

컴프 #2 수업에서 다루어지는 것들: 포인터(Pointer) #1

- 포인터
포인터는 메모리 주소를 가지고 있는 변수
변수는 메모리에 저장되며 메모리는 바이트 단위로 액세스 된다.

- 포인터 선언 및 값 설정
주소연산자: &
간접 (참조) 연산자: *
포인터 덧셈/뺄셈의 의미: ++, --, +, -

위의 내용을 메모리로 그리고 아래에 답하라.

p의 값은?

*p의 값은?

*p = 20; 이후 변화를 메모리에 반영하라.

i의 값은?

p++; 이후 변화를 메모리에 반영하라.

- 포인터와 배열
 - 배열의 이름은 배열 첫번째 원소의 주소값을 나타내는 포인터 상수
 - 포인터 연산(예: p++)의 의미와 역할

int array[3] = {10, 20, 30};

int *p = array;

예제 A)B)가 가장 포인터 적임!

// 예제 A) printf("%d\n", *p); p++; printf("%d\n", *p); p++; printf("%d\n", *p); p++;	// 예제 B)
// 예제 C)	// 예제 D)

- ✓ 위의 예제 A)B)C)D)는 같은 출력결과를 낸다.
차이점: A)B)는 p의 값이 변하고 C)D)는 p의 값이 변하지 않는다.

- ✓ 포인터 변수인 p는 마치 배열의 이름처럼 쓰일 수 있다.

*(p + i)	p + i
----------	-------

- more 포인터 연산: *p++ vs. (*p)++
아래의 실행결과를 메모리를 그리면서 예측해보라. 물론 A)는 예측 불가하다.
위의 A) 출력결과를 참조해서 그 후를 예측하라.

```
int main(void)
{
    int data[5] = {10, 20, 30, 40, 50};
    int *p = data;

    printf("%d %u %p\n\n", p, p, p); //A)

    printf("%d %d\n", p, *p);
    printf("%d\n", *p++);
    printf("%d %d\n\n", p, *p);

    p = data; // 다시 첫 번째 요소를 가리키게 한 후
    printf("%d %d\n", p, *p);
    printf("%d\n", (*p)++);
    printf("%d %d\n", p, *p);
}
```

- 포인터의 의미
 - 속도
 - 함수의 매개변수(call by reference)
 - 자기참조 구조체(2학년때 자료구조에서 배움)

※연산자 우선순위

기능별 분류	연산자	결합순서	우선순위
일차식	() [] -> .	→	1
단항 연산자	! ~++ -- - + (형명) * & sizeof	←	2
승제 연산자	* / %	→	3
가감 연산자	+ -	→	4
시프트 연산자	<< >>	→	5
비교 연산자	< <= > >=	→	6
등가 연산자	== !=	→	7
비트 연산자	&	→	8
	^	→	9
		→	10
논리 연산자	&&	→	11
		→	12
조건 연산자	? :	←	13
대입 연산자	= += -= *= /= %=	←	14
	>>= <<= &= ^= !=	←	14
кома 연산자	,	→	15

LAB 7 포인터(1)

- 포인터 기초
- 배열과 포인터

■ LAB7.0(포인터 기초)

아래의 프로그램을 디버깅 하면서

조사식을 사용하여 주석문에 밑줄을 채워넣어보라.

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

```
int main(void)
```

```
{
```

```
    char c = 'A';
```

```
    int n = 100;
```

```
    double d = 3.14;
```

```
    char *pC;
```

```
    int *pN;
```

```
    double *pD;
```

```
    pC = &c;
```

```
    pN = &n;
```

```
    pD = &d;
```

```
    // c, pC, *pC, n, pN, *pN, d, pD, *pD에 어떤 수가 들어가나 예상 후
```

```
    // 조사식창에 위의 이름들을 입력하여 확인
```

```
    // c: _____ pC: _____ *pC _____
```

```
    // n: _____ pN: _____ *pN _____
```

```
    // d: _____ pD: _____ *pD _____
```

```
    *pC = 'Z';
```

```
    *pN = 199;
```

```
    *pD = 3.1415;
```

```
    pC++;
```

```
    pN++;
```

```
    pD++;
```

```
    // pC, pN, pD의 값이 어떻게 바뀌었는가?
```

```
    // pC: _____ pN: _____ pD: _____
```

```
    pC = pC + 2;
```

```
    pN = pN + 2;
```

```
    pD = pD + 2;
```

```
    // pC, pN, pD의 값이 어떻게 바뀌었는가?
```

```
    // pC: _____ pN: _____ pD: _____
```

```
}
```

■ LAB7_1(포인터 기초) 가)나)… 순으로 차례로 코드를 작성해 보시오.

