TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/5/17
수업일수	56 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

행렬

-행렬의 종류

-행렬의 연산

-Gauss-Jordan 소거법

-Inverse Matrix(역행렬)

-Determinant(판별식)

-Crammer 공식

-행렬의 종류

정방행렬

: 행 개수 = 열 개수인 행렬

$$\begin{bmatrix} 2 & 7 & 4 \\ 7 & 0 & 3 \\ 4 & 5 & 2 \end{bmatrix}$$

대각행렬

: i=j인 행렬 요소들을 제외한 나머지가 모두 0인 행렬

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

단위행렬

: 대각행렬에서 i=j인 요소들이 모두 1인 행렬, 곱셈에 대한 항등식이다.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

전치행렬

: 단위행렬의 1인 대각선을 기준으로 대칭 한 행렬

$$\begin{bmatrix} 2 & 7 & 4 \\ 8 & 0 & 3 \\ 1 & 6 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 8 & 1 \\ 7 & 0 & 6 \\ 4 & 3 & 2 \end{bmatrix}$$

-행렬의 연산

행렬의 덧셈

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \\ 2 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 \\ 0 & 4 & 1 \\ 2 & 4 & 4 \end{bmatrix}$$

행렬의 뺄셈

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \\ 2 & 4 & 3 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ 2 & 4 & 2 \end{bmatrix}$$

행렬의 곱셈

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \\ 2 & 4 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 8 & 1 \\ 2 & 6 & 3 \\ 8 & 17 & 3 \end{bmatrix}$$

스칼라 배

$$2 * \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 0 \\ 0 & 2 & 0 \\ 4 & 6 & 2 \end{bmatrix}$$

-Gauss-Jordan 소거법

$2x + 4y + 4z = 12$

$6x + 2y + 2z = 16$ $\quad\longrightarrow\quad$ $\begin{bmatrix} 2 & 4 & 4 \\ 6 & 2 & 2 \\ 4 & 2 & 4 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 12 \\ 16 \\ 20 \end{bmatrix}$

$4x + 2y + 4z = 20$

$$\left[\begin{array}{ccc|c} 1 & 2 & 2 & 6 \\ 6 & 2 & 2 & 16 \\ 4 & 2 & 4 & 20 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 2 & 6 \\ 6 & 2 & 2 & 16 \\ 0 & -6 & -4 & -4 \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 2 & 2 & 6 \\ 0 & -10 & -10 & -20 \\ 0 & -6 & -4 & -4 \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 2 & 2 & 6 \\ 0 & 1 & 1 & 2 \\ 0 & -6 & -4 & -4 \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 2 & 2 & 6 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 2 & 8 \end{array}\right]$$

따라서 $x = 2, \quad y = -2, \quad z = 4$

-Inverse Matrix(역행렬)

$A|I \;\Rightarrow\; AA^{-1}|IA^{-1} \;\Rightarrow\; I|A^{-1}$

$$\left[\begin{array}{ccc|ccc} 2 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 9 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 2 & 1/2 & 0 & 0 \\ 0 & 3 & 9 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/2 & 0 & -2 \\ 0 & 3 & 9 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/2 & 0 & -2 \\ 0 & 1 & 3 & 0 & 1/3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/2 & 0 & -2 \\ 0 & 1 & 0 & 0 & 1/3 & -3 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$A^{-1} = \begin{bmatrix} 1/2 & 0 & -2 \\ 0 & 1/3 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

-Determinant(판별식)

: 정방행렬의 요소들을 평가한 스칼라 양. det(A)라 표시

$$A = \begin{bmatrix} 2 & 0 & 4 \\ 0 & 3 & 9 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\det(A) = 2 * (3 * 1 - 0 * 9) - 0 * (0 * 1 - 9 * 0) + 4 * (0 * 0 - 0 * 3)$$

$$= 6$$

-Crammer 공식

$$2x + 4y + 4z = 12$$

$$6x + 2y + 2z = 16 \qquad \rightarrow \qquad \begin{bmatrix} 2 & 4 & 4 \\ 6 & 2 & 2 \\ 4 & 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 12 \\ 16 \\ 20 \end{bmatrix}$$

