

몇번째 작은수

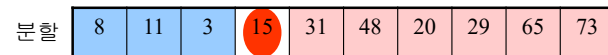
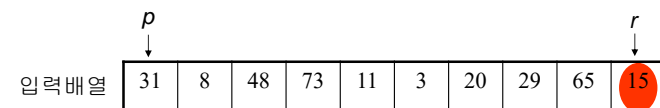
- 조금 거친 문제

Selection (i 번째 작은 수 찾기)

- 배열 $A[p \dots r]$ 에서 i 번째 작은 원소를 찾는다
- quick 정렬의 원리와 비슷

작동 예 1

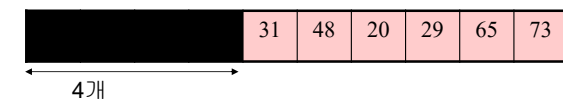
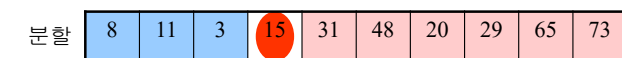
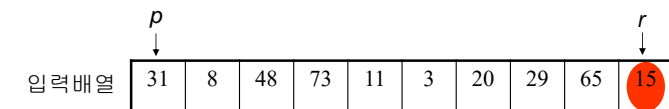
2번째 작은 원소 찾기



왼쪽 그룹에서 2번째 작은 원소를 찾는다

작동 예 2

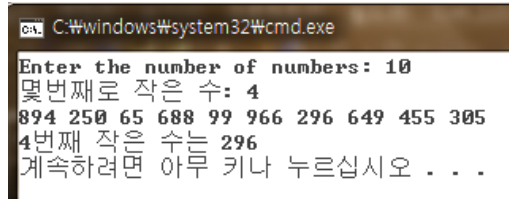
7번째 작은 원소 찾기



7번째(즉, 오른쪽 그룹에서 3번째) 작은 원소를 찾는다

HW(몇 번째 작은수)

- 10개의 난수(100보다 작은)를 발생시키고 몇 번째 작은 수를 찾을 것인가를 입력 받은 후 그 수를 출력하는 프로그램을 작성하라.
 - 10개의 난수도 출력해 보여준다.



```
C:\windows\system32\cmd.exe
Enter the number of numbers: 10
몇번째로 작은 수: 4
894 250 65 688 99 966 296 649 455 305
4번째 작은 수는 296
계속하려면 아무 키나 누르십시오 . . .
```

뽑기-revisited

- 어떤 방식으로 뽑을지를 생각하기
- Bucket size에 대해 고민하기

Q1. 수식선정

- 이전 문제인 <뽑기-수식나열>의 간단한 응용
- 봉순이는 임의의 양의 자연수를 입력 받아 1~n까지의 연속된 수열을 이용하여 입력 받은 숫자를 표현하고 싶어졌다. 이 때 수열을 이루는 숫자는 -/+ 의 부호를 가지며 반드시 1번씩 나와야 한다. 예를 들어 8을 입력 받으면 $-1+2+3+4$ 로 출력되어야 한다. 그런데 입력 숫자에 따라 복수 개의 수식으로 표현될 수 있다.
 - 예를 들어 7은 $-1+2-3+4+5$, $+1+2+3-4+5$ 와 같이 두 수식으로 표현된다.
 - 또한, $1+2-3+4+5+6-7+8-9$ 도 가능하다.

Q1. 수식선정

- 임의의 숫자를 입력 받아 앞에서 설명한 수식 중의 하나로 표현해주는 프로그램을 작성하시오. 복수 개가 가능하면 숫자의 개수가 제일 작은 것 중의 하나면 된다.
- 예) 7 [입력]
 - $-1+2-3+4+5$ 혹은
 - $+1+2+3-4+5$ 으로 출력.

포인터 2차원 동적 할당

2차원 포인터

■ 포인터의 포인터

- 1차원 포인터의 주소값을 저장하는 포인터의 포인터는 * 기호를 두 번씩 기술해서 선언한다.

