포인터 즉, 포인터를 잘못 지정해서 생기는 치명적인 오류를 방지할 수 있다는 것을 알 수 있습니다.

문제 3.(4)

3.(4) 코드 및 주석

```
#include <stdio.h>

void main(){
    char* pc; // char형 포인터 선언
    int* pi; // int형 포인터 선언
    float pf; // float형 포인터 선언
    double pd; // double형 포인터 선언

    char c; // char형 변수
    int i; // i형 변수
    float f; // float형 변수
    double d; // double형 변수
    printf("초기화 하기 전의 pc = %lu\n", pc); // char형 포인터를 unsigned long
형 데이터로 출력
```

```

    pc = &c;
    printf("초기화 한 후의 pc = %lu\n", pc); // 이하 동일 및 초기화 여부 확인
    *pc = 100; // pc가 가리키는 c의 값을 100으로 설정
    printf("c = %d\n", c); // c 출력
}

```

3.(4) 출력 화면 및 설명

```

q3_4.c:3:1: note: change return type to 'int'
void main(){
^~~~
int
q3_4.c:14:47: warning: format specifies type 'unsigned long' but the argument has type 'char *' [-Wformat]
    printf("초기화 하기 전의 pc = %lu\n", pc); // char형 포인터를 unsigned long 형 데이터로 출력
                                   ~~~~^~
                                   %s
q3_4.c:16:47: warning: format specifies type 'unsigned long' but the argument has type 'char *' [-Wformat]
    printf("초기화 한 후의 pc = %lu\n", pc); // 이하 동일 및 초기화 여부 확인
                                   ~~~~^~
                                   %s

3 warnings generated.
초기화 하기 전의 pc = 4339342224
초기화 한 후의 pc = 6131806943
c = 100
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 %

```

출력 화면을 보면 초기화 하기 전의 pc는 쓰레기 값이 할당된 pc의 값을 출력하고 있습니다.

하지만 char형 포인터 변수 pc에 c의 주소를 할당해 주었고, 이를 통해 포인터 변수 pc를 통해 c를 참조하여 100을 할당해 주었고 c의 값이 성공적으로 바뀐 것을 확인할 수 있습니다.

문제 3.(5)

3.(5) 코드 및 주석

```

#include <stdio.h>

void main(){
    char* pc; // char형 포인터 선언
    int* pi; // int형 포인터 선언
    float *pf; // float형 포인터 선언
    double *pd; // double형 포인터 선언

    char c; // char형 변수
    int i; // i형 변수
    float f; // float형 변수
    double d; // double형 변수
    printf("초기화 하기 전의 pc = %lu\n", pc); // char형 포인터를 unsigned long
형 데이터로 출력
    pc = &c;

```

```

printf("초기화 한 후의 pc = %lu\n", pc); // 이하 동일 및 초기화 여부 확인
*pc = 100; // pc가 가리키는 c의 값을 100으로 설정
printf("c = %d\n", c); // c 출력

printf("초기화 하기 전의 pi = %lu\n", pi); // int형 포인터를 unsigned long
형 데이터로 출력
pi = &i;
printf("초기화 한 후의 pi = %lu\n", pi); // 이하 동일 및 초기화 여부 확인
*pi = 100; // pi가 가리키는 c의 값을 100으로 설정
printf("i = %d\n", i); // i 출력

printf("초기화 하기 전의 pf = %lu\n", pf); // float형 포인터를 unsigned
long 형 데이터로 출력
pf = &f;
printf("초기화 한 후의 pf = %lu\n", pf); // 이하 동일 및 초기화 여부 확인
*pf = 100; // pf가 가리키는 f의 값을 100으로 설정
printf("f = %f\n", f); // f 출력

printf("초기화 하기 전의 pd = %lu\n", pd); // double형 포인터를 unsigned
long 형 데이터로 출력
pd = &d;
printf("초기화 한 후의 pc = %lu\n", pd); // 이하 동일 및 초기화 여부 확인
*pd = 100; // pd가 가리키는 d의 값을 100으로 설정
printf("d = %lf\n", d); // d 출력
}