$$4x + 2y + 4z = 20$$

$$\det(A) = 8 - 64 + 16 = -40$$

$$X = \begin{bmatrix} 12 & 4 & 4 \\ 16 & 2 & 2 \\ 20 & 2 & 4 \end{bmatrix}$$

$$\det(X) = 12 * 4 + 4 * (-24) + 4 * (-8) = -80$$

$$x = \frac{\det(X)}{\det(A)} = 2$$

$$Y = \begin{bmatrix} 2 & 12 & 4 \\ 6 & 16 & 2 \\ 4 & 20 & 4 \end{bmatrix}$$

$$\det(Y) = 2 * 24 + 12 * (-16) + 4 * (56) = 80$$

$$y = \frac{\det(Y)}{\det(A)} = -2$$

$$Z = \begin{bmatrix} 2 & 4 & 12 \\ 6 & 2 & 16 \\ 4 & 2 & 20 \end{bmatrix}$$

$$\det(Z) = 2 * 8 + 4 * (-56) + 12 * 4 = \ -160$$

$$\mathrm{z} = \frac{\det(Z)}{\det(A)} = 4$$

-matrix.h

```
#ifndef __MATRIX_H__
#define __MATRIX_H__

#include <stdio.h>

#include <math.h>

void print_mat(float (*R)[3])
{
        printf(" %lf,%lf,%lf ₩n %lf,%lf,%lf ₩n %lf,%lf,%lf",R[0][0], R[0][1], R[0][2],R[1][0],R[1][1],R[1][2],R[2][0],R[2][1],R[2][2]);
}

float det(float (*A)[3])
{

        return A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1])
                - A[0][1] * ((A[1][0] * A[2][2]) - (A[1][2] * A[2][0]))
                + A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
}

void add(float (*A)[3] , float (*B)[3], float (*R)[3])
{
        R[0][0] = A[0][0] + B[0][0];
        R[1][1] = A[1][1] + B[1][1];
        R[2][2] = A[2][2] + B[2][2];

        R[0][1] = A[0][1] + B[0][1];
        R[0][2] = A[0][2] + B[0][2];

        R[1][0] = A[1][0] + B[1][0];
```

```
	R[1][2] = A[1][2] + B[1][2];

	R[2][0] = A[2][0] + B[2][0];
	R[2][1] = A[2][1] + B[2][1];
}

void sub(float (*A)[3], float (*B)[3], float (*R)[3])
{
	R[0][0] = A[0][0] - B[0][0];
	R[1][1] = A[1][1] - B[1][1];
	R[2][2] = A[2][2] - B[2][2];

	R[0][1] = A[0][1] - B[0][1];
	R[0][2] = A[0][2] - B[0][2];

	R[1][0] = A[1][0] - B[1][0];
	R[1][2] = A[1][2] - B[1][2];

	R[2][0] = A[2][0] - B[2][0];
	R[2][1] = A[2][1] - B[2][1];
}

void mul(float (*A)[3], float (*B)[3], float(*R)[3])
{
	R[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0] + A[0][2]*B[2][0]
	R[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1] + A[0][2]*B[2][1]
	R[0][2] = A[0][0]*B[0][2] + A[0][1]*B[1][2] + A[0][2]*B[2][2]

	R[1][0] = A[1][0]*B[0][0] + A[1][1]*B[1][0] + A[1][2]*B[2][0]
	R[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1] + A[1][2]*B[2][1]
	R[1][2] = A[1][0]*B[0][2] + A[1][1]*B[1][2] + A[1][2]*B[2][2]
```

```
	R[2][0] = A[2][0]*B[0][0] + A[2][1]*B[1][0] + A[2][2]*B[2][0]
	R[2][1] = A[2][0]*B[0][1] + A[2][1]*B[1][0] + A[2][2]*B[2][1]
	R[2][2] = A[2][0]*B[0][2] + A[2][1]*B[1][0] + A[2][2]*B[2][2]
}

void trans(float (*A)[3], float (*R)[3])
{
	R[0][0] = A[0][0];
	R[1][1] = A[1][1];
	R[2][2] = A[2][2];

	R[0][1] = A[1][0];
	R[0][2] = A[2][0];

	R[1][0] = A[0][1];
	R[1][2] = A[2][1];

	R[2][0] = A[0][2];
	R[2][1] = A[1][2];

}
#endif
```

-matrix.c

```
#include "matrix.h"

int main(void)
{
	float res;
	float A[3][3] = {{2,0,4},{0,3,9},{0,0,1}};
	float B[3][3] = {{1,2,3},{1,2,3},{1,2,3}};
	float R[3][3] = {0};

	res = det(A);
	printf("%lf₩n",res);

	add(A,B,R);
	print_mat(R);
	return 0;
}
```

