자료구조
실습 레포트

제출일	20.04.23	**전 공**	컴퓨터소프트웨어공학과
과 목	자료구조 1	**학 번**	20184612
담당교수	홍 민 교수님	**이 름**	김동민

| 목 차 |

1. 희소행렬의 연산과 전치 프로그램

희소행렬의 연산과 전치 프로그램

- **두개의 희소 행렬 데이터를 data.txt파일에서 입력 받아, 두 행렬의 +, -, * 연산과 전치행렬을 구하는 프로그램을 작성한다. 단, 동적할당으로 2차원 배열을 생성하여 작성하여야 한다.**

1.1 문제 분석

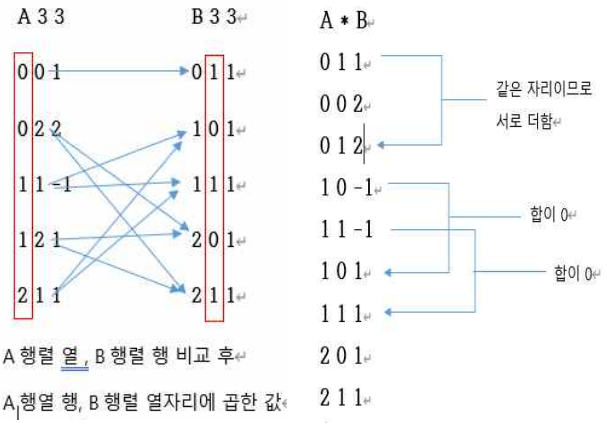
조건 1. data.txt파일로부터 희소 행렬을 입력받아온다.
조건 2. 이중 포인터를 동적 할당하여 2차원 배열을 생성한다.
조건 3. 행렬의 합, 차, 곱에 맞는 조건일 때만 수행한다.
조건 4. 행렬의 연산이 끝나면 전치행렬을 구하여 출력한다.

- 이 문제는 data파일에 행과 열이 있고 희소행렬이 저장되어 있다. 처음에 행렬의 행과 열을 입력받고, 희소행렬의 행은 저장되어있지 않기 때문에 행을 구해야 한다. 여기서 구한 행과 열은 3으로 이중 포인터를 동적 할당하여 희소행렬의 2차원 배열을 만들어야 한다. 그 다음 이중 for문을 이용해 희소행렬의 행과 열만큼 for문을 돌려서 data파일로부터 행렬을 입력받는다. 두 번째 행렬도 위와 같은 방법으로 입력받는다.

그리고 행과 열을 비교하고 조건을 확인하여 희소행렬의 덧셈, 뺄셈, 곱셈을 수행한다. 만약 덧셈, 뺄셈만 수행할 수 있거나 곱셈만 수행할 수 있는 조건이라면 가능한 것만 수행한다. 희소행렬의 연산이 끝나면, 처음에 입력받은 A행렬과 B행렬의 전치행렬을 구하여 희소행렬식으로 나타낸다.

희소 행렬의 곱 공식

희소행렬의 곱은 A행렬의 열과 B행렬의 행을 비교한다.
만약 같다면 A행렬의 행과 B행렬의 열자리에 그 자리의 데이터를 곱한 값을 넣는데, 만약 중간에 같은 자리가 나오게 되면 그 자리에 추가하여 값을 계산한다.
만약 추가했을 때 합이 0이 나온다면 그 자리는 제거하도록 한다.



