Ⅰ. Code 설명

1. OnFileOpen()

```
filename = dlg.GetPathName();
                fp = fopen(filename, "r");

                char temp_text[50];
                double GridScale;
                int index;
                Node input_node[50];
                NodeConnect input_connectNode;
                Calc_Structure;
                fscanf(fp, "%s", &temp_text);
                fscanf(fp, "%lf", &GridScale);
                fscanf(fp, "%s", &temp_text);
                index = 0;

                do
                {
                        fscanf(fp, "%s", &temp_text);
                        if(strcmp(temp_text, "q") == 0)
                                break;
                        fscanf(fp, "%lf", &input_node[index].Point.x);
                        fscanf(fp, "%lf", &input_node[index].Point.y);
                        fscanf(fp, "%lf", &input_node[index].Point.z);
                        fscanf(fp, "%s", &temp_text);
                        if(strcmp(temp_text, "FREE") == 0)
                        {
                                input_node[index].m_BC.CheckBC = 3;
                        }
                        else if(strcmp(temp_text, "FIX") == 0)
                        {
                                input_node[index].m_BC.CheckBC = 0;
                        }
                        else
                        {
                                input_node[index].m_BC.CheckBC = 2;
                        }
                        fscanf(fp, "%lf", &input_node[index].MX);
                        fscanf(fp, "%lf", &input_node[index].MY);
                        fscanf(fp, "%lf", &input_node[index].F);
                        input_node[index].PointOrder = index;
                        Calc_Structure.m_Node.push_back(&input_node[index]);

                        index ++;
                }while(1);
                fscanf(fp, "%s", &temp_text);
                index = 0;

                do
                {
                        int StartOrder, EndOrder;
                        fscanf(fp, "%s", &temp_text);
                        if(strcmp(temp_text, "q") == 0)
                                break;
```

```
                        fscanf(fp, "%d", &StartOrder);
                        fscanf(fp, "%d", &EndOrder);
                        fscanf(fp, "%s", &temp_text);
                        input_connectNode.Name = temp_text;

                        fscanf(fp, "%lf", &input_connectNode.m_Material.G);
                        fscanf(fp, "%lf", &input_connectNode.m_Material.E);
                        fscanf(fp, "%lf", &input_connectNode.m_Material.I);
                        fscanf(fp, "%lf", &input_connectNode.m_Material.J);
                        input_connectNode.Start = Calc_Structure.m_Node[StartOrder];
                        input_connectNode.End = Calc_Structure.m_Node[EndOrder];
                        Calc_Structure.m_NodeConnect.push_back(input_connectNode);
                        Grillage grillage(input_connectNode);
                        grillage.Create_K_Global();
                        Calc_Structure.m_Grillage.push_back(grillage);

                        index ++;
                }while(1);

                NodeStart = new Vector [Calc_Structure.m_NodeConnect.size()];
                NodeEnd = new Vector [Calc_Structure.m_NodeConnect.size()];
                for(int i = 0; i<Calc_Structure.m_NodeConnect.size(); i++)
                {
                        NodeStart[i] = Calc_Structure.m_NodeConnect[i].Start->Point;
                        NodeEnd[i] = Calc_Structure.m_NodeConnect[i].End->Point;
                }
                Calc_Structure.CreateGrillage();
                Calc_Structure.NodeDelta();
                Calc_Structure.NodeForce();
                int n_index = 0;
                for(int i = 0; i<Calc_Structure.delta.row; i++)
                {
                        if(i%3 == 0)
                                Calc_Structure.m_Node[n_index]->ThetaX =
Calc_Structure.Final_delta.GetElement(i,0);
                        else if(i%3 == 1)
                                Calc_Structure.m_Node[n_index]->ThetaY =
Calc_Structure.Final_delta.GetElement(i,0);
                        else if(i%3 == 2)
                        {
                                Calc_Structure.m_Node[n_index]->DeltaZ =
Calc_Structure.Final_delta.GetElement(i,0);
                                n_index++;
                        }
                }
```

이상은 입력 data로부터 각각 Structure Class에 있는 Node와 NodeConnect, BoundaryCondition, MaterialProperty class의 정보들로 값들을 입력받는 Code이다. 보면 q 값을 이용하여 파일의 끝을 찾아낼 때까지 무한roop을 돌도록 DoWhile 반복문을 사용하였으며, 모두 입력받은 후 Structure class의 함수들을 실행하여 delta와 Force를 계산함을 알 수 있다. 각각의 입력 정보들은 push_back 함수를 이용, 배열 형식으로 저장하였다. 이후 가시화 부분은 입력 data를 최종 node 좌표로 설정한 것으로 7차와 동일하므로 설명을 생략한다.