```

3.(5) 출력 화면 및 설명

```

q3_4.c:34:47: warning: format specifies type 'unsigned long' but the argument has type 'double *' [-Wformat]
printf("초기화 한 후의 pc = %lu\n", pd); // 이하 동일 및 초기화 여부 확인
                        ~~~~      ^~

9 warnings generated.
초기화 하기 전의 pc = 4312030096
초기화 한 후의 pc = 6159119071
c = 100
초기화 하기 전의 pi = 4307778800
초기화 한 후의 pi = 6159119064
i = 100
초기화 하기 전의 pf = 6642246507183524536
초기화 한 후의 pf = 6159119060
f = 100.000000
초기화 하기 전의 pd = 6159119104
초기화 한 후의 pc = 6159119048
d = 100.000000
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 %

```

문제 3.(4)에서 설명한 바와 같이 똑같이 작동하는 것을 볼 수 있습니다.

다만 float, double형은 각각 %f %lf로 출력해야 작동하는 것을 알 수 있는데 이는 float형과 double형이 char, int 형에 비해 저장 방식이 매우 다르기 때문입니다...

문제 3.(6)

3.(6) 코드 및 주석

```
#include <stdio.h>

void main(void){
    int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    int *p;

    p = a; // p에 a의 시작 주소를 치환한다.
    //이 문장은 포인터를 사용하여 배열 a의 첫째, 둘째, 셋째 원소들을 출력한다.
    printf("%d %d %d\n", *p, *(p+1), *(p+2));

    //이 문장은 배열 이름을 포인터처럼 사용하여 배열 a의 첫째, 둘째, 셋째 원소들을 출력한다.
    printf("%d %d %d\n", *a, *(a+1), *(a+2));

    //이 문장은 배열 a를 사용하여 같은 것을 출력한다.
    printf("%d %d %d\n", a[0], a[1], a[2]);
}
```

3.(6) 출력 화면 및 설명

```
zwoyoung@jang-won-yeong-ui-MacBookAir Q3 % cd "/Users/zwoyoung/Desktop/programing/datastruture/code/Q3/" &&
structure/code/Q3/"q3_6
q3_6.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(void){
^
q3_6.c:3:1: note: change return type to 'int'
void main(void){
^~~~
int
1 warning generated.
10 20 30
10 20 30
10 20 30
zwoyoung@jang-won-yeong-ui-MacBookAir Q3 %
```

출력 결과를 봐도 아시겠지만 한개의 원소가 int크기이고 그게 10개가 들어가는 배열 `a`를 선언하고 각각을 `a`의 시작 주소를 받은 `p`를 통해 `p`에 1씩 더해가면서 원소를 출력, 배열의 이름인 `a` 자체에 1씩 더해가면서 원소를 출력하고, `a`의 인덱스 접근 방식을 통해 출력을 해 보았습니다.

이러한 사실을 통해 알 수 있는 것은 `a`의 값을 똑같이 받았던 `p`가 `a`와 동일하게 작동했던 것을 보아 `a`는 배열의 시작 주소를 가지고 있음을 알 수 있고,

이러한 사실을 통해 배열의 인덱스 접근 방법인 대괄호[] 접근법도 비슷한 원리인 것을 알 수 있으며 이 3가지는 서로 유연하게 변화가 가능함을 알 수 있습니다... 배열의 이름과 포인터는 동일합니다.

문제 3.(7)

3.(7) 코드 및 주석

```
#include <stdio.h>

void main(void){
    // 문자열 str을 선언하고 문자열 안에 What is Pointer?의 크기만큼 공간 확보
    // 이후 해당 문자열 삽입
    char str[] = "What is Pointer?";
    // 문자열을 지정할 char형 포인터 p선언
    char* p;
    // 반복문 전용 i선언
    int i;

    // 배열의 첫번째 주소를 p에 할당
    p = str;
    // 널 문자가 발견될 때까지 반복
    for(i = 0; p[i] != NULL; i++){
        printf("%c", p[i]);
    }
}
```

