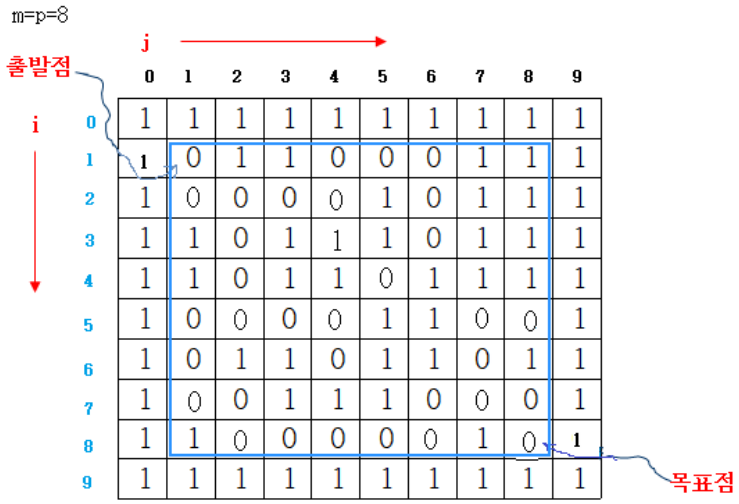
과제 2: Maze 경로 탐색

본 과제는 maze 환경에서 어떤 셀(start)에서 출발하여 어떤 다른 셀(destination)까지 갈 수 있는 경로를 탐색하는 프로그램을 개발하는 것을 목표로 한다. 스택의 구현은 반드시 연결스택을 사용하도록 한다.

[1] 과제 배경

- Maze: grid 기반 maze 환경에서 경로를 찾는 문제는 탐색(search)의 대표적인 문제이다. 우리는 m X p 사이즈의 grid를 가정한다. 이 안의 각 셀은 0 또는 1 을 가진다. 0 은 들어갈 수 있는(open) 셀, 1 은 막힌(closed) 셀을 나타낸다. 본 문제에서는 각 셀에서 8 가지 방향에 따라 옆 셀로의 이동을 시도할 수 있다고 가정한다. 프로그래밍 편의를 위해 우리는 m X p 크기의 grid 둘레를 1 을 가지는 boundary 셀들로 둘러 싼다. 아래 그림은 m=8, p=8 인 경우의 maze를 보여 준다(boundary를 포함하면 10 X 10 의 행렬). 이 maze를 위한 변수 선언은 다음과 같다(전역변수 이용).

```
int maze[m + 2][p + 2];  // 전역변수로 선언함. m, p 는 상수로 선언함.
```



	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	0	0	0	1	1	1
2	1	0	0	0	0	1	0	1	1	1
3	1	1	0	1	1	1	0	1	1	1
4	1	1	0	1	1	0	1	1	1	1
5	1	0	0	0	0	1	1	0	0	1
6	1	0	1	1	0	1	1	0	1	1
7	1	0	0	1	1	1	0	0	0	1
8	1	1	0	0	0	0	0	1	0	1
9	1	1	1	1	1	1	1	1	1	1

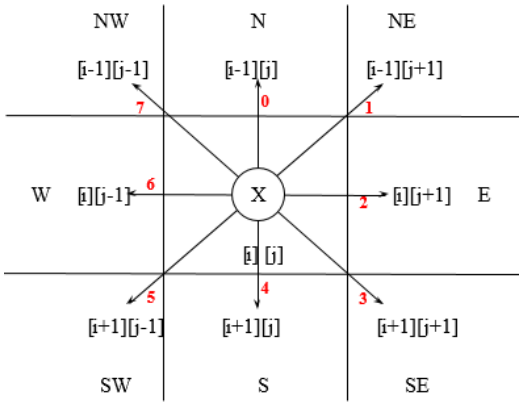
- 이동 방향 순서: 현재 방문하고 있는 셀에서 옆 셀로 이동을 위한 8 방향은 탐색 과정에서 다음 순서로 시도한다고 가정한다:

north(0), north-east(1),

east(2), south-east(3),

south(4), south-west(5),

west(6), north-west(7).



8 possible of moves from cell x

- 탐색 작업: 탐색 요청은 maze 내의 start cell [si, sj] 과 destination cell [di, dj] 을 지정하는 것으로 한다(본 실험에서 좌표 형식은 [row, column] 형식 이용). 이 요청에 따라 알고리즘은 시작셀 [si, sj]에서 목표셀 [di, dj]까지 갈 수 있는 경로 하나를 찾아서 출력한다 (경로는 순서대로 거치는 모든 셀의 좌표 리스트로 표시함).

