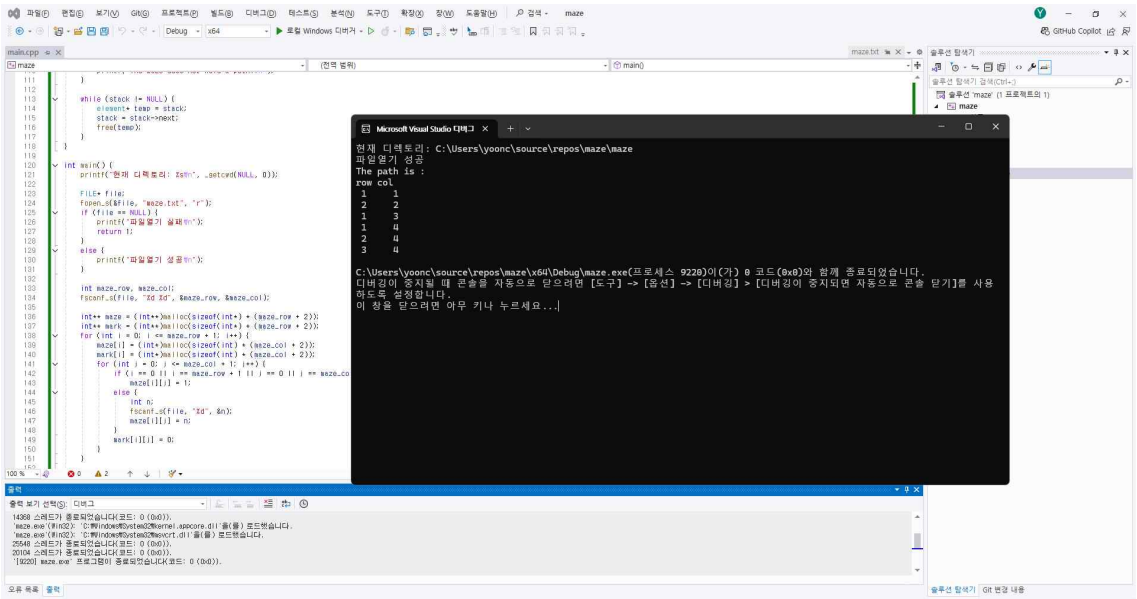
과제 1 보고서 20241630 윤찬영

기본적인 알고리즘 : maze에서 갈 수 있는 곳은 0, 갈 수 없는 벽은 1로 구성되어있고 mark는 이미 갔던 길을 표기하여 되돌아가지 않게 설정하는 것
m * p 미로를 풀기 위해 m+2 * p+2 maze를 만들어 외곽은 벽인 1로 설정하고 1,1에서 입구로 출발한다. maze와 같은 크기의 mark를 만들고 0으로 초기화한 후 이미 왔던 길에 포함되는 입구는 1,1로 기본설정한다.




```c
#include <stdio.h>
#include <direct.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct element {
//stack에 해당하는 구조체로 좌표와 이동 경로(dir)로 구성된다. (1,1,3) 이면 1,2를 가르키게 되는 것이다
    short int row;
    short int col;
    short int dir;
    struct element* next;
} element;

typedef struct { //stack의 dir와 관련된 것으로 0~7까지 북, 북동 ~ 북서 로 구성된다.
    short int vert;
    short int horiz;
} offsets;

offsets move[8] = {
```

```
    {-1, 0},   // N
    {-1, 1},   // NE
    {0, 1},    // E
    {1, 1},    // SE
    {1, 0},    // S
    {1, -1},   // SW
    {0, -1},   // W
    {-1, -1}   // NW
};

void push(element* top, element new_em) { //stack에 push하는 함수로 기존 start(top)에서 맨 끝까지 while문으로 이동한 후 맨끝의 next를 새로운 element로 연결하여 linked list를 연결한다.
    element* new_node = (element*)malloc(sizeof(element));
    *new_node = new_em;
    new_node->next = NULL;

    while (top->next != NULL)
        top = top->next;

    top->next = new_node;
}

element pop(element* top) {
//pop 함수는 linked list의 맨끝 element를 연결해제하고 그 element를 반환하는 함수이다. push와 마찬가지로 start(top)에서 맨끝으로 while로 이동하고 오른쪽에서 2번째 list의 next를 NULL로 함으로서 연결을 끊고 그 값을 리턴하고 free하여 메모리를 정리한다.