1.2 소스 코드

```c
/*
    작성일: 2021.04.20
    학번: 20184612
    작성자: 김동민
    프로그램명: 2중포인터 동적할당하고 파일로부터 희소행렬을 입력받아
                행렬의 전산수행과 전치행렬을 출력하는 프로그램
*/
#include <stdio.h>
#include <stdlib.h>
#pragma warning(disable:4996)

void Matrix_Print(int **, int, int);                                    //출력함수
void Sort_Matrix(int **, int);                                          //정렬함수
void Trans_Matrix(int **, int, int*, int*);                             //전치함수
void Free_Matrix(int **, int);                                          //메모리 동적할당 해제함수
void CAL_Matrix(int **, int **, int **, int, int, int, int);            //덧셈과 뺄셈 함수
//void SUB_Matrix(int **, int **, int **, int, int, int);
void MUL_Matrix(int **, int **, int **, int, int, int);                 //곱셈함수
void ADD(int **,int**,int***,int,int, int);                             //덧셈할때 필요한 함수들의 모음
void SUB(int **,int**,int***,int,int, int);                             //뺄셈 할때 필요한 함수들의 모음
void MUL(int **,int**,int***,int,int, int);                             //곱셈 할때 필요한 함수들의 모음
void attach(int *, int,int,int);                                        //결과행렬에 값을 전달하는 함수
int GET_sparse(int **,int**,int,int, int);                              //덧셈, 뺄셈행렬의 항을 반환하는 함수
int MUL_sparse(int **,int**,int,int);                                   //곱셈행렬의 항을 반환하는 함수
int **memory_allocation(int);                                           //동적할당을 반환하는 함수

int main(){
    FILE *fp;
    int tmp;
    int a_row,a_col, b_row, b_col, res_row,res_col;                     //full matrix의 행,열
    char a_name, b_name, temp;                                          //행렬의 이름
    int a_sparse=0, b_sparse=0;                                         //희소행렬의 항
    int i, j;
    int **A_matrix, **B_matrix;                                         //a희소행렬, b희소행렬
    int **res_matrix, res_sparse=0;                                     //결과희소행렬과 결과 희소행렬의 항

    fp=fopen("data.txt","r");
    if(fp==NULL){
        printf("파일이 열리지 않았습니다.₩n");
        return 0;
    }
    fscanf(fp,"%c%d%d",&temp, &tmp, &tmp);                              //A 행 열 입력(임시변수)
    while(!feof(fp)){
        fscanf(fp,"%d%d%d",&tmp, &tmp, &tmp);
        fscanf(fp, "%c",&temp);
        if(temp=='B')break;                                             //만약 temp가 B면 종료
        a_sparse++;                                                     //a희소행렬의 갯수++
    }
    fscanf(fp,"%d%d", &tmp, &tmp);                                      //b는 입력받을 >> 행과 열값만 임시변수에

    while(!feof(fp)){                                                   //data의 끝까지 반복
        fscanf(fp,"%d%d%d",&tmp, &tmp, &tmp);
        b_sparse++;                                                     //b희소행렬의 갯수++
    }
    A_matrix = memory_allocation(a_sparse);                             //a,b행렬 동적할당
    B_matrix = memory_allocation(b_sparse);
    rewind(fp);                                                         //첫값으로 되돌림
```

```
    fscanf(fp,"%c%d%d",&a_name, &a_row, &a_col);                    //행렬의 이름, 행렬의 행, 행렬의 열
    for(i=0; i<a_sparse; i++){
        for(j=0; j<3; j++){
            fscanf(fp,"%d",&A_matrix[i][j]);                        //a희소행렬을 data로부터 입력
        }
    }
    fscanf(fp, "%c",&temp);                                         //'\n'값 패스
    fscanf(fp,"%c%d%d",&b_name, &b_row, &b_col);                    //행렬의 이름, 행렬의 행, 행렬의 열

    for(i=0; i<b_sparse; i++){
        for(j=0; j<3; j++){
            fscanf(fp,"%d",&B_matrix[i][j]);                        //b희소행렬을 data로부터 입력
        }
    }
    fclose(fp);                                                     //파일 닫음

    printf("%c 행렬 = %d x %d\n", a_name, a_row, a_col);
    Matrix_Print(A_matrix, a_sparse, 3);                            //a 희소행렬 출력

    printf("%c 행렬 = %d x %d\n", b_name, b_row, b_col);
    Matrix_Print(B_matrix, b_sparse, 3);

    res_row = a_row;                                                //만약 덧셈시 (m*n)+(m*n) = m*n행렬
    res_col = b_col;                                                //만약 곱셈시 (m*n)x(n*p) = m*p행렬
                                                                    //앞행렬의 행 * 뒤행렬의 열일때 연산을 할 수 있음

    if((a_row==b_row)&&(a_col==b_col)&&(a_col==b_row)){                                  //m*m의 정방행렬일 경우 ->합,차,곱
        printf("행렬의 합 = %d x %d\n",res_row, res_col  );
        ADD(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 차 = %d x %d\n",res_row, res_col  );
        SUB(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 곱 = %d x %d\n", res_row, res_col);
        MUL(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
    }
    else if((a_row==b_row)&&(a_col==b_col)){                                             //(m*n)+(m*n)행렬일 경우 -> 합, 차
        printf("행렬의 합 = %d x %d\n",res_row, res_col  );
        ADD(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 차 = %d x %d\n",res_row, res_col  );
        SUB(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 곱셈은 계산 불가\n");

    }
    else if(a_col==b_row){                                                               //(m*n)*(n*p)행렬일 경우 -> 곱만 
        printf("행렬의 곱 = %d x %d\n", res_row, res_col);
        MUL(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 덧셈, 뺄셈은 계산 불가\n");
    }
    else{                                                                                //나머지는 모두 계산 불가
        printf("행렬의 덧셈, 뺄셈, 곱셈 모두 계산 불가\n");
    }
    printf("\n");
```

```
    Trans_Matrix(A_matrix, a_sparse, &a_row, &a_col);           //전치행렬(a희소행렬, 행값, 원래행의 주소, 원래 열의 주소)->행과 열이 바뀜
    printf("%c의 전치>>%d x %d\n",a_name,a_row, a_col);
    Matrix_Print(A_matrix, a_sparse, 3);                        //행렬 출력(전치확인)

    Trans_Matrix(B_matrix, b_sparse, &b_row, &b_col);
    printf("%c의 전치>>%d x %d\n",b_name,b_row, b_col);
    Matrix_Print(B_matrix, b_sparse, 3);

    Free_Matrix(A_matrix, a_sparse);                            //메모리 동적할당 해제
    Free_Matrix(B_matrix, b_sparse);

    return 0;
}
void ADD(int **a, int **b,int ***res_matrix, int a_sparse, int b_sparse, int res_sparse){   //매개변수(a희소, b희소, 결과희소의 주소(3중포인터),a희소 행, b희소 행

    res_sparse = GET_sparse( a, b, a_sparse, b_sparse, 1);              //행의 값 계산후 반환받음(덧셈일땐 마지막값1)
    *res_matrix = memory_allocation(res_sparse);                        //결과 행렬 동적할당(res_matrix의 값에 반환)
    CAL_Matrix(a, b, *res_matrix, a_sparse, b_sparse, res_sparse,1);    //덧셈행렬 계산(덧셈일땐 마지막값 1)
    Matrix_Print(*res_matrix, res_sparse, 3);
    Free_Matrix(*res_matrix, res_sparse);                               //결과 행렬 출력 후 동적할당 해제
}
void SUB(int **a, int **b,int ***res_matrix, int a_sparse, int b_sparse, int res_sparse){

    res_sparse = GET_sparse( a, b, a_sparse, b_sparse,2);               //행의 값 계산후 반환받음(뺄셈일땐 마지막값2)
    *res_matrix = memory_allocation(res_sparse);
    //SUB_Matrix(a, b, res_matrix, a_sparse, b_sparse, res_sparse);
    CAL_Matrix(a, b, *res_matrix, a_sparse, b_sparse, res_sparse,2);    //뺄셈행렬 계산(뺄셈일땐 마지막값 2)
    Matrix_Print(*res_matrix, res_sparse, 3);
    Free_Matrix(*res_matrix, res_sparse);
}
void MUL(int **a, int **b,int ***res_matrix, int a_sparse, int b_sparse, int res_sparse){
    res_sparse = MUL_sparse( a, b, a_sparse, b_sparse);
    *res_matrix = memory_allocation(res_sparse);
    MUL_Matrix(a, b, *res_matrix, a_sparse, b_sparse, res_sparse);
    Matrix_Print(*res_matrix, res_sparse, 3);
    Free_Matrix(*res_matrix, res_sparse);
}

int GET_sparse(int **a,int**b,int a_sparse,int b_sparse, int flag){
    int i,j;
    int temp =  a_sparse + b_sparse;
    for(i=0;i<a_sparse; i++){
        for(j=0; j<b_sparse; j++){
            if((a[i][0]==b[j][0])&&(a[i][1]==b[j][1])){                 //a행렬과 b행렬의 주치가 같을때
                if(flag==1&&(a[i][2]+b[j][2])==0) temp-=2;              //flag값이 1일때(행렬의 합 연산)
                else if(flag==2&&(a[i][2]-b[j][2])==0) temp-=2;         //flag값이 2일때(행렬의 차 연산)
                else temp--;
            }
        }
    }
    return temp;                                                        //희소행렬의 행반환
}
```

```
int MUL_sparse(int **a, int**b, int a_sparse, int b_sparse){
    int i,j,s;
    int temp = 0, count=0;
    int **ary;
    for(i=0; i<a_sparse; i++){
        for(j=0; j<b_sparse; j++){
            if(a[i][1]==b[j][0]){
                temp++;
            }
        }
    }
    ary = memory_allocation(temp);

    for(i=0; i<a_sparse; i++){
        for(j=0; j<b_sparse; j++){
            if(a[i][1]==b[j][0]){
                for(s=0; s<count; s++){
                    if(ary[s][0]==a[i][0]&&ary[s][1]==b[j][1]){
                        ary[s][2]+=a[i][2]*b[j][2];

