

HW 3 Report

Stack (미로탐색문제)

자료구조 및 알고리즘 이해 (19-2, C070-1) / 이정원 교수님

201120696 최제현

문제분석 (요구사항)

- Stack을 활용한 미로 탐색 문제 프로그램 완성
 - 미로의 구조는 미리 정의된다.
 - 미로의 구조와 크기가 바뀌어도 동작 가능하다.
 - 현재 위치에서 이동 가능한 부분으로 이동한다.
 - 이동하기 전에 현재 위치를 Stack에 저장한 후 이동한다.
 - 미로 탐색 과정을 알 수 있게, 매 탐색과 이동 후 미로와 스택을 출력한다.
- Stack 자료구조, 사용된 미로탐색 알고리즘의 대한 이해
- 구현된 소스코드 분석 및 이해
- 실행 결과 캡처 및 결과 분석

사용한 자료구조/알고리즘의 설명

1. Stack 자료구조

특징

- Stack이라는 이름처럼 쌓아놓은 더미 자료구조를 의미한다.
- 후입선출(LIFO: Last-In First-Out): 가장 최근에 들어온 데이터가 가장 먼저 나간다.
- 입력과 역순의 출력이 필요한 경우 적용할 수 있다.
 - ex. 에디터에서 되돌리기(undo) 기능, 함수호출에서 복귀주소 기억
- 배열, 연결리스트를 이용해 Stack을 구현할 수 있다.
 - 배열 방식: 코드를 단순하고 쉽게 구현할 수 있다는 장점이 있으나, 배열의 크기를 예측하기 힘들어 충분히 잡았을 경우 사용하지 않는 메모리 증가로 메모리의 낭비가 심해질 수 있는 단점이 있다.
 - 연결리스트 방식: 실행 중 필요한 만큼만 노드를 동적 할당하여 메모리를 사용하여 메모리의 낭비가 적고 효율적인 장점이 있으나, 구현이 복잡하다는 단점이 있다.

소스코드의 스택 구현 방식 설명

소스코드상 스택의 구현 방식은 StackType, position 두 개의 구조체를 사용하여 구현하였다. StackType 구조체로 스택을 만들 수 있고, StackType 내에 position 배열인 stack변수와 배열에서의 가장 최근에 입력된 자료의 위치를 가리키는 top 변수를 갖고 있다. 단순하고 쉽게 구현하고자 연결리스트가 아닌 배열로 스택을 구현하였다.

```
17  
18 typedef struct position {  
19     int x;  
20     int y;  
21 }position;  
22  
23 typedef struct {  
24     position *stack[MAX_STACK_SIZE];  
25     int top;  
26 }StackType;
```

Position
: X좌표와 Y좌표를 가진 구조체

StackType
: Position 구조체를 저장하는 스택 구조체
공백상태 → top = -1
n개 차있는 상태 → top = n-1 (0부터 시작)

