『제 24기 청학동』Tutoring 3주 주간 학습 보고서

학습활동 참여 점검			
팀명	2+3=5		
1회차 일시	2022.04.05 18:00 ~ 21:00	1회차 장소	804호
1회차 참석자	신수빈,변장무,이준호,이동현,권재현		
1회차 불참자			
2회차 일시		2회차 장소	
2회차 참석자			
2회차 불참자			
3회차 일시		3회차 장소	
3회차 참석자			
3회차 불참자			
4회차 일시		4회차 장소	
4회차 참석자			
4회차 불참자			
5회차 일시		5회차 장소	
5회차 참석자			
5회차 불참자			
학습목표	1) 배열, 2)구조체, 3)포인터		
과제물	여러 종류의 포인터에 대해 알아보기		

『제 24기 청학동』Tutoring 3주 주간 학습 보고서

학습활동 참여 점검	
팀명	2+3=5

3주차

1. 배열 (Array)

배열은 거의 모든 프로그래밍 언어에서 기본적으로 제공되는 데이터 타입이다.
배열은 기본이 되는 중요한 자료형으로서 많은 자료 구조들이 배열을 사용하여 구현된다.
배열은 동일한 타입의 데이터를 한 번에 여러 개 만들 때 사용된다.

Ex_1) int A[6];

1) 배열의 이름 : A

2) 배열의 타입 : 정수형 (한 요소 당 4Byte)

3) 배열의 크기 : 24Byte (6*4Byte)

4) 배열의 특성 : Auto형 데이터구조 (소속된 함수가 시작되면 메모리에 할당됨)

5) 배열의 scope : 배열이 소속된 함수 공간

6) 배열의 life-time : 배열이 메모리에 저장되어 있는 기간

7) 기타 등등

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

Base+0*sizeof(int) Base+1*sizeof(int) Base+2*sizeof(int) Base+3*sizeof(int) Base+4*sizeof(int) Base+5*sizeof(int)

Ex_2) int B[3][4]

B[0][0]	B[0][1]	B[0][2]	B[0][3]

Base+0*sizeof(int)+offset Base+1*sizeof(int)+offset Base+2*sizeof(int)+offset Base+3*sizeof(int)+offset

B[1][0]	B[1][1]	B[1][2]	B[1][3]

Base+4*sizeof(int)+offset Base+5*sizeof(int)+offset Base+6*sizeof(int)+offset Base+7*sizeof(int)+offset

B[2][0]	B[2][1]	B[2][2]	B[2][3]

Base+8*sizeof(int)+offset Base+9*sizeof(int)+offset Base+10*sizeof(int)+offset Base+11*sizeof(int)+offset

배열의 응용 1. 다항식 계산

Polynomial1

모든 차수에 대한 계수 값을 배열로 저장한다.

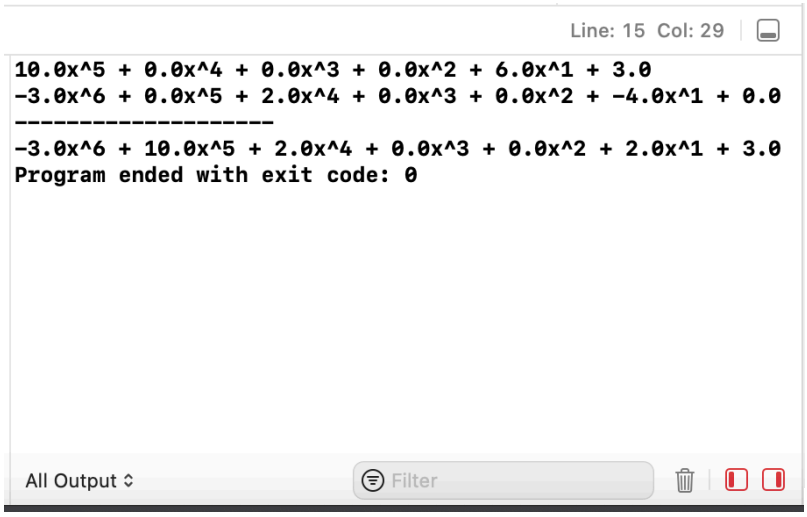
하나의 다항식을 하나의 배열로 표현한다.