                        break;
                    }
                }
                if(s==count){
                    ary[count][0]=a[i][0];
                    ary[count][1]=b[j][1];
                    ary[count][2]=a[i][2]*b[j][2];
                    count++;
                }
            }
        }
    }
    count=0;
    for(i=0; i<temp; i++){
        if(ary[i][2]==0)count++;
    }
    Free_Matrix(ary, temp);
    temp-=count;
    return temp;
}


void MUL_Matrix(int **a, int **b, int **res, int a_row, int b_row, int res_row){
    int i,j;
    int s=0, count=0;
    int temp=0;
    int **ary;

    for(i=0; i<a_row; i++){
        for(j=0; j<b_row; j++){
            if(a[i][1]==b[j][0]){
                temp++;
            }
        }
    }
    ary = memory_allocation(temp);
    for(i=0; i<a_row; i++){
        for(j=0; j<b_row; j++){
            if(a[i][1]==b[j][0]){
                for(s=0; s<count; s++){
                    if(ary[s][0]==a[i][0]&&ary[s][1]==b[j][1]){
                        ary[s][2]+=a[i][2]*b[j][2];
                        break;
                    }
                }
                if(s==count){
                    ary[count][0]=a[i][0];
                    ary[count][1]=b[j][1];
                    ary[count][2]=a[i][2]*b[j][2];
                    count++;
                }
            }
        }
    }
    i=0;
    s=0;
    while(s<res_row){
        if(ary[i][2]!=0){
            attach(res[s],ary[i][0],ary[i][1],ary[i][2]);
            s++;
        }
        i++;
    }
    Free_Matrix(ary, temp);
    Sort_Matrix(res, res_row);

}
```

```
void CAL_Matrix(int **a, int **b, int **res, int a_row, int b_row, int res_row, int flag){  //매개변수(a희소, b희소, 결과희소, a희소 행, b희소 행, 결과 희소 행, flag값(1빼면 덧셈
    int i=0,j=0;                                                                            //i= a행렬의 행번수,  j = b행렬의 행번수
    int s=0;                                                                                //결과 행렬의 행번수
    int temp;                                                                               //덧셈 혹은 뺄셈을 저장하는 임시변수
    while((i<a_row)&&(j<b_row)){

        if(a[i][0]==b[j][0]){                                                               //a행렬의 행과 b행렬의 행이 같을때
            if(a[i][1]>b[j][1]){                                                            //a행렬 보다 b행렬 열이 더 작을 때 -> a행렬이 더 앞에 오름
                if(flag==1) temp = b[j][2];                                                 //1빼 면 덧셈이므로 그냥 b값 대입, 2빼면 뺄셈이므로 -b값 대입
                else if(flag==2) temp = -b[j][2];
                attach(res[s], b[j][0],b[j][1],temp);                                       //결과행렬에 대입
                s++;
                j++;
            }
            else if(a[i][1]==b[j][1]){                                                      //a행렬의 행,열과 b행렬의 행,열이 모두 같을 때
                if(flag==1&&a[i][2]+b[j][2]!=0){                                            //flag값이 1이고 더했을때 0이 아닐때 덧셈수행
                    temp=a[i][2]+b[j][2];
                    attach(res[s], a[i][0],b[j][1],temp);                                   //결과 행렬에 저장
                    s++;
                    i++;
                    j++;
                }
                else if(flag==2&&a[i][2]-b[j][2]!=0){                                       //flag값이 2이고 뺐을때 0이 아닐때 뺄셈 수행
                    temp=a[i][2]-b[j][2];
                    attach(res[s], a[i][0],b[j][1],temp);
                    s++;
                    i++;
                    j++;
                }
                else{                                                                       //행렬의 합 모든 차가 0이므로 결과행렬에 합당하지 않음.
                    i++;
                    j++;
                }
            }
            else{                                                                           //a행렬 보다 b행렬 열이 더 클 때 -> a행렬이 더 앞에 오름
                attach(res[s], a[i][0],a[i][1],a[i][2]);                                    //a행렬 결과행렬에 대입
                s++;
                i++;
            }
        }
        else if(a[i][0]<b[j][0]){                                                           //a행렬의 행이 더 작을때 ->a행렬이 더 앞에 오름. a행렬 대입
            attach(res[s], a[i][0],a[i][1],a[i][2]);
            s++;
            i++;
        }
        else{                                                                               //a행렬의 행이 더 클때 ->b행렬이 더 앞에 오름 b행렬 대입
            if(flag==1) temp = b[j][2];                                                     //1빼 면 덧셈이므로 그냥 b값 대입, 2빼면 뺄셈이므로 -b값 대입
            else if(flag==2) temp = -b[j][2];
            attach(res[s], b[j][0],b[j][1],temp);
            s++;
            j++;

        }

    }
    while(i<a_row){                                                                         //a의 나머지 항목들 이동
        attach(res[s], a[i][0],a[i][1],a[i][2]);
        s++;
        i++;
    }
    while(j<b_row){                                                                         //b의 나머지 항목들 이동
        if(flag==1) temp = b[j][2];
        else if(flag==2) temp = -b[j][2];                                                   //1빼 면 덧셈이므로 그냥 b값 대입, 2빼면 뺄셈이므로 -b값 대입
        attach(res[s], b[j][0],b[j][1],temp);
        s++;
        j++;
    }
}*/


void attach(int *a, int a, int b, int c){
    a[0]=a;                                                                                 //a는 결과행렬, a[0]값에 a, a[1]값에 b, a[2]값에 c대입
    a[1]=b;
    a[2]=c;
}

int **memory_allocation(int sparse){                                                        //동적할당 함수(매개변수-희소행렬의 행)
    int i,j;
    int **a;
    if(sparse<0){                                                                           //만약 매개변수 희소행렬의 행이 0보다 작으면 동적할당 불가
        printf("메모리가 존재하지 않습니다.₩n");
        return 0;
    }

    a = (int **)malloc(sizeof(int*)*sparse);                                                //이중포인터 a를 희소행렬의 행으로 동적할당
    for(i=0; i<sparse; i++){                                                                //희소행렬의 열갯수 3이므로 for문을 돌려 a[i]를 3으로 동적할당
        a[i] = (int*)malloc(sizeof(int)*3);
        for(j=0; j<3; j++){                                                                 //동적할당과 함께 0으로 초기화
            a[i][j]=0;
        }
    }
    return a;                                                                               //동적할당한 이중포인터 a를 반환
}

void Sort_Matrix(int **matrix, int row){                                                    //정렬함수(매개변수 - 희소행렬, 희소행렬의 행갯)
    int i,r,tmp,k;
    for(i=0; i<row; i++){
        for(r=0; r<row; r++){
            if(matrix[i][0]<matrix[r][0]){                                                  //만약 현재 행렬의 행갯보다 위에 있는 행렬이 더 크다면 정렬
                for(k=0; k<3; k++){
                    tmp=matrix[i][k];
                    matrix[i][k] = matrix[r][k];
                    matrix[r][k]=tmp;
                }
            }
            if((matrix[i][0]==matrix[r][0]) && (matrix[i][1]<matrix[r][1]) ){           //정렬을 모두 마친 후 행렬의 열에 대하여 정렬, 만약 행갯이 같고 열갯이
                for(k=0; k<3; k++){
                    tmp=matrix[i][k];
                    matrix[i][k] = matrix[r][k];
                    matrix[r][k]=tmp;
                }
            }
        }
    }
}
```

```
void Trans_Matrix(int **matrix, int sparse_row, int* row, int *col){          //전치 함수(매개변수 - 희소행렬, 희소행렬의 행값, full-matrix의 행과 열의 주소)
    int i, tmp;
    tmp = *row;                                                               //full-matrix행 열값을 전치
    *row = *col;
    *col = tmp;

    for(i=0; i<sparse_row; i++){                                              //희소행렬의 행, 열값을 전치
        tmp=matrix[i][1];
        matrix[i][1] = matrix[i][0];
        matrix[i][0]=tmp;
    }
    Sort_Matrix(matrix, sparse_row);                                          //전치행렬을 정렬 -> 정렬함수
}

void Matrix_Print(int **matrix, int row, int col){                            //출력함수(매개변수-희소 행렬, 희소행렬의 행, 열값)
    int i,j;
    for(i=0; i<row; i++){
        for(j=0; j<col; j++){
            if(j==2) printf(" :");                                            //j가 2일때 :을 출력하여 위치와 데이터 구분
            printf("%3d", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
void Free_Matrix(int **matrix, int row){                                      //메모리 동적할당 해제 함수(매개변수 - 희소행렬, 희소행렬 행값)
    int i;
    for(i=0; i<row; i++){                                                     //행만큼 for문을 돌며 free
        free(matrix[i]);
    }
    free(matrix);                                                             //행값 free
}
```