자료형 **포인터 변수명;

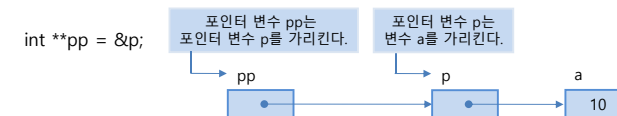
포인터의 포인터 기본 형식

EX) int **pp;

- 포인터와 관련된 연산자인 &과 *는 +와 - 사이 같은 관계다. & 연산자를 변수 앞에 붙이면 차원이 점점 올라간다. 포인터 변수 p는 변수 a를 가리킨다.



- 반대로 * 연산자를 변수 앞에 붙이면 차원이 점점 내려간다. 이 때 주의할 점은 * 연산자는 중복해서 붙일 수 있는데, & 연산자는 변수 앞에 한 번만 붙일 수 있다는 점이다.



2차원 포인터 사용하기

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

```
void main()
```

```
{
```

```
    int a=5;
```

```
    int *p;
```

```
    int **pp;
```

```
    p=&a;
```

```
    pp=&p;
```

```
    printf(" p : %X\t &a : %X\n", p, &a );
```

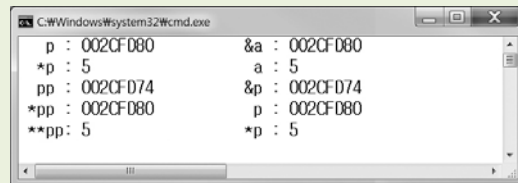
```
    printf(" *p : %d\t &a : %d\n", *p, a );
```

```
    printf(" pp : %X\t &p : %X\n", pp, &p );
```

```
    printf(" **pp : %X\t *pp : %X\n", **pp, p );
```

```
    printf(" ***pp : %X\t *p : %X\n", ***pp, *p );
```

```
}
```



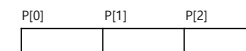
03 2차원 포인터

■ 1차원 포인터를 저장하는 포인터 배열

❶ int *p1, *p2, *p3;

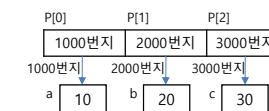
❷ int *p[3];

- ❷처럼 작성하면 원소 3개를 저장할 수 있는 배열이 생성되며, 각 원소에는 1차원 포인터를 저장할 수 있다.



- 각 원소에 1차원 포인터를 저장해 보자.

```
int a=10, b= 20, c=30;
int *p[3]={&a, &b, &c};
```



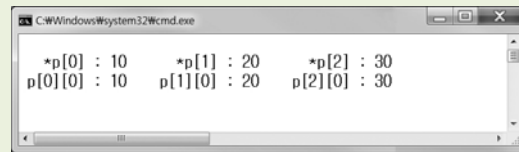
- 각 원소의 내용은 주소이므로 10, 20, 30을 출력하려면 각 원소 앞에 * 연산자를 기술해야 한다. 즉, p[0]==&a이므로 *p[0]==a가 된다.

1차원 포인터를 저장하는 포인터 배열

```
#include <stdio.h>
void main()
{
    int a=10, b= 20, c=30; // 정수형 변수
    // 포인터 배열에 변수의 주소를 저장해 둔다.
    int *p[3]={&a, &b, &c};

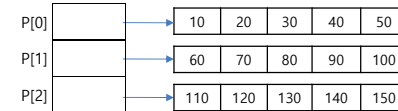
    // 배열 원소에 * 연산자로 정수값을 얻어온다.
    printf("\n *p[0] : %d", *p[0]);
    printf("\n *p[1] : %d", *p[1]);
    printf("\n *p[2] : %d", *p[2]);

    // * 연산자 대신 []로 정수값을 얻어온다.
    printf("\n p[0][0] : %d", p[0][0]);
    printf("\n p[1][0] : %d", p[1][0]);
    printf("\n p[2][0] : %d", p[2][0]);
    printf("\n");
}
```



03 2차원 포인터

- 포인터 배열에 1차원 배열명인 배열의 시작 주소를 저장해 보자.
 Int a[5]={10, 20, 30, 40, 50};
 Int b[5]={60, 70, 80, 90, 100};
 Int c[5]={110, 120, 130, 140, 150};
 Int *p[3]={a, b, c}; // a == &a[0], 즉 배열명은 배열 첫번째 원소의 주소
- 배열명 자체가 포인터이기 때문에 & 연산자 없이 배열명을 포인터 배열의 초기값으로 준다.