1. Node.h , BoundaryCondition.h , NodeConnect.h , MaterialProperty.h

이상의 class들은 header 파일과 cpp 파일이 모두 존재하지만, 체계적인 class 변수로만 이용하였지 이 class 를 이용하여 함수 및 명령을 실행하지는 않았다. 각각 Node의 정보, 각 Node에 해당하는 BoundaryCondition와 Beam의 정보와 그에 해당하는 Beam 재료의 성질에 대한 정보를 다룬다.

BoundaryCondition에서는 각각 Fix, Free, Simple 에 대한 구분을 각각 0, 3, 2으로 해 놓았다. 이는 각 Node의 delta, θ_x, θ_y, δ_z 에 대하여 BoundaryCondition 에 따라 모르는 변위의 개수를 구분하여 놓은 것이다.

2. Grillage.cpp

```
void Grillage::Create_K_Local()              //Local 좌표계의강성매트릭스생성
{
        K_Local.element[0][0] = Material.G*Material.J/L;
        K_Local.element[0][3] = -Material.G*Material.J/L;

        K_Local.element[1][1] = 4.0*Material.E*Material.I/pow(L, 1.0);
        K_Local.element[1][2] = -6.0*Material.E*Material.I/pow(L, 2.0);
        K_Local.element[1][4] = 2.0*Material.E*Material.I/pow(L, 1.0);
        K_Local.element[1][5] = 6.0*Material.E*Material.I/pow(L, 2.0);

        K_Local.element[2][1] = -6.0*Material.E*Material.I/pow(L, 2.0);
        K_Local.element[2][2] = 12.0*Material.E*Material.I/pow(L, 3.0);
        K_Local.element[2][4] = -6.0*Material.E*Material.I/pow(L, 2.0);
        K_Local.element[2][5] = -12.0*Material.E*Material.I/pow(L, 3.0);
(중략)

void Grillage::Create_T()           //좌표변환매트릭스생성
{
        T.element[0][0] = cos(theta);
        T.element[0][1] = sin(theta);

        T.element[1][0] = -sin(theta);
        T.element[1][1] = cos(theta);

        T.element[2][2] = 1.0;
(중략)

void Grillage::Create_K_Global()    //Global 좌표계의최종매트릭스생성
{
        Create_K_Local();
        Create_T();
        K_Global = T.Transpose()*K_Local*T;
}
```

Grillage 문제를 해결하는 데 핵심인 강성방정식을 만드는 cpp이다. 복잡한 과정들이 많지만, 요약하면 각 Node에서의 입력된 힘과 모멘트 성분을 받아 F = Kd 에서 K Matrix에 대입하여 주는 것이다. K Matrix 는 3*(Node 개수) by 3*(Node 개수) 의 정방행렬로 만들어지게 되고, 이를 기본 좌표축과 회전된 각도에 따라 회전변환 시켜 주는 T Matrix를 이용하여 Force 정보 Matrix와 delta 정보 Matrix 사이의 관계를 말해주는 K Matrix를 생성하여 주는 것이 이 Grillage.cpp의 목적이다.