1.3 소스 코드 분석

```
/*
    작성일: 2021.04.20
    학번: 20184612
    작성자: 김동민
    프로그램명: 2중포인터 동적할당하고 파일로부터 희소행렬을 입력받아
                행렬의 연산수행과 전치행렬을 출력하는 프로그램
*/
#include <stdio.h>
#include <stdlib.h>
#pragma warning(disable:4996)

void Matrix_Print(int **, int, int);
void Sort_Matrix(int **, int);
void Trans_Matrix(int **, int, int*, int*);
void Free_Matrix(int **, int);
void CAL_Matrix(int **, int **, int **, int, int, int, int);
//void SUB_Matrix(int **, int **, int **, int, int, int);
void MUL_Matrix(int **, int **, int **, int, int, int);
void ADD(int **,int**,int***,int,int, int);
void SUB(int **,int**,int***,int,int, int);
void MUL(int **,int**,int***,int,int, int);
void attach(int *, int,int,int);
int GET_sparse(int **,int**,int,int, int);
int MUL_sparse(int **,int**,int,int);
int **memory_allocation(int);
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성한다.
2. 필요한 헤더를 포함한다.
3. 필요한 함수들을 선언한다.

- Matrix_print함수는 희소행렬을 출력하는 함수이다.
- Sort_Matrix함수는 행렬의 전치 후 정렬하는 함수이다.
- Trans_Matrix는 전치행렬을 수행하는 함수이다.
- Free_Matrix는 메모리 동적할당을 해제하는 함수이다.
- CAL_Matrix는 희소행렬의 덧셈과 뺄셈을 수행하는 함수이다.
- MUL_Matrix는 희소행렬의 곱셈을 수행하는 함수이다.
- ADD, SUB, MUL함수는 각각의 계산에 필요한 함수들을 한곳에 모아놓은 함수로 희소행렬의 행을 계산하는 것부터 메모리동적할당, 계산, 출력, 동적할당 해제를 하는 함수를 가지고 있다.
- attach함수는 결과 값 행렬에 값을 넣는 함수이다.
- GET_sparse와 MUL_sparse함수는 희소행렬의 행을 연산하여 반환하는 함수이다. GET은 덧셈과 뺄셈의 행을 계산한다.
- **memory_allocation함수는 희소행렬의 행만큼 동적 할당하여 이중포인터를 반환하는 함수이다.

```
int main( ){
    FILE *fp;
    int tmp;
    int a_row,a_col, b_row, b_col, res_row,res_col;
    char a_name, b_name, temp;
    int a_sparse=0, b_sparse=0;
    int i, j;
    int **A_matrix, **B_matrix;
    int **res_matrix, res_sparse=0;

    fp=fopen("data.txt","r");
    if(fp==NULL){
        printf("파일이 열리지 않았습니다.₩n");
        return 0;
    }
```

4. 필요한 변수들을 선언한다.

- fp는 FILE구조체를 가리키는 포인터 변수이다.
- int tmp와 char temp는 희소행렬의 행 값을 구할 때 임시로 사용하는 변수이다.
- a_row부터 res_col까지의 변수는 본래 full matrix행렬의 행과 열을 나타낸다.
- a_name과 b_name은 행렬의 이름을 가지는 변수이다.
- a_sparse와 b_sparse는 희소행렬의 행 값을 가지는 변수이다.
- i와 j는 for문을 돌리기 위한 변수이다.
- **A_matrix와 **B_matrix는 희소행렬을 받을 이중포인터이다. sparse변수에 행 값을 저장하면 이것으로 동적할당을 한다.
- **res_matrix와 res_sparse는 결과행렬과 결과행렬의 행을 저장한다. 결과 행렬의 행을 구하는 것과 결과행렬 동적할당은 행렬의 연산을 수행할 때만 사용한다.

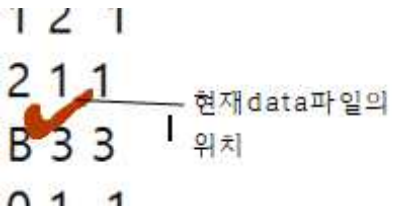
5. data파일을 읽기 전용으로 개방한다.
만약 fp가 NULL이면 “파일이 열리지 않았습니다.”를 화면에 출력하고 프로그램을 종료한다.