선생님 코드

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


        void init_mat(float (*A)[3]) //랜덤 값으로 배열 채워 넣는 함수
        {
            int i, j;


            for(i = 0; i < 3; i++)
                for(j = 0; j < 3; j++)
                    A[i][j] = rand() % 4;
        }


        void print_mat(float (*R)[3]) // 배열 프린트하는 함수
        {
            int i, j;


            for(i = 0; i < 3; i++)
            {
                for(j = 0; j < 3; j++)
                    printf("%10.4f", R[i][j]);
                printf("₩n");
            }
            printf("₩n");
```

```
}


void add_mat(float (*A)[3], float (*B)[3], float (*R)[3]) //배열의 각 요소를 더하는 함수
{
	int i, j;


	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j] + B[i][j];
}


void sub_mat(float (*A)[3], float (*B)[3], float (*R)[3]) //배열의 각 요소를 빼는 함수
{
	int i, j;


	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
			R[i][j] = A[i][j] - B[i][j];
}


void scale_mat(float scale_factor, float (*A)[3], float (*R)[3]) //배열의 각 요소에 스칼라 값을 곱하는 함수
{
	int i, j;


	for(i = 0; i < 3; i++)
		for(j = 0; j < 3; j++)
```

```
		R[i][j] = scale_factor * A[i][j];
}


#if 0
A[0][0]  A[0][1]  A[0][2]		B[0][0]  B[0][1]  B[0][2]
A[1][0]  A[1][1]  A[1][2]		B[1][0]  B[1][1]  B[1][2]
A[2][0]  A[2][1]  A[2][2]		B[2][0]  B[2][1]  B[2][2]


A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0]		A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1]
	A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2]
A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0]		A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1]
	A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2]
A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0]		A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1]
	A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2]
#endif


void mul_mat(float (*A)[3], float (*B)[3], float (*R)[3]) //배열의 곱
{
	R[0][0] = A[0][0]*B[0][0]+A[0][1]*B[1][0]+A[0][2]*B[2][0];
	R[0][1] = A[0][0]*B[0][1]+A[0][1]*B[1][1]+A[0][2]*B[2][1];
	R[0][2] = A[0][0]*B[0][2]+A[0][1]*B[1][2]+A[0][2]*B[2][2];


	R[1][0] = A[1][0]*B[0][0]+A[1][1]*B[1][0]+A[1][2]*B[2][0];
	R[1][1] = A[1][0]*B[0][1]+A[1][1]*B[1][1]+A[1][2]*B[2][1];
	R[1][2] = A[1][0]*B[0][2]+A[1][1]*B[1][2]+A[1][2]*B[2][2];


	R[2][0] = A[2][0]*B[0][0]+A[2][1]*B[1][0]+A[2][2]*B[2][0];
```

```
		R[2][1] = A[2][0]*B[0][1]+A[2][1]*B[1][1]+A[2][2]*B[2][1];
		R[2][2] = A[2][0]*B[0][2]+A[2][1]*B[1][2]+A[2][2]*B[2][2];
	}


	float det_mat(float (*A)[3]) //판별식 구하는 함수. float 값을 리턴
	{
		return A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1]) +
				A[0][1] * (A[1][2] * A[2][0] - A[1][0] * A[2][2]) +
				A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
	}


	void trans_mat(float (*A)[3], float (*R)[3]) //전치행렬을 만드는 함수
	{
		R[0][0] = A[0][0];
		R[1][1] = A[1][1];
		R[2][2] = A[2][2];


		R[0][1] = A[1][0];
		R[1][0] = A[0][1];


		R[0][2] = A[2][0];
		R[2][0] = A[0][2];


		R[2][1] = A[1][2];
		R[1][2] = A[2][1];
	}
```

```c
#if 0
	R[0][1] = A[1][2] * A[2][0] - A[1][0] * A[2][2];
	R[0][2] = A[1][0] * A[2][1] - A[1][1] * A[2][0];


	R[1][0] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
	R[1][2] = A[0][1] * A[2][0] - A[0][0] * A[2][1];


	R[2][0] = A[0][1] * A[1][2] - A[0][2] * A[1][1];
	R[2][1] = A[0][2] * A[1][0] - A[0][0] * A[1][2];
#endif


void adj_mat(float (*A)[3], float (*R)[3]) //adj 를 구하는 함수
{
	R[0][0] = A[1][1] * A[2][2] - A[1][2] * A[2][1];
	R[0][1] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
	R[0][2] = A[0][1] * A[1][2] - A[0][2] * A[1][1];