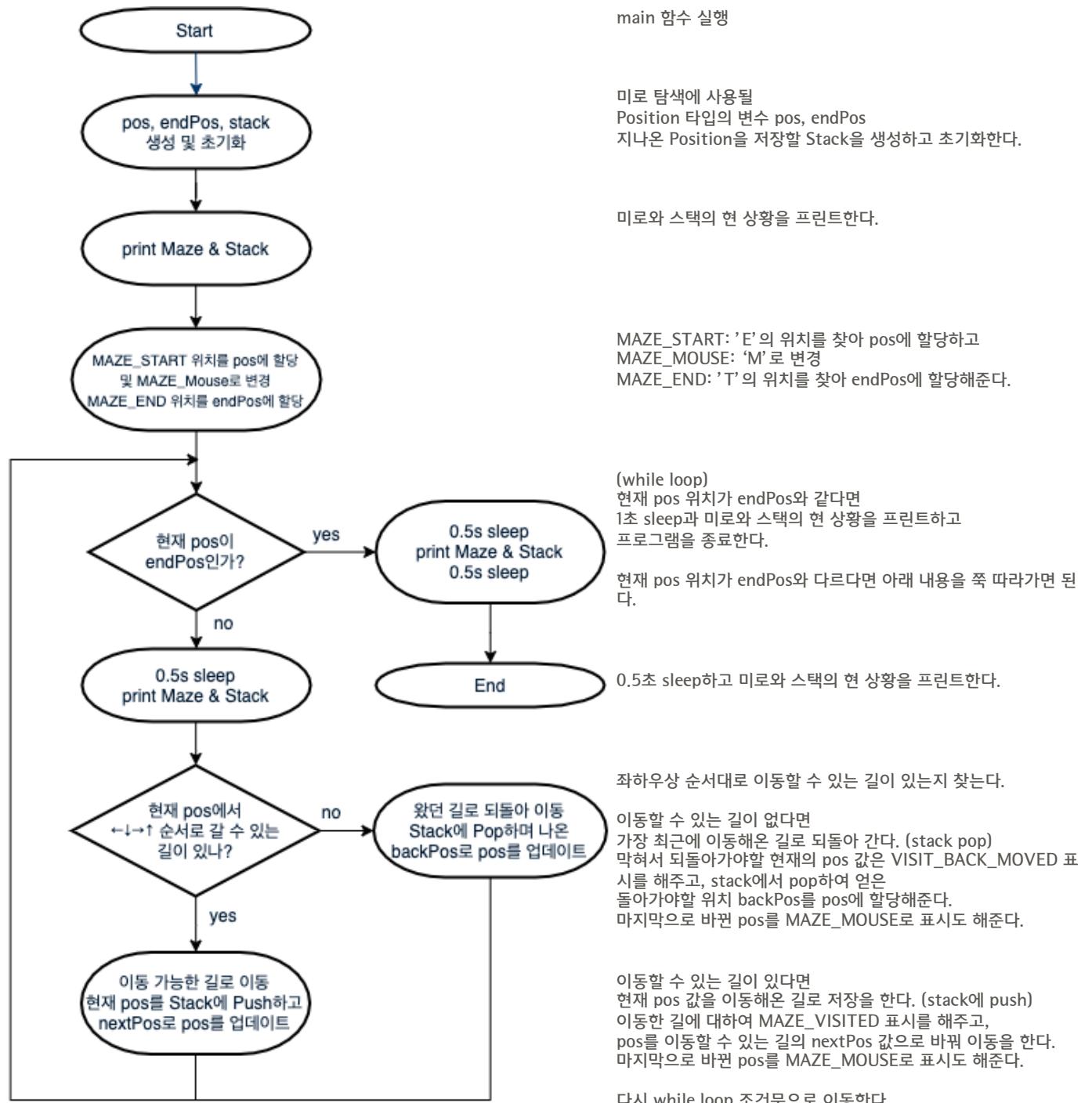
구현된 Stack의 ADT(Abstract Data Type)은 다음과 같다.

- 객체: n개의 position형의 요소들의 선형 리스트
- 연산:
 - init(s) ::= top 포인터를 -1로 만들어 리스트를 초기화한다.
 - is_empty(s) ::= 스택이 비어있는지 검사한다.
 - is_full(s) ::= 스택이 가득차있는지 검사한다.
 - push(s, item) ::= 스택의 맨 위에 요소 item을 추가한다.
 - pop(s) ::= 스택의 맨 위에 있는 요소를 삭제하며 반환한다.
 - peek(s) ::= 스택의 맨 위에 있는 요소를 삭제하지 않고 반환한다.
 - printStack(s) ::= 스택을 프린트한다.

2. 미로 탐색 알고리즘

구현한 알고리즘 설명

미로 탐색의 중요 알고리즘을 되도록 핵심적인 로직위주로 플로우차트를 그렸고, 설명하였다.