가) 아래와 같은 코드가 컴파일 되어 아래의 출력문이 나오도록 포인터 변수 선언 코드를 추가하고 가)-1과 같은 출력이 나오도록 포인터만을 이용해서 출력문을 완성하라.

나) 아래의 출력문이 나오도록 앞에서 선언한 포인터를 이용하여 값을 적절히 변경하는 코드를 추가하시오.

➔ 이때, VS의 조사식 창에 위의 변수 x, c, f를 입력해 보자(혹은, 마우스를 이용하여 조사식 창에 위치시켜 보자) 그리고 이들 변수들의 값을 조사식 창에서 확인해 보자(배열의 경우 각 엘리먼트의 값이 잘 보이는지 꼭 확인한다.)

➔ 조사식 창에 변수 px, pc, pd를 입력해보고 값(포인터가 가르키는 값)을 확인해보자. 세 변수는 포인터 변수들이기에 그 값이 hexa값으로 표시된다. 윈도우의 계산기 프로그램을 이용하여 이 hexa값들은 십진수로 얼마인지를 적어보고, 이 값이 의미하는 바가 무엇인지 생각해 보자.

다) 포인터 px를 이용하여 배열 x에 있는 엘리먼트의 값을 모두 합산하여 sum에 저장하는 부분을 추가하라.

➔ sum 값이 변화되는 다음 문장에 Breakpoint를 설정(F9), F5를 계속 클릭하면서 반복이 진행되면서 sum값이 변화하는 것을 살펴보자.

```
#include <stdio.h>
int main(void)
```

```
{
```

```
    int x[] = {0, 0, 1, 2, 3};
```

```
    char c[] = "BCDE";
```

```
    double f = 2.3;
```

```
    int i, sum = 0;
```

```
    // 가) 추가
```

```
    px = x;
```

```
    pc = c;
```

```
    pd = &f;
```

```
    printf("가-1) %d %c %.1f\n", x[0], c[0], f);
```

```
    printf("가-2) _____);
```

```
    // 나) 추가
```

```
    printf("나) %d %c %.11f\n", x[0], c[0], f);
```

```
    // 다) 추가
```

```
    printf("다) sum = %d\n", sum); // sum의 값은 5이다. x[0]의 값은 위에서 이미 -1로 수정
```

```
    // 라)
```

```
    px = x;
```

```
    for (i = 0; i < sizeof(x) / sizeof(int); i++)
```

```
    {
```

```
        *px = 100;
```

```
        px++; // 라-B)
```

```
    }
```

```
    // 마)
```

```
    pc = c;
```

```
    for (i = 0; i < sizeof(c) / sizeof(char); i++)
```

```
    {
```

```
        *pc = 'X';
```

```
        pc++; // 마-B)
```

```
    }
```

라) px 값의 변화 살펴보기.

➔ 위 코드의 라-B)에도 Breakpoint를 설정하라(F9 사용). F5를 계속 누르면서 x값의 변화를 관찰하라. 또한, 수행 시에 px의 값이 어떻게 변화하는지 종이에 적어보라. 그리고, for 문의 수행 후에 x의 값이 어떻게 변했는지 조사식 창을 사용하여 확인해 보라.

마) pc 값의 변화 살펴보기

➔ 라)에서와 같이 코드의 마-B)에도 Breakpoint를 설정하라(F9 사용). F5를 계속 누르면서 c값의 변화를 관찰하라. 수행시에 pc 값이 어떻게 변화하는지 종이에 적어보라. 그리고, for 문 수행 후 c의 값이 어떻게 변했는지 조사식 창을 사용하여 확인해 보라.

라)번과 마)번에서 px와 pc의 값이 서로 다르게 변화함을 알 수 있을 것이다. 그 이유는 무엇인가?

■ LAB7_2 아래 코드를 작성하고 질문에 답해보라

```
#include <stdio.h>

void main(void)
{
    int n[] = {-1, 1, 2, 3, 4 };
    int *p1 = n;
    int *p2 = p1;

    printf("%d\n", n[0]);
    printf("%d\n", (*p1)++);
    printf("%d\n", n[0]);

    printf("%d\n", *p2++);
    // *(간접지정 연산)과 ++은 우선순위(p69)가 같다. 이때 결합방향이 ← 이므로 오른 쪽에
    // 위치한 ++를 먼저 수행하고 그다음 *를 수행한다. 즉 *(p2++)와 같다.
    printf("%d\n", *p2);
    printf("%d\n", **p1);
    printf("%d\n", *p1);
}
```

가) 코드를 보고, 출력될 결과를 예상하여 종이에 적어보라.