	R[1][0] = A[1][2] * A[2][0] - A[1][0] * A[2][2];
	R[1][1] = A[0][0] * A[2][2] - A[0][2] * A[2][0];
	R[1][2] = A[0][2] * A[1][0] - A[0][0] * A[1][2];


	R[2][0] = A[1][0] * A[2][1] - A[1][1] * A[2][0];
	R[2][1] = A[0][1] * A[2][0] - A[0][0] * A[2][1];
	R[2][2] = A[0][0] * A[1][1] - A[0][1] * A[1][0];
}
```

```
bool inv_mat(float (*A)[3], float (*R)[3])
{
        float det;


        det = det_mat(A);


        if(det == 0.0) //판별식이 0 이어서 역함수를 구할 수 없으면 0 반환
                return false;


        adj_mat(A, R);
#ifdef __DEBUG__
        printf("Adjoint Matrix₩n");
        print_mat(R);
#endif
        scale_mat(1.0 / det, R, R); R 배열에 판별식의 역수를 곱한 값을 R 에 넣음.


        return true;
}


void molding_mat(float (*A)[3], float *ans, int idx, float (*R)[3])
{
        int i, j;


        for(i = 0; i < 3; i++)
        {
```

```
		for(j = 0; j < 3; j++)
		{
			if(j == idx)
				continue;
			R[i][j] = A[i][j];
		}


		R[i][idx] = ans[i];
	}
}


void crammer_formula(float (*A)[3], float *ans, float *xyz)
{
	float detA, detX, detY, detZ;
	float R[3][3] = {};


	detA = det_mat(A);


	molding_mat(A, ans, 0, R);
#ifdef __DEBUG__
	print_mat(R);
#endif
	detX = det_mat(R);


	molding_mat(A, ans, 1, R);
#ifdef __DEBUG__
	print_mat(R);
```

```
#endif
        detY = det_mat(R);


        molding_mat(A, ans, 2, R);
#ifdef __DEBUG__
        print_mat(R);
#endif
        detZ = det_mat(R);


        xyz[0] = detX / detA;
        xyz[1] = detY / detA;
        xyz[2] = detZ / detA;
}


void print_vec3(float *vec)
{
        int i;


        for(i = 0; i < 3; i++)
                printf("%10.4f", vec[i]);


        printf("₩n");
}


void create_3x4_mat(float (*A)[3], float *ans, float (*R)[4])
{
```

```
		int i, j;


		for(i = 0; i < 3; i++)
		{
			for(j = 0; j < 3; j++)
				R[i][j] = A[i][j];


			R[i][3] = ans[i];
		}
}


void print_3x4_mat(float (*R)[4])
{
		int i, j;


		for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 4; j++)
			printf("%10.4f", R[i][j]);
		printf("₩n");
	}
	printf("₩n");
}


void adjust_3x4_mat(float (*A)[4], int idx, float (*R)[4])
{
		int i, j;
```

```
		float div_factor;


		for(i = idx + 1; i < 3; i++)
		{
			//div_factor = -A[idx][idx] / A[idx + 1][idx];
			//div_factor = -A[idx + 1][idx] / A[idx][idx];
			//div_factor = -A[i][0] / A[idx][0];
			div_factor = -A[i][idx] / A[idx][idx];
			printf("div_factor = %f₩n", div_factor);


			for(j = 0; j < 4; j++)
				R[i][j] = A[idx][j] * div_factor + A[i][j];
		}
	}


	void finalize(float (*R)[4], float *xyz)
	{
		xyz[2] = R[2][3] / R[2][2];
		xyz[1] = (R[1][3] - R[1][2] * xyz[2]) / R[1][1];
		xyz[0] = (R[0][3] - R[0][2] * xyz[2] - R[0][1] * xyz[1]) / R[0][0];
	}


	void gauss_elimination(float (*A)[3], float *ans, float *xyz)
	{
		float R[3][4] = {};


		create_3x4_mat(A, ans, R);
```

```
#if __DEBUG__
        print_3x4_mat(R);
#endif


        adjust_3x4_mat(R, 0, R);
#if __DEBUG__
        print_3x4_mat(R);
#endif


        adjust_3x4_mat(R, 1, R);
#if __DEBUG__
        print_3x4_mat(R);
#endif


        finalize(R, xyz);
}


void create_3x6_mat(float (*A)[3], float (*R)[6])
{
        int i, j;


        for(i = 0; i < 3; i++)
                for(j = 0; j < 3; j++)
                {
                        R[i][j] = A[i][j];
```