- 각 원소의 내용은 주소이므로 * 연산자를 기술하면 각 1차원 배열의 첫 번째 원소 내용이 출력된다.
 p[0]==a==&a[0]이므로 *p[0]==*a==&a[0]==a[0]이 되고
 p[1]==b==&b[0]이므로 *p[1]==*b==&b[0]==b[0]이 되고
 p[2]==c==&c[0]이므로 *p[2]==*c==&c[0]==c[0]이 된다.
- *p[0], *p[1], *p[2]와 같은 포인터 표현은 p[0][0], p[1][0], p[2][0]와 같이 배열처럼 표현할 수 있으므로 만일 각 1차원 배열의 두번째 원소를 출력하려면 다음과 같이 표현할 수 있다.
 p[0][1], p[1][1], p[2][1]

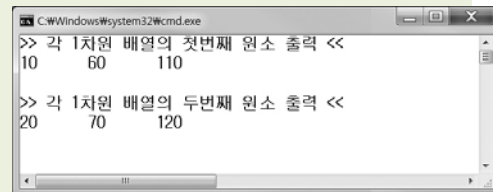
포인터 배열에 1차원 배열의 주소값 저장하기

```
#include <stdio.h>
void main()
{
    int a[5]={ 10, 20, 30, 40, 50};
    int b[5]={ 60, 70, 80, 90, 100};
    int c[5]={110, 120, 130, 140, 150};

    int *p[3]={a, b, c};

    printf(">> 각 1차원 배열의 첫번째 원소 출력 << \n");
    printf("%d %d %d\n", p[0][0], p[1][0], p[2][0]);

    printf(">> 각 1차원 배열의 두번째 원소 출력 << \n");
    printf("%d %d %d\n", p[0][1], p[1][1], p[2][1] );
}
```



03 2차원 포인터

2차원 배열과 포인터 변수

- 배열 원소의 주소값을 출력하려면?
 • 배열 원소의 주소값을 출력하려면 배열에 첨자를 지정한 원소에 &를 붙이면 된다. 즉, a[0][0]의 주소값을 알고 싶으면 &a[0][0]이라고 하면 된다.

2차원 배열에서 첨자를 생략하는 형태

- 2차원 배열에서 첨자를 생략하는 형태는 2가지다. 열만 생략하든지, 행과 열을 동시에 생략해야 한다
 • 2차원 배열 int a[3][4];에 대해서 열만 생략하면 행이 3개가 된다.
 a[0], a[1], a[2]
- 2차원 배열에서 열을 생략하고 행만 지목하면 원소값 대신 주소가 출력되며, 이 주소 들은 16 바이트 차이가 난다. a[0]과 a[1] 사이에 16 바이트 차이가 있고, a[1]과 a[2] 사 이에도 16 바이트 차이가 난다. 이 주소들은 각 행의 시작 주소와 일치한다. 행 1개가 열 4개로 구성되어 있고 정수형 배열이므로 16바이트 (4바이트×4개) 차이가 나는 것 이다.
 a[n] == &a[n][0]
- 2차원 배열에서 열을 생략하면 각 행의 시작 주소를 의미한다. n은 행의 위치를 알려 주는 첨자다.

2차원 배열의 주소값 출력하기

```
include <stdio.h>
#define ROW 3
#define COL 4
void main() {
    int a[ROW][COL] = {
        {90, 85, 95, 100},
        {75, 95, 80, 90},
        {90, 80, 70, 60}
    };
    int r,c;
    printf("2차원 배열에 저장된 원소들의 주소\n");
    printf("-----");
    for(r= 0; r<ROW; r++)
    {
        printf("\n %d행: ", r);
        for( c= 0; c<COL; c++) {
            printf("\t %X", &a[r][c]); // 배열의 주소값 출력
        }
        printf("\n");
    }
}
```

0행 :	001EF738	001EF73C	001EF740	001EF744
1행 :	001EF748	001EF74C	001EF750	001EF754
2행 :	001EF758	001EF75C	001EF760	001EF764