```
}
fscanf(fp,"%c%d%d",&temp, &tmp, &tmp);
while(!feof(fp)){
    fscanf(fp,"%d%d%d",&tmp, &tmp, &tmp);
    fscanf(fp, "%c",&temp);
    if(temp=='B')break;
    a_sparse++;
}
fscanf(fp,"%d%d", &tmp, &tmp);

while(!feof(fp)){
    fscanf(fp,"%d%d%d",&tmp, &tmp, &tmp);
    b_sparse++;
}
A_matrix = memory_allocation(a_sparse);
B_matrix = memory_allocation(b_sparse);
rewind(fp);
```

6. 희소행렬의 행을 구하는 과정이다.

먼저 char temp와 int tmp를 이용해 이름과 본래 행렬의 값을 받는다. 그리고 !feof를 이용하여 data로부터 임시 변수에 값을 받으면서 a희소행렬의 행 값을 1씩 증가시킨다. 그리고 만약 temp에 B가 들어오면 반복을 종료한다.



여기서 반복이 종료되면 data파일은 B다음 위치에 있다. 그러므로 행과 열값만 임시변수에 담고, !feof를 통해 파일이 끝날 때 까지 반복하여 b희소행렬의 행 값을 증가시킨다.

7. 위에서 계산한 희소행렬의 행을 통해 A희소행렬과 B희소행렬을 동적 할당한다. memory_allocation함수는 매개변수로 행을 전달하면, 함수 안에서 행에 대하여 동적 할당하여 이중포인터를 반환하는 함수이다.
동적할당으로 인해 A희소행렬과 B희소행렬은 이차원 배열의 값을 가진다.

8. 파일 입력이 모두 끝났으므로 파일의 맨 처음으로 되돌리는 rewind함수를 호출한다.

```
fscanf(fp,"%c%d%d",&a_name, &a_row, &a_col);
for(i=0; i<a_sparse; i++){
    for(j=0; j<3; j++){
        fscanf(fp,"%d",&A_matrix[i][j]);
    }
}
fscanf(fp, "%c",&temp);
fscanf(fp,"%c%d%d",&b_name, &b_row, &b_col);

for(i=0; i<b_sparse; i++){
    for(j=0; j<3; j++){
        fscanf(fp,"%d",&B_matrix[i][j]);
    }
}
fclose(fp);

printf("%c 행렬 = %d x %d₩n", a_name, a_row, a_col);
Matrix_Print(A_matrix, a_sparse, 3);

printf("%c 행렬 = %d x %d₩n", b_name, b_row, b_col);
Matrix_Print(B_matrix, b_sparse, 3);
```

9. 이제 data로부터 값을 저장한다.
a_name에 행렬의 이름, a_row와 a_col에는 원래 행렬의 행, 열값을 입력받고, 이중 for문으로 A희소행렬에 값을 입력받는다. 바깥쪽 for문은 위에서 구한 행으로 반복하고 희소행렬의 열은 무조건 3이기 때문에 안쪽 for문은 3으로 반복한다.
그 다음 B의 값을 입력받아야 하는데, 바로 “%c”로 입력받으면 data는 ‘\n'을 가리켜서 '\n'값을 입력받게 된다. 따라서 “%c”로 한번만 임시변수에 data('\n')를 입력받은 후 B행렬의 이름과 행, 열값과 이중 for문으로 희소행렬의 값을 입력받는다.

10. 파일 입력이 모두 끝났고, 파일을 더 이상 쓸 일이 없기 때문에 fclose를 통해 파일을 닫아주도록 한다.

11. 행렬이 잘 입력되었는지 확인한다.
행렬의 이름과 행, 열을 출력하고, Matrix_print함수를 통해 희소행렬이 data로부터 잘 입력되었는지 확인한다. Matrix_Print의 매개변수로는 이중 포인터인 matrix희소행렬과 행의 값, 열의 값을 전달하도록 한다.

```
    res_row = a_row;
    res_col = b_col;

    if((a_row==b_row)&&(a_col==b_col)&&(a_col==b_row)){
        printf("행렬의 합 = %d x %d₩n",res_row, res_col  );
        ADD(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 차 = %d x %d₩n",res_row, res_col  );
        SUB(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 곱 = %d x %d₩n", res_row, res_col);
        MUL(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
    }
    else if((a_row==b_row)&&(a_col==b_col)){
        printf("행렬의 합 = %d x %d₩n",res_row, res_col  );
        ADD(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 차 = %d x %d₩n",res_row, res_col  );
        SUB(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 곱셈은 계산 불가₩n");

    }
    else if(a_col==b_row){
        printf("행렬의 곱 = %d x %d₩n", res_row, res_col);
        MUL(A_matrix, B_matrix,&res_matrix, a_sparse, b_sparse, res_sparse);
        printf("행렬의 덧셈, 뺄셈은 계산 불가₩n");
    }
    else{
        printf("행렬의 덧셈, 뺄셈, 곱셈 모두 계산 불가₩n");
    }
    printf("₩n");
```

12. A+B행렬의 경우 (m*n)+(m*n)의 식이고 결과는 (m*n)행렬이고, A*B 행렬의 경우 (m*n)*(n*n)의 식이고 결과는 (m*p)행렬이다. 따라서 결과행렬의 행 값은 A행렬의 행 값이고, 결과행렬의 열값은 B행렬의 열값임을 알 수 있다. 따라서 res_row엔 a_row를, res_col엔 b_col의 값을 넣는다.

13. ((a_row==b_row)&&(a_col==b_col))&&(a_col==b_row)의 경우 모든 행렬의 행과 열이 동일한 (m*m)행렬, 따라서 정방행렬임을 알 수 있다. 정방행렬은 행렬의 합, 차, 곱 모두 가능하다. 따라서 덧셈함수, 뺄셈함수, 곱셈함수를 모두 호출한다.

14. (a_row==b_row)&&(a_col==b_col)은 (m*n)+(m*n)의 행렬로 행렬의 덧셈과 뺄셈만 가능하고 A 행렬의 열과 B 행렬의 행이 다르기 때문에 곱은 불가능하다 . 따라서 덧셈함수, 뺄셈함수만 호출하고 행렬의 곱은 계산 불가라는 메시지를 보인다.

15. (a_col==b_row)은 (m*n)*(n*p)의 행렬로 A 행렬의 열과 B 행렬의 행이 같기 때문에 행렬의 곱만 가능하다. 이 경우 각 행렬의 행과 열이 다르기 때문에 덧셈과 뺄셈은 불가능하다. 따라서 행렬의 곱 함수만 호출하고 합과 차는 계산이 불가 한다는 메시지를 보인다.

16. 행렬의 합, 차, 곱은 모두 매개변수로 A희소행렬, B희소행렬, 결과행렬, A희소 행 값, B희소 행 값, 결과 희소 행 값을 공통적으로 전달한다.
여기서 결과행렬은 아직 동적할당하지 않은 상태이기 때문에 주소를 전달하고, 함수에선 삼중포인터로 전달받아 동적할당 할 것이다.

17. 만약 이 조건에 모두 부합하지 않는다면 행렬의 합, 차, 곱이 모두 계산 불가라는 메시지를 화면에 출력하고 연산을 종료한다.

```
    Trans_Matrix(A_matrix, a_sparse, &a_row, &a_col);
    printf("%c의 전치>>%d x %d₩n",a_name,a_row, a_col);
    Matrix_Print(A_matrix, a_sparse, 3);

    Trans_Matrix(B_matrix, b_sparse, &b_row, &b_col);
    printf("%c의 전치>>%d x %d₩n",b_name,b_row, b_col);
    Matrix_Print(B_matrix, b_sparse, 3);

    Free_Matrix(A_matrix, a_sparse);
    Free_Matrix(B_matrix, b_sparse);

    return 0;
}
```

18. 행렬의 계산이 끝나면 전치행렬을 구한다. 전치행렬의 값을 구하기 위해선 Trans_Matrix함수를 호출한다. 매개변수로는 희소행렬과, 희소행렬의 행 값 그리고 본래 행과 열의 값의 주소를 전달한다. 2*3행렬을 전치하면 3*2행렬이 되듯이, 전치행렬을 하면 본래 행렬의 행 값과 열값이 뒤바뀌기 때문에 주소를 전달하여 값을 바꾼다.

19. 전치행렬을 구했으면 행렬의 이름과 행, 열값을 출력하여 전치가 되었는지 확인한다. Print함수를 통해서도 전치가 잘 되었는지 확인한다.

20. 모든 프로그램이 끝나면 Free_Matrix함수를 호출하여 메모리 동적할당을 해제한다. 매개변수로 희소행렬과 행 값을 전달하면 함수에서 동적할당을 해제할 것이다.

**함수의 구현 부