3. Structure.cpp

1) CreateGrillage()

```
void Structure::CreateGrillage()
{
	NumOfNode = m_Node.size();
	TotalMatrix.CreateMatrix(3*NumOfNode, 3*NumOfNode);

	for(int i=0; i<(int)(m_Grillage.size()); i++)
	{
		Matrix part1;
		part1.CreateMatrix(3, 3);
		for(int j=0; j<3; j++)
		{
			for(int k=0; k<3; k++)
			{
				part1.element[j][k] = m_Grillage[i].K_Global.element[j][k];
			}
		}
		Matrix part2;
		part2.CreateMatrix(3, 3);
		for(int j=0; j<3; j++)
		{
			for(int k=0; k<3; k++)
			{
				part2.element[j][k] = m_Grillage[i].K_Global.element[j][k+3];
			}
		}
		Matrix part3;
		part3.CreateMatrix(3, 3);
		for(int j=0; j<3; j++)
		{
			for(int k=0; k<3; k++)
			{
				part3.element[j][k] = m_Grillage[i].K_Global.element[j+3][k];
			}
		}
		Matrix part4;
		part4.CreateMatrix(3, 3);
		for(int j=0; j<3; j++)
		{
			for(int k=0; k<3; k++)
			{
				part4.element[j][k] = m_Grillage[i].K_Global.element[j+3][k+3];
			}
		}

		int Num_Startnode = m_Grillage[i].Start_nodeNumber;
		int Num_Endnode = m_Grillage[i].End_nodeNumber;
		int sub_SE = Num_Endnode - Num_Startnode;

		TotalMatrix.AddSetPartofMatrix(3*Num_Startnode, 3*Num_Startnode, part1);
		TotalMatrix.AddSetPartofMatrix(3*Num_Startnode, 3*Num_Startnode + 3*sub_SE,
part2);
		TotalMatrix.AddSetPartofMatrix(3*Num_Startnode + 3*sub_SE, 3*Num_Startnode,
part3);
		TotalMatrix.AddSetPartofMatrix(3*Num_Startnode + 3*sub_SE, 3*Num_Startnode +
```

```
3*sub_SE, part4);
        }
}
```

이는 Grillage.cpp에서 만들어진 각 Node별 강성행렬들을 중첩하는 함수이다. 여기서 각 Node의 강성행렬인 3 by 3 행렬들을 Matrix class의 AddSetPartofMatrix의 함수를 이용하여 전체 강성행렬 TotalMatrix 안에 대입하였다.

2) Nodedelta()

```
void Structure::NodeDelta()
{
        int temp = 0;
        for(int i=0; i<NumOfNode; i++)
                temp += m_Node[i]->m_BC.CheckBC;

        delta.CreateMatrix(temp, 1);
        temp_Force.CreateMatrix(temp, 1);

        Matrix temp_TotalMat;
        temp_TotalMat.CreateMatrix(temp, temp);

        temp_TotalMat = TotalMatrix;

        Matrix Temp;
        Temp.CreateMatrix(TotalMatrix.row, TotalMatrix.column);

        int index = 0;
        int n = 0;
        for(int i=0; i<NumOfNode; i++)
        {
                // SIMPLE
                if(m_Node[i]->m_BC.CheckBC == 2)
                {
                        temp_TotalMat = temp_TotalMat.deleteRowAndColumn(3*i+2-n, 3*i+2-n,
temp_TotalMat);

                        temp_Force.element[index][0] =  m_Node[i]->MX;
                        temp_Force.element[index+1][0] =  m_Node[i]->MY;
                        index += 2;
                        n += 1;
                }
                // FREE
                else if(m_Node[i]->m_BC.CheckBC == 3)
                {
                        temp_Force.element[index][0] =  m_Node[i]->MX;
                        temp_Force.element[index+1][0] =  m_Node[i]->MY;
                        temp_Force.element[index+2][0] =  m_Node[i]->F;
                        index += 3;
                }
                // FIX
                else
                {
                        temp_TotalMat = temp_TotalMat.deleteRowAndColumn(3*i+0-n, 3*i+0-n,
temp_TotalMat);
                        temp_TotalMat = temp_TotalMat.deleteRowAndColumn(3*i-n, 3*i-n,
temp_TotalMat);
```

```
                        temp_TotalMat = temp_TotalMat.deleteRowAndColumn(3*i-n, 3*i-n,
temp_TotalMat);
                        n += 3;
                }
        }

        delta = temp_TotalMat.inverseMatrix()*temp_Force;
}
```

위의 CreateGrillage() 함수를 이용하여 전체 강성행렬을 구한 후, 우리가 모르는 delta 성분만을 끄집어 내어 새로운 강성행렬을 만들어 temp_TotalMat에 저장한 후에 이를 이용하여 우리가 아는 delta를 구하는 것이다. Gillage 분석에서는 Force를 알고 delta를 모르는 부분이거나 혹은 delta를 알고 Force를 모르는 Node이기에 이러한 방법을 사용할 수 있다. 우리가 delta성분을 모르기에 알고 있는 Force 성분과 강성행렬을 이용하여 delta성분을 구하는 함수이다.

