Xilinx Zynq FPGA, TI DSP, MCU기반의 프로그래밍 및 회로 설계 전문가 과정

강사 - Innov (이상훈)
gcccompil3r@gmail.com
학생 - 이유성
dbtjd1102@naver.com

*벡터의 연산

*vector 곱셈 : 1.스칼라곱 2.내적 3.외적 4.텐서 연산(우주선 만들 때 -> 공기역학)

*그램 슈미트 정규 직교화

*벡터 내적과 외적 코딩

-벡터 내적 : 서로 수직하는지 아닌지 확인할 때
-외적 : (스칼라 값)*방향 ->증명 -> http://j1w2k3.tistory.com/1158

외적 내적 코딩

```
#include<stdio.h>

int Inner(int *arr1, int *arr2);

void Outer(int *res2 , int *arr1, int *arr2);

int main(void)
{
	int arr1[3] = {1,2,3} ;
	int arr2[3] = {2,3,4} ;

	int res1 = Inner(arr1 , arr2 ) ;

	printf("%d\n" ,res1);

	int res2[3];
//	for(int i = 0; i <3 ;i++){

	//	res2[i] =
	 Outer(res2,arr1,arr2);
	}
	printf("(%d)*i (%d)*j (%d)*k \n" ,res2[0],res2[1],res2[2] );

	return 0;
}

int Inner(int *arr1, int *arr2)
{
	int i,ret=0 ;
	for(i = 0 ; i < 3; i++){

		ret += arr1[i] * arr2[i];
	}
```

```
	return ret;
}

void Outer(int *res2 , int *arr1, int *arr2)
{
	int i, ret = 0;

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

		res2[i] = arr1[(i+1)%3]*arr2[(i+2)%3]-arr1[(i+2)%3]*arr2[(i+1)%3];


	}

}
```

```
*cat vector_3d.c

#include "vector_3d.h"
#include <stdio.h>

void print_vec3(vec3 R)
{
	printf("x = %f, y = %f, z = %f\n", R.x, R.y, R.z);
}

int main(void)
{
	vec3 A = {3, 2, 1};
	vec3 B = {1, 1, 1};
	vec3 R = {0, 0, 0, vec3_add, vec3_sub};

	R.add(A, B, &R);
	print_vec3(R);

	R.sub(A, B, &R);
	print_vec3(R);

	return 0;
}
```

*cat vector_3d.h

```
 #ifndef __VECTOR_3D_H__
#define __VECTOR_3D_H__

#include <stdio.h>
#include <math.h>

typedef struct vector3d vec3;

struct vector3d
{
   float x;
   float y;
   float z;

   void (* add)(vec3, vec3, vec3 *);
   void (* sub)(vec3, vec3, vec3 *);
   void (* scale)(float, vec3, vec3 *);
   float (* dot)(vec3, vec3);
   void (* cross)(vec3, vec3, vec3 *);
   void (* print)(vec3);

   void (* gramschmidt)(vec3 *, vec3 *, vec3);
};

void vec3_add(vec3 a, vec3 b, vec3 *r)
{
   r->x = a.x + b.x;
   r->y = a.y + b.y;
   r->z = a.z + b.z;
}

void vec3_sub(vec3 a, vec3 b, vec3 *r)
{
   r->x = a.x - b.x;
   r->y = a.y - b.y;
   r->z = a.z - b.z;
}

void vec3_scale(float factor, vec3 a, vec3 *r)
{
   r->x = a.x * factor;
   r->y = a.y * factor;
   r->z = a.z * factor;
}
```

```
float vec3_dot(vec3 a, vec3 b)
{
   return a.x * b.x + a.y * b.y + a.z * b.z;
}

void vec3_cross(vec3 a, vec3 b, vec3 *r)
{
   r->x = a.y * b.z - a.z * b.y;
   r->y = a.z * b.x - a.x * b.z;
   r->z = a.x * b.y - a.y * b.x;
}

void print_vec3(vec3 r)
{
   printf("x = %f, y = %f, z = %f\n", r.x, r.y, r.z);
}

float magnitude(vec3 v)
{
   return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

void gramschmidt_normalization(vec3 *arr, vec3 *res, vec3 r)
{
   vec3 scale1 = {};
   float dot1, mag1;

   mag1 = magnitude(arr[0]);
   r.scale(1.0 / mag1, arr[0], &res[0]);
   r.print(res[0]);

   mag1 = magnitude(res[0]);
   dot1 = r.dot(arr[1], res[0]);
   r.scale(dot1 * (1.0 / mag1), res[0], &scale1);
   r.sub(arr[1], scale1, &res[1]);
   r.print(res[1]);
}

#endif
```