```
void ADD(int **a, int **b,int ***res_matrix, int a_sparse, int b_sparse, int res_sparse){

    res_sparse = GET_sparse( a, b, a_sparse, b_sparse, 1);
    *res_matrix = memory_allocation(res_sparse);
    CAL_Matrix(a, b, *res_matrix, a_sparse, b_sparse, res_sparse,1);
    Matrix_Print(*res_matrix, res_sparse, 3);
    Free_Matrix(*res_matrix, res_sparse);
}
void SUB(int **a, int **b,int ***res_matrix, int a_sparse, int b_sparse, int res_sparse){

    res_sparse = GET_sparse( a, b, a_sparse, b_sparse,2);
    *res_matrix = memory_allocation(res_sparse);
    //SUB_Matrix(a, b, res_matrix, a_sparse, b_sparse, res_sparse);
    CAL_Matrix(a, b, *res_matrix, a_sparse, b_sparse, res_sparse,2);
    Matrix_Print(*res_matrix, res_sparse, 3);
    Free_Matrix(*res_matrix, res_sparse);
}
void MUL(int **a, int **b,int ***res_matrix, int a_sparse, int b_sparse, int res_sparse){
    res_sparse = MUL_sparse( a, b, a_sparse, b_sparse);
    *res_matrix = memory_allocation(res_sparse);
    MUL_Matrix(a, b, *res_matrix, a_sparse, b_sparse, res_sparse);
    Matrix_Print(*res_matrix, res_sparse, 3);
    Free_Matrix(*res_matrix, res_sparse);
}
```

21. 여기 있는 함수는 덧셈함수, 뺄셈함수, 곱셈함수이다.
함수의 매개변수로는 a희소행렬, 희소행렬과 삼중포인터로 결과행렬의 주소를 가져오고, a의 행, b의 행, 결과행렬의 행을 매개변수로 가져온다.

22. ADD와 SUB함수에는 행을 계산하여 결과행렬의 행을 반환하는 GET함수와 결과행렬을 동적 할당하는 함수, 계산함수, 출력함수, 마지막으로 결과행렬의 메모리를 해제하는 함수가 존재한다.
결과행렬은 현재 삼중포인터로 원래 결과행렬의 주소를 가지고 있기 때문에 결과행렬을 사용할 땐 *을 붙여 값으로 사용한다.
- 행 반환 함수와 계산함수의 경우, 중복되는 코드가 많아 한 함수에 작성하고 마지막 매개변수가 1일 때 덧셈을 수행하고, 2일 때 뺄셈을 수행하도록 작성하였다.

23. MUL함수에는 행을 계산하여 결과행렬의 행을 반환하는 함수와 동적할당, 계산함수, 출력함수, 메모리 해제 함수가 있다. 행 반환 함수나 계산함수의 경우 덧셈, 뺄셈과 다른 코드로 계산하기 때문에 따로 작성하였다.
결과행렬은 현재 삼중포인터로 원래 결과행렬의 주소를 가지고 있기 때문에 결과행렬을 사용할 땐 *을 붙여 값으로 사용한다.

```
int GET_sparse(int **a,int**b,int a_sparse,int b_sparse, int flag){
    int i,j;
    int temp =  a_sparse + b_sparse;
    for(i=0;i<a_sparse; i++){
        for(j=0; j<b_sparse; j++){
            if((a[i][0]==b[j][0])&&(a[i][1]==b[j][1])){
                if(flag==1&&(a[i][2]+b[j][2])==0) temp-=2;
                else if(flag==2&&(a[i][2]-b[j][2])==0) temp-=2;
                else temp--;
            }
        }
    }
    return temp;
}
```

24. 덧셈, 뺄셈 계산 시 결과 희소행렬의 행을 구하여 반환하는 함수이다. 매개변수로는 A희소, B희소, a희소 행, b희소 행, flag가 1일 땐 덧셈, flag가 2일 땐 뺄셈계산을 한다.

25. temp에 a희소 행과 b희소 행을 더한다.
만약 a행이 5이고 b행이 6이면 최대로 나올 수 있는 값이 11이기 때문이다 그리고 for문을 돌리며 만약 같은 행과 열이 있다면 행을 빼는 식으로 프로그래밍 해보았다. 만약 a행렬의 행과 b행렬의 행이 같다면 같은 행, 열이 중복된 것이기 때문에 temp를 1씩 뺀다.

26. 만약 덧셈계산 때(flag가 1일 때) a행과 b행의 합이 0이면 같은 행, 열이 중복되고, 0이기 때문에 희소행렬에선 제외하므로 -2를 한다.
같은 원리로 뺄셈계산 때(flag가 2일 때) a행과 b행의 차가 0이면 0이면 같은 행, 열이 중복되고, 0이기 때문에 희소행렬에서 제외한다. 따라서 temp의 값을 -2한다.

27. for문을 모두 돌면 temp에 저장된 값은 덧셈 혹은 뺄셈의 결과 행렬의 행 값과 같을 것이다. 따라서 temp를 반환하여 결과 행렬의 행 값에 대입하면 된다.

```
int MUL_sparse(int **a, int**b, int a_sparse, int b_sparse){
    int i,j,s;
    int temp = 0, count=0;
    int **ary;
    for(i=0; i<a_sparse; i++){
        for(j=0; j<b_sparse; j++){
            if(a[i][1]==b[j][0]){
                temp++;
            }
        }
    }
    ary = memory_allocation(temp);

    for(i=0; i<a_sparse; i++){
        for(j=0; j<b_sparse; j++){
            if(a[i][1]==b[j][0]){
                for(s=0; s<count; s++){
                    if(ary[s][0]==a[i][0]&&ary[s][1]==b[j][1]){
                        ary[s][2]+=a[i][2]*b[j][2];

                        break;
                    }
                }
                if(s==count){
                    ary[count][0]=a[i][0];
                    ary[count][1]=b[j][1];
                    ary[count][2]=a[i][2]*b[j][2];
                    count++;
                }
            }
        }
    }
    count=0;
    for(i=0; i<temp; i++){
        if(ary[i][2]==0)count++;
    }
    Free_Matrix(ary, temp);
    temp-=count;
    return temp;
}
```

28. 곱셈계산 시 결과행렬의 행 값을 구하여 반환하는 함수이다.
매개 변수로는 A희소, B희소, a희소 행, b희소 행을 받아온다.
희소행렬의 곱셈계산을 할 때, A행렬의 열과 B행렬의 행이 같을 때 계산이 성립한다. 따라서 A행렬의 열과 B행렬의 행이 같을 때 temp값을 1씩 더하면 곱셈 계산 시 최대로 나올 수 있는 수가 나온다.

29. 이중포인터 임시변수 ary를 최대로 나올 수 있는 수인 temp로 동적 할당한다. 왜냐하면 ary에 먼저 곱 계산을 한 후 결과 값이 0인 나온 수를 빼서 temp에 저장할 것이기 때문이다.

30. 곱셈을 수행하기 위해선 먼저 A행렬의 열과 B행렬의 행이 같아야 한다. 만약 A행렬의 열과 B행렬의 행이 같다면, 한번 for문을 돌며 원래 ary에 저장되어 있는 행, 열과 같은지 확인한다. 만약 여기서 같다면 그 자리에 a의 데이터 값과 b의 데이터 값을 곱하여 더하고 break로 for문을 빠져나온다.

31. 만약에 원래 저장되어 있던 행, 열과 모두 달라 for문을 빠져나오게 된다면, s값은 ary행렬의 길이와 같기 때문에 ary의 끝자리에 값을 추가하게 된다.

32. ary에 모든 값이 입력되면, count값을 0으로 설정하고 총 temp의 개수만큼 for문을 돌린다. 여기서 만약 ary데이터 값이 0이 나오면 count를 1씩 증가시킨다.

33. ary의 이용이 모두 끝났으므로 ary는 동적할당 해제를 해준다. 매개변수로는 ary와 temp를 전달한다.
위에서 계산으로 ary임시 희소행렬의 값에서 값이 0일 때를 모두 셌으므로 temp에서 이를 빼면 0이 아닌 수들만 남을 것이다. 그러므로 temp에서 count를 빼고 temp를 리턴한다.
그러면 temp값은 결과 희소행렬의 행으로 전달 될 것이다.

```
void CAL_Matrix(int **a, int **b, int **res, int a_row, int b_row, int res_row, int flag){
    int i=0,j=0;
    int s=0;
    int temp;
    while((i<a_row)&&(j<b_row)){