    if (top->next == NULL) {
        printf("스택이 비어 있습니다 (pop 실패)\n");
        exit(1); // 안전하게 종료
    }

    element* prev = top;
    element* curr = top->next;

    while (curr->next != NULL) {
        prev = curr;
        curr = curr->next;
    }

    element ret = *curr;
    prev->next = NULL;
    free(curr);
    return ret;
```

```
}

void path(int** maze, int** mark, int EXIT_ROW, int EXIT_COL) {
//maze를 탈출하는 함수
    int row, col, nextRow, nextCol, dir;
    bool found = false;

    element* stack = (element*)malloc(sizeof(element));
    stack->row = -1; stack->col = -1; stack->dir = -1;
    //stack의 맨 처음은 dummy list로 만들어 -1로 초기화한다.
    stack->next = NULL;
    element start = { 1, 1, 0, NULL };
    mark[1][1] = 1;
    push(stack, start);

    element position;

    while (stack->next != NULL && !found) {
//출구를 찾거나 길이 없어 dummy stack밖에 안남았을 때까지 반복한다
        position = pop(stack);
//dir가 8이상이 되었을 때 실행하여 pop을 하여 전 상태로 돌아가는 역할을 한다

        row = position.row; col = position.col; dir = position.dir;

        while (dir < 8 && !found) {
//출구를 찾거나 dir가 8미만 일 때까지 반복 dir가 8이상이란 것은 동서남북 어디로도 이동할 수 없다는 것이다
            nextRow = row + move[dir].vert;
            nextCol = col + move[dir].horiz;

            if (nextRow == EXIT_ROW && nextCol == EXIT_COL) { //출구를 찾았는가?
                found = true;
            }
            else if (!maze[nextRow][nextCol] && !mark[nextRow][nextCol]) {
//이동할 위치가 maze의 벽이 아닌가 and mark가 1이 아닌가 즉 이미 왔던 길이 아닌가 둘 다 0 즉 이동할 수 있다면
                mark[nextRow][nextCol] = 1;
                position.row = row; position.col = col;
                position.dir = ++dir;
                push(stack, position);

                row = nextRow; col = nextCol; dir = 0;
//이동 한 후 push, dir 초기화 row col은 이동한 row col로 설정
```

```
                }
                else {
                    dir++;
//이동할 수 없다면 다른 dir
                }
            }
        }

        if (found) {
            printf("The path is : \n");
            printf("row col \n");

            for (element* p = stack->next; p != NULL; p = p->next)
                printf("%2d %5d\n", p->row, p->col);

            printf("%2d %5d\n", row, col);
            printf("%2d %5d\n", EXIT_ROW, EXIT_COL); //왔던 길 출력
        }
        else {
            printf("-1\n“); //못 찾았으면 -1 출력
        }

        while (stack != NULL) { //stack 메모리 정리
            element* temp = stack;
            stack = stack->next;
            free(temp);
        }
}

int main() {
    printf("현재 디렉토리: %s\n", _getcwd(NULL, 0));

    FILE* file;
    fopen_s(&file, "maze.txt", "r"); //maze.txt 파일 열기
    if (file == NULL) {
        printf("파일열기 실패\n");
        return 1;
    }
    else {
        printf("파일열기 성공\n");
    }

    int maze_row, maze_col;
```

```
    fscanf_s(file, "%d %d", &maze_row, &maze_col); // 첫 줄 = maze의 row, col

    int** maze = (int**)malloc(sizeof(int*) * (maze_row + 2));
//maze의 테두리 벽을 포함한 동적할당
    int** mark = (int**)malloc(sizeof(int*) * (maze_row + 2));
//mark도 동적할당
    for (int i = 0; i <= maze_row + 1; i++) {
        maze[i] = (int*)malloc(sizeof(int) * (maze_col + 2));
        mark[i] = (int*)malloc(sizeof(int) * (maze_col + 2));
        for (int j = 0; j <= maze_col + 1; j++) {
            if (i == 0 || i == maze_row + 1 || j == 0 || j == maze_col + 1)
                maze[i][j] = 1; //테두리는 1로 초기화
            else {
                int n;
                fscanf_s(file, "%d", &n); //받은 maze.txt 값을 maze에 넣기
                maze[i][j] = n;
            }
            mark[i][j] = 0;
        }
    }

    fclose(file);

    path(maze, mark, maze_row, maze_col);

    for (int i = 0; i <= maze_row + 1; i++) {
        free(maze[i]);
        free(mark[i]);
    }
    free(maze);
    free(mark);

    return 0;
}
```