C-프로그래밍

Project#3

C211123 이준선

2022.05.31

Program 1) Random Square Matrix Generation & Calculation

아래의 예시와 같이 동작하는 프로그램을 구현하려고 한다.
먼저 아래 메뉴를 표시하고 각 메뉴는 다음과 같이 동작한다.

-- Menu --
1. Random Square Matrix Generation
2. Transpose (T)
3. Rotation (90, 180, 270, 360 degree)
4. Inverse (^-1)
5. Calculation (+, -, *)
6. Exit

"1. Random Square Matrix Generation" 선택 시, 정방행렬(square matrix)의 행(또는 열)의 길이 (2~3)를 입력 받아, 0 ~ 99 범위내에서 난수 (Random number)를 생성하여 두 정방행렬 X, Y 를 출력한다. 난수 발생은 교재 407-410 참조.
"2. Transpose (T)" 선택 시, X, Y 의 전치행렬(Transpose)를 출력한다.
"3. Rotation (0, 90, 180, 270, 360 degree)" 선택 시 (0, 90, 180, 270, 360) 중 원하는 각도를 입력 받아 X, Y 를 오른쪽(시계방향)으로 입력 받은 각도만큼 회전한 행렬을 출력한다. (교재 406 페이지 참고)
"4. Inverse (^-1)" 선택 시 각 행렬 X, Y 의 역행렬(inverse)를 출력한다. 이때, 행렬식(Determinant)을 구할 필요 있음. 역행렬이 존재하지 않을 때는 그 메시지를 출력한다. (예, "Matrix X is not invertible")
"5. Calculation (+, -, *)" 선택 시 ('+', '-', '*') 중 원하는 연산을 입력 받아 그 결과를 출력한다.
"6. Exit" 선택 시 프로그램 종료한다.
각 메뉴는 반드시 함수 형태로 구현하여야 한다. Call by reference 형태로 배열을 전달하고 접근한다. 1 번 메뉴 선택 전에 다른 메뉴를 선택했을 시 행렬이 존재하지 않는다는 Error 메시지 출력하게끔 한다. 함수의 연산은 정확한지 계산기 등으로 확인한다.

<소스 코드>

**보고서 맨 뒤에 전체 소스 코드 첨부*

**일부 중요한 함수 및 부분만 부분 인용함*

```
// 행렬 X, Y 를 표시하기 위한 매크로
#define X 0
#define Y 1

// 행렬의 개수, 기본적으로 X, Y 임으로 두 개이다.
#define MAT_SIZE 2

// selectMenu 에서 사용되는, 행렬이 존재하는 지 판별하고 존재하지 않을 경우 에러
메시지를 출력하고 리턴하는 매크로
#define MATRIX_EXIST(flag) if(flag == 0){ puts("Error: Random Square Matrix Not
Ready"); return; }

// malloc 함수에서 NULL 이 리턴되었을 때를 처리하기 위한 매크로
// 만약 동적 할당에 실패하여 malloc 에서 NULL 이 발생하였을 때, 에러 메시지를
출력한다.
#define _MALLOC(ptr, size) if((ptr = malloc(size)) == NULL) { puts("Error:
Matrix Generation Failure");}

// 계산 옵션 목록 상수
enum { PLUS, MINUS, MULTIPLICATIOIN };

// 메뉴 목록 상수
enum { RandomSquareMatrixGeneration = 1, Transpose, Rotation, Inverse,
Calculation, Exit };
```

 # 메인 함수

```
int main(void)
{

	int** MATRIX[MAT_SIZE] = {NULL, NULL}; ///>MATRIX[X], [Y]를 담을
int**의 배열
	int SQUARE = 0; ///>정방 행렬의 열(행) 수

	int menuSelection = 0; ///>메뉴 선택지

	// menuSelection 이 Exit 이 아닐 동안 계속 반복해서 메뉴를 입력받는다.
	while (menuSelection != Exit) {
		printMenu();
		menuSelection = getMenuSelection();
		selectMenu(menuSelection, MATRIX, &SQUARE);
	}

	//프로그램이 종료될 때, 행렬을 모두 해제하고 종료한다.
	for (int i = 0; i < MAT_SIZE; i++) {
		if(MATRIX[i] != NULL) {
```

```
                        deleteMatrix(MATRIX[i], SQUARE);
                }
        }

        return 0;
}

/********
* function : selectMenu
* purpose : menuSelection 을 바탕으로 지정된 메뉴 함수를 호출한다.
* parameters
*       int menuSelection : 메뉴 선택지
*       int** matrix[] : 행렬 X, Y              ///> 이후 생략
*       int* square : 행렬의 열(또는 행)
* descriptions
*       메뉴 목록은 printMenu() 또는 enum 참고
*       만약 menuSelection 이 올바른 메뉴 선택지가 아닐 경우, 안내문을 출력하고
반환한다.
*       행렬이 존재하지 않는 상태에서 1 번 이외 메뉴를 선택할 경우 안내문을 출력하고
반환한다. 이때 MATRIX_EXIST 매크로 사용
* note
*       square 을 포인터로 받는다.
*       RandomSquareMatrixGeneration 메뉴 실행 시 메인함수에서 선언된 square 을
직접 수정한다.
*       EXIT 이 선택되어도 행렬 MATRIX 를 따로 해제하지는 않는다. 메인함수에서 따로
해제해줘야 한다.
*********/
void selectMenu(int menuSeleciton, int** MATRIX[], int* square)
{
        static int isMatrixGenerated = 0; ///> 행렬이 이미 있는 지 확인하는
bool 값, 0 이면 없고, 1 이면 있다는 뜻이다.
        int SQUARE = *square; ///>포인터로 받은 square 을 안전하게 사용하기 위해
SQUARE 에 복사한다.

        switch (menuSeleciton) {
        case RandomSquareMatrixGeneration:
                //이미 생성된 행렬이 존재할 시, 기존 행렬을 삭제하고 다시 생성한다.
                if (isMatrixGenerated != 0) {
                        for (int i = 0; i < MAT_SIZE; i++) {
                                deleteMatrix(MATRIX[i], SQUARE);
                        }
                }
                *square = randomSquareMatrixGeneration(MATRIX);
                isMatrixGenerated = 1;
                break;
        case Transpose:
                MATRIX_EXIST(isMatrixGenerated)
                transpose(MATRIX, SQUARE);
                break;
        case Rotation:
                MATRIX_EXIST(isMatrixGenerated)
                rotation(MATRIX, SQUARE);
                break;
        case Inverse:
```

```
                MATRIX_EXIST(isMatrixGenerated)
                inverse(MATRIX, SQUARE);
                break;
        case Calculation:
                MATRIX_EXIST(isMatrixGenerated)
                calculation(MATRIX, SQUARE);
                break;
        case Exit:
                puts("Exit, thank you");
                return;
        default:
                //메뉴 선택 목록에 없을 시, 안내문 출력
                puts("Thou selected wrong menu, try again");
        }
}
```

메뉴 1 randomSquareMatrixGeneration

```
/********
* function : randomSquareMatrixGeneration
* purpose : menu 1_RandomSquareMatrixGeneration 기능 수행
* return : SQUARE, 행렬의 열(=행)
* description
*       안내문을 출력하고 SQUARE 값을 입력받는다.
*       generateBlandkedMatrix 함수로 행렬을 생성한다.
*       fillMatrixWithRandom 함수로 행렬을 무작위 숫자로 채운다.
*       행렬 X, Y 를 출력한다.
*       입력받는 SQUARE 을 반환한다.
*********/
int randomSquareMatrixGeneration(int** MATRIX[])
{
        int SQUARE = 0;
        printf("Input the number of rows (2 or 3): ");
        scanf_s("%d", &SQUARE);

        for (int i = 0; i < MAT_SIZE; i++) {
                MATRIX[i] = generateBlankedMatrix(SQUARE);
        }

        srand((unsigned int)time(NULL));
        for (int i = 0; i < MAT_SIZE; i++) {
                fillMatrixWithRandom(MATRIX[i], SQUARE);
        }

        printf("X = ");
        printMatrix(MATRIX[X], SQUARE);

        printf("Y = ");
        printMatrix(MATRIX[Y], SQUARE);

        return SQUARE;
}
```

```
/********
* function : generateBlankedMatrix
* purpose : SQUARE * SQUARE 의 정방 행렬을 생성한다.
* return : 생성된 int** 형 matrix
* description
*       동적할당으로 2 차원 배열 생성
*       동적할당 실패를 대비해 _MALLOC 매크로 사용
* note
*       0 으로 초기화 안 한다.
*********/
int** generateBlankedMatrix(int SQUARE)
{
        int** matrix = NULL;
        _MALLOC(matrix, sizeof(int*) * SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                _MALLOC(matrix[i], sizeof(int) * SQUARE);
        }
        return matrix;
}


/*******
* function : fillMatrixWithRandom
* purpose : 행렬의 모든 요소를 난수로 채운다.
* parameters
*       int** matrix : 0 으로 초기화할 대상 행렬
* description
*       0 부터 99 까지의 임의의 정수로 행렬을 채운다.
* note
*       srand 로 시드값 초기화하지 않음. 미리 srand 로 시드값을 설정해야 함.
********/
void fillMatrixWithRandom(int** matrix, int SQUARE)
{
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        matrix[i][j] = rand() % 100;
                }
        }
}
```

메뉴 2 Transpose

```
/********
* function : transpose
* purpose : menu 2_Transpose 기능 수행
* description
*       X 와 Y 의 전치행렬을 출력한다.
* note
*       결과 행렬을 반환하지는 않는다.
*********/
void transpose(int** MATRIX[], int SQUARE)
{
        printf("Transpose X and Y\n");

        printf("X^T = ");
```

```
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        printf("%2d ", MATRIX[X][j][i]);
                }
                printf("\n");
                printf("      ");
        }
        printf("\n");

        printf("Y^T = ");
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        printf("%2d ", MATRIX[Y][j][i]);
                }
                printf("\n");
                printf("      ");
        }
        printf("\n");
}
```

 # 메뉴 3 Rotation

```
/********
* function : rotation
* purpose : menu 3_Rotation 기능 수행
* description
*       getDegree() 함수로 각도를 입력받고, 이를 통해 회전수를 계산한다.
*       회전수만큼 rotation 한 결과를 출력한다. (rotate90 함수 이용)
*********/
void rotation(const int** MATRIX[], int SQUARE)
{
        int degree = getDegree();
        int rotationCycle = degree / 90; ///> 회전 바퀴 수

        // 회전시킬 행렬 2 개를 생성한다.
        int** rotatedMatrix[MAT_SIZE] = { generateBlankedMatrix(SQUARE),
generateBlankedMatrix(SQUARE) };

        // 회전시킬 행렬에 원본 행렬을 복사한다.
        for (int i = 0; i < MAT_SIZE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        for (int k = 0; k < SQUARE; k++) {
                                rotatedMatrix[i][j][k] = MATRIX[i][j][k];
                        }
                }
        }

        // rotate Matrix for rotation Cycle
        // ex : if degree == 180, cycle is 2 and rotate for 2 times
        for (int k = 0; k < MAT_SIZE; k++) { // k 는 X 와 Y
                for (int i = 0; i < rotationCycle; i++) {
                        rotate90(rotatedMatrix[k], SQUARE);
                }
        }
```

```
        printf("X = ");
        printMatrix(rotatedMatrix[X], SQUARE);

        printf("Y = ");
        printMatrix(rotatedMatrix[Y], SQUARE);

        for (int i = 0; i < MAT_SIZE; i++) {
                deleteMatrix(rotatedMatrix[i], SQUARE);
        }
}

/*******
* function : rotate90
* purpose : 입력받은 행렬을 90 도만큼 시계방향 회전시킨다.
* parameters
*       int** originalMatrix : 90 도 회전시킬 행렬
********/
void rotate90(int** matrix, int SQUARE)
{
        int** rotatedMatrix = generateBlankedMatrix(SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        rotatedMatrix[j][SQUARE - i - 1] = matrix[i][j];
                }
        }

        // 원본 행렬 직접 수정
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        matrix[i][j] = rotatedMatrix[i][j];
                }
        }
        deleteMatrix(rotatedMatrix, SQUARE);
}
```