2차원 배열에 행만 지정해서 출력하기

```
#include <stdio.h>
#define ROW 3
#define COL 4
void main()
{
    int a[ROW][COL] = {
        {90, 85, 95, 100},
        {75, 95, 80, 90},
        {90, 80, 70, 60}
    };
    int r;
    printf("2차원 배열의 각 행의 첫번째 열의 주소\n");
    printf("-----");
    for( r= 0; r<ROW; r++){
        printf("\n %d행 ", r);
        printf("\t a[%d] = %X", r, a[r]);
        printf("\t &a[%d][0] = %X", r, &a[r][0]);
    }
    printf("\n\n 각 행의 첫번째 열에 위치한 원소\n");
    printf("-----\n");
    printf("\t *a[0]=90\t *a[1]=75\t *a[2]=90\n");
}
```

0행 :	a[0] = 001FFCD0	&a[0][0] = 001FFCD0
1행 :	a[1] = 001FFCE0	&a[1][0] = 001FFCE0
2행 :	a[2] = 001FFCF0	&a[2][0] = 001FFCF0

*a[0]=90	*a[1]=75	*a[2]=90
----------	----------	----------

03 2차원 포인터

2차원 배열에서의 배열명

- * 연산자를 두 번 붙이면 배열의 첫 번째 원소가 출력된다.
**a==a[0][0]
- 2차원 배열명에 포인터 연산자 +를 사용해 보자.
 - a+1은 2차원 배열의 시작주소보다 16바이트 큰 주소가 구해지고, a+2는 32바이트 큰 주소가 구해진다. 이렇게 16바이트씩 증가한다는 것은 행 단위로 주소를 계산한다는 의미가 된다. 배열명이 2차원 포인터이므로 더하기 연산을 한 결과 역시 2차원 포인터가 되며, 이 포인터는 행 단위의 주소를 계산할 수 있는 2차원 포인터다.

2차원 배열의 원소에 도입한 포인터 개념

- 2차원 배열명에 포인터 관련 연산자인 *, + 만을 사용해 배열의 원소를 사용해서 얻어낼 수 있다.
- a[r][c]==*(*(a+r)+c)
- a가 2차원 배열명이라면 이는 2차원 포인터다. 여기에 r을 더해 a+r한 결과는 역시 2차원 포인터로서 r번째 행의 시작 주소다. 여기에 계속 더하기를 하면 주소가 행 단위로 계산되어 차원을 낮추려고 * 연산자를 붙인다.
*(a+r)+c는 r번째 행의 시작 주소를 기준으로 4바이트씩 c번 떨어진 위치의 주소를 계산한다. 메모리 상에 2차원 배열이 각 행에 대한 열을 하나씩 증가시키는 구조로 할당되므로 2차원을 1차원으로 떨어뜨린 후 여기에 더하기를 해나간다. 1차원 포인터에 더하기 연산을 하면 4바이트 단위로 주소가 증가하고, 최종적으로 계산된 주소인 *(a+r)+c에 * 연산자를 한 번 더 붙이면 그 위치의 값을 출력한다.

2차원 배열명의 의미 파악하기

```
#include <stdio.h>
#define ROW 3
#define COL 4
void main()
{
    int a[ROW][COL] = {
        {90, 85, 95, 100},
        {75, 95, 80, 90},
        {90, 80, 70, 60}
    };
    printf(" a : %X\n", a);
    printf(" *a : %X\n", *a);
    printf(" **a : %d\n", **a);
    printf("-----\n");
    printf(" a + 1 : %X\n", a + 1);
    printf(" a + 2 : %X\n", a + 2);
}
```

a :	0029F748
*a :	0029F748
**a :	90

a + 1 :	0029F758
a + 2 :	0029F768

배열의 원소를 포인터 연산자를 이용해서 출력하기

```
#include <stdio.h>
#define ROW 3
#define COL 4
void main()
{
    int a[ROW][COL] = {
        {90, 85, 95, 100},
        {75, 95, 80, 90},
        {90, 80, 70, 60}
    };
    int r, c;
    for(r=0; r<ROW; r++){
        for(c=0; c<COL; c++) {
            printf("*(a+%d)+%d):%d \t", r, c, *(a+r)+c );
        }
        printf("\n");
    }
}
```

```
C:\Windows\system32\cmd.exe
*(a+0)+0):90      *(a+0)+1):85      *(a+0)+2):95      *(a+0)+3):100
*(a+1)+0):75      *(a+1)+1):95      *(a+1)+2):80      *(a+1)+3):90
*(a+2)+0):90      *(a+2)+1):80      *(a+2)+2):70      *(a+2)+3):60
```

03 2차원 포인터

- 2차원 포인터 변수는 어떻게 선언해야 하는가?