        if(a[i][0]==b[j][0]){
            if(a[i][1]>b[j][1]){
                if(flag==1) temp = b[j][2];
                else if(flag==2) temp = -b[j][2];
                attach(res[s], b[j][0],b[j][1],temp);
                s++;
                j++;
            }
            else if(a[i][1]==b[j][1]){
                if(flag==1&&a[i][2]+b[j][2]!=0){
                    temp=a[i][2]+b[j][2];
                    attach(res[s], a[i][0],b[j][1],temp);
                    s++;
                    i++;
                    j++;
                }
                else if(flag==2&&a[i][2]-b[j][2]!=0){
                    temp=a[i][2]-b[j][2];
                    attach(res[s], a[i][0],b[j][1],temp);
                    s++;
                    i++;
                    j++;
                }
                else{
                    i++;
                    j++;
                }
            }
            else{
                attach(res[s], a[i][0],a[i][1],a[i][2]);
                s++;
                i++;
            }
        }
```

34. 희소행렬의 덧셈과 뺄셈을 수행하는 함수이다.
매개변수로는 A희소행렬, B희소행렬, a희소 행, b희소 행, flag값(1일 땐 덧셈, 2일 땐 뺄셈 수행)을 가진다.
원래 덧셈 뺄셈을 따로 계산했지만 중복되는 코드가 많아 한 번에 작성하였다. s는 결과행렬의 행을 나타내고, i는 A의 행, j는 B의 행을 나타낸다.

35. 첫 조건은 A행렬 행과 B행렬 행이 같을 때이다.
첫 번째 조건의 안에서 A행렬의 열과 B행렬의 열을 비교한다.
A행렬의 열과 B행렬의 열을 비교하여 B행렬이 더 작으면 B가 더 앞에 있으므로 B를 대입하는데 만약 flag값이 1이면 덧셈이므로 결과행렬에 b를 대입하고 만약 2이면 뺄셈이므로 결과행렬에 -b를 대입한다.
b를 결과행렬에 대입했으므로 결과행렬의 위치를 나타내는 s와 b의 위치를 나타내는 j를 각각 1씩 증가시킨다.

36. attach함수는 결과행렬에 값을 대입할 때 사용하는 함수이다.
attach의 매개변수로 결과행렬의 행 값만 보내고 나머지는 대입할 행, 열, 데이터 값을 전달한다.

37. A행렬의 행과 B행렬의 행을 비교하여 둘이 같다면,

① 만약 flag가 1일 때 a와 b를 더하여 0이 아니라면 결과행렬의 행에 a행, 열에 b열, temp에 합 계산을 하여 결과 행렬에 저장한다.
a와 b와 s모두 사용했으므로 모두 1씩 증가시킨다.

② 만약 flag가 2일 때 a와 b를 빼서 0이 아니라면 결과행렬의 행에 a행, 열에 b열, temp에 차 계산을 하여 결과 행렬에 저장한다.
a와 b와 s모두 사용했으므로 모두 1씩 증가시킨다.

③ 만약 더하거나 뺐을 때 값이 0이 나온다면 결과행렬에 저장하지 않고 i값과 j값을 증가시킨다.

38. A행렬의 행과 B행렬의 행을 비교하여 A행렬이 더 작다면, A가 더 앞에 있으므로 A를 대입한다. A는 앞자리이기 때문에 flag값에 관계없이 +이므로 결과행렬에 바로 대입하면 된다.
a행렬을 결과 행렬에 대입했으므로 i와 s를 1씩 증가시킨다.

```
        else if(a[i][0]<b[j][0]){
            attach(res[s], a[i][0],a[i][1],a[i][2]);
            s++;
            i++;
        }
        else{
            if(flag==1) temp = b[j][2];
            else if(flag==2) temp = -b[j][2];
            attach(res[s], b[j][0],b[j][1],temp);
            s++;
            j++;

        }

    }
    while(i<a_row){
        attach(res[s], a[i][0],a[i][1],a[i][2]);
        s++;
        i++;
    }
    while(j<b_row){
        if(flag==1) temp = b[j][2];
        else if(flag==2) temp = -b[j][2];
        attach(res[s], b[j][0],b[j][1],temp);
        s++;
        j++;
    }
}
```

39. 두 번째 조건은 a행이 b행보다 작을 때이다.
이 경우 A행렬이 더 앞에 있으므로 결과 값에 a값을 저장하면 된다.
A행렬은 덧셈, 뺄셈에 관계없이 +이므로 결과행렬에 바로 저장해도 된다.
a행렬을 결과 행렬에 대입했으므로 i와 s를 1씩 증가시킨다.

40. 마지막 조건은 b행이 a행보다 작을 때이다.
이 경우는 B행렬이 더 앞에 있으므로 결과 값에 b값을 저장하는데, 만약 flag값이 1이면 덧셈이므로 temp에 +b를 대입하여 결과에 저장하고 만일 flag값이 2이면 뺄셈이므로 temp에 -b를 대입하여 결과에 저장한다.
b행렬을 결과 행렬에 대입했으므로 j와 s를 1씩 증가시킨다.

41. 반복 문을 모두 돌고나면 A행렬엔 값이 없는데 B행렬엔 값이 남아있거나, 반대로 A행렬에 값이 남아있는 경우가 있을 수 있다.
이때 남은 값들을 결과행렬에 대입하는 코드이다.
이때도 a의 경우는 바로 결과행렬에 넣지만, b의 경우는 flag값이 1이냐 2이냐에 따라서 temp값이 달라진다.

42. A행렬과 B행렬의 위치에 따라 결과행렬을 만들었기 때문에, 이렇게 결과행렬을 만들고 나면, 결과행렬은 정렬하지 않아도 오름차순 정렬이 되어 있을 것이다.

```
void MUL_Matrix(int **a, int **b, int **res, int a_row, int b_row, int res_row){
    int i,j;
    int s=0, count=0;
    int temp=0;
    int **ary;

    for(i=0; i<a_row; i++){
        for(j=0; j<b_row; j++){
            if(a[i][1]==b[j][0]){
                temp++;
            }
        }
    }
    ary = memory_allocation(temp);
    for(i=0; i<a_row; i++){
        for(j=0; j<b_row; j++){
            if(a[i][1]==b[j][0]){
                for(s=0; s<count; s++){
                    if(ary[s][0]==a[i][0]&&ary[s][1]==b[j][1]){
                        ary[s][2]+=a[i][2]*b[j][2];
                        break;
                    }
                }
                if(s==count){
                    ary[count][0]=a[i][0];
                    ary[count][1]=b[j][1];
                    ary[count][2]=a[i][2]*b[j][2];
                    count++;
                }
            }
        }
    }
    i=0;
    s=0;
    while(s<res_row){
        if(ary[i][2]!=0){
            attach(res[s],ary[i][0],ary[i][1],ary[i][2]);
            s++;
        }
        i++;
    }
    Free_Matrix(ary, temp);
    Sort_Matrix(res, res_row);

}
```

43. 곱셈계산을 수행하는 함수이다.
매개변수로는 A희소행렬, B희소행렬, 동적할당 완료한 결과행렬, a희소 행, b희소 행, 결과 희소 행이 전달된다.

44. 위 곱셈결과 희소행렬의 행을 뽑을 때와 똑같이 진행한다.
a열과 b행이 같을 때, 최대로 나올 수 있는 경우 temp를 구하여 임시 변수 ary를 동적 할당한다. 행렬의 행을 뽑을 때와 마찬가지로 임시변수 ary에 곱셈 행렬을 했을 시 저장되는 값을 저장한다.
그러면 ary에는 곱셈행렬을 계산했을 시 결과가 0이 되는 것과 0이 되지 않는 값 모두가 포함되어 있을 것이다.

45. i와 s를 0으로 설정하고, s가 결과행렬의 행까지 반복하도록 while문을 작성한다. s는 결과행렬의 위치를 나타내고 i는 임시변수 ary의 위치를 나타내는 변수가 될 것이다.

46. ary의 2열에 있는 데이터가 0이 아니라면 곱셈 계산 시 0이 아니라는 소리와 같다. 따라서 i를 증가시키며 데이터가 0이 아니라면 결과행렬에 저장하도록 한다. 결과행렬에 저장했다면, s++을 하여 결과 행은 다음 위치로 넘어가게 된다.
여기서도 attach함수를 이용하여 결과에 대입해주도록 한다.

47. 임시변수 ary는 사용이 모두 끝났으므로 Free함수를 이용하여 메모리를 해제한다.
이대로 결과행렬 출력을 하게 되면 정렬이 되지 않아 순서가 뒤죽박죽인 경우가 있었다. 따라서 정렬함수를 호출하여 결과행렬을 오름차순 정렬하도록 한다.

```
void attach(int *s, int a, int b, int c){
    s[0]=a;
    s[1]=b;
    s[2]=c;
}

int **memory_allocation(int sparse){
    int i,j;
    int **a;
    if(sparse<=0){
        printf("메모리가 존재하지 않습니다.₩n");
        return 0;
    }

    a = (int **)malloc(sizeof(int*)*sparse);
    for(i=0; i<sparse; i++){
        a[i] = (int*)malloc(sizeof(int)*3);
        for(j=0; j<3; j++){
            a[i][j]=0;
        }
    }
    return a;
}
```

48. attach함수는 매개변수로 포인터변수 하나와 값 3개를 입력받는다.
포인터 변수로 결과행렬의 행 값만을 전달하면, attach함수에서 결과행렬의 행 값과 열값, 데이터 값에 맞는 위치에 나머지 값 3개를 저장하게 된다.

49. 희소행렬 메모리 동적할당을 하는 함수이다, 매개변수로는 희소행렬의 행을 입력받는다.
만약 희소행렬의 행이 0보다 작다면 동적할당을 할 수 없으므로 0을 리턴한다. 만약 1이상이라면, 임시 이중포인터 a를 희소행렬의 행과 3으로 동적할당하여 2차원 배열을 생성한다.
동적할당하며 행렬의 값도 0으로 설정한다.

50. 동적할당을 완료하면, 할당한 이중포인터를 리턴한다.