메뉴 4 Inverse

```
/********
* function : inverse
* purpose : menu 4_Inverse 기능 실행
* description
*       getInverseMatrix() 함수로 역행렬을 계산하고, 만약 역행렬이 존재하면 이를
출력한다.
* description
*       getInverseMatrix()를 통해 X, Y 가 역행렬이 존재하는 지를 각각
판단한다(NULL 일 경우 존재하지 않는 것임).
*       이를 printDoubleMatrix() 로 출력한다.
*********/
void inverse(int** MATRIX[], int SQUARE)
{
        double** inverseMatrix[MAT_SIZE];       ///> 역행렬 X, Y 를 저장할
double** [] 형 행렬
        inverseMatrix[X] = getInverseMatrix(MATRIX[X], SQUARE);
        inverseMatrix[Y] = getInverseMatrix(MATRIX[Y], SQUARE);
```

```
	// 행렬 X, Y 가 모두 역행렬이 존재할 때
	if (inverseMatrix[X] != NULL && inverseMatrix[Y] != NULL) {
		printf("X and Y are Invertible, \n");
		printf("Inverse X and Y\n");

		printf("X^(-1) = ");
		printDoubleMatrix(inverseMatrix[X], SQUARE);

		printf("Y^(-1) = ");
		printDoubleMatrix(inverseMatrix[Y], SQUARE);
	}
	// 행렬 X 가 역행렬이 존재하지 않고, 행렬 Y 만 역행렬이 존재할 때
	else if (inverseMatrix[X] == NULL && inverseMatrix[Y] != NULL) {
		printf("X is not invertible but Y is invertible, \n");
		printf("Inverse Y\n");

		printf("Y^(-1) = ");
		printDoubleMatrix(inverseMatrix[Y], SQUARE);
	}
	// 행렬 X 만 역행렬이 존재하고, 행렬 Y 는 역행렬이 존재하지 않을 때
	else if (inverseMatrix[X] != NULL && inverseMatrix[Y] == NULL) {
		printf("X is invertible but Y is not invertible, \n");
		printf("Inverse X\n");

		printf("X^(-1) = ");
		printDoubleMatrix(inverseMatrix[X], SQUARE);
	}
	// 기타 상황, 행렬 X, Y 가 모두 역행렬이 존재하지 않을 때
	else {
		printf("X and Y are not invertible\n");
		deleteMatrix(inverseMatrix[X], SQUARE);
		deleteMatrix(inverseMatrix[Y], SQUARE);
	}

	for (int i = 0; i < MAT_SIZE; i++) {
		deleteMatrix(inverseMatrix[i], SQUARE);
	}
}

/********
* function : getInverseMatrix
* purpose : 역행렬을 반환한다.
* parameters
*	int** matrix : 원본 행렬
* return
*	double** resultMatrix, 역행렬
*	또는 역행렬이 존재하지 않으면 NULL 반환
* description
*	행렬식 determinant 를 구해서 역행렬을 계산하는 것은, 행렬의 크기에 따라 너무
복잡해지므로,
*	보다 간편하고 컴퓨터 친화적인 Gauss-Jordan 소거법을 이용하여 역행렬을
계산한다.
*	단, 행렬식이 0 인지를 확인하여 해당 행렬의 역행렬을 가지는 지를 검사하기는
한다.
```

```
*       구한 역행렬이 실제 matrix 의 역행렬인지 알기 위해 resultMatrix * matrix == IdentityMatrix(단위 행렬)인지를 확인한다.
*       만약 resultMatrix * matrix != IdentityMatrix 이면 matrix 는 올바른 역행렬이 존재하지 않는 것이다. NULL 을 리턴한다.
* note
*       원본 matrix 는 수정되지 않는다.
*********/
double** getInverseMatrix(const int** matrix, int SQUARE) {
        if (getDeterminant(matrix, SQUARE) == 0) return NULL; // Det (Matrix) == 0 이면 역행렬이 존재할 수 없음

        const double ERROR = 1.0e-10; ///> 부동소수점 오류 기준값

        // *note : generateDoubleMatrix()는 0 으로 초기화를 동시에 진행함.
        double** resultMatrix = generateDoubleMatrix(SQUARE); ///> 역행렬 결과를 저장할 행렬
        double** MatrixCopy = generateDoubleMatrix(SQUARE); ///> 원본 행렬의 복사본, 가우스 조던 소거법을 이용하기에 이 행렬을 수정한다.

        // MatrixCopy 에 원본 matrix 복사
        // int -> double 형으로 형변환하여 복사한다.
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        MatrixCopy[i][j] = (double)matrix[i][j];
                }
        }

        // 계산 결과가 저장되는 resultMatrix 행렬을 단위행렬로 초기화
        for (int i = 0; i < SQUARE; i++)
                for (int j = 0; j < SQUARE; j++)
                        resultMatrix[i][j] = (i == j) ? 1 : 0;

        // 대각요소를 0 이 아닌 수로 만든다.
        // 가우스 조던 방법의 해가 부정(해가 무수히 많음)일 경우를 대비한 과정이다.
        for (int i = 0; i < SQUARE; i++) {
                if (-ERROR < MatrixCopy[i][i] && MatrixCopy[i][i] < ERROR) { // MatrixCopy[i][i] == 0
                        for (int k = 0; k < SQUARE; k++) {
                                if (-ERROR < MatrixCopy[k][i] && MatrixCopy[k][i] < ERROR) // MatrixCopy[k][i] == 0
                                        continue;
                                for (int j = 0; j < SQUARE; j++) {
                                        MatrixCopy[i][j] += MatrixCopy[k][j];
                                        resultMatrix[i][j] += resultMatrix[k][j];
                                }
                                break;
                        }
                        if (-ERROR < MatrixCopy[i][i] && MatrixCopy[i][i] < ERROR) {
                                //수정 사항을 모두 완료했음에도 대각선 요소가 0 -> 가우스-조던 방법 불가능
                                deleteMatrix(resultMatrix, SQUARE);
                                deleteMatrix(MatrixCopy, SQUARE);
```

```
                        return NULL;
                    };
                }
        }


        /* Gauss-Jordan elimination */
        for (int i = 0; i < SQUARE; i++) {
                // 대각 요소를 1 로 만듦
                double constant = MatrixCopy[i][i]; ///> 대각선 요소의 값 저장
                for (int j = 0; j < SQUARE; j++) {
                        MatrixCopy[i][j] /= constant;   // MatrixCopy[i][i] 를
1 로 만드는 작업
                        resultMatrix[i][j] /= constant;   // i 행 전체를
MatrixCopy[i][i] 로 나눔
                }

                // i 행을 제외한 k 행에서 MatrixCopy[k][i] 를 0 으로 만드는 단계
                for (int k = 0; k < SQUARE; k++) {
                        if (k == i) continue;      // 자기 자신의 행은 건너뜀
                        if (MatrixCopy[k][i] == 0) continue;   // 이미 0 이 되어
있으면 건너뜀

                        // MatrixCopy[k][i] 행을 0 으로 만듦
                        constant = MatrixCopy[k][i];
                        for (int j = 0; j < SQUARE; j++) {
                                MatrixCopy[k][j] = MatrixCopy[k][j] -
MatrixCopy[i][j] * constant;
                                resultMatrix[k][j] = resultMatrix[k][j] -
resultMatrix[i][j] * constant;
                        }
                }
        }

        deleteMatrix(MatrixCopy, SQUARE);

        return resultMatrix;
}
```

메뉴 5 Calculation

```
/********
* function : calculation
* purpose : menu 5_Calculation 의 기능 실행
* description
*       getCalculationSelection() 을 이용해 계산 옵션(+, -, * 중 하나)를 선택받고,
그에 맞는 연산 결과를 출력한다.
*       연산 결과는 calc_함수들이 담당함(함수 자체에서 print 하지는 않음)
*********/
void calculation(int** MATRIX[], int SQUARE)
{
        int calculationSelection = getCalculationSelection();
        switch (calculationSelection) {
```

```
        case PLUS:
                calc_plus(MATRIX, SQUARE);
                break;
        case MINUS:
                calc_minus(MATRIX, SQUARE);
                break;
        case MULTIPLICATIOIN:
                calc_multiplication(MATRIX, SQUARE);
                break;
        default:
                puts("ERROR : Wrong Calculation menu selected");
        }
}

/********
* function : calc_multiplication
* purpose : calculation menu 중 *(곱셈) 메뉴 기능 구현
* description
*       행렬 X, Y 를 곱한 결과를 출력한다.
* note
*       printMatrix 함수를 이용하기에, 출력 정렬이 깨지는 경우가 발생한다.
*       결과 행렬을 반환하지는 않는다.
*********/
void calc_multiplication(int** MATRIX[], int SQUARE)
{
        int** resultMatrix = generateBlankedMatrix(SQUARE);
        fillMatrixWithZero(resultMatrix, SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        for (int k = 0; k < SQUARE; k++) {
                                resultMatrix[i][j] += (MATRIX[X][i][k] *
MATRIX[Y][k][j]);
                                // i, j 칸에 대해 i 행과 j 열을 k 로 순환하며 곱한
결과를 모두 합한다.
                        }
                }
        }

        printf("X * Y = ");
        printMatrix(resultMatrix, SQUARE);
        deleteMatrix(resultMatrix, SQUARE);
}
```

**** Calc_Minus, Calc_Plus 함수는 생략***

<출력 결과>

```
Microsoft Visual Studio 디버그 콘솔
-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 3
Error: Random Square Matrix Not Ready
-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 1
Input the number of rows (2 or 3): 3
X = 86 22 69
    72 73 25
    48 82 12

Y = 93 99 51
    31  6 64
    87 74 18

-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 2
Transpose X and Y
X^T = 86 72 48
      22 73 82
      69 25 12

Y^T = 93 31 87
      99  6 74
      51 64 18
```

```
-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 3
Input the degree to rotate: 90
X = 48 72 86
    82 73 22
    12 25 69

Y = 87 31 93
    74  6 99
    18 64 51

-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 4
X and Y are Invertible,
Inverse X and Y
X^(-1) = -0.016299  0.074888 -0.062295
          0.004665 -0.031654  0.039124
          0.033320 -0.083245  0.065169

Y^(-1) = -0.029675  0.012773  0.038664
          0.032124 -0.017716 -0.028027
          0.011362  0.011099 -0.016100
```

```
-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 5
Input the calculation (+, -, or *): +
X + Y = 179 121 120
    103 79 89
    135 156 30

-- Menu --
1. Random Square Matrix Generation
2. Transpose(T)
3. Rotation(90, 180, 270, 360 degree)
4. Inverse(^ -1)
5. Calculation(+, -, *)
6. Exit
Choose the item you want: 6
Exit, thank you

C:₩Users₩love5₩source₩repos₩MatrixGenerationAndCalc
ulcation₩x64₩Debug₩MatrixGenerationAndCalculcation.
exe(프로세스 10112개)이(가) 종료되었습니다(코드: 0
개).
이 창을 닫으려면 아무 키나 누르세요...
```

<코드 설명>

함수의 자세한 작동 방식에 대한 것은 코드 내 주석으로 대체함.

1. RandomSquareMatrixGeneration

   SQUARE 값을 입력받고, SQUARE * SQUARE 크기의 정방 행렬을 생성한다. 정방 행렬의 요소는 모두 0~99 사이의 임의의 정수로 채운다.

   `srand((unsigned int)time(NULL));` 을 통해 랜덤성을 구현한다.

   generateBlankedMatrix() 함수에서, SQUARE의 값만큼 동적 할당한다.

   `#define _MALLOC(ptr, size) if((ptr = malloc(size)) == NULL)`
   `{ puts("Error: Matrix Generation Failure");}` 을 통해, 만약 동적할당이 실패할 경우 에러메시지를 출력한다. 다만 동적할당이 실패할 정도로 메모리가 부족한 경우라면, 입력된 SQUARE의 값이 비정상적으로 크거나 OS 자체가 비정상적인 상황에 처해있는 경우라 예측되기에 그에 대한 예외 처리는 하지 않고 오류를 일으키게끔 하였다.

2. Transpose

   transpose() 함수의 경우, 새로 matrix를 만들고, printMatrix() 함수를 이용해 출력하는 것보다 정렬에 용이하고 더 직관적이기에, transpose된 matrix를 새로 반환하는 함수를 만들기보다, 직접 출력하도록 하였다.

   ```
   for (int i = 0; i < SQUARE; i++) {
            for (int j = 0; j < SQUARE; j++) {
                    printf("%2d ", MATRIX[X][j][i]);
            }
            printf("\n");
            printf("      ");
    }
   ```

   와 같이 i, j 값의 위치를 바꾸어 출력함으로써 전치 행렬을 출력한다.