- 2차원 배열명을 int **p로 선언한 포인터 변수에 저장하면 에러가 발생한다.

```
int a[3][4];
int **p;
p=a;
```

Cannot convert 'int[4] *' to 'int **'

- 2차원 포인터지만 2차원 배열을 가리키려면 각 행이 열 몇 개로 구성되었는지에 대한 정보가 있어야 하므로 다음과 같이 선언해야 한다.

```
int (*p)[4];
p=a;
```

- 2차원 배열이 1차원 배열의 모임이므로 열이 4개로 구성된 형태의 배열을 가리키는 포인터 변수 p를 선언한 후 여기에 2차원 배열명을 대입한다. 2차원 배열의 시작 주소값인 a를 p에 대입했으므로 p로 2차원 배열 a의 원소들은 다음과 같이 출력할 수 있게 된다.

```
printf("*(p+%d)+%d):%d \t", r, c, (*(p+r)+c));
```

03 2차원 포인터

- p는 선언 시 한 행이 열 4개로 구성된 배열을 가리키는 포인터로 선언했기 때문에 p+1을 하면 16바이트 (4*4)가 증가한다. p+r한 결과는 r번째 행의 시작 주소값이고 2차원 포인터다. 여기에 계속 더하기를 하면 행 단위로 주소가 계산되므로 차원을 낮추려고 * 연산을 붙인 후 더하기 연산을 한다. *(p+r)+c는 r번째 행의 시작 주소를 기준으로 4바이트씩 c번 떨어진 위치의 주소를 계산하고 최종적으로 계산된 주소에 * 연산을 한 번 더 붙인 *(p+r)+c는 그 위치의 값을 출력한다.

포인터 변수 p
2차원 배열의 시작 주소값이 저장되어 있는
4바이트짜리 기억공간

	0열	1열	2열	3열
0행	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1행	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2행	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- 2차원 배열의 주소를 저장할 포인터 변수의 선언은 반드시 한 행에 포함된 열의 개수를 명시해야 한다.

2차원 배열의 동적 할당

사용자로부터
2차원 배열의 크기를
입력 받아

배열을 동적 할당 받고

하나씩 정수를
입력 받은 후

합을 출력하는
프로그램을 작성하라.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int row,col,i,j;
    int sum = 0;
    int **p;
    printf("2차원 배열의 크기?\n");
    scanf("%d", &row); // r행 c열 2차원 배열
    scanf("%d", &col); // r행 c열 2차원 배열

    p = (int**)malloc(sizeof(int*) * row); // r행 동적 할당
    for(i = 0; i < row; i++)
        p[i] = (int*)malloc(sizeof(int) * col); // 각 행마다 c열 할당

    for(i = 0; i < row; i++)
        for(j = 0; j < col; j++)
            scanf("%d", &p[i][j]);

    for(i = 0; i < row; i++)
        for(j = 0; j < col; j++)
            sum += p[i][j];
    printf("합 = %d\n", sum);