나) 이제 빌드를 하여 프로그램을 수행해 보라. 그리고, 자신의 예상과 비교해 보라.

다) 다시 이제 F10 키로 한 라인씩 수행해 보면서, 자신의 예상과 틀린 부분을 하나하나 확인해 보고 왜 다른 결과가 나온 것인지 이해하도록 해 보시오(이해가 안된 부분은 질문을 통해 해결하라).

잔소리) 위의 코드 수행 결과 정도는 완전히 이해 할 수 있어야 중간고사 문제를 풀 수 있다.

■ LAB7_3 여지껏 포인터를 실습하면서 어렵다고, 복잡하다고 느꼈을 것이다. 위의 문제들은 포인터를 연습하기위한 인위적인 문제들이었다. 이제, 포인터의 *기본적인* 활용법을 익혀보자. 1차원 배열과 2차원 배열에 포인터를 활용하는 연습이다. A) 부분은 LAB7_1의 라)에서 다룬 부분이다.

```
#include <stdio.h>

int main(void)
{
    int *pi;
    int arr1[] = {10, 20, 30, 40, 50};
    int arr2[][3] = {{1, 2, 3}, {10, 20, 30}, {100, 200, 300}, {1000, 2000, 3000}};

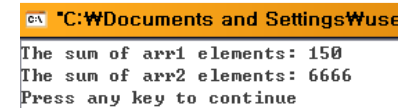
    int i, sum;

    sum = 0;
    pi = arr1; // pi = &arr1[0];
    for (i = 0; i < sizeof(arr1)/sizeof(int); i++)
        _____ // A) arr1을 사용하지말고 pi을 사용하라.

    printf("The sum of arr1 elements: %d\n", sum);

    sum = 0;
    pi = (int *)arr2; // pi = &arr[0][0]
    for (i = 0; i < sizeof(arr2)/sizeof(int); i++)
        _____ // B) arr2를 사용하지말고 pi를 사용하라.

    printf("The sum of arr2 elements: %d\n", sum);
}
```



```
C:\Windows and Settings\Wuse
The sum of arr1 elements: 150
The sum of arr2 elements: 6666
Press any key to continue
```

가) 디버거를 사용하여 위의 프로그램의 흐름 및 포인터 및 sum의 변화들을 관찰하라.

나) A)와 B)를 비교하라.

1차원배열 arr1, 2차원 배열 arr2는 둘다 동일하게 엘리먼트들은 메모리에 연속적으로 저장된다. 이 예제에서는 이처럼 연속적으로 저장된 엘리먼트들을 차례로 스캔하는데있어 포인터 pi와 포인터 연산 ++이 쓰임새를 익힌다. 이처럼 포인터는 배열의 엘리먼트들처럼 연속된 메모리의 내용들을 접근하는데 유용하게 사용된다

■ LAB7_4 (1차원 배열과 포인터)아래의 주어진 지시에 따르시오.(hw 5_1과 연관됨)

크기가 10인 data 배열에 난수를 넣고(generateData 함수)

이를 출력하고,(printData함수)

엘리먼트들의 합을 구하여(totalData함수)

출력하는 프로그램을 작성하라.

주어진 변수 외에는 어떤 변수도 추가로 사용하지 마시오. 그리고 data는 더 사용하지 말고 p나 pi를 이용하여 (p 나 pi 를 변화시키면서... 교안의 예제 B)처럼) data 배열 값에 접근/수정한다. 또한 [,]도 더 사용하지 말라.

➔ 프로그램을 완성한 후, 메인 함수 부분을 F10으로 수행해 보라. 변수 data의 엘리먼트 변화를 VS를 통해 확인해 보라. (조사식 창에 data를 추가해본다)

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

```
void generateData();
void printData();
int totalData();
int data[10]; // 계산 수행에 사용할 전역 변수
```

```
int main(void)
{
    srand(200); // random 값 출력에 사용하는 함수. Seed 값을 부여
    generateData();

    printf("발생된 10개의 난수:\n");
    printData();

    printf("10개 난수의 합 = %d \n", totalData());
}
```

```
// generateData 함수는 data[0]..data[9]에 난수를 넣는다.
void generateData()
{
    int k;
    int *p;
    p = data; //혹은 p = &data[0]
    for (k = 0; k < 10; k++) // Index 0..9까지 난수값 부여
        ...
}
```

```
// printData 함수는 data[0]..data[9]의 값을 출력한다.
void printData()
{
    int *pi = data;
    int i;

    for (i = 0; i < 10; i++)
        ...
}
```

```
// totalData 함수는 data[0]에서 data[9]까지의 값을 모두 더해 그 결과를 반환한다.
int totalData()
{
    int *pi = data, sum = 0;
    int i;

    for (i = 0; i < 10; i++)
        ...
}
```

HW 7 포인터(1)

■ HW7_1 (1차원 배열과 포인터) LAB7_4에

10개의 엘리먼트중 가장 큰 수를 찾아 출력하는 기능을 추가하는 프로그램을 작성하라.
위의 LAB7_4 처럼 포인터를 사용하라.