3. Rotation

   rotate90() 회전수만큼 반복하는 방식으로 구현하였다.

   `int rotationCycle = degree / 90;`

   을 통해 회전 바퀴 수를 구한다. 예를 들어 degree == 90이면 rotationCycle = 1, degree == 180 이면 rotationCycle = 2인 방식이다.

   rotationCycle만큼 rotate90()함수를 반복한다.

   rotate90() 함수는 다음과 같은 방식으로 작동한다.

   (1, 1), (1, 2), (1, 3)
   (2, 1), (2, 2), (2, 3)
   (3, 1), (3, 2), (3, 3)

   의 행렬을 90도 회전시켰을 때 다음과 같이 배치된다.

   (3, 1), (2, 1), (1, 1)
   (3, 2), (2, 2), (1, 2)
   (3, 3), (2, 3), (1, 3)

   즉 기존의 **행**이 회전 후 **열**이 되며, 회전 후 **열**은 SQUARE - 1 에서 회전 전 행을 뺀 값과 같음을 알 수 있다.

이를 다음과 같이 구현한다.

```
for (int i = 0; i < SQUARE; i++) {
        for (int j = 0; j < SQUARE; j++) {
                rotatedMatrix[j][SQUARE - i - 1] = matrix[i][j];
        }
}
```

4. Inverse

getInverseMatrix()함수를 통해 역행렬을 구한다. getInverseMatrix() 함수는 역행렬이 존재할 시 역행렬을, 역행렬을 구할 수 없을 시 NULL 을 반환한다.

getInverseMatrix() 함수 내에서 getDeterminant() 를 통해 행렬식 Det 을 구한다. 만약 행렬식이 0 이면 역행렬이 존재할 수 없으므로 NULL 을 반환한다.

역행렬을 구할 때는 가우스-조던(요르단) 방식을 사용한다. 원래 행렬의 대각요소가 0 이면, 임의의 방식으로 0 이 아닌 수로 만들어야 한다.

-ERROR < MatrixCopy[i][i] && MatrixCopy[i][i] < ERROR

위 식은 MatrixCopy[i][i] == 0 의 의미를 갖는 식이다. MatrixCopy[i][i]는 double 형이기에 부동소수점 오류를 갖는다. 따라서 MatrixCopy[i][i] == 0 으로 했을 때 오류가 발생할 수 있기에 적당히 작은 오차는 무시하는 방식으로 식을 작성한다. 한편 double 형은 소수점 아래 15 번 째까지 오차를 저장할 수 있기 때문에, ERROR = 10^{-10} 의 상수로 선언하였다.

```
const double ERROR = 1.0e-10;
```

5. Calculation

+, -, * 연산 중 하나를 수행한다.

Program 2) Telephone Book Manager

"전화번호관리 프로그램"을 구현하려고 한다. 먼저 "이름(성)"과 "전화번호", "이메일 주소"를 멤버로 가지는 구조체를 정의하고, 구조체 배열 또는 구조체 포인터로 프로그램을 구성한다. 최대 입력 가능 전화번호 수는 10 개로 한정한다.

<소스코드>

**보고서 맨 뒤에 전체 소스 코드 첨부*

**일부 중요한 함수 및 부분만 부분 인용함*

#헤더

```
// 정렬 양식을 출력하는 매크로
#define SHOWLINE printf("Num. %-15s %-15s %s\n", "Name", "Tel", "Email");
puts("-----------------------------------------");
// 문자열 크기
#define STRING_SIZE 256

/*******
* 메뉴 선택지 상수
* 이 enum 을 참조하는 함수:
*       printMenu(), selectMenu
********/
enum { Insert = 1, Delete, Search, PrintAll, Save, Exit };
```

#구조체

```
/*******
* struct : MEMBER
* description
*       이름, 전화번호, 이메일 주소, order 로 이루어진 구조체
*       이중 연결 리스트이나, 원형 연결 리스트는 아니다.
*       즉 head->prev = NULL 이고, last->next = NULL 이 저장되어야 한다.
********/
typedef struct _MEMBER {
        struct _MEMBER* next; ///> 다음 노드 저장
        struct _MEMBER* prev; ///> 이전 노드 저장
        char name[STRING_SIZE]; ///> 이름
        char telephoneNumber[STRING_SIZE]; ///> 전화번호
        char emailAddress[STRING_SIZE]; ///> 이메일 주소
        int order; ///> 멤버의 순서이자 번호. 생성 순서대로 번호를 부여받는다.
} MEMBER;
```

#메인 함수

```
int main(void)
{
        int menuSelection = 0; ///> 메뉴 선택지
        MEMBER* head = malloc(sizeof(MEMBER)); ///> 주소록의 head 노드
```

```
        initializeHead(head); // head 노드를 기본값으로 초기화한다.

        while (menuSelection != Exit) { // menuSelection 이 EXIT 이면 반복문을 탈출한다
                printMenu();
                menuSelection = getMenuSelection();
                selectMenu(head, menuSelection);
        }

        deleteAllData(head);
        return 0;
}
```

#메뉴 선택

```
/********
* function : selectMenu
* purpose : selection 에 대응하는 메뉴 기능을 실행시킨다.
* parameters
*       MEMBER* head : 주소록의 헤더 노드
*       int selection : 메뉴 선택
* description :
*       selection 에 대응하는 기능(함수)를 실행한다.
*       만약 selection 에 대응하는 메뉴가 없을 경우, 에러 메시지를 출력한다.
*       Exit 을 선택한 경우, 굿바이 메시지를 출력하고 반복문을 종료한다.
*       selection 목록은 enum 참고
* note :
*       EXIT 을 선택한 경우에도 함수 자체가 프로그램을 종료하진 않는다. 메인 함수에서
종료해주어야 한다.
*       EXIT 을 선택한 경우에도 주소록의 메모리를 해제하지는 않는다. 메인 함수에서
해제해주어야 한다.
********/
void selectMenu(MEMBER* head, int selection)
{
        switch (selection) {
        case Insert:
                insert(head);
                break;
        case Delete:
                deleteMember(head);
                break;
        case Search:
                search(head);
                break;
        case PrintAll:
                printAllData(head);
                break;
        case Save:
                save(head);
                break;
        case Exit:
                puts("Exit, thanks you!");
                return;
        default:
                puts("Thou selected wrong menu");
        }
```

```
}
```

1. Insert

```
/*******
* function : insert
* purpose : menu-1. insert 의 구현, 멤버의 정보를 입력받고 주소록에 추가한다.
* parameters :
*       MEMBER* head : 주소록의 헤드 노드
* description :
*       안내문을 출력하며, 이름, 전화번호, 이메일 주소 순으로 입력받는다.
*       memberOrder 은 기본적으로 1 에서부터 순차적으로 부여하기 위한 변수이지만,
*       resetOrder() 함수를 사용해서 새로 추가할 때마다 모든 주소록의 order 를
재부여하는 기능이 있다.
*       appendMember() 함수를 사용해서 입력받는 데이터에 부합되는 멤버를 주소록에
추가한다.
*
*       주소록의 크기를 10 으로 제한한다.
*       즉 order 가 10 보다 크면 주소록을 추가하지 않는다.
* note :
*       STRING_SIZE 보다 큰 바이트 크기의 문자열이 입력되었을 시 이에 대한 에러를
처리하지 않는다.
********/
void insert(MEMBER* head)
{
        char name[STRING_SIZE];
        char telephoneNumber[STRING_SIZE];
        char emailAddress[STRING_SIZE];
        static int memberOrder = 1; ///> memberOrder 의 순서를 부여하기 위한 변수,
초기화되지 않고, 함수가 호출될 때마다 증가한다.

        printf("[INSERT]\n");
        printf("Input Name: ");
        gets_s(name, sizeof(name));
        printf("Input Tel. Number: ");
        gets_s(telephoneNumber, sizeof(telephoneNumber));
        printf("Input Email Address: ");
        gets_s(emailAddress, sizeof(emailAddress));

        appendMember(head, name, telephoneNumber, emailAddress, memberOrder);
        resetOrder(head, 1);

        // 주소록의 크기를 제한하는 기능
        // 마지막 멤버의 order 가 10 보다 크면 해당 멤버를 삭제하고, 안내 메시지를
출력한다.
        int maxMemberOrder = 10;
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
        }
        if (cur->order > maxMemberOrder) {
                printf("Telephone book can save maximum %d members\n",
maxMemberOrder);
                deleteTargetMember(cur);
```

```
        }
}

/********
* function : appendMember
* purpose : 주소록에 멤버를 추가한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
*       const char* name, telephoneNumber, emailAddress
*       const int order
* description :
*       주소록 리스트의 맨 끝에 새 멤버 노드를 추가한다.
*       만약 새 노드 생성에 실패할 경우, 에러 메시지를 출력한 후 함수를 종료한다.
*       노드 생성에 성공할 경우, 매개변수로 받은 name, telephoneNumber,
emailAddress, order 값을 갖는 멤버가 추가된다.
*       추가된 멤버의 next 포인터는 NULL 이고, prev 는 바로 직전 노드를 참조한다.
*       바로 직전 노드의 next 포인터를 추가된 노드를 참조한다.
* note :
*       새 노드 생성에 실패해도 예외를 던지지 않는다. 에러 메시지만 출력한다.
*********/
void appendMember(MEMBER* head, const char* name, const char* telephoneNumber,
const char* emailAddress, const int order)
{
        MEMBER* newMember = malloc(sizeof(MEMBER));

        if (newMember == NULL) {
                puts("ERROR : node creation failed");
                return;
        }
        newMember->prev = NULL;
        newMember->next = NULL;

        // 데이터 취합부
        strcpy_s(newMember->name, sizeof(newMember->name), name);
        strcpy_s(newMember->telephoneNumber, sizeof(newMember-
>telephoneNumber), telephoneNumber);
        strcpy_s(newMember->emailAddress, sizeof(newMember->emailAddress),
emailAddress);
        newMember->order = order;

        // head node 의 다음이 없을 경우, 즉 추가된 노드가 첫 노드일 경우
        if (head->next == NULL) {
                newMember->prev = head;
                newMember->next = NULL;

                head->next = newMember;
                head->prev = NULL;
        }
        else {
                MEMBER* cur = head; ///> cur 은 head 부터 끝헤드까지 탐색한다.
                while (cur->next != NULL) {
                        cur = cur->next;
                }
```

```
                // 주소록 끝에 newMember 를 연결한다.
                cur->next = newMember;
                newMember->prev = cur;
        }
}
```

2. Delete

```
/******
* function : deleteMember
* purpose : menu-2. delete 의 기능 구현, 이름을 입력받고 해당 이름에 일치하는 멤버
노드를 주소록에서 삭제한다.
* parameters : 주소록의 헤드 노드
* description
*       안내 메시지를 출력 후 이름을 입력받는다.
*       만약 이름에 일치하는 멤버가 주소록에 없으면 안내문을 출력한다.
*       만약 이름에 일치하는 멤버가 주소록에 있으면 deleteTargetMember 를 통해 해당
멤버 노드를 삭제한다.
*******/
void deleteMember(MEMBER* head)
{
        puts("[DELETE]");

        char targetName[STRING_SIZE];
        printf("Input Name to Delete: ");
        gets_s(targetName, sizeof(targetName));

        MEMBER* targetMember = findMember(head, targetName);
        if (targetMember == NULL) {
                printf("Data for Name “%s” is not exist\n", targetName);
        }
        else {
                deleteTargetMember(targetMember);
                resetOrder(head, 1);
        }
}

/*******
* function : deleteTargetMember
* purpose : target 에 해당하는 멤버 노드를 주소록에서 삭제한다.
* parameters
*       MEMBER* target : 지우려는 대상 멤버
* description
*       target 의 prev(이전) 노드와 next(이후) 노드를 적절히 연결한 후,
free()함수를 통해 target 을 해제한다.
*       만약 target 이 마지막 노드일 시, prev 노드가 끝임을 나타내기 위해 prev->next
= NULL 로 설정한다.
* note
*       매개변수가 head 노드가 아닌 target 노드다.
*       이 함수에는 target 이 헤드 노드인지를 검사하는 기능이 없다.
********/
void deleteTargetMember(MEMBER* target)
{
        // 만약 target 이 마지막 노드라면, prev->next 를 NULL 로 설정한다.
```

```
        if (target->next == NULL) {
                target->prev->next = NULL;
        }

        // 만약 target 이 중간에 있는 노드라면, target 의 prev 와 next 노드를 서로
연결한다.
        else {
                target->prev->next = target->next;
                target->next->prev = target->prev;
        }

        free(target);
}
```