단점 : 대부분의 항의 개수가 0일 때, 공간의 낭비가 심하다.

다항식 계산 1.1

```
#include
#define MAX(a,b) (((a)>(b))?(a):(b))
#define MAX_DEGREE 101

typedefstruct
{
    int degree;
    float coef [MAX_DEGREE];
} polynomial;

polynomial poly_add1(polynomial A, polynomial B)
{
    polynomial C;
    int Apos = 0, Bpos = 0, Cpos = 0;
    int degree_a = A.degree;
    int degree_b = B.degree;
    C.degree = MAX(A.degree, B.degree);

    while(Apos <= A.degree && Bpos <= B.degree)
    {
        if(degree_a > degree_b)
        {
            C.coef[Cpos++] = A.coef[Apos++];
            degree_a--;
        }

        else if(degree_a == degree_b)
        {
            C.coef[Cpos++] = A.coef[Apos++] + B.coef[Bpos++];
            degree_a--; degree_b--;
        }
```

```
        else
        {
            C.coef[Cpos++] = B.coef[Bpos++];
            degree_b--;
        }
    }
    return C;
}

void print_poly(polynomial p)
{
    int i;

    for(i=p.degree;i>0;i--)
        printf("%3.1fx^%d + ", p.coef[p.degree-i], i);

    printf("%3.1f ₩n", p.coef[p.degree]);
}

int main(void)
{
    polynomial a = { 5, {10, 0, 0, 0, 6, 3} };
    polynomial b = { 6, {-3, 0, 2, 0, 0, -4, 0} };
    polynomial c;

    print_poly(a);
    print_poly(b);

    c = poly_add1(a, b);

    printf("--------------------₩n");
    print_poly(c);

    return 0;
}
```