- 탐색 기법: 많이 사용되는 기본적인 탐색 기법으로는 깊이우선 탐색(DFS; depth-first search) 과 넓이우선 탐색(BFS; breadth-first search)이 있다. 전자는 스택(stack), 후자는 큐(queue)를 이용하여 구현할 수 있다. 본 과제에서는 교과서에

서 소개한 것처럼 스택을 이용하는 깊이우선(DFS) 기법을 이용하도록 한다. DFS 에서는 탐색과정에서 도달한 현재 셀에서 여러 방향으로 이동이 가능할 때 그 중 번호가 빠른 방향으로 이동한다. 이때 이동 직전에 방문한 셀(과 다음 방향)은 스택에 저장한다. 새로 도달한 셀에서도 같은 방식으로 다음 셀로 이동한다. 어떤 셀에서 이동을 시도할 때 모든 방향이 실패했을 경우에는 이 셀로 오게 된 직전의 셀로 돌아 가야 한다. 이를 backtracking 이라 한다. 즉 더 이상 시도할 방향이 없는 셀에서 backtracking 이 일어나게 된다. backtracking의 구현은 스택에서 한 셀을 스택에서 pop 하여 그 셀로 돌아 가서 (저장한) 다음 방향부터 시도하는 것으로 달성한다.

DFS 방식은 아무 경로든 가리지 않고 먼저 발견되는 경로를 결과로 출력한다 (주: BFS 는 최단 경로를 찾아 줌). 따라서 거리가 먼 경로를 찾아 주는 경우도 있다. (자세한 동작 방식은 textbook 참고.)

- 스택의 구현: 본 과제에서는 스택의 구현을 위해 배열(array) 대신 linked list 를 사용하도록 한다 (연결스택 사용). 변수 top 은 탐색 함수 path 의 지역변수로 선언한다. 스택 내에는 시작셀에서 현재 방문 중인 셀의 직전 셀까지 거쳐온 셀들이 순서대로 저장되어 있다.

- mark 정보: path 탐색과정에서 이미 방문한 셀들이 무엇인지 알아야 할 필요가 있다. 이를 위해 우리는 행렬 maze 와 같은 사이즈의 행렬 mark 를 이용한다. 매번 탐색 작업 직전에 mark 의 모든 셀 값은 0 으로 한다. 탐색과정에서 셀이 방문되면 셀의 mark 값을 1 로 변경한다.

```
int mark[m + 2][p + 2];   // 전역변수로 선언함.
```

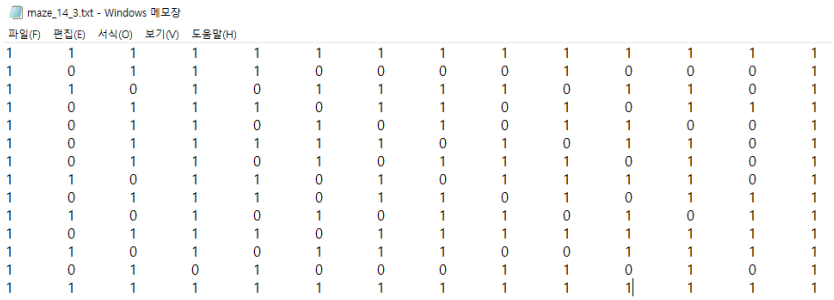
[2] 프로그램 개발 관련 사항

개발할 프로그램의 진행 순서는 다음과 같다.

(a) maze 를 아래 그림과 같이 준비한다. 파일은 (m+2) X (p+2) 사이즈의 maze의 각 셀의 정보를 가진다. 이 정보를 텍스트 파일에 넣는다. 각 숫자 사이는 공백문자를 넣는다.

main 함수는 맨 첫 부분에서 이 텍스트 파일로 부터 maze 데이터를 읽어 들인다. (fopen, fscanf 이용).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	0	1	1	1	0	0	0	0	1	0	0	0	1
3	1	1	0	1	0	1	1	1	1	0	1	1	0	1
4	1	0	1	1	1	0	1	1	0	1	0	1	1	1
5	1	0	1	1	0	1	0	1	0	1	1	0	0	1
6	1	0	1	1	1	1	1	0	1	0	1	1	0	1
7	1	0	1	1	0	1	0	1	1	1	0	1	0	1
8	1	1	0	1	1	0	1	0	1	1	1	1	0	1
9	1	0	1	1	1	0	1	1	0	1	0	1	1	1
10	1	1	0	1	0	1	0	1	1	0	1	0	1	1
11	1	0	1	1	1	0	1	1	1	1	1	1	1	1
12	1	1	0	1	0	1	1	1	0	0	1	1	1	1
13	1	0	1	0	1	0	0	0	1	1	0	1	0	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1



(주: 본 과제에서는 이 maze 데이터를 이용할 것을 권장함.)

(b) main 함수는 아래 예와 같이 탐색작업 수행 loop(반복문)를 수행한다.

- 각 iteration 마다 콘솔 창에 좌표 입력 프롬프트을 내보낸다.