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

```
void generateData();
void printData();
int totalData();
int maxData();
int data[10]; // 계산 수행에 사용할 전역 변수
```

```
int main(void)
{
    srand(200);
    generateData();
    printf("발생된 10개의 난수:\n");
    printData();
    printf("10개 난수의 합 = %d \n", totalData());
    printf("10개 난수중 가장 큰 수 = %d \n", maxData());
}
```

```
C:\WINDOWS\system32\cmd.exe
발생된 10개의 난수:
91 80 60 71 86 11 63 86 0 14
10개 난수의 합 = 562
10개 난수중 가장 큰 수 = 91
계속하려면 아무 키나 누르십시오 . . .
```

■ HW7_2 (2차원 배열과 포인터)

다음과 같은 2차원 배열 `int data[3][10]`에
0에서 99사이의 난수를 저장한후(`generateData`함수)
출력하고(`printData`함수)
전체의 합을 계산(`totalData`함수)해서
출력하는 프로그램을 작성하라.

```
C:\WINDOWS\system32\cmd.exe
91 80 60 71 86 11 63 86 0 14
40 42 37 46 22 31 59 46 92 22
17 18 65 2 0 23 50 77 70 91

전체의 합은: 1412
계속하려면 아무 키나 누르십시오 . . .
```

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

void generateData();
void printData();
int totalData();
static int data[3][10]; // 계산 수행에 사용할 정적 변수

void main()
{
    srand(200); // random 값 출력에 사용하는 함수. Seed 값을 부여

    generateData();
    printData();
    printf("전체의 합은: %d\n", totalData());
}

void generateData()
{
    int *p = &data[0][0];
    int i;
    for (i = 0; i < 30; i++)
        ...
}

void printData()
{
    int i, j;
    int *p = &data[0][0];
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 10; j++)
            ...
    }
}

int totalData()
{
    int *p = &data[0][0];
    int i, total = 0;
    for (i = 0; i < 30; i++)
        ...
}
```

LAB1(재귀)

- (실습) 자연수 n 을 입력받아 $sum = 12 + 22 + \dots + n^2$ 를 계산하여 출력하되 재귀 함수로 구현해보라. `int sum(int n);`

실행예:

5

160

실행예:

6

222

- (숙제) 위의 문제에서 각 항을 출력하도록 코드를 수정하라.

실행예:

5

12 22 32 42 52

160

실행예:

6

12 22 32 42 52 62

222

LAB2(재귀)

- 10진수를 2진수로 변환하여 출력하는 함수를 작성하되 재귀 호출에 의해 구현하라.

`void to_binary(int n);`

실행예:

16

1 0 0 0 0

실행예:

15

1 1 1 1

PROJECT 04-2(SNS 발전)

■ Project4_2 (지구촌은 몇 단계 ?)

지난 과제에서 1촌의 1촌 즉 2촌까지의 관계를 표현하는 프로그램을 과제로 하였다. 이번 과제는 지난 과제를 활용하여 모든 사용자가 최대 몇 촌 관계인지 알아내는 과제를 할 것이다. 과거 연구에 따르면 SNS 상의 사용자는 평균 5명 정도의 1촌을 맺고 있고, 5~6 단계를 거치면 모든 사용자가 다른 사용자와 연결될 수 있다고 한다. 이번 과제는 랜덤하게 생성된 1촌 관계가 몇 번의 연결을 통하면 모두 알게 되는지를 구하는 프로그램을 작성하는 것이다. 즉, 1촌 관계를 표시하는 행렬 A가 몇 번의 논리곱을 하면 모든 원소가 1이 되는지를 구하는 것이다.

이번 과제도 main 함수는 주어지고 나머지 함수를 작성하는 것이 여러분이 해야할 일이다.

아래 오른편의 실행에는 사용자를 15으로 한 경우 모두가 친구가 되는 과정을 보여주는 예이다. 모두가 친구가 되면(즉, 모두 1이 되면) 실행을 끝내는 것으로 한다.

```
#define NUM_OF_MEMBERS 15
```

```
// 나중에 70, 100, 300,, 500, 1000으로 바꾸어 실행시켜 보라
```

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

// 여기서는 함수 정의를 앞부분에 배치한다. 즉 원형은 불필요하다.