3. Search

```
/********
* function : search
* purpose : menu- search 기능 구현, 이름을 입력받고 일치하는 멤버를 주소록에서 찾아
정보를 출력한다.
* parameters
*       head : 주소록의 헤드 노드
* description :
*       안내 메지지를 출력하고, 찾으려는 대상의 이름을 입력받는다.
*       findMember()함수를 통해 대상이 되는 멤버를 찾는다. 이를 targetMember 로
받는다.
*       만약 targetMember 가 NULL 이면, 에러 메시지를 출력하고 프로그램을 종료한다.
*       만약 targetMember 가 존재하면, 출력 양식에 맞추어 targetMember 의 정보를
출력한다.
*       출력 시 printMember() 함수, SHOWLINE 매크로 사용
*********/
void search(MEMBER* head)
{
        puts("[SEARCH]");
        char targetName[STRING_SIZE];
        printf("Input Name to Search: ");
        gets_s(targetName, sizeof(targetName));

        MEMBER* targetMember = findMember(head, targetName);
        if (targetMember == NULL) {
                printf("Error: No data found for Name “%s”\n", targetName);
                return;
        }
        else {
                SHOWLINE;
                printMember(targetMember);
        }
}


/*********
* function : findMember
* purpose : name 에 부합하는 멤버를 찾고 이를 반환한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
```

```
*       const char* name : 찾으려는 멤버의 이름
* return : 찾으려는 멤버 구조체 노드
* description
*       head 에 연결된 리스트 중에서 name 이 일치하는 멤버를 찾아 반환한다.
*       만약 일치하는 name 이 없다면, NULL 을 반환한다.
**********/
MEMBER* findMember(MEMBER* head, const char* name)
{
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
                if (strcmp(cur->name, name) == 0) {
                        return cur;
                }
        }

        return NULL;
}
```

4. PrintAll

```
/********
* function : printAllData
* purpose : 주소록의 모든 멤버를 양식에 맞추어 출력한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
* description
*       안내문을 추력한 후, 양식에 맞추어 모든 멤버를 출력한다.
*       출력 양식은 SHOWLINE 매크로에 맞추어 출력하며, 출력 시 printMember() 함수를
사용해 각 멤버를 출력한다.
*********/
void printAllData(MEMBER* head)
{
        puts("[Print All Data]");

        SHOWLINE;
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
                printMember(cur);
        }

        printf("\n");
}

/*******
* function : printMember
* purpose : 단일 member 의 정보를 출력 양식에 맞추어 출력한다.
* parameters
*       MEMBER* member : 출력할 대상 멤버의 노드
* description
*       대상 멤버를 출력 양식에 맞추어 출력한다.
*       이때 출력 양식은 SHOWLINE 매크로의 양식을 따른다.
*       만약 member 가 존재하지 않으면 에러 메시지를 출력하고 return 한다.
* note
```

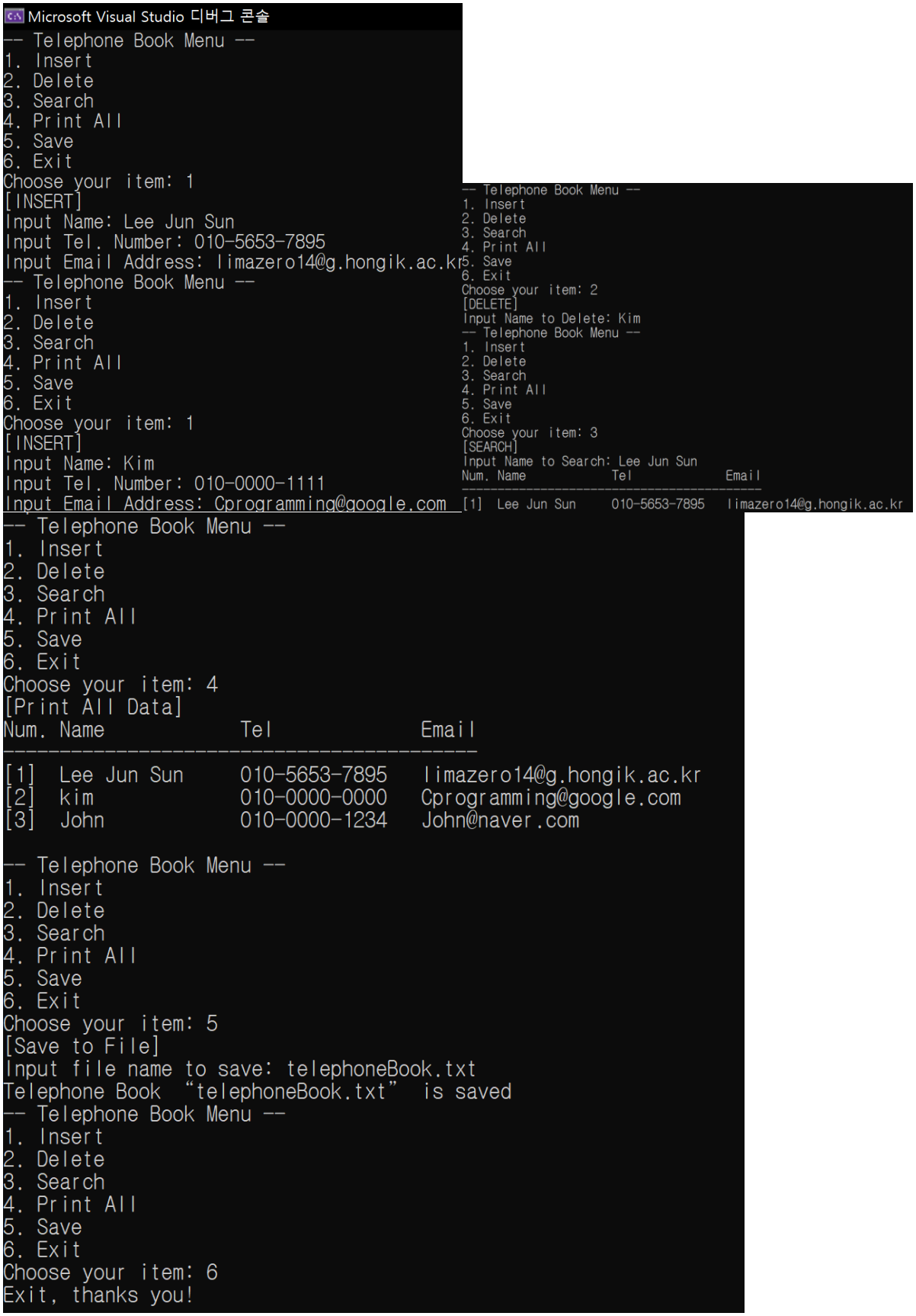
```
*       SHOWLINE 매크로가 수정되면 이 함수의 출력 양식도 같이 수정되어야 한다.
*       parameter 가 head 노드가 아니다! 이 함수에는 member 가 헤드 노드인지 검사하는
기능이 없다.
********/
void printMember(const MEMBER* member)
{
        if (member == NULL) {
                puts("ERROR : no data for such member");
                return;
        }
        printf("[%d]  %-15s %-15s %s\n", \
                member->order,\
                member->name, \
                member->telephoneNumber, \
                member->emailAddress);
}
```

5. Save

```
/********
* function : save
* purpose : 주소록을 파일로 저장한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
* description
*       파일 이름을 입력받고, fileName 파일에 주소록을 저장한다.
*       이때 저장 양식은 SHOWLINE 양식을 따르지 않고, 따로 정렬한다.
*       콘솔의 글자수와 파일의 글자수는 일부 차이가 있어, SHOWLINE 양식보다 더 크게
정렬함.
*       파일을 여는 데 실패했을 경우, 에러 메시지를 출력하고 반환한다.
* note
*       파일 저장 양식의 유동성을 확보하기 위해 .txt 를 붙이지 않는다. 입력자가 직접
"fileName.txt"로 입력해야 한다.
*********/
void save(MEMBER* head)
{
        puts("[Save to File]");

        char fileName[STRING_SIZE] = ""; ///> 파일 이름
        printf("Input file name to save: ");
        gets_s(fileName, sizeof(fileName));

        FILE* bookFile = 0;
        fopen_s(&bookFile, fileName, "wt");

        if (bookFile == NULL) {
                printf("ERROR: %s open failed\n", fileName);
                return;
        }

        // 정렬 양식에 맞추어 입력하기 위해, fprintf 함수 이용한다.
        fprintf(bookFile, "Num. % -20s % -20s %s\n", "Name", "Tel", "Email");
        MEMBER* cur = head;
        while (cur->next != NULL) {
```

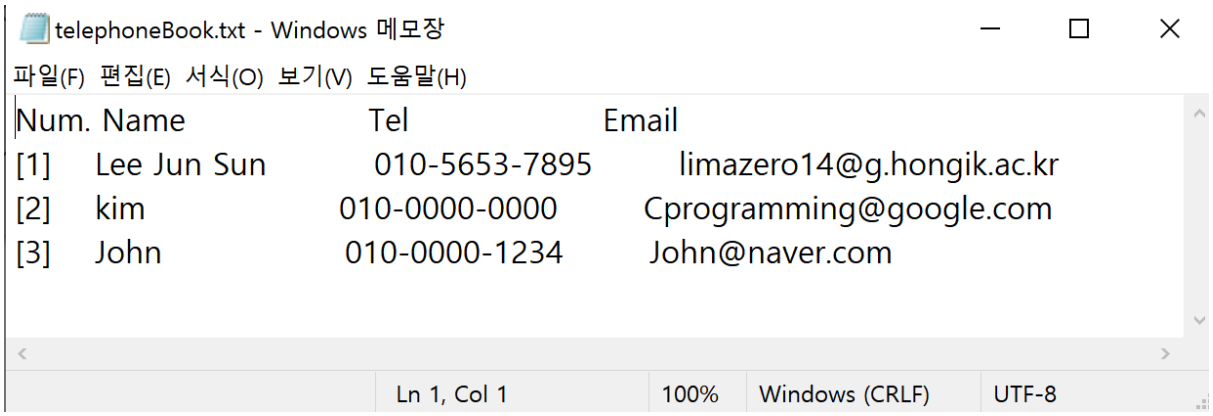
```
                cur = cur->next;
                fprintf(bookFile, "[%d]    %-20s %-20s %s\n", \
                        cur->order, \
                        cur->name, \
                        cur->telephoneNumber, \
                        cur->emailAddress);
        }

        fclose(bookFile);
        printf("Telephone Book “%s” is saved\n", fileName);
}
```

<실행 결과>



```
Microsoft Visual Studio 디버그 콘솔
-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 1
[INSERT]
Input Name: Lee Jun Sun
Input Tel. Number: 010-5653-7895
Input Email Address: limazero14@g.hongik.ac.kr
-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 1
[INSERT]
Input Name: Kim
Input Tel. Number: 010-0000-1111
Input Email Address: Cprogramming@google.com
-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 4
[Print All Data]
Num. Name            Tel              Email
-----------------------------------------
[1]  Lee Jun Sun     010-5653-7895    limazero14@g.hongik.ac.kr
[2]  kim             010-0000-0000    Cprogramming@google.com
[3]  John            010-0000-1234    John@naver.com

-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 5
[Save to File]
Input file name to save: telephoneBook.txt
Telephone Book "telephoneBook.txt" is saved
-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 6
Exit, thanks you!
```

```
-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 2
[DELETE]
Input Name to Delete: Kim
-- Telephone Book Menu --
1. Insert
2. Delete
3. Search
4. Print All
5. Save
6. Exit
Choose your item: 3
[SEARCH]
Input Name to Search: Lee Jun Sun
Num. Name            Tel              Email
-----------------------------------------
[1]  Lee Jun Sun     010-5653-7895    limazero14@g.hongik.ac.kr
```



<코드 설명>

* 코드의 자세한 작동 방식은 주석으로 대체함

1. MEMBER 구조체
   이중 연결 리스트로 계획하였다. 코드의 직관성을 위해 원형 리스트로 구현하지는 않았다.
   next 는 다음 노드를, prev 는 이전 노드를 참조한다.
   member 의 이름, 전화번호, 이메일 주소, 순번을 저장한다.
   이때 순번은 생성 순서대로 부여받는 번호이지만, 멤버가 삭제될 때 resetOrder() 함수를 통해 다시 부여받는 경우도 있다.
2. Insert
   appendMember() 함수는 주소록 리스트 끝에다가 새 멤버를 추가한다.
   문제 조건 상 주소록의 최대 멤버 수는 10 개이다. 따라서 새로 추가된 멤버의 order 가 10 을 초과하면 해당 멤버를 삭제하고 추가하지 않는다.
3. Delete
   findMember() 함수로 삭제 대상 노드를 찾는다. deleteTargetMember() 함수로 대상 노드를 삭제한다. 동시에 resetOrder() 함수로 순번을 재정렬한다.
   resetOrder(head, startNum) 함수는 startNum 을 시작으로 order 을 다시 정렬하는 함수이다.
   deleteTargetMember() 함수가 실질적으로 노드를 지운다. 이때 적절히 대상 이전 노드와 이후 노드를 연결한다.
4. Search
   findMember() 함수를 이용해 대상을 찾고 출력한다.
   findMember(head, name) 함수는 name 과 동일한 이름을 갖는 노드를 반환하는 함수이다. 만약 name 이 주소록 내 없다면 NULL 을 반환한다.
5. PrintAll
   printMember() 함수를 이용해 주소록의 모든 멤버를 출력한다.
6. Save
   주소록의 데이터를 파일로 저장한다.