3) NodeForce()

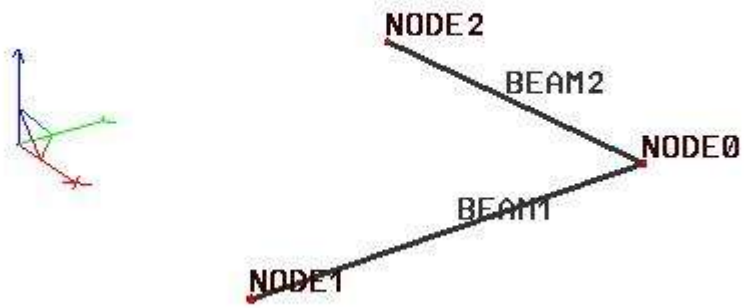
```
void Structure::NodeForce()
{
        Final_delta.CreateMatrix(3*NumOfNode, 1);
        Final_Force.CreateMatrix(3*NumOfNode, 1);

        int index = 0;
        int n = 0;
        for(int i=0; i<NumOfNode; i++)
        {
                // SIMPLE
                if(m_Node[i]->m_BC.CheckBC == 2)
                {
                        Final_delta.element[index+n][0] = delta.element[index][0];
                        Final_delta.element[index+n+1][0] = delta.element[index+1][0];

                        index += 2;
                        n += 1;
                }
                // FREE
                else if(m_Node[i]->m_BC.CheckBC == 3)
                {
                        Final_delta.element[index+n][0] = delta.element[index][0];
                        Final_delta.element[index+n+1][0] = delta.element[index+1][0];
                        Final_delta.element[index+n+2][0] = delta.element[index+2][0];

                        index += 3;
                }
                // FIX
                else
                {
                        n += 3;
                }
        }
        for (int i=0; i<3*NumOfNode; i++)
        {
                for (int j=0; j<3*NumOfNode; j++)
                {
                        Final_Force.element[i][0] +=
TotalMatrix.element[i][j]*Final_delta.element[j][0];
```

```
                }
            }
}
```

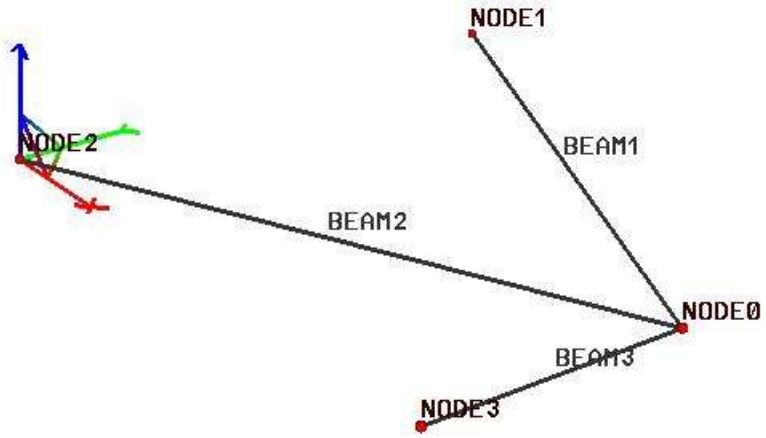
앞에서 구한 delta와 전체 강성행렬을 이용, Node에 걸리는 전체 힘과 Moment를 구하는 함수이다. 앞에서 미지의 delta를 구하였고, 전체 강성행렬을 회전축에 따라 다르게 구성하였기에 각 Node에 대한 주어진 Grillage에 작용되는 Force와 Moment를 구할 수 있게 되었다. 이렇게 힘과 모멘트를 구하면 주어진 BoundaryCondition대로 힘과 모멘트가 작용하고, delta가 주어짐을 확인할 수 있다.