```
                        if(i == j)
                            R[i][j + 3] = 1;
                        else
                            R[i][j + 3] = 0;
                }
}


void print_3x6_mat(float (*R)[6])
{
        int i, j;


        for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 6; j++)
            printf("%10.4f", R[i][j]);
        printf("₩n");
    }
    printf("₩n");
}


void adjust_3x6_mat(float (*A)[6], int idx, float (*R)[6])
{
    int i, j;
    float div_factor, scale;


        scale = A[idx][idx];
```

```
        for(i = idx + 1; i < 3; i++)
        {
            //div_factor = -A[idx][idx] / A[idx + 1][idx];
            //div_factor = -A[idx + 1][idx] / A[idx][idx];
            //div_factor = -A[i][0] / A[idx][0];
            div_factor = -A[i][idx] / A[idx][idx];
            printf("div_factor = %f₩n", div_factor);


                    if(div_factor == 0.0)
                            continue;


            for(j = 0; j < 6; j++)
                R[i][j] = A[idx][j] * div_factor + A[i][j];
        }


            for(j = 0; j < 6; j++)
                    R[idx][j] = A[idx][j] / scale;
}


void gauss_elim_mat(float (*A)[3], float (*R)[3])
{
        float mid[3][6] = {};


        create_3x6_mat(A, mid);
#if __DEBUG__
        print_3x6_mat(mid);
#endif
```

```c
        adjust_3x6_mat(mid, 0, mid);
#if __DEBUG__
    print_3x6_mat(mid);
#endif


        adjust_3x6_mat(mid, 1, mid);
#if __DEBUG__
    print_3x6_mat(mid);
#endif
}


int main(void)
{
        bool inv_flag;


        float test[3][3] = {{2.0, 0.0, 4.0}, {0.0, 3.0, 9.0}, {0.0, 0.0, 1.0}};
        float stimul[3][3] = {{2.0, 4.0, 4.0}, {6.0, 2.0, 2.0}, {4.0, 2.0, 4.0}};
        float ans[3] = {12.0, 16.0, 20.0};
        float xyz[3] = {};


        float A[3][3] = {};
        float B[3][3] = {};
        float R[3][3] = {};


        srand(time(NULL));
```

```
printf("Init A Matrix₩n");
init_mat(A);
print_mat(A);


printf("Init B Matrix₩n");
init_mat(B);
print_mat(B);


printf("A + B Matrix₩n");
add_mat(A, B, R);
print_mat(R);


printf("A - B Matrix₩n");
sub_mat(A, B, R);
print_mat(R);


printf("Matrix Scale(A)₩n");
scale_mat(0.5, A, R);
print_mat(R);


printf("AB Matrix₩n");
mul_mat(A, B, R);
print_mat(R);
```

```
printf("det(A) = %f₩n", det_mat(A));
printf("det(B) = %f₩n", det_mat(B));


printf("₩nA^T(Transpose) Matrix₩n");
trans_mat(A, R);
print_mat(R);


printf("B^T(Transpose) Matrix₩n");
trans_mat(B, R);
print_mat(R);


printf("A Inverse Matrix₩n");
inv_flag = inv_mat(A, R);
if(inv_flag)
        print_mat(R);
else
        printf("역행렬 없다!₩n");


printf("test Inverse Matrix₩n");
inv_flag = inv_mat(test, R);
if(inv_flag)
        print_mat(R);
else
        printf("역행렬 없다!₩n");


printf("크래머 공식 기반 연립 방정식 풀기!₩n2x + 4y + 4z = 12₩n6x + 2y + 2z = 16₩n4x + 2y + 4z = 20₩n");
crammer_formula(stimul, ans, xyz);
```

```c
		print_vec3(xyz);


		printf("가우스 소거법 기반 연립 방정식 풀기!(문제 위의 것과 동일함)₩n");
		gauss_elimination(stimul, ans, xyz);
		print_vec3(xyz);


		printf("가우스 소거법으로 역행렬 구하기!₩n");
		gauss_elim_mat(test, R);
		print_mat(R);


		return 0;
}
```