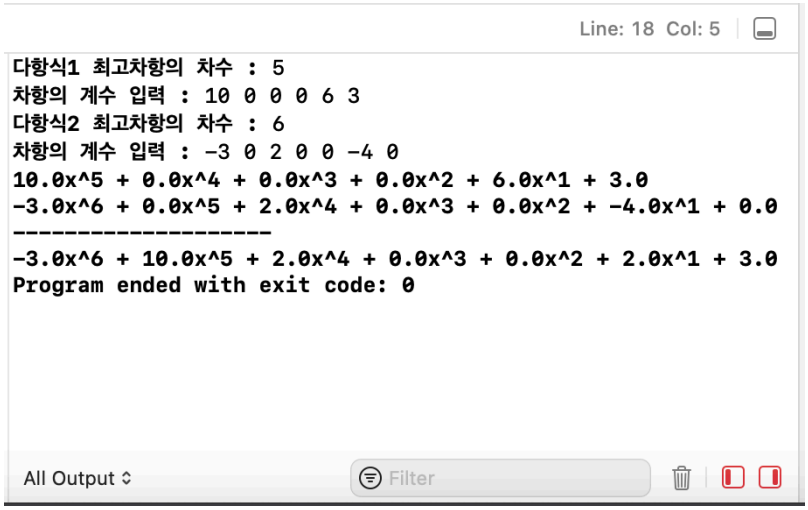


다항식 계산 1.2 (1.1을 직접 입력받는 응용ver)

```
#include
#define MAX(a,b) (((a)>(b))?(a):(b))
#define MAX_DEGREE 101

typedefstruct
{
    int degree;
    float coef [MAX_DEGREE];
} polynomial;

polynomial poly_add1(polynomial A, polynomial B)
{
    polynomial C;
    int Apos = 0, Bpos = 0, Cpos = 0;
    int degree_a = A.degree;
    int degree_b = B.degree;
    C.degree = MAX(A.degree, B.degree);

    while(Apos <= A.degree && Bpos <= B.degree)
    {
        if(degree_a > degree_b)
        {
            C.coef[Cpos++] = A.coef[Apos++];
            degree_a--;
        }
```

```
        else if(degree_a == degree_b)
        {
            C.coef[Cpos++] = A.coef[Apos++] + B.coef[Bpos++];
            degree_a--; degree_b--;
        }

        else
        {
            C.coef[Cpos++] = B.coef[Bpos++];
            degree_b--;
        }
    }
    return C;
}

void print_poly(polynomial p)
{
    int i;

    for(i=p.degree;i>0;i--)
        printf("%3.1fx^%d + ", p.coef[p.degree-i], i);

    printf("%3.1f ₩n", p.coef[p.degree]);
}

intmain(void)
{
    polynomial a, b, c;
    int i, j;

    printf("다항식1 최고차항의 차수 : ");
    scanf("%d", &a.degree);
    printf("차항의 계수 입력 : ");
    for(i=0;i<=a.degree;i++)
        scanf("%f", &a.coef[i]);
```

```
    printf("다항식2 최고차항의 차수 : ");
    scanf("%d", &b.degree);
    printf("차항의 계수 입력 : ");
    for(j=0;j<=b.degree;j++)
        scanf("%f", &b.coef[j]);

    print_poly(a);
    print_poly(b);

    c = poly_add1(a, b);

    printf("--------------------₩n");
    print_poly(c);

    return 0;
}
```



Polynomial2

다항식에서 0이 아닌 항만을 배열에 저장한다.

하나의 배열로 여러 개의 다항식을 나타낼 수 있다.

다항식의 항을 (계수, 차수) 형식으로 구분하여 배열에 저장한다.

장점 : 메모리 공간을 효율적으로 이용할 수 있다.

단점 : 다항식의 연산들이 복잡해진다.

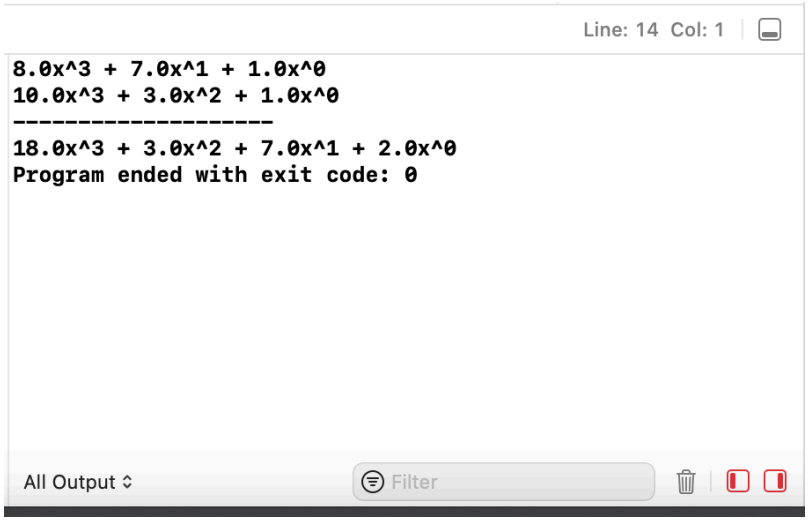
다항식 계산 2.1

```
#include
#include
```

```
#define MAX_TERMS 101

typedefstruct
{
    float coef;
    int expon;
} polynomial;

polynomial terms[MAX_TERMS] = { {8, 3}, {7, 1}, {1, 0}, {10, 3}, {3, 2}, {1, 0} };
int avail = 6;

charcompare(int a, int b)
{
    if(a > b) return '>';
    else if(a == b) return '=';
    elsereturn'<';
}

void attach(float coef, int expon)
{
    if(avail > MAX_TERMS)
    {
        fprintf(stderr, "항의 개수가 너무 많음₩n");
        exit(1);
    }

    terms[avail].coef = coef;
    terms[avail].expon = expon;
    avail++;
}

void poly_add2(int As, int Ae, int Bs, int Be, int *Cs, int *Ce)
{
    float tempcoef;
    *Cs = avail;

    while(As <= Ae && Bs <= Be)
```

```
        switch (compare(terms[As].expon, terms[Bs].expon))
        {
            case '>' :
                attach(terms[As].coef, terms[As].expon);
                As++;
                break;

            case '=' :
                tempcoef = terms[As].coef + terms[Bs].coef;
                if(tempcoef)
                    attach(tempcoef, terms[As].expon);
                As++;
                Bs++;
                break;

            case '<' :
                attach(terms[Bs].coef, terms[Bs].expon);
                Bs++;
                break;
        }

    for(;As<=Ae;As++)
        attach(terms[As].coef, terms[As].expon);

    for (;Bs<=Be;Bs++)
        attach(terms[Bs].coef, terms[Bs].expon);

    *Ce = avail - 1;
}

voidprint_poly(int s, int e)
{
    int i;
    for(i=s;i<e;i++)
        printf("%3.1fx^%d + ", terms[i].coef, terms[i].expon);
    printf("%3.1fx^%d₩n", terms[e].coef, terms[e].expon);
}
```

```
intmain(void)
{
    int As = 0, Ae = 2, Bs = 3, Be = 5, Cs, Ce;

    poly_add2(As, Ae, Bs, Be, &Cs, &Ce);
    print_poly(As, Ae);
    print_poly(Bs, Be);
    printf("-------------------₩n");
    print_poly(Cs, Ce);

    return 0;
}
```