Ⅱ. 결과 화면

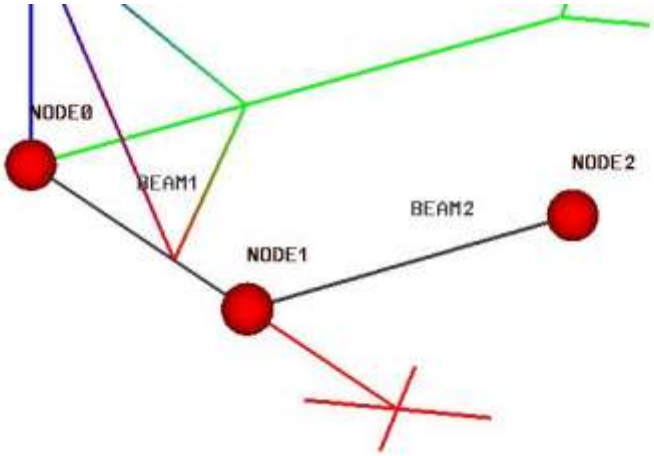
1. test1 (0.1배 scale!!!)



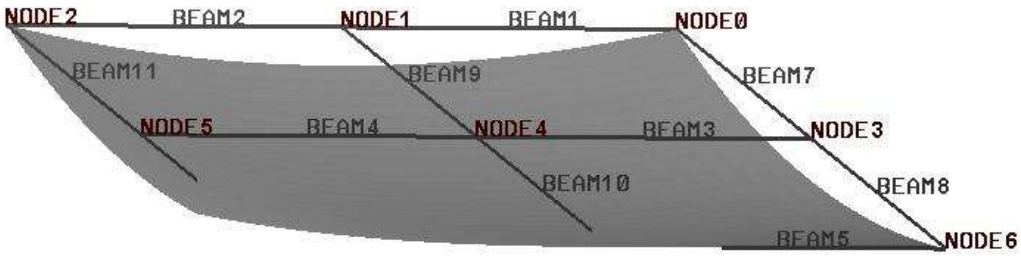
2. test2 (0.1배 scale!!!)



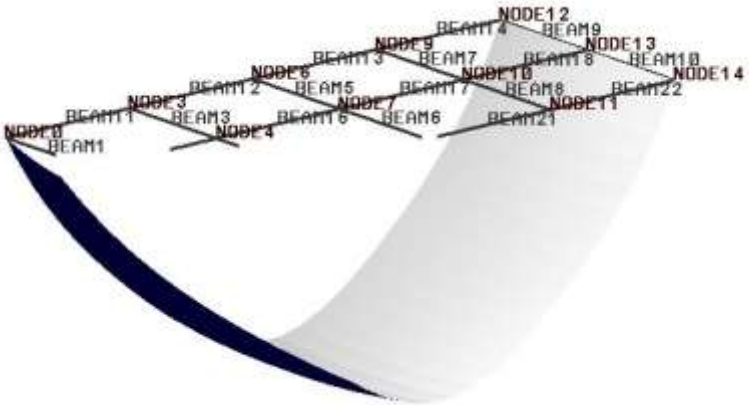
3. test3



4. test 4 (test5와 거의 동일)

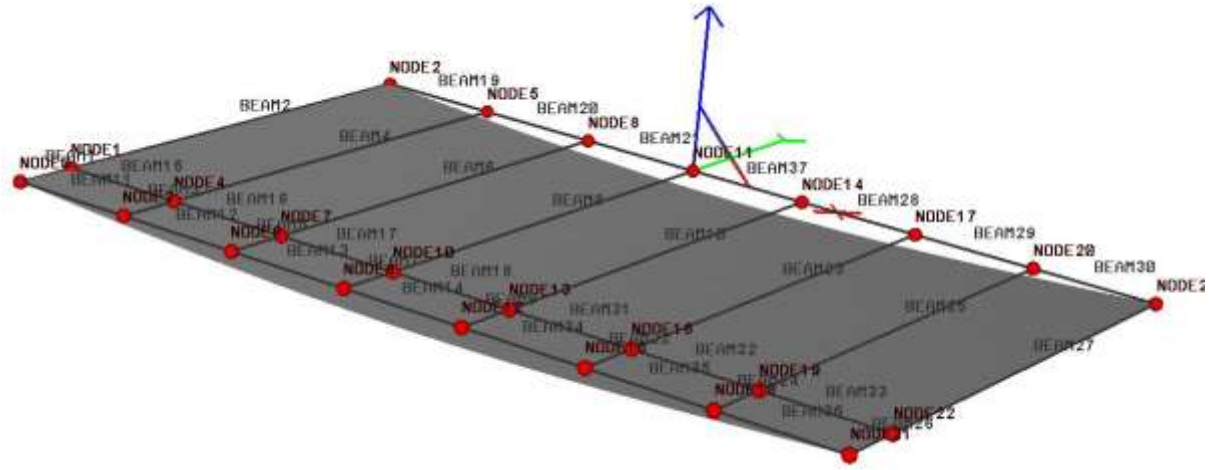


5. test 6



6. test7 (Cargo)]

① 결과 화면



② 결과값 (input & output 폴더 안에 존재합니다.)