- 이 요청에 시작점의 좌표(row, col), 목표점의 좌표(row, col)을 구성하는 4 개의 정수를 (각 수 사이는 공백문자로 구분) 를 넣는다: sy sx dy dx (아래 코드 참고)

```
printf("출발점, 목표점은?(4개의 정수; 종료는 Control+D Enter): ");
res = scanf("%d%d%d%d", &starty, &startx, &endy, &endx);
if (res != 4) break;
```

- 입력으로 4 개의 수 대신 control+D, enter 의 두 글자를 치면 프로그램을 종료한다.

- main 은 4 개의 좌표 정보를 넘기면서 함수 path 를 호출하여 탐색을 수행한다. 아래 매개변수에서 [sy,sx] 가 출발셀, [dy,dx] 가 목표셀임.

```
void path(int sy, int sx, int dy, int dx) { ..... }
```

  이 함수 path 의 수행 과정은 다음과 같다.

  . 먼저 출발셀, 목표셀 모두 open 셀 즉 maze 값이 0 이 아니면 탐색을 거부하고 바로 종료한다.

  . 빈(empty) 스택에 출발셀 정보 [sy,sx, dir = 0] 를 넣는다.

  . mark 정보를 초기화 한다 (모든 셀의 mark 값으로 0 을 넣는다).

  . 출발점 셀의 mark 를 1 로 변경한다. (출발점을 항상 제일 먼저 반드시 방문하므로.)

  . 스택에서 출발셀을 꺼내서 탐색 작업을 수행한다(저장된 dir 방향부터 시도). 이 탐색 과정은 2 개의 중첩된 while 문을 사용함.)

  . 탐색 완료 후 탐색 결과를 출력한다. 출발셀에서 현재셀 직전의 셀까지의 경로는 스택에 저장되어 있다. 이 정보와 현재셀 그리고 목표셀 정보를 출력한다. 만약 경로가 존재하지 않는다면 "경로없음"을 출력한다.

[3] 매번의 탐색명령에 따른 출력할 정보

매번의 경로 탐색작업의 중간 및 완료 후 에 다음 사항을 출력하도록 한다. 못 찾은 경우는 "경로없음"을 출력한다.

(a) 탐색 상황 실시간 표시(위 콘솔창 그림 참고). 여기는 개발하지 않아도 감점 없음. 개발하면 부가점 제공할 것임.

  먼저 매 탐색 명령 수행 전에 먼저 행렬 maze 를 화면에 출력한다(0/1. 숫자 사이에 공백문자 없음).

  그 뒤 셀을 방문할 때마다 그 셀의 위치를 * 로 덮어 쓴다(그리고 1 초 동안 아무 일도 안하고 멈춘다).

  (단, 시작셀은 '>' 글자, 목표셀은 '<' 로 표시한다.)

  backtracking 이 일어나면 즉시 그 셀에 글자 '$' 를 덮어 쓴다 (그 후 1초 멈춤).

  이렇게 하면 탐색 진행 과정을 실시간으로 볼 수 있다.

(b) 찾아낸 경로를 시작셀부터 목표셀까지 좌표를 순서대로 표시한다. 예를 들면 다음과 같은 형식을 취한다.

  한 줄에 최대 6개이 셀 좌표만 출력한다.

  [1,1] [1,2] . . . ...[3,7]

  [9,1] ..... [8,8]

  (주: 이 작업을 위해서는 스택의 연결리스트를 reverse 해주는 처리가 필요함.)

(c) 찾은 경로의 길이 (시작셀부터 목표셀까지의 찾은 경로에 포함된 셀의 수)를 출력.

(d) 탐색 중에 backtracking 이 일어 난 셀의 수를 출력.

주: 실행화면(콘솔창)의 한 예는 다음과 같다.

[4] 제출물:

- 소스프로그램(.cpp 화일)
- 실행화면(캡쳐이미지)
- 보고서 (개발 중에 겪은 어려웠던 점, 이를 극복하는 과정 및 수단, 받은 도움, 얻은 소득 등에 대한 소회를 기술한다).

[5] 아래는 프로그래밍 참고사항

- 상수, 타입선언, 전역변수들

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include <Windows.h>  //gotoxy() 함수를 사용하기 위한 헤더

#define m 12
#define p 12
#define Max_dir  8   // 총 가능한 이동방향의 수
#define Timestep 1000 // 시간단위는 ms 이므로 1000 이면 1 초를 멈추게 됨.

typedef struct Aoff {
    short int vert;
    short int horiz;
} offsets;

typedef struct  St_element {    // 스택에 저장할 데이터
    short int row;
    short int col;
    short int dir;
} ty_element;

typedef struct ListNode* listPointer;
typedef struct ListNode {       // 연결스택의 노드
    ty_element data;
    listPointer link;
} listNode;

//                            N      NE       E      SE       S       SW       W       NW
offsets move[Max_dir] = { {-1,0}, {-1,1},  {0,1},  {1,1},   {1,0},  {1,-1},  {0,-1}, {-1,-1} };

int maze[m + 2][p + 2];     // maze 행렬. 전역변수로 선언함.
int mark[m + 2][p + 2];     // mark 행렬
```

- 경로 탐색 함수 path:
  주의: 스택의 변수 top 은 함수 path 내에서 **지역변수**로 선언한다.