다항식 계산 2.2 (1.1을 직접 입력받는 응용ver)

```
#include
#include

#define MAX_TERMS 101

typedefstruct
{
    float coef;
    int expon;
} polynomial;

polynomial terms[MAX_TERMS];
```

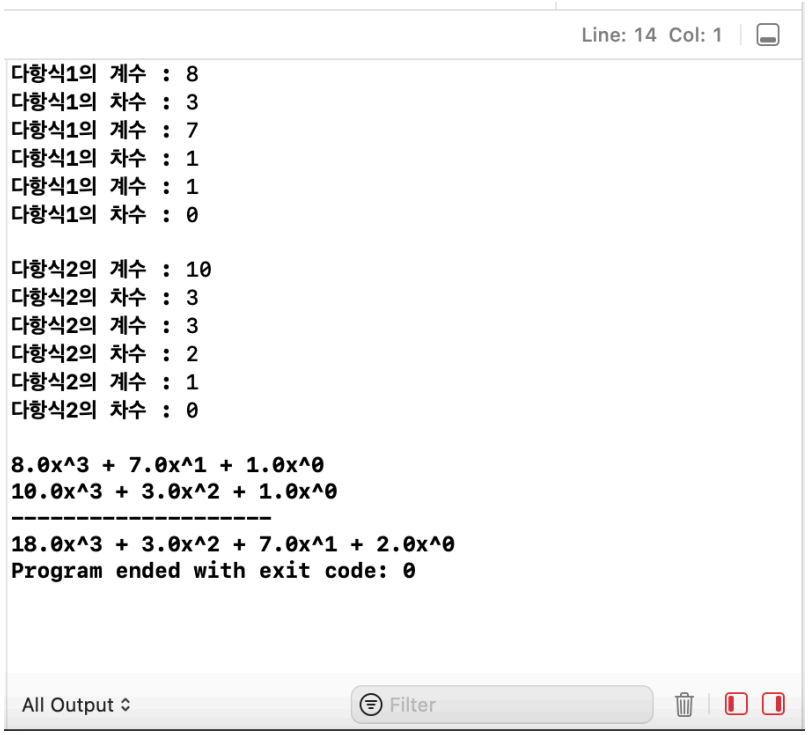
학습 내용

```
int As, Ae, Bs, Be, Cs, Ce, avail;

charcompare(int a, int b)
{
    if(a > b) return '>';
    else if(a == b) return '=';
    elsereturn'<';
}

void attach(float coef, int expon)
{
    terms[avail].coef = coef;
    terms[avail].expon = expon;
    avail++;
}

void poly_add2(int As, int Ae, int Bs, int Be, int *Cs, int *Ce)
{
    float tempcoef;
    *Cs = avail;

    while(As <= Ae && Bs <= Be)
        switch (compare(terms[As].expon, terms[Bs].expon))
        {
            case '>' :
                attach(terms[As].coef, terms[As].expon);
                As++;
                break;

            case '=' :
                tempcoef = terms[As].coef + terms[Bs].coef;
                if(tempcoef)
                    attach(tempcoef, terms[As].expon);
                As++;
                Bs++;
                break;
```

```
            case '<' :
                attach(terms[Bs].coef, terms[Bs].expon);
                Bs++;
                break;
        }

    for(;As<=Ae;As++)
        attach(terms[As].coef, terms[As].expon);

    for (;Bs<=Be;Bs++)
        attach(terms[Bs].coef, terms[Bs].expon);

    *Ce = avail - 1;
}

voidprint_poly(int s, int e)
{
    int i;
    for(i=s;i<e;i++)
        printf("%3.1fx^%d + ", terms[i].coef, terms[i].expon);
    printf("%3.1fx^%d₩n", terms[e].coef, terms[e].expon);
}

intmain(void)
{
    float v1;
    int v2;

    avail = As = 0;

    while(1)
    {
        printf("다항식1의 계수 : ");
        scanf("%f", &v1);
        printf("다항식1의 차수 : ");
        scanf("%d", &v2);
```

```
        terms[avail].coef = v1;
        terms[avail].expon = v2;

        avail++;

        if(v2 == 0)
        {
            Ae = avail - 1;
            Bs = avail;
            break;
        }
    }

    printf("₩n");

    while(1)
    {
        printf("다항식2의 계수 : ");
        scanf("%f", &v1);
        printf("다항식2의 차수 : ");
        scanf("%d", &v2);

        terms[avail].coef = v1;
        terms[avail].expon = v2;

        avail++;

        if(v2 == 0)
        {
            Be = avail - 1;
            break;
        }
    }

    printf("₩n");

    poly_add2(As, Ae, Bs, Be, &Cs, &Ce);
```

```
    print_poly(As, Ae);
    print_poly(Bs, Be);
    printf("--------------------₩n");
    print_poly(Cs, Ce);

    return 0;
}
```