    for(i = 0; i < row; i++)
        free(p[i]); // 각 열을 반환
    free(p); // 행을 반환
}
```


행렬 · 동적할당

행렬(matrix)

- 여러 문제에 많이 이용되는 행렬은 행(m)과 열(n)로 구성된 자료구조이다.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- m x n 행렬은 2차원 배열 A[m][n]으로 표현한다.
- 예: 3 x 4 행렬

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

	[0]	[1]	[2]	[3]
[0]	1	2	3	4
[1]	5	6	7	8
[2]	9	10	11	12

행렬(matrix)

• 전치행렬

- 행렬 A의 모든 원소의 위치(i, j)를 (j, i)로 교환하여 m x n 행렬을 n x m 행렬로 변환한 행렬

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \xrightarrow{\text{전치행렬로 변환}} \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}$$

■ 행렬곱

$$\begin{array}{c} m \times n \text{ 행렬 } A \\ \begin{bmatrix} A_{0,0} & A_{0,1} & \dots & A_{0,n-1} \\ A_{1,0} & A_{1,1} & \dots & A_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m-1,0} & A_{m-1,1} & \dots & A_{m-1,n-1} \end{bmatrix} \end{array} \times \begin{array}{c} n \times p \text{ 행렬 } B \\ \begin{bmatrix} B_{0,0} & B_{0,1} & \dots & B_{0,p-1} \\ B_{1,0} & B_{1,1} & \dots & B_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n-1,0} & B_{n-1,1} & \dots & B_{n-1,p-1} \end{bmatrix} \end{array} = \begin{array}{c} m \times p \text{ 행렬 } C \\ \begin{bmatrix} C_{0,0} & C_{0,1} & \dots & C_{0,p-1} \\ C_{1,0} & C_{1,1} & \dots & C_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m-1,0} & C_{m-1,1} & \dots & C_{m-1,p-1} \end{bmatrix} \end{array}$$

$$C_{0,0} = A_{0,0} \times B_{0,0} + A_{0,1} \times B_{1,0} + \dots + A_{0,n-1} \times B_{n-1,0}$$

$$C_{i,j} = A_{i,0} \times B_{0,j} + A_{i,1} \times B_{1,j} + \dots + A_{i,n-1} \times B_{n-1,j}$$

Lab(행렬합_정적할당)

- 3 x 3인 2차원 배열을 A, B를 입력 받아
 - 두 배열의 합을 구하는 프로그램을 작성하시오.

- 정적할당을 사용
- A와 B의 행렬원소의 값을 입력 받는다
- 입력/처리/출력 부분을 각각 함수화 하라

- void readMatrix(int a[][3])
- void matrixAdd(int a[][3], int b[][3], int x[][3])
- void printMatrix(int a[][3])

```
C:\windows\system32\cmd.e
<3 x 3> 행렬 A 입력:
1 1 1
10 10 10
100 100 100
<3 x 3> 행렬 B 입력:
1 1 1
2 2 2
3 3 3
행렬합:
2 2 2
12 12 12
103 103 103
계속하려면 아무 키나 누!
```

Lab(행렬합_동적할당)

- 배열의 행의 개수(r)와 열의 개수(c)를 입력 받고
 - (r x c)의 2차원 배열을 A, B를 입력 받아
 - 두 배열의 합을 구하여 출력하는 프로그램을 작성하시오.

- 동적 할당: 입력한 사이즈의 행렬을 위해 동적으로 2차원 배열을 할당 (malloc)
- 동적으로 할당한 행렬을 반환 (free)
- Lab(행렬합_정적할당) 에서 사용한 함수를 그대로 사용해도 되는가? (답: 매 변수만 바꾸면 됨)

```
#include <stdio.h>
#include <stdlib.h>
void matrixAdd(int **a, int **b, int **x, int r, int c) {...}

void printMatrix(int **a, int r, int c) {...}
void readMatrix(int **a, int r, int c) {...}
int main(void)
{
    int **A, **B;
    int **X; // A + B
    int aRow, aCol;
    int i, j;

    printf("Enter 행렬 A의 행과 열의 개수: ");
    scanf("%d %d", &aRow, &aCol);

    // 동적으로 행렬 A(aRow x aCol)와 B(aRow x aCol)와
    // X(aRow x aCol)를 생성
    ...
    // A B 행렬값 입력 및 x 행렬 초기화
    ...
    matrixAdd(A, B, X, r, c);
    printf("행렬합:\n");
    printMatrix(X, r, c); printf("\n");
}
```

HW(행렬곱_동적할당)

- 행렬 A와 C에 대해서
 - 두 배열의 곱
 - 행렬 A의 전치행렬
- 을 구하여 출력하는 프로그램을 작성하시오.

```
C:\windows\system32\cmd.exe
Enter 행렬 A의 행과 열의 개수: 3 4
Enter 행렬 C의 행과 열의 개수<C의 행은 4이어야>: 4 2
3 x 4 행렬 A 입력:
1 1 1 1
2 2 2 2
3 3 3 3
4 x 2 행렬 C 입력:
1 2
1 2
1 2
1 2
행렬곱:
4 8
8 16
12 24
전치행렬:
1 2 3
1 2 3
1 2 3
1 2 3
```

- A, B 각각에 대한 행의 개수 열의 개수 입력 받는다 (이때 행렬곱 계산이 되려면 A의 열의 개수와 C의 행의 개수가 같아야 한다)
- 동적 할당: 입력한 사이즈의 행렬을 위해 동적으로 2차원 배열을 할당 (malloc)
- 동적으로 할당한 행렬을 반환 (free)