Node	X_Position	Y_Position	Z_Position	X_Moment	Y_Moment	Z_Moment	X_theta	Y_theta	Z_delta
0	-11.400000	-16.120000	0.000000	-0.000000	0.000000	16259.305211	-0.006051	0.065523	0.000000
1	-11.400000	-14.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.004841	0.064464	-0.011551
2	-11.400000	0.000000	0.000000	0.000000	0.000000	8740.694789	0.003021	0.066565	0.000000
3	-7.600000	-16.120000	0.000000	0.000000	-0.000000	-2500.000000	-0.003311	0.058517	-0.240790
4	-7.600000	-14.120000	0.000000	0.000000	0.000000	-2500.000000	-0.002683	0.057539	-0.246874
5	-7.600000	0.000000	0.000000	0.000000	0.000000	-2500.000000	0.001755	0.059436	-0.244108
6	-3.800000	-16.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.001846	0.039660	-0.430343
7	-3.800000	-14.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.001565	0.039329	-0.433756
8	-3.800000	0.000000	0.000000	0.000000	0.000000	-2500.000000	0.000731	0.040346	-0.436738
9	0.000000	-16.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.001347	0.013988	-0.533786
10	0.000000	-14.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.001194	0.013905	-0.536331
11	0.000000	0.000000	0.000000	-0.000000	0.000000	-2500.000000	0.000244	0.014225	-0.541927
12	3.800000	-16.120000	0.000000	0.000000	0.000000	-2500.000000	-0.001347	-0.013988	-0.533786
13	3.800000	-14.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.001194	-0.013905	-0.536331
14	3.800000	0.000000	0.000000	0.000000	-0.000000	-2500.000000	0.000244	-0.014225	-0.541927
15	7.600000	-16.120000	0.000000	0.000000	-0.000000	-2500.000000	-0.001846	-0.039660	-0.430343
16	7.600000	-14.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.001565	-0.039329	-0.433756
17	7.600000	0.000000	0.000000	0.000000	0.000000	-2500.000000	0.000731	-0.040346	-0.436738
18	11.400000	-16.120000	0.000000	0.000000	0.000000	-2500.000000	-0.003311	-0.058517	-0.240790
19	11.400000	-14.120000	0.000000	-0.000000	0.000000	-2500.000000	-0.002683	-0.057539	-0.246874
20	11.400000	0.000000	0.000000	0.000000	0.000000	-2500.000000	0.001755	-0.059436	-0.244108
21	15.200000	-16.120000	0.000000	-0.000000	0.000000	16259.305211	-0.006051	-0.065523	0.000000
22	15.200000	-14.120000	0.000000	-0.000000	-0.000000	-2500.000000	-0.004841	-0.064464	-0.011551
23	15.200000	0.000000	0.000000	-0.000000	-0.000000	8740.694789	0.003021	-0.066565	0.000000