배열의 응용 2. 희소 행렬

Matrix1

2차원 배열을 이용하여 배열의 전체 요소를 저장하는 방법이다.

장점 : 행렬을 저장하여 행렬에 대한 연산들을 간단하게 구현할 수 있다.

단점 : 대부분의 값들이 0인 희소 행렬인 경우, 메모리 공간 낭비가 많다.

희소행렬1

```
#include
#define ROWS 3
#define COLS 3

int r, c;

void matrix_transpose(int A[ROWS][COLS], int B[ROWS][COLS])
{
```

```
    for(r=0;r<ROWS;r++)
        for(c=0;c<COLS;c++)
            B[c][r] = A[r][c];
}

void matrix_print(int A[ROWS][COLS])
{
    printf("====================₩n");
    for(r=0;r<ROWS;r++)
    {
        for(c=0;c<COLS;c++)
            printf("%d ", A[r][c]);
        printf("₩n");
    }
    printf("====================₩n");
}

intmain(void)
{
    int array1[ROWS][COLS] = { {2, 3, 0}, {8, 9, 1}, {7, 0, 5} };
    int array2[ROWS][COLS];

    matrix_transpose(array1, array2);
    matrix_print(array1);
    matrix_print(array2);

    return 0;
}
```

Matrix2

행렬의 값(요소)들을 저장하는 방법이다.

행렬의 값이 0인 경우가 많다면 0이 아닌 값만 저장하고, 행렬의 값이 0인 경우가 적다면 모든 값을 저장하는 것이 유리하다.

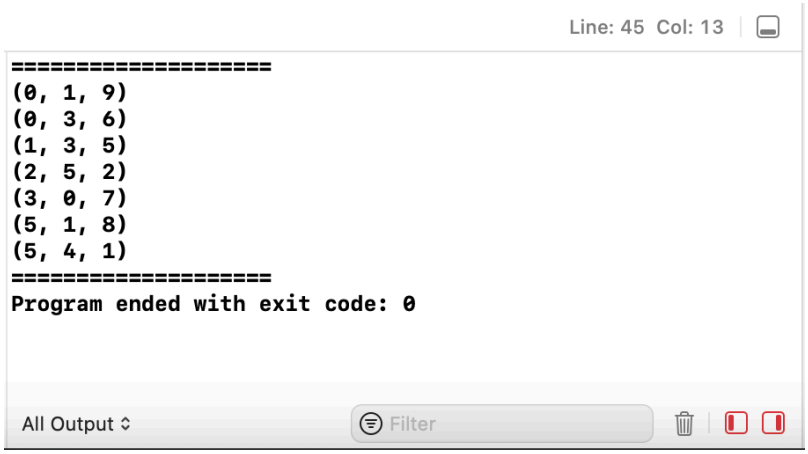
장점 : 희소 행렬의 경우, 메모리 공간을 절약할 수 있다.