정렬은 console 에서 출력하는 것과 조금 다른데, 이유는 UTF-8 등에서는 한글과 영어가 각각 차지하는 공간이 console 에서 조금 다르기 때문이다.

<전체 소스 코드>

Program 1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

// 행렬 X, Y 를 표시하기 위한 매크로
#define X 0
#define Y 1

// 행렬의 개수, 기본적으로 X, Y 임으로 두 개이다.
#define MAT_SIZE 2

// selectMenu 에서 사용되는, 행렬이 존재하는 지 판별하고 존재하지 않을 경우 에러 메시지를 출력하고 리턴하는 매크로
#define MATRIX_EXIST(flag) if(flag == 0){ puts("Error: Random Square Matrix Not Ready"); return; }

// malloc 함수에서 NULL 이 리턴되었을 때를 처리하기 위한 매크로
// 만약 동적 할당에 실패하여 malloc 에서 NULL 이 발생하였을 때, 에러 메시지를 출력한다.
#define _MALLOC(ptr, size) if((ptr = malloc(size)) == NULL) { puts("Error: Matrix Generation Failure");}

// 계산 옵션 목록 상수
enum { PLUS, MINUS, MULTIPLICATIOIN };

// 메뉴 목록 상수
enum { RandomSquareMatrixGeneration = 1, Transpose, Rotation, Inverse, Calculation, Exit };

void printMenu();
void selectMenu(int menuSeleciton, int** [], int*);
int getMenuSelection();
void printMatrix(const int** matrix, int);
void printDoubleMatrix(const double** matrix, int);
int randomSquareMatrixGeneration(int** []);
int** generateBlankedMatrix(int);
double** generateDoubleMatrix(int);
void fillMatrixWithZero(int** matrix, int);
void fillMatrixWithRandom(int** matrix, int);
void deleteMatrix(void** matrix, int);
void transpose(int** [], int);
void rotation(int** [], int);
int getDegree();
void rotate90(int**, int);
void inverse(int** [], int);
double** getInverseMatrix(const int** matrix, int);
int getDeterminant(const int** matrix, int);
void calculation(int** [], int);
int getCalculationSelection();
void calc_plus(int** [], int);
void calc_minus(int** [], int);
void calc_multiplication(int** [], int);
```

```c
int main(void)
{

	int** MATRIX[MAT_SIZE] = {NULL, NULL}; ///>MATRIX[X], [Y]를 담을 int**의 배열
	int SQUARE = 0; ///>정방 행렬의 열(행) 수

	int menuSelection = 0; ///>메뉴 선택지

	// menuSelection 이 Exit 이 아닐 동안 계속 반복해서 메뉴를 입력받는다.
	while (menuSelection != Exit) {
		printMenu();
		menuSelection = getMenuSelection();
		selectMenu(menuSelection, MATRIX, &SQUARE);
	}

	//프로그램이 종료될 때, 행렬을 모두 해제하고 종료한다.
	for (int i = 0; i < MAT_SIZE; i++) {
		if(MATRIX[i] != NULL) {
			deleteMatrix(MATRIX[i], SQUARE);
		}
	}

	return 0;
}

/*********
* function : printMenu
* purpose : 메뉴 선택지를 출력한다.
**********/
void printMenu()
{
	puts("-- Menu --");
	puts("1. Random Square Matrix Generation");
	puts("2. Transpose(T)");
	puts("3. Rotation(90, 180, 270, 360 degree)");
	puts("4. Inverse(^ -1)");
	puts("5. Calculation(+, -, *)");
	puts("6. Exit");
}

/********
* function : selectMenu
* purpose : menuSelection 을 바탕으로 지정된 메뉴 함수를 호출한다.
* parameters
*	int menuSelection : 메뉴 선택지
*	int** matrix[] : 행렬 X, Y		///> 이후 생략
*	int* square : 행렬의 열(또는 행)
* descriptions
*	메뉴 목록은 printMenu() 또는 enum 참고
*	만약 menuSelection 이 올바른 메뉴 선택지가 아닐 경우, 안내문을 출력하고 반환한다.
*	행렬이 존재하지 않는 상태에서 1 번 이외 메뉴를 선택할 경우 안내문을 출력하고 반환한다.
이때 MATRIX_EXIST 매크로 사용
* note
*	square 을 포인터로 받는다.
```

```
*		RandomSquareMatrixGeneration 메뉴 실행 시 메인함수에서 선언된 square을 직접 수정한다.
*		EXIT이 선택되어도 행렬 MATRIX를 따로 해제하지는 않는다. 메인함수에서 따로 해제해줘야 한다.
*********/
void selectMenu(int menuSeleciton, int** MATRIX[], int* square)
{
	static int isMatrixGenerated = 0; ///> 행렬이 이미 있는 지 확인하는 bool값, 0이면 없고, 1이면 있다는 뜻이다.
	int SQUARE = *square; ///>포인터로 받은 square을 안전하게 사용하기 위해 SQUARE에 복사한다.

	switch (menuSeleciton) {
	case RandomSquareMatrixGeneration:
		//이미 생성된 행렬이 존재할 시, 기존 행렬을 삭제하고 다시 생성한다.
		if (isMatrixGenerated != 0) {
			for (int i = 0; i < MAT_SIZE; i++) {
				deleteMatrix(MATRIX[i], SQUARE);
			}
		}
		*square = randomSquareMatrixGeneration(MATRIX);
		isMatrixGenerated = 1;
		break;
	case Transpose:
		MATRIX_EXIST(isMatrixGenerated)
		transpose(MATRIX, SQUARE);
		break;
	case Rotation:
		MATRIX_EXIST(isMatrixGenerated)
		rotation(MATRIX, SQUARE);
		break;
	case Inverse:
		MATRIX_EXIST(isMatrixGenerated)
		inverse(MATRIX, SQUARE);
		break;
	case Calculation:
		MATRIX_EXIST(isMatrixGenerated)
		calculation(MATRIX, SQUARE);
		break;
	case Exit:
		puts("Exit, thank you");
		return;
	default:
		//메뉴 선택 목록에 없을 시, 안내문 출력
		puts("Thou selected wrong menu, try again");
	}
}

/********
* function : getMenuSelection
* purpose : 키보드로 메뉴 선택지를 입력받는다.
* description
*		안내문을 출력하고 메뉴 선택지를 정수로 입력받는다.
* note
*		올바르지 않은 메뉴를 선택했을 시 이를 검사하는 기능은 없다. 오직 입력받은 integer만 반환한다.
```

```
*		입력 버퍼 자동적으로 비워주는 기능 포함되어 있다.
*********/
int getMenuSelection()
{
	int selection = 0;
	printf("Choose the item you want: ");
	scanf_s("%d", &selection);
	while (getchar() != '\n'); //입력 버퍼 비우기
	return selection;
}

/*******
* function : printMatrix
* purpose : 행렬을 출력한다.
* parameters
*		int** matrix : 출력할 행렬
*		int SQUARE : 행렬의 열(또는 행) ///> 이하 생략
* description
*		행렬을 출력한다. 이때 줄간격은 기본적으로
*		"X = " 다음에 출력하는 것을 전제로 정렬한다.
* note
*		int** []을 매개변수로 받지 않는다. 즉 MATRIX[X] 이런 식으로 전달한다.
********/
void printMatrix(const int** matrix, int SQUARE)
{
	for (int i = 0; i < SQUARE; i++) {
		for (int j = 0; j < SQUARE; j++) {
			printf("%2d ", matrix[i][j]);
		}
		printf("\n");
		printf("    "); // 정렬 기능
	}
	printf("\n");
}

/********
* function : printDoubleMatrix
* purpose : double 형 행렬을 출력한다.
* parameters
*		double** matrix : double 형으로 이루어진 행렬
* description
*		printMatrix 의 double 형 출력 버전
*		출력 시 기본적으로 "X^(-1) = " 에 맞추는 것을 전제로 정렬한다.
*********/
void printDoubleMatrix(const double** matrix, int SQUARE)
{
	for (int i = 0; i < SQUARE; i++) {
		for (int j = 0; j < SQUARE; j++) {
			printf("%9f ", matrix[i][j]);
		}
		printf("\n");
		printf("         "); //정렬 기능
	}
	printf("\n");
}
```

```c
/********
* function : randomSquareMatrixGeneration
* purpose : menu 1_RandomSquareMatrixGeneration 기능 수행
* return : SQUARE, 행렬의 열(=행)
* description
*        안내문을 출력하고 SQUARE 값을 입력받는다.
*        generateBlandkedMatrix 함수로 행렬을 생성한다.
*        fillMatrixWithRandom 함수로 행렬을 무작위 숫자로 채운다.
*        행렬 X, Y 를 출력한다.
*        입력받는 SQUARE 을 반환한다.
*********/
int randomSquareMatrixGeneration(int** MATRIX[])
{
        int SQUARE = 0;
        printf("Input the number of rows (2 or 3): ");
        scanf_s("%d", &SQUARE);

        for (int i = 0; i < MAT_SIZE; i++) {
                MATRIX[i] = generateBlankedMatrix(SQUARE);
        }

        srand((unsigned int)time(NULL));
        for (int i = 0; i < MAT_SIZE; i++) {
                fillMatrixWithRandom(MATRIX[i], SQUARE);
        }

        printf("X = ");
        printMatrix(MATRIX[X], SQUARE);

        printf("Y = ");
        printMatrix(MATRIX[Y], SQUARE);

        return SQUARE;
}

/********
* function : generateBlankedMatrix
* purpose : SQUARE * SQUARE 의 정방 행렬을 생성한다.
* return : 생성된 int** 형 matrix
* description
*        동적할당으로 2 차원 배열 생성
*        동적할당 실패를 대비해 _MALLOC 매크로 사용
* note
*        0 으로 초기화 안 한다.
*********/
int** generateBlankedMatrix(int SQUARE)
{
        int** matrix = NULL;
        _MALLOC(matrix, sizeof(int*) * SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                _MALLOC(matrix[i], sizeof(int) * SQUARE);
        }
        return matrix;
}

/********
```

```
* function : generateDoubleMatrix
* purpose : SQUARE * SQUARE 의 double 형 정방 행렬을 생성한다.
* return : 생성된 double** 형 matrix
* description
*        generateBlankedMatrix 의 double 형 버전
*        모든 요소들을 0.0 으로 초기화한다.
*********/
double** generateDoubleMatrix(const int SQUARE)
{
        double** matrix = NULL;
        _MALLOC(matrix, sizeof(double*) * SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                _MALLOC(matrix[i], sizeof(double) * SQUARE);
                for (int j = 0; j < SQUARE; j++) {
                        matrix[i][j] = 0.0;
                }
        }
        return matrix;
}