```
void print_links(int data[][NUM_OF_MEMBERS])
{
    //앞의 프로젝트에서 정의
}
```

```
void matrix_multiplication(int data1[][NUM_OF_MEMBERS],
                           int data2[][NUM_OF_MEMBERS],
                           int result[][NUM_OF_MEMBERS])
{
    //앞의 프로젝트에서 정의
}
```

```
void matrix_copy(int dest[][NUM_OF_MEMBERS],
                 int src[][NUM_OF_MEMBERS])
{
    //구현해야할 함수
    //src의 모든 내용을 dest로 복사
}
```

```
int check_links(int data[][NUM_OF_MEMBERS])
{
    //구현해야할 함수
    //data의 모든 원소가 1이면 1 리턴
    //하나라도 0이면 0 리턴
}
```

cmd C:\windows\system32\cmd

초기 1촌 상태:

```
110011000100000
110000001101000
001010000100000
000100111000010
101010000010000
100001000010011
000100101100000
000100010001001
010100101000100
111000100101000
000011000010101
010000010101100
000000001011110
000101000000111
000001010010011
```

1 steps:

```
111011101111011
111111111101100
111010100111000
010101111101111
111011000110101
110111010110111
111100111101110
010101111111111
110100111111110
111111111101100
101011011011111
111100111111111
010111111111111
100101111011111
10011110110011111
```

2 steps:

```
111111111111111
111111111111111
111111111111101
111101111111111
111011111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
110111111111111
111111111111111
```

3 steps:

```
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
111111111111111
```

It takes 3 steps.

계속하려면 아무 키

```
int main( void )
```

```
{
```

```
    int link_data[NUM_OF_MEMBERS][NUM_OF_MEMBERS] = {0};
    int link_data2[NUM_OF_MEMBERS][NUM_OF_MEMBERS] = {0};
    int link_result[NUM_OF_MEMBERS][NUM_OF_MEMBERS] = {0};
```

```
    int i = 0, j = 0;
    int num_of_steps=0;
    int ALL_ONES=0;
```

```
    //srand( (unsigned int)time(NULL) );
    srand(100);
    for(i = 0; i<NUM_OF_MEMBERS; i+ + )
        link_data[i][i] = 1; // 역 대각선 셀들은 모두 1로(자신은 자기 자신과 1촌)
```

```
    for(i = 0; i<NUM_OF_MEMBERS; i+ + )
    {
```

```
        j=0;
        while ( j<2 )
        {
            int new_link = rand()%NUM_OF_MEMBERS;
            if( new_link != i )
            {
                link_data[i][new_link] = 1;
                link_data[new_link][i] = 1;
                j+ + ;
            }
        }
    }
```

```
    //printf("Wn초기 1촌 상태:Wn");
    //print_links(link_data); // 사용자가 70명 이상이면 출력이 매끄럽지 않으니 주석문으로 처리한다.
    matrix_copy(link_data2, link_data );
    while(1)
    {
```

```
        num_of_steps+ + ;
```

```
        matrix_multiplication2(link_data, link_data2, link_result);
```

```
        printf("Wn%d steps:Wn", num_of_steps);
        // print_links(link_result); // 사용자가 70명 이상이면 주석처리
```

```
        ALL_ONES = check_links(link_result);
        if( ALL_ONES ) break;
```

```
        matrix_copy(link_data2, link_result);
```

```
    }
    printf("It takes %d steps.Wn", num_of_steps);
}
```

사용자를 70으로 바꾸고 (배열 출력을 생략하여) 실행시킨 결과는 다음과 같다. 즉, 70명의 사용자가 평균 4명의 1촌을 가지고 있을 때, 5단계를 거치면 모든 다른 사용자와 연결될 수 있다는 것을 의미한다.

이제 NUM_OF_MEMBERS을 70(5 steps)에서 200(6 steps), 300, 500으로 키워보면서 프로그램을 수행시켜보아라. 아마 1000쯤 되면 몹시 수행 속도가 느림을 느낄 것이다. 이것은 matrix 논리곱을 하는 operation이 복잡도가 크기 때문이다.

100만 사용자면 어떻게 할까? 이산수학, 알고리즘, 병렬 프로그래밍, 빅데이터가 이 문제를 푸는 도구를 제공하니 해당 수업을 열심히 수강하길 바란다.

cmd C:\windows\system32\cmd.exe

1 steps:

2 steps:

3 steps:

4 steps:

5 steps:
It takes 5 steps.
계속하려면 아무 키나 누르