소스코드 분석

기존 제공된 코드

초기 선언 코드들

```
1 #include <stdio.h>
2 #include <Windows.h>
3
4 #define SLEEP_TIME 500
5
6 #define MAX_STACK_SIZE 20
7 #define MAZE_WIDTH 10
8 #define MAZE_HEIGHT 10
9
10 #define MAZE_ROAD ' '
11 #define MAZE_WALL '#'
12 #define MAZE_VISITED '.'
13 #define MAZE_BACK_MOVED 'X'
14 #define MAZE_MOUSE 'M'
15 #define MAZE_START 'E'
16 #define MAZE_END 'T'
17
18 typedef struct position {
19     int x;
20     int y;
21 }position;
22
23 typedef struct {
24     position *stack[MAX_STACK_SIZE];
25     int top;
26 }StackType;
27
28 char maze[MAZE_HEIGHT][MAZE_WIDTH] = {
29     { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' },
30     { 'E', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
31     { '#', '#', '#', ' ', '#', ' ', '#', '#', '#', '#' },
32     { '#', '#', '#', ' ', ' ', ' ', '#', ' ', '#', '#' },
33     { '#', '#', '#', '#', ' ', '#', ' ', ' ', '#', '#' },
34     { '#', '#', '#', '#', ' ', '#', ' ', ' ', '#', '#' },
35     { '#', '#', '#', '#', ' ', '#', ' ', ' ', '#', '#' },
36     { '#', '#', '#', '#', ' ', '#', ' ', ' ', '#', '#' },
37     { '#', '#', '#', '#', ' ', '#', ' ', ' ', '#', '#' },
38     { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' }
39 };
40
41 void init(StackType *s);
42 int isEmpty(StackType *s);
43 int isFull(StackType *s);
44 void push(StackType *s, position *item);
45 position* pop(StackType *s);
46 position* peek(StackType *s);
47
48 position* makePosition(int x, int y);
49
50 void printMaze();
51 void printStack(StackType *s);
52 position* positionSearch(position pos);
53 void positionMove(StackType *s, position *pos, position *nextPos);
54 void positionBackMove(StackType *s, position *pos);
55 int isInMaze(int x, int y);
```

사용 라이브러리 include

전처리 매크로 정의

공통 SLEEP_TIME 설정 값

배열로 구현한 스택의 MAX_STACK_SIZE 값

MAZE width, height 크기 정보, 심볼 값

struct type 설정

위치 정보를 표시할 자료형 position

position 정보를 배열로 저장하여 스택으로 사용되는 자료형 StackType

전역 변수 maze로 미로 형태 할당

함수 프로토타입 선언

StackType 스택의 연산 함수 init, isEmpty, isFull, push, pop, peek

position 구조체를 만들어주는 makePosition

미로 탐색 상황을 보기 위한 프린트 함수 printMaze, printStack

미로 탐색을 위한 함수 positionSearch, positionMove, positionBackMove

미로 안에 있는지 확인하기 위한 함수 isInMaze

main 함수

```
50
57 int main(void)
58 {
59     int itrX, itrY;
60     position pos, endPos;
61     StackType stack;
62     position *tempPos;
63
64     init(&stack);
65
66     printMaze();
67     printStack(&stack);
68
69     for (itrY = 0; itrY < MAZE_HEIGHT; itrY++) {
70         for (itrX = 0; itrX < MAZE_WIDTH; itrX++) {
71             if (maze[itrY][itrX] == MAZE_START) {
72                 pos.x = itrX;
73                 pos.y = itrY;
74                 maze[itrY][itrX] = MAZE_MOUSE;
75             }
76             if (maze[itrY][itrX] == MAZE_END) {
77                 endPos.x = itrX;
78                 endPos.y = itrY;
79             }
80         }
81     }
82
83     while (pos.x != endPos.x || pos.y != endPos.y) {
84         Sleep(SLEEP_TIME);
85         printMaze();
86         printStack(&stack);
87
88         tempPos = positionSearch(pos);
89         if (tempPos)
90             positionMove(&stack, &pos, tempPos);
91         else
92             positionBackMove(&stack, &pos);
93     }
94     Sleep(SLEEP_TIME);
95     printMaze();
96     printStack(&stack);
97     Sleep(SLEEP_TIME);
98
99     return 0;
100 }
```

앞서 미로 탐색 알고리즘 플로우차트에서 설명한 내용과 같으므로 따로 중복된 설명은 하지 않겠다.
추가적으로 tempPos 임시 position 타입의 변수를 이용해 받아온 값을 이용해 move할지, backMove할지를 결정해준다.

Stack 연산 함수들

```
104 void init(StackType *s) {
105     s->top = -1;
106 }
107
108 int is_empty(StackType *s) {
109     return (s->top == -1);
110 }
111
112 int is_full(StackType *s) {
113     return (s->top == (MAX_STACK_SIZE - 1));
114 }
115
116 void push(StackType *s, position *item) {
117     if (is_full(s)) {
118         fprintf(stderr, "스택 포화 에러\n");
119         return;
120     }
121     else s->stack[++(s->top)] = item;
122 }
123
124 position* pop(StackType *s) {
125     if (is_empty(s)) {
126         fprintf(stderr, "스택 공백 에러\n");
127         exit(1);
128     }
129     else return s->stack[(s->top)--];
130 }
131
132 position* peek(StackType *s) {
133     if (is_empty(s)) {
134         fprintf(stderr, "스택 공백 에러\n");
135         exit(1);
136     }
137     else return s->stack[s->top];
138 }
```