/*******
* function : fillMatrixWithZero
* purpose : 행렬의 모든 요소를 0 으로 초기화한다.
* parameters
        int** matrix : 0 으로 초기화할 대상 행렬
********/
void fillMatrixWithZero(int** matrix, int SQUARE)
{
        for (int i = 0; i < SQUARE; i++) {
                memset(matrix[i], 0, sizeof(int) * SQUARE);
        }
}

/*******
* function : fillMatrixWithRandom
* purpose : 행렬의 모든 요소를 난수로 채운다.
* parameters
*        int** matrix : 0 으로 초기화할 대상 행렬
* description
*        0 부터 99 까지의 임의의 정수로 행렬을 채운다.
* note
*        srand 로 시드값 초기화하지 않음. 미리 srand 로 시드값을 설정해야 함.
********/
void fillMatrixWithRandom(int** matrix, int SQUARE)
{
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        matrix[i][j] = rand() % 100;
                }
        }
}

/********
* function : deleteMatrix
* purpose : 2 차원 행렬을 해제한다.
* paremeters
```

```
*       void** matrix : 해제하고자 하는 대상 행렬
*********/
void deleteMatrix(void** matrix,int SQUARE)
{
        if (matrix == NULL) return;
        for (int i = 0; i < SQUARE; i++) {
                if(matrix[i] != NULL)
                        free(matrix[i]);
        }

        free(matrix);
}

/********
* function : transpose
* purpose : menu 2_Transpose 기능 수행
* description
*       X 와 Y 의 전치행렬을 출력한다.
* note
*       결과 행렬을 반환하지는 않는다.
*********/
void transpose(int** MATRIX[], int SQUARE)
{
        printf("Transpose X and Y\n");

        printf("X^T = ");
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        printf("%2d ", MATRIX[X][j][i]);
                }
                printf("\n");
                printf("      ");
        }
        printf("\n");

        printf("Y^T = ");
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        printf("%2d ", MATRIX[Y][j][i]);
                }
                printf("\n");
                printf("      ");
        }
        printf("\n");
}

/********
* function : rotation
* purpose : menu 3_Rotation 기능 수행
* description
*       getDegree() 함수로 각도를 입력받고, 이를 통해 회전수를 계산한다.
*       회전수만큼 rotation 한 결과를 출력한다. (rotate90 함수 이용)
*********/
void rotation(const int** MATRIX[], int SQUARE)
{
        int degree = getDegree();
```

```
	int rotationCycle = degree / 90; ///> 회전 바퀴 수

	// 회전시킬 행렬 2 개를 생성한다.
	int** rotatedMatrix[MAT_SIZE] = { generateBlankedMatrix(SQUARE),
generateBlankedMatrix(SQUARE) };

	// 회전시킬 행렬에 원본 행렬을 복사한다.
	for (int i = 0; i < MAT_SIZE; i++) {
		for (int j = 0; j < SQUARE; j++) {
			for (int k = 0; k < SQUARE; k++) {
				rotatedMatrix[i][j][k] = MATRIX[i][j][k];
			}
		}
	}

	// rotate Matrix for rotation Cycle
	// ex : if degree == 180, cycle is 2 and rotate for 2 times
	for (int k = 0; k < MAT_SIZE; k++) { // k 는 X 와 Y
		for (int i = 0; i < rotationCycle; i++) {
			rotate90(rotatedMatrix[k], SQUARE);
		}
	}

	printf("X = ");
	printMatrix(rotatedMatrix[X], SQUARE);

	printf("Y = ");
	printMatrix(rotatedMatrix[Y], SQUARE);

	for (int i = 0; i < MAT_SIZE; i++) {
		deleteMatrix(rotatedMatrix[i], SQUARE);
	}
}

/********
* function : getDegree
* purpose : 정수형 각도를 입력받고, 이를 반환한다.
* return : degree
* description
	90 도, 180 도, 270, 360 도 중 하나를 입력받는다.
	만약 다른 각도가 입력되었을 시 안내문을 출력하고 다시 입력받는다.
*********/
int getDegree()
{
	int degree = -1;
	while (!(degree == 0 || degree == 90 || degree == 180 || degree == 270 ||
degree == 360)) {
		printf("Input the degree to rotate: ");
		scanf_s("%d", &degree);
		if (!(degree == 0 || degree == 90 || degree == 180 || degree == 270
|| degree == 360)) {
			printf("Wrong degree, try again\n");
		}
	}
	return degree;
}
```

```c
/*******
* function : rotate90
* purpose : 입력받은 행렬을 90 도만큼 시계방향 회전시킨다.
* parameters
*       int** originalMatrix : 90 도 회전시킬 행렬
********/
void rotate90(int** matrix, int SQUARE)
{
        int** rotatedMatrix = generateBlankedMatrix(SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        rotatedMatrix[j][SQUARE - i - 1] = matrix[i][j];
                }
        }

        // 원본 행렬 직접 수정
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        matrix[i][j] = rotatedMatrix[i][j];
                }
        }
        deleteMatrix(rotatedMatrix, SQUARE);
}

/********
* function : inverse
* purpose : menu 4_Inverse 기능 실행
* description
*       getInverseMatrix() 함수로 역행렬을 계산하고, 만약 역행렬이 존재하면 이를 출력한다.
* description
*       getInverseMatrix()를 통해 X, Y 가 역행렬이 존재하는 지를 각각 판단한다(NULL 일 경우
존재하지 않는 것임).
*       이를 printDoubleMatrix() 로 출력한다.
*********/
void inverse(int** MATRIX[], int SQUARE)
{
        double** inverseMatrix[MAT_SIZE];   ///> 역행렬 X, Y 를 저장할 double** [] 형
행렬
        inverseMatrix[X] = getInverseMatrix(MATRIX[X], SQUARE);
        inverseMatrix[Y] = getInverseMatrix(MATRIX[Y], SQUARE);

        // 행렬 X, Y 가 모두 역행렬이 존재할 때
        if (inverseMatrix[X] != NULL && inverseMatrix[Y] != NULL) {
                printf("X and Y are Invertible, \n");
                printf("Inverse X and Y\n");

                printf("X^(-1) = ");
                printDoubleMatrix(inverseMatrix[X], SQUARE);

                printf("Y^(-1) = ");
                printDoubleMatrix(inverseMatrix[Y], SQUARE);
        }
        // 행렬 X 가 역행렬이 존재하지 않고, 행렬 Y 만 역행렬이 존재할 때
        else if (inverseMatrix[X] == NULL && inverseMatrix[Y] != NULL) {
                printf("X is not invertible but Y is invertible, \n");
```

```
                printf("Inverse Y\n");

                printf("Y^(-1) = ");
                printDoubleMatrix(inverseMatrix[Y], SQUARE);
        }
        // 행렬 X만 역행렬이 존재하고, 행렬 Y는 역행렬이 존재하지 않을 때
        else if (inverseMatrix[X] != NULL && inverseMatrix[Y] == NULL) {
                printf("X is invertible but Y is not invertible, \n");
                printf("Inverse X\n");

                printf("X^(-1) = ");
                printDoubleMatrix(inverseMatrix[X], SQUARE);
        }
        // 기타 상황, 행렬 X, Y가 모두 역행렬이 존재하지 않을 때
        else {
                printf("X and Y are not invertible\n");
                deleteMatrix(inverseMatrix[X], SQUARE);
                deleteMatrix(inverseMatrix[Y], SQUARE);
        }

        for (int i = 0; i < MAT_SIZE; i++) {
                deleteMatrix(inverseMatrix[i], SQUARE);
        }
}

/********
* function : getInverseMatrix
* purpose : 역행렬을 반환한다.
* parameters
*       int** matrix : 원본 행렬
* return
*       double** resultMatrix, 역행렬
*       또는 역행렬이 존재하지 않으면 NULL 반환
* description
*       행렬식 determinant를 구해서 역행렬을 계산하는 것은, 행렬의 크기에 따라 너무
복잡해지므로,
*       보다 간편하고 컴퓨터 친화적인 Gauss-Jordan 소거법을 이용하여 역행렬을 계산한다.
*       단, 행렬식이 0인지를 확인하여 해당 행렬의 역행렬을 가지는 지를 검사하기는 한다.
*       구한 역행렬이 실제 matrix의 역행렬인지 알기 위해 resultMatrix * matrix ==
IdentityMatrix(단위 행렬)인지를 확인한다.
*       만약 resultMatrix * matrix != IdentityMatrix 이면 matrix는 올바른 역행렬이
존재하지 않는 것이다. NULL을 리턴한다.
* note
*       원본 matrix는 수정되지 않는다.
*********/
double** getInverseMatrix(const int** matrix, int SQUARE) {
        if (getDeterminant(matrix, SQUARE) == 0) return NULL; // Det (Matrix) == 0 이면
역행렬이 존재할 수 없음

        const double ERROR = 1.0e-10; ///> 부동소수점 오류 기준값

        // *note : generateDoubleMatrix()는 0으로 초기화를 동시에 진행함.
        double** resultMatrix = generateDoubleMatrix(SQUARE); ///> 역행렬 결과를 저장할
행렬
        double** MatrixCopy = generateDoubleMatrix(SQUARE); ///> 원본 행렬의 복사본,
가우스 조던 소거법을 이용하기에 이 행렬을 수정한다.
```

```
        // MatrixCopy 에 원본 matrix 복사
        // int -> double 형으로 형변환하여 복사한다.
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        MatrixCopy[i][j] = (double)matrix[i][j];
                }
        }

        // 계산 결과가 저장되는 resultMatrix 행렬을 단위행렬로 초기화
        for (int i = 0; i < SQUARE; i++)
                for (int j = 0; j < SQUARE; j++)
                        resultMatrix[i][j] = (i == j) ? 1 : 0;

        // 대각요소를 0 이 아닌 수로 만든다.
        // 가우스 조던 방법의 해가 부정(해가 무수히 많음)일 경우를 대비한 과정이다.
        for (int i = 0; i < SQUARE; i++) {
                if (-ERROR < MatrixCopy[i][i] && MatrixCopy[i][i] < ERROR) { //
MatrixCopy[i][i] == 0
                        for (int k = 0; k < SQUARE; k++) {
                                if (-ERROR < MatrixCopy[k][i] && MatrixCopy[k][i] <
ERROR) // MatrixCopy[k][i] == 0
                                        continue;
                                for (int j = 0; j < SQUARE; j++) {
                                        MatrixCopy[i][j] += MatrixCopy[k][j];
                                        resultMatrix[i][j] += resultMatrix[k][j];
                                }
                                break;
                        }
                        if (-ERROR < MatrixCopy[i][i] && MatrixCopy[i][i] < ERROR) {
                                //수정 사항을 모두 완료했음에도 대각선 요소가 0 ->
가우스-조던 방법 불가능
                                deleteMatrix(resultMatrix, SQUARE);
                                deleteMatrix(MatrixCopy, SQUARE);

                                return NULL;
                        };
                }
        }


        /* Gauss-Jordan elimination */
        for (int i = 0; i < SQUARE; i++) {
                // 대각 요소를 1 로 만듦
                double constant = MatrixCopy[i][i]; ///> 대각선 요소의 값 저장
                for (int j = 0; j < SQUARE; j++) {
                        MatrixCopy[i][j] /= constant;   // MatrixCopy[i][i] 를 1 로
만드는 작업
                        resultMatrix[i][j] /= constant;   // i 행 전체를
MatrixCopy[i][i] 로 나눔
                }

                // i 행을 제외한 k 행에서 MatrixCopy[k][i] 를 0 으로 만드는 단계
                for (int k = 0; k < SQUARE; k++) {
                        if (k == i) continue;      // 자기 자신의 행은 건너뜀
```

```
                        if (MatrixCopy[k][i] == 0) continue;   // 이미 0 이 되어
있으면 건너뜀