단점 : 다양한 행렬 연산들을 구현하기가 어려워진다.

희소행렬 2

```
#include
#include

#define MAX_TERMS 100

typedefstruct
{
    int row, col, value;
} element;

typedefstructSparseMatrix
{
    element data[MAX_TERMS];
    int rows, cols, terms;
} SparseMatrix;

SparseMatrixmatrix_transpose2(SparseMatrix a)
{
    SparseMatrix b;

    int c, i, bindex;

    b.rows = a.rows;
    b.cols = a.cols;
    b.terms = a.terms;

    if(a.terms > 0)
    {
        bindex = 0;
        for(c=0;c<a.cols;c++)
        {
            for(i=0;i<a.terms;i++)
```

```
            {
                if(a.data[i].col == c)
                {
                    b.data[bindex].row = a.data[i].col;
                    b.data[bindex].col = a.data[i].row;
                    b.data[bindex].value = a.data[i].value;
                    bindex++;
                }
            }
        }
    }

    return b;
}

voidmatrix_print(SparseMatrix a)
{
    int i;
    printf("====================₩n");
    for(i=0;i<a.terms;i++)
        printf("(%d, %d, %d) ₩n", a.data[i].row, a.data[i].col, a.data[i].value);
    printf("====================₩n");
}

intmain(void)
{
    SparseMatrix m = { { {0, 3, 7}, {1, 0, 9}, {1, 5, 8}, {3, 0, 6}, {3, 1, 5}, {4, 5, 1}, {5, 2, 2}
}, 6, 6, 7};

    SparseMatrix result;

    result = matrix_transpose2(m);
    matrix_print(result);

    return 0;
}
```



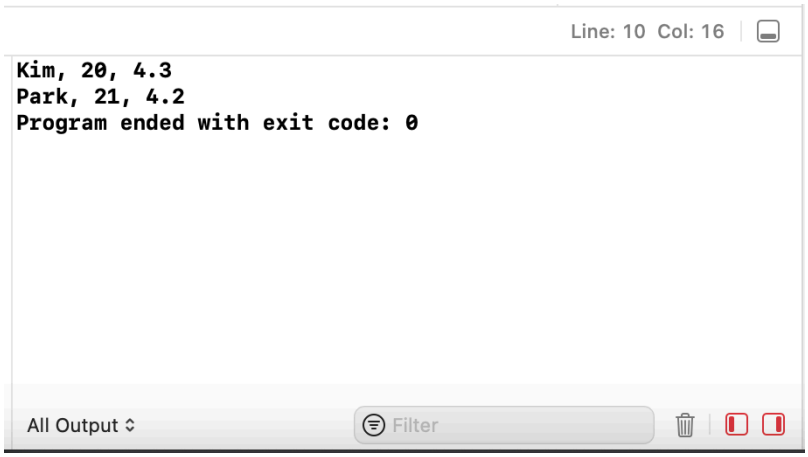
2. 구조체 (Structure)

복잡한 객체에는 다양한 타입의 데이터들이 한데 묶어져서 있다.

배열이 같은 타입의 데이터 모임이라면 구조체는 다른 타입의 데이터를 묶는 방법이다.

```
#include

typedefstructstudentTag
{
    char name[10];
    int age;
    double gpa;
} student;

voidprint_student(student p)
{
    printf("%s, %d, %0.1f ₩n", p.name, p.age, p.gpa);
}

intmain (void)
{
    student a = {"Kim", 20, 4.3};
    student b = {"Park", 21, 4.2};

    print_student(a);
    print_student(b);
```

```
    return 0;
}
```



자체 참조 구조체(Self-Referential Structure)

필드 중에 자기 자신을 가리키는 포인터가 존재하는 구조체
연결 리스트, 트리 구조에 많이 사용된다.