3.(7) 출력 화면 및 설명

```
ⓧ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 % cd "/Users/zwonyoung/Desktop/programing/datastruture/code/Q3/" &&
struture/code/Q3/"q3_7
q3_7.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(void){
^
q3_7.c:3:1: note: change return type to 'int'
void main(void){
^~~~
int
q3_7.c:10:21: warning: comparison between pointer and integer ('char' and 'void *') [-Wpointer-integer-compar
    for(i = 0; p[i] != NULL; i++){
                ^ ~~~~~
2 warnings generated.
What is Pointer?
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 %
```

위쪽의 양중맞은 오류창들은 무시해주시면 감사하겠습니다.

c언어는 기본적으로 배열의 크기를 명시하지 않았을때 형식이 정해진 문자열을 넣을 경우에는 그 문자열의 크기를 자동적으로 할당하고 그 문자열을 집어넣을 수 있습니다. 따라서 What is Pointer?라는 내용이 문자열 str에 저장됩니다.

이후 문자열을 가리킬 char형 포인터 p를 선언하고, 반복문 증감식에 활용할 i도 선언해 줍니다.

그리고 p에 문자열 str의 처음 시작 주소를 넣어주고 p의 인덱스 문법으로 p[i]와 같이 반복문을 돌려주면

What is Pointer?

를 출력하고 종료하는 것을 알 수 있는데, 문자열은 선언하는 순간부터 맨 뒤쪽에 널값이 추가되는 것을 알 수 있습니다.

문제 3.(8)

3.(8) 코드 및 주석

```
#include <stdio.h>

// 포인터 배열 pa 선언, 배열의 크기는 넣어주는 값에 따라 할당
char* pa[] = {
    "에러 1",
    "에러 2",
    "에러 3",
    "에러 4",
    "에러 5",
    "에러 6"
};

// 파라미터로 int형 정수 err_num을 받는 함수 error를 선언하고
// pa 배열의 index에 err_num을 집어넣고 그 인덱스에 해당하는 값을 출력
void error(int err_num){
    printf(pa[err_num]);
}

// error 함수에 1을 넘겨주며 호출
void main(){
    error(1);
}
```

3.(8) 출력 화면 및 설명

```
q3_8.c:16:1: note: change return type to 'int'
void main(){
^~~~
int
2 warnings generated.
에러 2
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 %
```

보이시는 것과 같이 전역변수 포인터 배열 `pa`를 선언하고 `pa`의 각 요소가 "에러 1", "에러 2", "에러 3", "에러 4", "에러 5", "에러 6"을 가리키게 하고 이 요소들의 개수만큼의 포인터를 가지는 배열의 크기 공간을 자동으로 할당하여 포인터 배열 `pa`를 선언한 모습을 볼 수 있습니다.(물론 각자의 문자열의 크기도 자동으로 배정됩니다.)

여기서 주의할 점은 배열 `pa`는 각 요소가 문자열의 시작 주소를 가리키고 있음을 인지하여야 합니다.

<code>char *pa[]</code>	<code>char</code>	<code>char</code>	<code>char</code>
에	러	1	\0
	char	char	char
에	러	2	\0
	char	char	char
에	러	3	\0
...	

그리고 `error` 함수를 선언하고 정수형 `err_num`으로 값을 받고 있고 그 값을 `pa` 배열의 인덱스 값으로 사용하여 `pa` 배열 요소중 하나의 값을 출력하고 있습니다.

그리고 메인 함수에서는 이 `error` 함수를 인수 1을 넘겨주며 호출하고 있습니다. 따라서 `pa` 배열의 1번 인덱스에 해당하는 값을 출력하게 되며 두번째 요소인 "에러 2"를 출력하며 프로그램을 종료하고 있습니다.

`printf`는 기본적으로 %연산자를 통해 변수를 호출하여 출력하는 것이 기본적이거나, 굳이 %연산자를 쓰지 않고 바로 출력이 가능합니다.