                        // MatrixCopy[k][i] 행을 0 으로 만듦
                        constant = MatrixCopy[k][i];
                        for (int j = 0; j < SQUARE; j++) {
                                MatrixCopy[k][j] = MatrixCopy[k][j] -
MatrixCopy[i][j] * constant;
                                resultMatrix[k][j] = resultMatrix[k][j] -
resultMatrix[i][j] * constant;
                        }
                }
        }

        deleteMatrix(MatrixCopy, SQUARE);

        return resultMatrix;
}

/********
* function : getDeterminant
* purpose : 행렬의 행렬식(Det)를 구한다
* parameters
*       int** matrix : 행렬 M
* return : 행렬의 행렬식
*********/
int getDeterminant(const int** matrix, int SQUARE)
{
        if (SQUARE == 1) return matrix[0][0]; // 1 * 1 행렬일 시 그 값 그대로 반환

        int*** minor_Matrix = malloc(sizeof(int**) * SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                minor_Matrix[i] = generateBlankedMatrix(SQUARE);
        }
        for (int k = 0; k < SQUARE; k++) {
                for (int i = 0; i < SQUARE - 1; i++)
                        for (int j = 0; j < SQUARE; j++) {
                                if (j < k)
                                        minor_Matrix[k][i][j] = matrix[i + 1][j];
                                else if (j > k)
                                        minor_Matrix[k][i][j - 1] = matrix[i +
1][j];
                        }
        }
        int sum = 0;
        int test = 1;
        for (int k = 0; k < SQUARE; k++) {
                sum += test * matrix[0][k] * getDeterminant(minor_Matrix[k], SQUARE -
1);
                test *= -1;
        }

        for (int i = 0; i < SQUARE; i++) {
                deleteMatrix(minor_Matrix[i], SQUARE);
        }
        return sum;
```

```
}

/********
* function : calculation
* purpose : menu 5_Calculation 의 기능 실행
* description
*        getCalculationSelection() 을 이용해 계산 옵션(+, -, * 중 하나)를 선택받고, 그에
맞는 연산 결과를 출력한다.
*        연산 결과는 calc_함수들이 담당함(함수 자체에서 print 하지는 않음)
*********/
void calculation(int** MATRIX[], int SQUARE)
{
        int calculationSelection = getCalculationSelection();
        switch (calculationSelection) {
        case PLUS:
                calc_plus(MATRIX, SQUARE);
                break;
        case MINUS:
                calc_minus(MATRIX, SQUARE);
                break;
        case MULTIPLICATIOIN:
                calc_multiplication(MATRIX, SQUARE);
                break;
        default:
                puts("ERROR : Wrong Calculation menu selected");
        }
}

/********
* function : getCalculationSelection
* purpose : calculationSelection 을 입력받고 반환하는 함수
* return : calculationSelction(PLUS, MINUS, MULTIPLICATION) 반환
* description
*        getchar() 함수를 통해 +, -, * 중 하나를 입력받는다.
*        입력받은 이후, 나머지 입력은 모두 버퍼를 비운다.
*        +면 PLUS, -면 MINUS, *면 MULTIPLICATION 을 반환한다.
*        모두 아니면, 잘못 선택되었다는 안내문을 출력한 후 다시 입력받는다.
* note
*        입력버퍼를 비우는 기능은 있으나, 여러 개의 문자가 입력된 상황에서 에러가 발생할 수
있다.
*********/
int getCalculationSelection()
{
        char selection; ///> char 형 메뉴 선택지
        while (1) {
                printf("Input the calculation (+, -, or *): ");
                selection = getchar();
                while (getchar() != '\n'); // 입력버퍼 비우기
                if (selection == '+') return PLUS;
                else if (selection == '-') return MINUS;
                else if (selection == '*') return MULTIPLICATIOIN;
                else puts("Thou selected wrong menu. Try again"); //올바르지 않은
선택지면 에러메시지 출력
        }
}
```

```
/********
* function : calc_plus
* purpose : calculation menu 중 +(덧셈) 메뉴 기능 구현
* description
*        행렬 X, Y를 더한 결과를 출력한다.
* note
*        printMatrix 함수를 이용하기에, 출력 정렬이 깨지는 경우가 발생한다.
*        결과 행렬을 반환하지는 않는다.
*********/
void calc_plus(int** MATRIX[], int SQUARE)
{
        int** resultMatrix = generateBlankedMatrix(SQUARE); ///>덧셈 결과가 저장될 행렬
        fillMatrixWithZero(resultMatrix, SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        resultMatrix[i][j] = MATRIX[X][i][j] + MATRIX[Y][i][j];
                }
        }

        printf("X + Y = ");
        printMatrix(resultMatrix, SQUARE);
        deleteMatrix(resultMatrix, SQUARE);
}

/********
* function : calc_minus
* purpose : calculation menu 중 -(뺄셈) 메뉴 기능 구현
* description
*        행렬 X, Y를 뺀 결과를 출력한다.
* note
*        printMatrix 함수를 이용하기에, 출력 정렬이 깨지는 경우가 발생한다.
*        결과 행렬을 반환하지는 않는다.
*********/
void calc_minus(int** MATRIX[], int SQUARE)
{
        int** resultMatrix = generateBlankedMatrix(SQUARE);
        fillMatrixWithZero(resultMatrix, SQUARE);
        for (int i = 0; i < SQUARE; i++) {
                for (int j = 0; j < SQUARE; j++) {
                        resultMatrix[i][j] = MATRIX[X][i][j] - MATRIX[Y][i][j];
                }
        }

        printf("X + Y = ");
        printMatrix(resultMatrix, SQUARE);
        deleteMatrix(resultMatrix, SQUARE);
}

/********
* function : calc_multiplication
* purpose : calculation menu 중 *(곱셈) 메뉴 기능 구현
* description
*        행렬 X, Y를 곱한 결과를 출력한다.
* note
*        printMatrix 함수를 이용하기에, 출력 정렬이 깨지는 경우가 발생한다.
*        결과 행렬을 반환하지는 않는다.
```

```
*********/
void calc_multiplication(int** MATRIX[], int SQUARE)
{
	int** resultMatrix = generateBlankedMatrix(SQUARE);
	fillMatrixWithZero(resultMatrix, SQUARE);
	for (int i = 0; i < SQUARE; i++) {
		for (int j = 0; j < SQUARE; j++) {
			for (int k = 0; k < SQUARE; k++) {
				resultMatrix[i][j] += (MATRIX[X][i][k] *
MATRIX[Y][k][j]);
				// i, j 칸에 대해 i 행과 j 열을 k 로 순환하며 곱한 결과를
모두 합한다.
			}
		}
	}