Node	X	Y	Z	X_Moment	Y_Moment	Z_Force	X_Theta	Y_Theta	Z_Delta
0	-11.40	-16.12	0.00	0.00	-0.00	16259.31	-0.00605	0.06552	0.00000
1	-11.40	-14.12	0.00	-0.00	0.00	-2500.00	-0.00484	0.06446	-0.01155
2	-11.40	0.00	0.00	-0.00	0.00	8740.69	0.00302	0.06657	0.00000
3	-7.60	-16.12	0.00	-0.00	0.00	-2500.00	-0.00331	0.05852	-0.24079
4	-7.60	-14.12	0.00	-0.00	0.00	-2500.00	-0.00268	0.05754	-0.24687
5	-7.60	0.00	0.00	-0.00	0.00	-2500.00	0.00176	0.05944	-0.24411
6	-3.80	-16.12	0.00	-0.00	0.00	-2500.00	-0.00185	0.03966	-0.43034
7	-3.80	-14.12	0.00	-0.00	0.00	-2500.00	-0.00156	0.03933	-0.43376
8	-3.80	0.00	0.00	0.00	0.00	-2500.00	0.00073	0.04035	-0.43674
9	0.00	-16.12	0.00	0.00	0.00	-2500.00	-0.00135	0.01399	-0.53379
10	0.00	-14.12	0.00	0.00	0.00	-2500.00	-0.00119	0.01391	-0.53633
11	0.00	0.00	0.00	0.00	0.00	-2500.00	0.00024	0.01422	-0.54193
12	3.80	-16.12	0.00	0.00	0.00	-2500.00	-0.00135	-0.01399	-0.53379
13	3.80	-14.12	0.00	-0.00	0.00	-2500.00	-0.00119	-0.01391	-0.53633
14	3.80	0.00	0.00	0.00	0.00	-2500.00	0.00024	-0.01422	-0.54193
15	7.60	-16.12	0.00	-0.00	0.00	-2500.00	-0.00185	-0.03966	-0.43034
16	7.60	-14.12	0.00	-0.00	0.00	-2500.00	-0.00156	-0.03933	-0.43376
17	7.60	0.00	0.00	0.00	0.00	-2500.00	0.00073	-0.04035	-0.43674
18	11.40	-16.12	0.00	0.00	-0.00	-2500.00	-0.00331	-0.05852	-0.24079
19	11.40	-14.12	0.00	0.00	0.00	-2500.00	-0.00268	-0.05754	-0.24687

값을 비교하면 소수점 버림 단위를 제외한 값이 동일함을 확인할 수 있다.

Ⅲ. 후기

이번 Grillage Program 의 경우 6차와 7차의 연장선 상에 있는 듯 하면서 아니었던 과제였습니다. 실제로 문제 풀기에는 꽤나 복잡했던 Grillage 강성방정식에 비해 연속되고 반복된 과정을 컴퓨터가 처리해 주는 Program에서는 생각보다 단순했던 과제였던 것 같습니다. 특히 BoundaryCondition을 각각의 Node에 어떻게 저장해야 할지, 그리고 그것을 어떻게 사용해야 할지 처음에는 감이 쉽게 잡히지 않았지만, 강성방정식을 사용하여 푼 28강의 Grillage 예제를 이용하여 조교님이 주신 알고리즘과 비교, 그러한 과정을 통해 조금이나마 내용을 더 잘 알수 있게 되었습니다.

프로그램을 작성할 때는 많아 보였지만, 정작 보고서를 쓸 때는 "Grillage 강성방정식을 만들고, 회전변환한 후, 이를 중첩하여, 미지의 delta를 구하고, Force를 구한다." 는 쉬운 알고리즘을 따라간다는 것 말고는 설명할 것이 없었습니다. 이는 수없이 많은 node를 계산하는 것, 그리고 강성방정식을 만들어 중첩하는 것이 어려운 것이지 그 밖에 계산하는 것은 그리 어렵지 않은 부분이라는 생각이 들었습니다. Grillage를 처음 보았을 때는 전산선박설계시간에 배운 많은 내용중에서도 제일 어렵다는 생각이 들었지만 박광필 조교님의 설명과 정채윤 조교님의 보강 덕분인지 쉬운 것을 알았습니다.

역시나 이번 프로그램도 MFC 프로그램의 문제점과 같이 도중에 Debugging이 힘든 점은 마찬가지였지만, 주위 Grillage 공부를 해 놓은 친구들과 토론해 가면서 프로그램을 작성하여 실수가 다른 프로그램들에 비해 없었던 점이 다행이었던 것 같습니다. 이번 학기동안 연구실 일에도 바쁘셨을텐데 저희에게 신경 써 주신 정채윤 조교님, 그리고 마지막 grillage 강의를 맡아 주신 박광필 조교님, 그리고 항상 저희에게 많은 것을 가르쳐 주시려는 열정이 있으시고 덕분에 저희 실력을 더욱 늘게 해 주시는 이규열 교수님 정말 이번 학기 수고 많으셨습니다.