```c
//  미로 경로 탐색 함수.
//  [sy,sx] 에서 시작하여 [dy, dx] 에 도달하는 경로를 찾는다.
int path(int sy, int sx, int dy, int dx)
{
    listPointer top = NULL;  // 스택의 top 변수. 초기에는 NULL.
    int i, j, row, col, nextRow, nextCol, dir, basex, basey, ocount;
    int found, num_bktrack = 0, path_length = 0;
    int EXIT_ROW = dy, EXIT_COL = dx;  // 목표점.
    ty_element position;

    if (maze[sy][sx] == 1 || maze[dy][dx] == 1) {
        printf("입력오류: 출발점이나 목표점이 막힌 셀입니다.₩n");
        return 0;
    }

    // 먼저 미로를 화면에 그린다.
    CONSOLE_SCREEN_BUFFER_INFO presentCur;
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &presentCur);
    basex = presentCur.dwCursorPosition.X; // 베이스 좌표의 x.
    basey = presentCur.dwCursorPosition.Y; // 베이스 좌표의 y.

    for (i = 0; i < m + 2; i++) {
        for (j = 0; j < p + 2; j++) {
            gotoxy(j + basex, i + basey);
            printf("%1d", maze[i][j]);
        }
    }

    // 빈 스택에 출발점을 넣는다.(초기화)
    position.row = sy; position.col = sx;  position.dir = 0;
    push(&top, position);

    // mark 정보초기화. 바운더리를 제외한 내부 셀에 모두 0 (방문안함)으로 초기화한다.
    for (i = 0; i < m; i++) {
        for (j = 0; j < p; j++) {
            mark[1+i][1+j] = 0;
```

- 위 3-(a) 구현에 필요한 정보

(a) gotoxy 함수: 매개변수 x 에 원하는 셀의 column, y 에 그 셀의 row 정보를 주면서 호출한다. 콘솔창의 맨 좌상 모서리를 위치 [0,0] 으로 간주한 좌표 상에서 커서를 [y,x] 셀 위치에 놓는다(즉 수직으로 y 행, 수평으로 x 글자를 이동한 위치).

```
void gotoxy(int x, int y)
{
    COORD Pos = { x ,y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}
```

이 함수를 이용하여 원하는 셀 위치에 커서를 놓고, 다음 printf 문으로 거기에 원하는 문자 1 개를 덮어 쓸 수 있다.

(b) 기준점 찾기

maze 행렬을 화면에 표시하려면 먼저 기준점(base position)을 알아야 한다(즉 콘솔창에서 현재 커서의 위치를 알아내어 이를 기준점으로 하고자 한다). 매번의 탐색 명령마다 이 기준점 찾기 작업을 한번 수행하여야 한다.

```
CONSOLE_SCREEN_BUFFER_INFO presentCur;
GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &presentCur);
basex = presentCur.dwCursorPosition.X; // 기준점의 x 좌표
basey = presentCur.dwCursorPosition.Y;  // 기준점의 y 좌표
```

위 코드를 수행하면 기준점 [basey, basex] 를 구할 수 있다. 그 다음에 다음과 같은 코드를 수행하면 maze 를 콘솔창에 출력할 수 있다.

```
for (i = 0; i < m + 2; i++) {
        for (j = 0; j < p + 2; j++) {
            gotoxy(j + basex, i + basey);
            if (maze[i][j] == 1)
               ch = '1';
            else
               ch = '0';
            printf("%c", ch);
        }
}
```

(c) 컴퓨터 수행을 일정 시간 멈추기:

```
Sleep(Timestep);   // Timestep 은 상수로 단위는 milisecond. 1000 이면 1 초 멈춤.
```

(d) 탐색 작업 중에 maze 의 셀 위치에 글자 표시하기:

예를 들어 셀 (row, col) 로 탐색이 이동하였다 하자. 이 셀의 위치에 글자 * 를 표시하려면 다음 코드를 수행한다.

```
gotoxy(col + basex, row + basey);
printf("*");
```

(e) 이 프로그램에 필요한 include 파일들 :

```
#include<stdio.h>
#include<stdlib.h>
#include <Windows.h>  //gotoxy(),GetConsoleScreenBufferInfo() 등의 사용에 필요함.
```