	printf("X * Y = ");
	printMatrix(resultMatrix, SQUARE);
	deleteMatrix(resultMatrix, SQUARE);
}
```

Program 2)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 정렬 양식을 출력하는 매크로
#define SHOWLINE printf("Num. %-15s %-15s %s\n", "Name", "Tel", "Email"); puts("-------
-----------------------------------");
// 문자열 크기
#define STRING_SIZE 256

/*******
* 메뉴 선택지 상수
* 이 enum 을 참조하는 함수:
*        printMenu(), selectMenu
********/
enum { Insert = 1, Delete, Search, PrintAll, Save, Exit };

/*******
* struct : MEMBER
* description
*        이름, 전화번호, 이메일 주소, order 로 이루어진 구조체
*        이중 연결 리스트이나, 원형 연결 리스트는 아니다.
*        즉 head->prev = NULL 이고, last->next = NULL 이 저장되어야 한다.
********/
typedef struct _MEMBER {
        struct _MEMBER* next; ///> 다음 노드 저장
        struct _MEMBER* prev; ///> 이전 노드 저장
        char name[STRING_SIZE]; ///> 이름
        char telephoneNumber[STRING_SIZE]; ///> 전화번호
        char emailAddress[STRING_SIZE]; ///> 이메일 주소
        int order; ///> 멤버의 순서이자 번호. 생성 순서대로 번호를 부여받는다.
} MEMBER;

void printMenu();
void selectMenu(MEMBER*, int);
int getMenuSelection();
void initializeHead(MEMBER* head);
void printAllData(MEMBER*);
void printMember(MEMBER*);
void insert(MEMBER*);
void appendMember(MEMBER* list, const char* name, const char* telephoneNumber, const
char* emailAddress, int order);
MEMBER* findMember(MEMBER* list, const char* name);
void search(MEMBER* head);
void deleteAllData(MEMBER* head);
void deleteTargetMember(MEMBER*);
void deleteMember(MEMBER* head);
void resetOrder(MEMBER*, int);
void save(MEMBER* head);


int main(void)
{
        int menuSelection = 0; ///> 메뉴 선택지
        MEMBER* head = malloc(sizeof(MEMBER)); ///> 주소록의 head 노드
```

```
	initializeHead(head); // head 노드를 기본값으로 초기화한다.

	while (menuSelection != Exit) { // menuSelection 이 EXIT 이면 반복문을 탈출한다
		printMenu();
		menuSelection = getMenuSelection();
		selectMenu(head, menuSelection);
	}

	deleteAllData(head);
	return 0;
}

/*********
* function : printMenu
* purpose : 메뉴 목록을 출력한다.
**********/
void printMenu()
{
	puts("-- Telephone Book Menu --");
	printf("%d. Insert\n", Insert);
	printf("%d. Delete\n", Delete);
	printf("%d. Search\n", Search);
	printf("%d. Print All\n", PrintAll);
	printf("%d. Save\n", Save);
	printf("%d. Exit\n", Exit);
}

/********
* function : selectMenu
* purpose : selection 에 대응하는 메뉴 기능을 실행시킨다.
* parameters
*	MEMBER* head : 주소록의 헤더 노드
*	int selection : 메뉴 선택
* description :
*	selection 에 대응하는 기능(함수)를 실행한다.
*	만약 selection 에 대응하는 메뉴가 없을 경우, 에러 메시지를 출력한다.
*	Exit 을 선택한 경우, 굿바이 메시지를 출력하고 반복문을 종료한다.
*	selection 목록은 enum 참고
* note :
*	EXIT 을 선택한 경우에도 함수 자체가 프로그램을 종료하진 않는다. 메인 함수에서
종료해주어야 한다.
*	EXIT 을 선택한 경우에도 주소록의 메모리를 해제하지는 않는다. 메인 함수에서
해제해주어야 한다.
********/
void selectMenu(MEMBER* head, int selection)
{
	switch (selection) {
	case Insert:
		insert(head);
		break;
	case Delete:
		deleteMember(head);
		break;
	case Search:
		search(head);
		break;
```

```
        case PrintAll:
                printAllData(head);
                break;
        case Save:
                save(head);
                break;
        case Exit:
                puts("Exit, thanks you!");
                return;
        default:
                puts("Thou selected wrong menu");
        }
}

/*********
* function : getMenuSelection
* purpose : menuSelection 을 입력받고 반환한다.
* description :
*       안내 메시지를 출력하고, 정수 값을 입력받고 이를 반환한다.
*       입력버퍼를 자체적으로 비운다. 즉 '\n'(엔터) 를 입력 버퍼에서 제거하는 기능이 포함되어
있다.
* note :
*       자체적으로 올바른 메뉴가 선택되었는 지 확인하는 기능은 없다.
*********/
int getMenuSelection()
{
        int menuSelection = 0;
        printf("Choose your item: ");
        scanf_s("%d", &menuSelection);

        while (getchar() != '\n');
        // clear the stdin buffer

        return menuSelection;
}

/*********
* function : initializeHead
* purpose : head 노드를 초기화한다.
* parameters : 주소록의 head node
* description:
*       head 노드가 처음 생성되었을 때, head 노드를 초기화하는 함수이다.
*       prev 와 next 포인터를 NULL 로 초기화한다.
*       order = -1
*       name, emailAddress, telephoneNumber 을 각각 HEAD NAME, HEAD ADDRESS, 000-000-
0000 으로 초기화한다.
**********/
void initializeHead(MEMBER* head)
{
        head->prev = NULL;
        head->next = NULL;
        head->order = -1;
        strcpy_s(head->name, sizeof(head->name), "HEAD NAME");
        strcpy_s(head->emailAddress, sizeof(head->emailAddress), "HEAD ADDRESS");
        strcpy_s(head->telephoneNumber, sizeof(head->telephoneNumber), "000-000-
0000");
```

```
}

/********
* function : appendMember
* purpose : 주소록에 멤버를 추가한다.
* parameters
*        MEMBER* head : 주소록의 헤드 노드
*        const char* name, telephoneNumber, emailAddress
*        const int order
* description :
*        주소록 리스트의 맨 끝에 새 멤버 노드를 추가한다.
*        만약 새 노드 생성에 실패할 경우, 에러 메시지를 출력한 후 함수를 종료한다.
*        노드 생성에 성공할 경우, 매개변수로 받은 name, telephoneNumber, emailAddress,
order 값을 갖는 멤버가 추가된다.
*        추가된 멤버의 next 포인터는 NULL 이고, prev 는 바로 직전 노드를 참조한다.
*        바로 직전 노드의 next 포인터를 추가된 노드를 참조한다.
* note :
*        새 노드 생성에 실패해도 예외를 던지지 않는다. 에러 메시지만 출력한다.
*********/
void appendMember(MEMBER* head, const char* name, const char* telephoneNumber, const
char* emailAddress, const int order)
{
        MEMBER* newMember = malloc(sizeof(MEMBER));

        if (newMember == NULL) {
                puts("ERROR : node creation failed");
                return;
        }
        newMember->prev = NULL;
        newMember->next = NULL;

        // 데이터 취합부
        strcpy_s(newMember->name, sizeof(newMember->name), name);
        strcpy_s(newMember->telephoneNumber, sizeof(newMember->telephoneNumber),
telephoneNumber);
        strcpy_s(newMember->emailAddress, sizeof(newMember->emailAddress),
emailAddress);
        newMember->order = order;

        // head node 의 다음이 없을 경우, 즉 추가된 노드가 첫 노드일 경우
        if (head->next == NULL) {
                newMember->prev = head;
                newMember->next = NULL;

                head->next = newMember;
                head->prev = NULL;
        }
        else {
                MEMBER* cur = head; ///> cur 은 head 부터 끝헤드까지 탐색한다.
                while (cur->next != NULL) {
                        cur = cur->next;
                }

                // 주소록 끝에 newMember 를 연결한다.
                cur->next = newMember;
                newMember->prev = cur;
```

```
        }
}

/*********
* function : findMember
* purpose : name 에 부합하는 멤버를 찾고 이를 반환한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
*       const char* name : 찾으려는 멤버의 이름
* return : 찾으려는 멤버 구조체 노드
* description
*       head 에 연결된 리스트 중에서 name 이 일치하는 멤버를 찾아 반환한다.
*       만약 일치하는 name 이 없다면, NULL 을 반환한다.
**********/
MEMBER* findMember(MEMBER* head, const char* name)
{
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
                if (strcmp(cur->name, name) == 0) {
                        return cur;
                }
        }

        return NULL;
}

/********
* function : search
* purpose : menu- search 기능 구현, 이름을 입력받고 일치하는 멤버를 주소록에서 찾아 정보를
출력한다.
* parameters
*       head : 주소록의 헤드 노드
* description :
*       안내 메지지를 출력하고, 찾으려는 대상의 이름을 입력받는다.
*       findMember()함수를 통해 대상이 되는 멤버를 찾는다. 이를 targetMember 로 받는다.
*       만약 targetMember 가 NULL 이면, 에러 메시지를 출력하고 프로그램을 종료한다.
*       만약 targetMember 가 존재하면, 출력 양식에 맞추어 targetMember 의 정보를 출력한다.
*       출력 시 printMember() 함수, SHOWLINE 매크로 사용
*********/
void search(MEMBER* head)
{
        puts("[SEARCH]");
        char targetName[STRING_SIZE];
        printf("Input Name to Search: ");
        gets_s(targetName, sizeof(targetName));

        MEMBER* targetMember = findMember(head, targetName);
        if (targetMember == NULL) {
                printf("Error: No data found for Name “%s”\n", targetName);
                return;
        }
        else {
                SHOWLINE;
                printMember(targetMember);
        }
```

```
}

/********
* function : deleteAllData
* purpose : head 노드에 연결된 주소록의 모든 노드 메모리를 해제한다.
* parameters : 주소록의 헤드 노드
* description
*        head 노드에 연결된 주소록의 모든 노드 메모리를 해제한다.
*        주로 주소록이 필요 없을 때나, 프로그램을 종료할 때 사용한다.
* note
*        deleteMember, deleteTargetMember 와는 역할이 다르다. 주소록의 모든 데이터를
삭제한다.
*        head 노드를 포함해서 삭제한다. 따로 head 노드를 해제하려 하지 말 것
*********/
void deleteAllData(MEMBER* head)
{
        MEMBER* cur = head;
        MEMBER* next = NULL; ///> cur을  free()로 해제하기에, 임시로 다음 노드를 가리킬
next 포인터가 필요하다.
        while (cur != NULL) {
                next = cur->next; // cur을 해제하기 전, 임시로 next를 cur->next 노드로
저장한다.
                free(cur);
                cur = next;
        }
}
/******
* function : deleteMember
* purpose : menu-2. delete의 기능 구현, 이름을 입력받고 해당 이름에 일치하는 멤버 노드를
주소록에서 삭제한다.
* parameters : 주소록의 헤드 노드
* description
*        안내 메시지를 출력 후 이름을 입력받는다.
*        만약 이름에 일치하는 멤버가 주소록에 없으면 안내문을 출력한다.
*        만약 이름에 일치하는 멤버가 주소록에 있으면 deleteTargetMember를 통해 해당 멤버
노드를 삭제한다.
*******/
void deleteMember(MEMBER* head)
{
        puts("[DELETE]");

        char targetName[STRING_SIZE];
        printf("Input Name to Delete: ");
        gets_s(targetName, sizeof(targetName));

        MEMBER* targetMember = findMember(head, targetName);
        if (targetMember == NULL) {
                printf("Data for Name “%s” is not exist\n", targetName);
        }
        else {
                deleteTargetMember(targetMember);
                resetOrder(head, 1);
        }
}
/********
* function : resetOrder
```

```
* purpose : MEMBER 구조체의 order(순서)를 오름차순으로 재정렬한다.
* parameters
*       MEMBER* head : 주소록 리스트의 헤드
*       const int startNumber : 시작 순서, 이 숫자를 시작으로 오름차순 정렬한다.
* description
*       startNum 을 기준으로, head 노드에 연결되어 있는 연결 리스트의 order 을 오름차순
정렬한다.
*       주로 노드가 삭제되었을 때 등 순번에 대한 재정렬이 필요한 경우 사용한다.
*********/
void resetOrder(MEMBER* head, int startNumber)
{
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
                cur->order = startNumber;
                startNumber++;
        }
}

/********
* function : save
* purpose : 주소록을 파일로 저장한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
* description
*       파일 이름을 입력받고, fileName 파일에 주소록을 저장한다.
*       이때 저장 양식은 SHOWLINE 양식을 따르지 않고, 따로 정렬한다.
*       콘솔의 글자수와 파일의 글자수는 일부 차이가 있어, SHOWLINE 양식보다 더 크게 정렬함.
*       파일을 여는 데 실패했을 경우, 에러 메시지를 출력하고 반환한다.
* note
*       파일 저장 양식의 유동성을 확보하기 위해 .txt 를 붙이지 않는다. 입력자가 직접
"fileName.txt"로 입력해야 한다.
*********/
void save(MEMBER* head)
{
        puts("[Save to File]");

        char fileName[STRING_SIZE] = ""; ///> 파일 이름
        printf("Input file name to save: ");
        gets_s(fileName, sizeof(fileName));

        FILE* bookFile = 0;
        fopen_s(&bookFile, fileName, "wt");

        if (bookFile == NULL) {
                printf("ERROR: %s open failed\n", fileName);
                return;
        }

        // 정렬 양식에 맞추어 입력하기 위해, fprintf 함수 이용한다.
        fprintf(bookFile, "Num. % -20s % -20s %s\n", "Name", "Tel", "Email");
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
                fprintf(bookFile, "[%d]    %-20s %-20s %s\n", \
                        cur->order, \
```

```
                        cur->name, \
                        cur->telephoneNumber, \
                        cur->emailAddress);
        }

        fclose(bookFile);
        printf("Telephone Book “%s” is saved\n", fileName);
}

/*******
* function : deleteTargetMember
* purpose : target 에 해당하는 멤버 노드를 주소록에서 삭제한다.
* parameters
*       MEMBER* target : 지우려는 대상 멤버
* description
*       target 의 prev(이전) 노드와 next(이후) 노드를 적절히 연결한 후, free()함수를 통해
target 을 해제한다.
*       만약 target 이 마지막 노드일 시, prev 노드가 끝임을 나타내기 위해 prev->next =
NULL 로 설정한다.
* note
*       매개변수가 head 노드가 아닌 target 노드다.
*       이 함수에는 target 이 헤드 노드인지를 검사하는 기능이 없다.
********/
void deleteTargetMember(MEMBER* target)
{
        // 만약 target 이 마지막 노드라면, prev->next 를 NULL 로 설정한다.
        if (target->next == NULL) {
                target->prev->next = NULL;
        }

        // 만약 target 이 중간에 있는 노드라면, target 의 prev 와 next 노드를 서로 연결한다.
        else {
                target->prev->next = target->next;
                target->next->prev = target->prev;
        }

        free(target);
}

/********
* function : printAllData
* purpose : 주소록의 모든 멤버를 양식에 맞추어 출력한다.
* parameters
*       MEMBER* head : 주소록의 헤드 노드
* description
*       안내문을 추력한 후, 양식에 맞추어 모든 멤버를 출력한다.
*       출력 양식은 SHOWLINE 매크로에 맞추어 출력하며, 출력 시 printMember() 함수를 사용해
각 멤버를 출력한다.
*********/
void printAllData(MEMBER* head)
{
        puts("[Print All Data]");

        SHOWLINE;
        MEMBER* cur = head;
        while (cur->next != NULL) {
```

```
                cur = cur->next;
                printMember(cur);
        }

        printf("\n");
}

/*******
* function : printMember
* purpose : 단일 member 의 정보를 출력 양식에 맞추어 출력한다.
* parameters
*       MEMBER* member : 출력할 대상 멤버의 노드
* description
*       대상 멤버를 출력 양식에 맞추어 출력한다.
*       이때 출력 양식은 SHOWLINE 매크로의 양식을 따른다.
*       만약 member 가 존재하지 않으면 에러 메시지를 출력하고 return 한다.
* note
*       SHOWLINE 매크로가 수정되면 이 함수의 출력 양식도 같이 수정되어야 한다.
*       parameter 가 head 노드가 아니다! 이 함수에는 member 가 헤드 노드인지 검사하는 기능이
없다.
********/
void printMember(const MEMBER* member)
{
        if (member == NULL) {
                puts("ERROR : no data for such member");
                return;
        }
        printf("[%d]  %-15s %-15s %s\n", \
                member->order,\
                member->name, \
                member->telephoneNumber, \
                member->emailAddress);
}

/*******
* function : insert
* purpose : menu-1. insert 의 구현, 멤버의 정보를 입력받고 주소록에 추가한다.
* parameters :
*       MEMBER* head : 주소록의 헤드 노드
* description :
*       안내문을 출력하며, 이름, 전화번호, 이메일 주소 순으로 입력받는다.
*       memberOrder 은 기본적으로 1 에서부터 순차적으로 부여하기 위한 변수이지만,
*       resetOrder() 함수를 사용해서 새로 추가할 때마다 모든 주소록의 order 를 재부여하는
기능이 있다.
*       appendMember() 함수를 사용해서 입력받는 데이터에 부합되는 멤버를 주소록에 추가한다.
*
*       주소록의 크기를 10 으로 제한한다.
*       즉 order 가 10 보다 크면 주소록을 추가하지 않는다.
* note :
*       STRING_SIZE 보다 큰 바이트 크기의 문자열이 입력되었을 시 이에 대한 에러를 처리하지
않는다.
********/
void insert(MEMBER* head)
{
        char name[STRING_SIZE];
        char telephoneNumber[STRING_SIZE];
```

```
        char emailAddress[STRING_SIZE];
        static int memberOrder = 1; ///> memberOrder 의 순서를 부여하기 위한 변수,
초기화되지 않고, 함수가 호출될 때마다 증가한다.

        printf("[INSERT]\n");
        printf("Input Name: ");
        gets_s(name, sizeof(name));
        printf("Input Tel. Number: ");
        gets_s(telephoneNumber, sizeof(telephoneNumber));
        printf("Input Email Address: ");
        gets_s(emailAddress, sizeof(emailAddress));

        appendMember(head, name, telephoneNumber, emailAddress, memberOrder);
        resetOrder(head, 1);

        // 주소록의 크기를 제한하는 기능
        // 마지막 멤버의 order 가 10 보다 크면 해당 멤버를 삭제하고, 안내 메시지를 출력한다.
        int maxMemberOrder = 10;
        MEMBER* cur = head;
        while (cur->next != NULL) {
                cur = cur->next;
        }
        if (cur->order > maxMemberOrder) {
                printf("Telephone book can save maximum %d members\n",
maxMemberOrder);
                deleteTargetMember(cur);
        }
}
```