문제 3.(9)

3.(9) 코드 및 주석

```
#include <stdio.h>

void swap1(int i, int j){
    int temp; // i 와 j를 바꾸기 위해 값을 임시로 저장할 temp 선언

    temp = i; // i의 값을 temp에 저장
    i = j; // j의 값을 i에 저장
    j = temp; // temp의 값을 j에 저장
}

void swap2(int *i, int *j){
    int temp; // i 와 j를 바꾸기 위해 값을 임시로 저장할 temp 선언

    temp = *i; // 포인터 i가 가리키는 주소에 할당된 값을 temp에 저장
    *i = *j; // 포인터 i가 가리키는 주소에 할당된 값을 포인터 i가 가리키는 주소에 할당된
값에 저장
```



```

        *j = temp; // temp에 저장된 값을 포인터 j가 가리키는 주소에 할당된 값에 저장
    }

    void main(){
        //바꿀 두 값의 변수 선언
        int num1, num2;

        num1 = 100; // 100 할당
        num2 = 200; // 200 할당

        swap1(num1, num2); // swap1 함수에 '값' 넘겨주기
        printf("num1 : %d num2 %d\n", num1, num2); // 값이 제대로 바뀌었나 확인하는
printf
        swap2(&num1, &num2); // swap2 함수에 값의 '주소' 넘겨주기
        printf("num1 : %d num2 %d\n", num1, num2); // 값이 제대로 바뀌었나 확인하는
printf
    }

```

3.(9) 출력 화면 및 설명

```

⊗ zwonyoung@jang-won-yeong-ui-MacBookAir gwnu_JAVA_00P % cd "/Users/zwonyoung/Desktop/programing/datastruture/code/Q3/"q3_9
q3_9.c:19:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main(){
^
q3_9.c:19:1: note: change return type to 'int'
void main(){
^~~~
int
1 warning generated.
num1 : 100 num2  200
num1 : 200 num2  100
○ zwonyoung@jang-won-yeong-ui-MacBookAir Q3 % 

```

문제의 코드를 설명하자면 두 변수 int형 num1과 num2를 선언하고 각각에 100, 200을 할당하여 각각 swap 1함수에는 각각의 변수의 '값'을 swap2 함수에는 각각의 변수의 '주소'를 넘겨준 것을 알 수 있습니다.

그리고 결과를 보면 swap1 함수는 원래의 의도였던 서로의 값의 위치를 바꾸는 행동을 실패하였고, swap2 함수는 성공한 것을 볼 수 있습니다.

그 이유를 지금부터 설명해보겠습니다.

기본적으로 포인터 변수는 주소값을 저장할 수 있고 그 주소값을 통해 주소값이 가지고 있는 값을 참조하거나

변경 할 수 있습니다.

그리고 함수는 그 함수 내에서 생성된 변수, 데이터들은 return문을 통해 반환하지 않으면 사라집니다.

위의 코드의 swap2를 자세히 살펴보면 num1, num2의 '값'을 '복사'해서 각각의 파라미터 i와 파라미터 j에 저장하였고, 그 값을 통해 i의 값을 temp에 저장하고, 다시 j의 값을 i에 옮기고 temp의 값을 j에 옮김으로서 i와 j의 값 교환을 성공적으로 진행하였지만, 함수가 아무런 값을 반환하지 않고 그대로 종료되므로 그동안 함수 내에서 새로이 생성된 i, j(i와 j를 새로 생성하고 여기에 인수의 값을 넣은 것입니다.) temp와 그 값은 함수가 종료되는 순간 날아가 버립니다...

그러나 swap2함수는 각각의 값의 주소를 넘겨주었고 그 '주소'를 '복사하여' int 포인터형 변수 i와 j에 각각 할당해 준 모습을 볼 수 있습니다. 이후 똑같이 값을 임시 저장할 temp를 선언해주고 포인터형 변수 i에 저장된 '주소'로 그 주소값에 저장된 '값'을 간접적으로 접근하여 그 값을 temp에 저장하였고,

포인터형 변수 i에 저장된 '주소'로 그 주소값에 저장된 '값'을 간접적으로 접근하여 동일한 방식으로 접근한 j가 가리키는 주소의 값을 넣어주었고,

이후 temp의 값을 j가 가리키는 주소의 값에 집어넣고 함수를 종료하였습니다.

이렇게 해주게 되면 temp, i, j(여기서의 i와 j는 주소값을 가지고 있는 포인터 변수이다.)는 함수가 종료되는 순간 날아가 버리지만, 포인터를 통해 직접 main 함수 안에 있는 '값'에 접근하여 직접 바꿔주었으므로 두개의 값의 교환은 성공적으로 이루어진 모습을 볼 수 있습니다.