```
void Sort_Matrix(int **matrix, int row){
    int i,r,tmp,k;
    for(i=0; i<row; i++){
        for(r=0; r<row; r++){
            if(matrix[i][0]<matrix[r][0]){
                for(k=0; k<3; k++){
                    tmp=matrix[i][k];
                    matrix[i][k] = matrix[r][k];
                    matrix[r][k]=tmp;
                }
            }
            if((matrix[i][0]==matrix[r][0]) && (matrix[i][1]<matrix[r][1]) ){
                for(k=0; k<3; k++){
                    tmp=matrix[i][k];
                    matrix[i][k] = matrix[r][k];
                    matrix[r][k]=tmp;
                }
            }
        }
    }
}
void Trans_Matrix(int **matrix, int sparse_row, int* row, int *col){
    int i, tmp;
    tmp = *row;
    *row = *col;
    *col = tmp;

    for(i=0; i<sparse_row; i++){
        tmp=matrix[i][1];
        matrix[i][1] = matrix[i][0];
        matrix[i][0]=tmp;
    }
    Sort_Matrix(matrix, sparse_row);
}
```

51. 행렬을 오름차순 정렬하는 함수이다.
매개변수로는 희소행렬과 희소행렬 행 값을 전달받는다.
먼저 i와 r을 행만큼 이중 for문을 돌며 만약 현재 행렬의 행 값보다 행렬의 행 값이 더 작다면 for문을 이용해 행, 열, 데이터 값을 모두 맞바꾼다.

52. 결과를 테스트하던 중, 오름차순 정렬은 완료 되었지만 0 1이 0 0 보다 먼저 나오는 경우가 발생했다. 그래서 행에 관하여 정렬을 모두 완료 하였다면, 행이 서로 같을 때 열에 대하여 오름차순 정렬을 한다. 행이 같고 열이 더 작은 경우를 비교하고 조건 문으로 작성하여 위와 같이 위치를 맞바꾼다.

53. 전치행렬을 수행하는 함수이다.
매개변수로는 희소행렬과 희소행렬 행 값, full-matrix의 행과 열값을 포인터로 전달받아 주소를 저장한다.
3*2의 행렬을 전치행렬을 수행하면 2*3이 되므로 단순히 값에 의한 호출로 데이터를 바꾸는 게 아닌 주소를 받아와 주소 안의 값을 바꿔야 한다.

54. 먼저 임시변수 tmp를 이용해 행의 주소에 있는 값과 열의 주소에 있는 값을 서로 뒤바꾼다.
그리고 희소행렬의 행만큼 for문을 돌려 희소행렬의 행 값과 열값을 서로 바꾼다. 데이터는 뒤바뀐 행, 열값에 그대로 있기 때문에 데이터 값은 바꾸지 않는다.

55. 전치가 완료 되었다면, 위의 정렬함수를 출력하여 행렬이 오름차순 정렬이 될 수 있도록 한다.

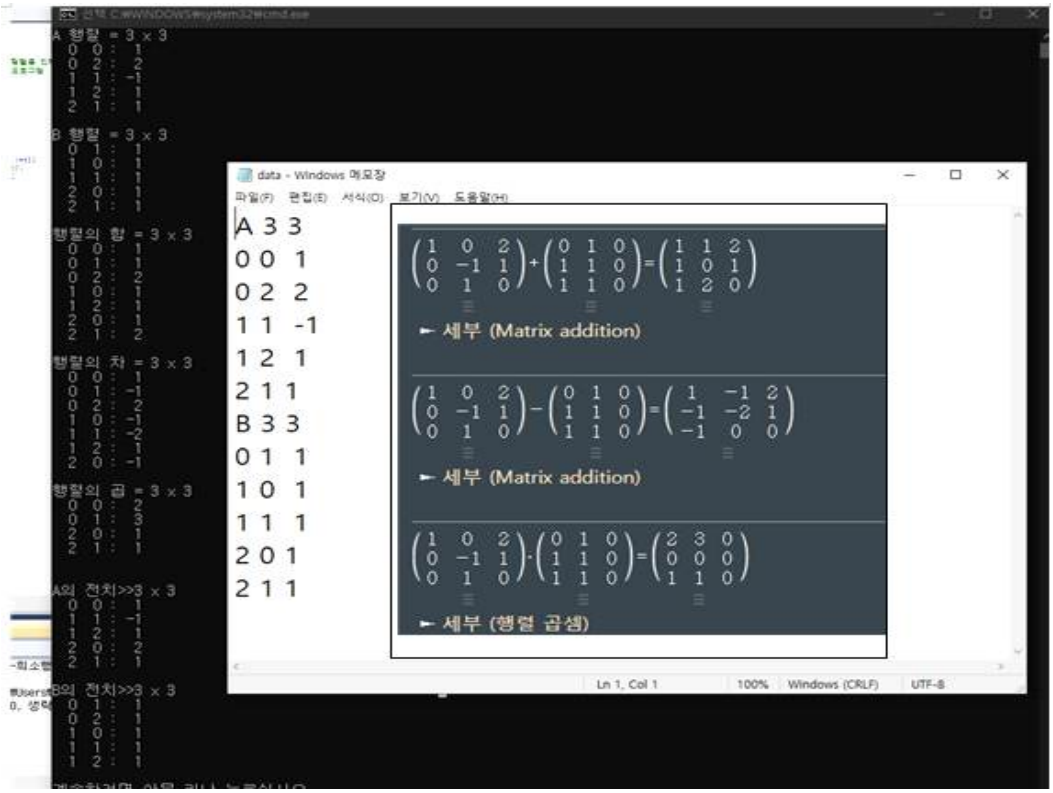
```
void Matrix_Print(int **matrix, int row, int col){
    int i,j;
    for(i=0; i<row; i++){
        for(j=0; j<col; j++){
            if(j==2) printf(" :");
            printf("%3d", matrix[i][j]);
        }
        printf("₩n");
    }
    printf("₩n");
}
void Free_Matrix(int **matrix, int row){
    int i;
    for(i=0; i<row; i++){
        free(matrix[i]);
    }
    free(matrix);
}
```

56. 행렬을 출력하는 함수이다.
매개변수로는 희소행렬 이중포인터와, 희소행렬의 행, 열값을 가져온다.
입력받은 행과 열만큼 이중for문을 돌며 데이터를 출력한다. 여기서 열값은 3이기 때문에 만약 j가 2라면 :를 출력하여 위치와 데이터를 구분해줄 것이다.

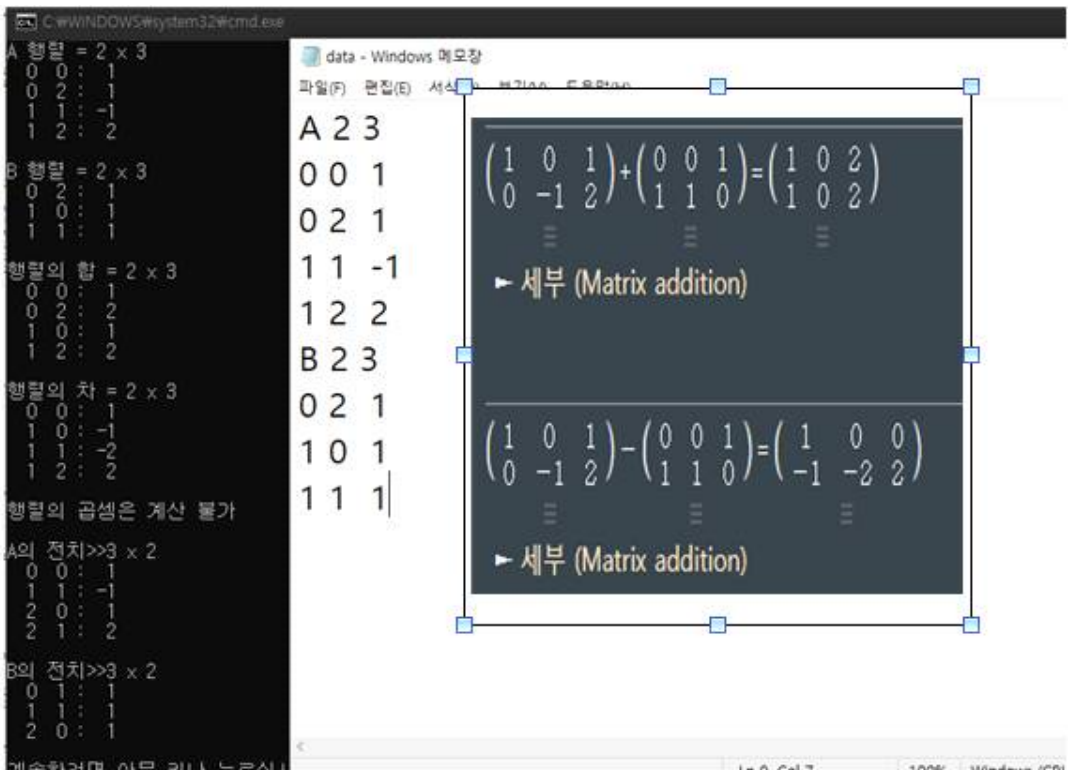
57. 메모리 동적할당을 해제하는 함수이다.
매개변수로는 희소행렬과 희소행렬의 행 값을 가져온다.
처음 동적할당 할 때에 행으로 동적할당하고 행만큼 for문을 돌며 3으로 동적 할당한 것의 반대로 for문을 돌며 free를 해주고 for문을 전부 돌았다면 마지막으로 행 값을 free를 해준다.

1.4 실행 창 - 오른쪽행렬은 실제 계산 값

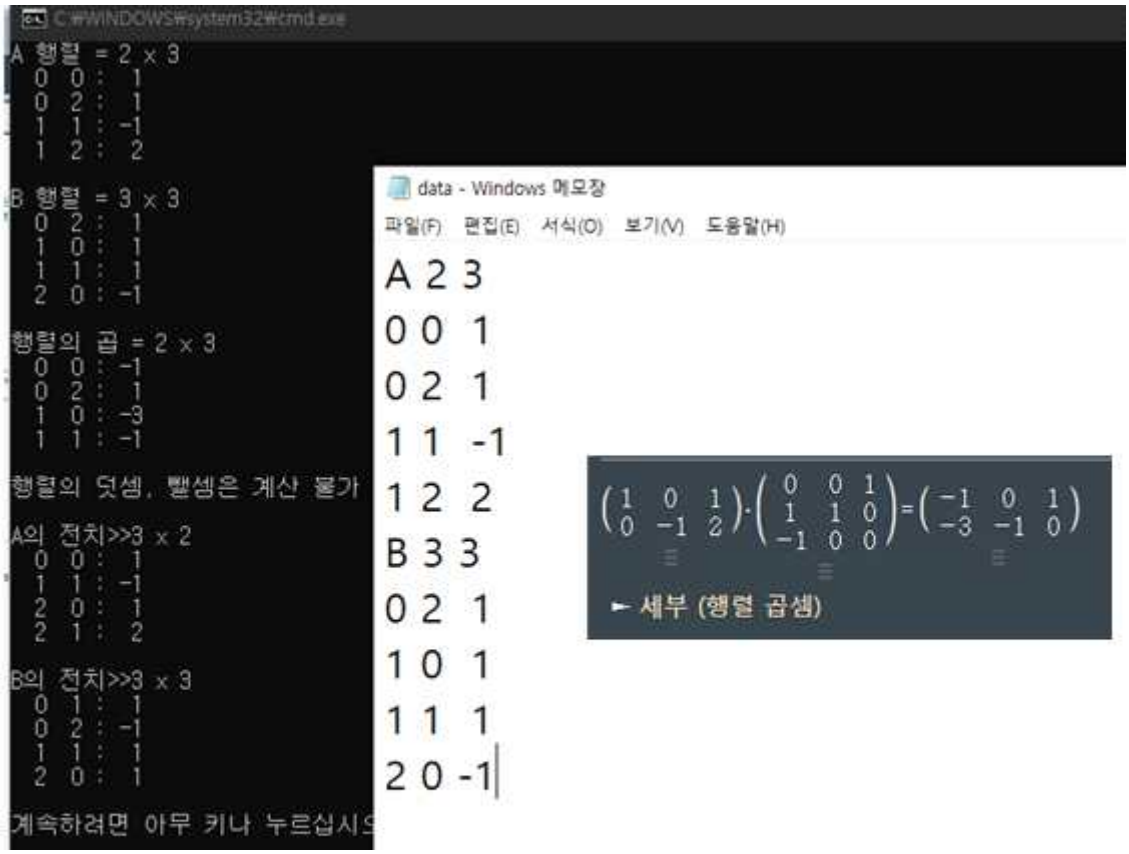
- (m*m)의 정방행렬일 때(덧셈, 뺄셈, 곱셈, 전치계산)



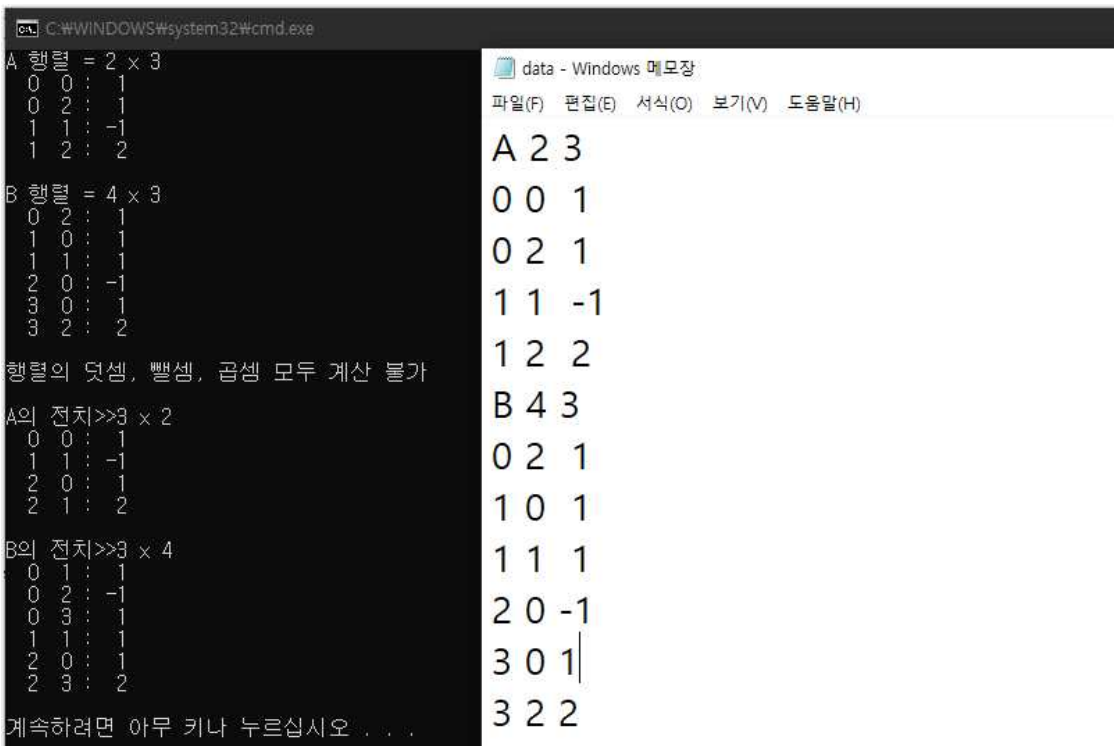
- (m*n)+(m*n)의 행렬일 때(덧셈, 뺄셈만 계산)



- (m*n)*(n*p)의 행렬일 때(곱셈만 계산)



- 계산 불가 행렬일 때 (전치행렬만 출력)



1.5 느낀 점

바로 전 과제인 행렬의 연산을 할 땐 비교적 쉽게 했지만, 이번 Mid Term Project는 보자마자 쉽지 않겠다는 생각이 먼저 들었고, 실제로 해보니 제가 생각했던 것보다 더 어려웠습니다. 평소엔 FULL_Matrix를 이용하여 코딩해왔지만, 이번엔 사용하지 않고 희소행렬 데이터로만 프로그래밍 하려다보니, 익숙하지가 않아 프로그래밍 하는 데에 많은 어려움이 있었던 것 같습니다.

가장 어려웠다고 생각되는 부분은 곱셈이었는데, 덧셈과 뺄셈의 경우 어느 정도 계산을 할 수 있었지만, 희소행렬 곱을 할 때 규칙이 있다는 것을 찾기 위해 몇 시간동안 계산을 하기도 했습니다. 특히 중간에 더했을 때 0이 나오는 경우를 처리하는 것이 많이 힘들었습니다.

이번 텀 프로젝트를 통해 제가 평소에 부족했던 부분을 채울 수 있는 계기가 되었던 것 같습니다. 메모리 동적할당이나 정렬에서 저 스스로 부족하다고 생각했던 것들을 보완할 수 있었습니다. 특히 정렬에선 처음엔 맨 행자리에 대해서만 정렬하여 열자리의 순서가 바뀌어 있는 경우가 있었는데 알고리즘을 계속 생각하다 행과 열자리의 값에 맞게 정렬을 할 수 있게 되었습니다.
함수와 포인터에 관해서도 많이 이해를 할 수 있었습니다. 이중포인터를 반환하는 함수를 만들어 동적할당 된 포인터를 반환하거나 동적할당 하지 않은 결과행렬을 삼중포인터로 함수로 가져가 함수에서 동적할당, 해제 등을 하며 제가 어렵다고 생각했던 포인터에 대해서도 많은 것을 배울 수 있는 과제였다고 생각합니다.

아쉽다고 생각했던 부분도 있었습니다. 오랫동안 생각을 하다 곱셈 알고리즘을 만들었는데, 삼중for문이 나와서 for문을 더 줄여볼 수는 없을까 하고 생각했습니다. 좀만 더 고민해봤다면 결과를 찾았을 수도 있었겠지만 아쉽게도 레포트 제출 날이 다가와서 여기서 마무리하게 되었습니다. 그렇지만 저는 레포트 제출 이후에도 이 부분뿐만 아니라 다른 함수나 알고리즘에 대해서도 더 빠르고 간편하게 하는 방법에 대해 고민해 볼 것입니다.

이상 “희소행렬의 연산과 전치 프로그램” 레포트를 마치겠습니다.
- 감사합니다. -