3. 포인터 (Pointer)

포인터는 다른 변수의 주소를 가지고 있는 변수이다.
모든 변수는 메모리 공간에 저장되고 메모리의 각 바이트에는 주소가 매겨져 있다.
이 주소는 포인터에 저장된다.
주소는 컴퓨터에 따라 다를 수 있으므로 포인터 변수는 대개 정확한 숫자가 아닌 화살표로 그려진다.
모든 변수는 주소를 가지고 있다.
컴퓨터 메모리는 바이트로 구성되어 있고 각 바이트마다 순차적으로 주소가 매겨져 있다.

&연산자 : 변수의 주소를 지정한다.

*연산자 : 포인터가 가리키는 장소의 내용을 지정한다.

p
    // 포인터

*p
    // 포인터가 가리키는 (장소에 저장된) 값

*p++
    // 포인터가 가리키는 값을 가져온 다음, 포인터를 1만큼 증가한다.

(*p)++
    // 포인터가 가리키는 값을 증가시킨다.

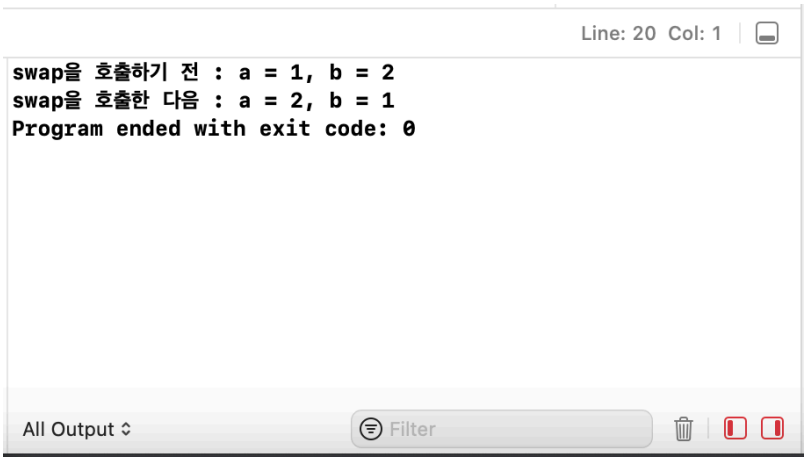
int a;
    // 정수 변수 선언

int *p;
    // 정수 포인터 선언

int **pp;

```
    // 정수 포인터(가 저장된 장소를 가리키는) 포인터 선언
p = &a;
    // 변수 a가 저장된 주소를 포인터 p에 저장
pp = &p;
    // 포인터 p가 저장된 주소를 포인터 pp에 저장

void *p;
    // p는 아무것도 가리키지 않는 포인터
int *pi;
    // pi는 정수 변수를 가리키는 포인터
float *pf;
    // pf는 실수 변수를 가리키는 포인터
char *pc;
    // pc는 문자 변수를 가리키는 포인터
int **pp;
    // pp는 포인터를 가리키는 포인터
struct test *ps;
    // ps는 test 타입의 구조체를 가리키는 포인터
void (*f)(int);
    // f는 함수를 가리키는 포인터

    #include <stdio.h>

    void swap(int *px, int *py)
    {
        int tmp;
        tmp = *px;
        *px = *py;
        *py = tmp;
    }

    intmain(void)
    {
        int a = 1, b = 2;
        printf("swap을 호출하기 전 : a = %d, b = %d ₩n", a, b);
        swap(&a, &b);
        printf("swap을 호출한 다음 : a = %d, b = %d ₩n", a, b);

        return 0;
```

```
}
```


Line: 20 Col: 1
swap을 호출하기 전 : a = 1, b = 2
swap을 호출한 다음 : a = 2, b = 1
Program ended with exit code: 0
All Output
Filter

『제 24기 청학동』Tutoring 3주 주간 학습 보고서

학습활동 참여 점검	
팀명	2+3=5
성찰활동 잘한 점	.
성찰활동 개선할 점	.
구성원 학습활동 사진	
다음 학습할 내용	리스트 - 1) 배열 리스트, 2) 단순 연결 리스트, 3) 원형 연결 리스트, 4) 이중 연결 리스트, 5) 연결리스트로 구현한 스택, 6) 연결리스트로 구현한 큐