배열을 이용한 스택이므로 StackType 구조체 인스턴스의 top 변수를 이용해 스택의 현재 top 위치를 표현한다.
또한, 함수 전체적으로 포인터 변수로 매개변수를 받아 같은 주소를 바라보는 스택의 값을 직접 수정하도록 하였다.

init: 초기엔 비어있으므로 -1 값을 할당해준다.

is_empty: top이 -1이면 비어있는 상황이다.

is_full: top이 MAX_STACK_SIZE-1 위치에 있다면 더 이상 배열에 넣을 수 있는 공간이 없다.

push: 스택이 먼저 가득찼는지 확인하고, 가득찼다면 포화 에러 메세지를 프린트하고, push 동작을 무시한다. 공간이 있으면, 현재의 top 위치 값은 먼저 1 더하고 해당 top 위치에 item을 넣는다.

pop: 스택이 먼저 비어있는지 확인하고, 비어있다면 공백 에러 메세지를 프린트하고, 프로그램을 종료시킨다. item이 있다면 현재의 top 위치의 item 값을 반환하고, top 위치 값을 1을 빼준다. (실제 stack 배열 상에서는 pop 된 값이 존재한다.)

peek: pop과 기능이 거의 동일하지만, 현재 스택의 top 위치 값의 변화는 없다. 단순히 top item에 뭐있나 보기 위한 용도

makePosition

```
140 position* makePosition(int x, int y) {
141     position* pos;
142     pos = (position*)malloc(sizeof(position));
143     if (!pos)
144     {
145         fprintf(stderr, "position 할당 에러\n");
146         exit(1);
147     }
148     pos->x = x;
149     pos->y = y;
150
151     return pos;
152 }
153 }
```

입력 x, y 값을 가지는 position 동적 할당 변수를 반환하는 함수
동적할당이 실패하면 할당 에러 메세지가 프린트되고, 프로그램이 종료된다.

print 함수들

```
155 void printMaze() {
156     int itrY, itrX;
157
158     system("cls");
159     for (itrY = 0; itrY < MAZE_HEIGHT; itrY++) {
160         for (itrX = 0; itrX < MAZE_WIDTH; itrX++) {
161             printf(" %c ", maze[itrY][itrX]);
162         }
163         printf("\n");
164     }
165 }
166
167 void printStack(StackType *s) {
168     int itr;
169     printf("%4s|(%3s,%3s)|\n", "", " X ", " Y ");
170     for (itr = MAX_STACK_SIZE - 1; itr > s->top; itr--) {
171         printf("%4d|(%3s,%3s)|\n", itr, "", "");
172         for (; itr >= 0; itr--)
173             printf("%4d|(%3d,%3d)|\n", itr, s->stack[itr]->x, s->stack[itr]->y);
174     }
175 }
```

printMaze: 콘솔을 clear하고, MAZE_HEIGHT & MAZE_WIDTH 값 만큼 이중 for-loop를 돌리며 Maze를 캐릭터 하나하나 프린트한다.
printStack: stack을 저장된 값을 프린트한다. itr 변수를 공통으로 사용하는 for-loop이 두개 있는데, 첫번째 for-loop은 MAX_STACK_ZIE - 1부터 1씩 빼가면서 현재 stack의 top + 1 까지 비어있는 스택을 프린트하고, 두번째 for-loop은 0번까지 stack의 x, y 값을 반복하여 출력한다.

isInMaze 함수

```
235
236 int isInMaze(int x, int y) {
237     if (x < 0 || x > MAZE_WIDTH)
238         return 0;
239     if (y < 0 || y > MAZE_HEIGHT)
240         return 0;
241     return 1;
242 }
```

입력 받은 x, y 각각 0 이상 MAZE_WIDTH, MAZE_HEIGHT 이하에 들어가는지 확인하여 미로 범위를 벗어나지 않았는지 확인한다.

본인 작성 코드

positionSearch 함수

```
176 position* positionSearch(position pos) {
177     int x, y;
178
179     x = pos.x - 1;
180     y = pos.y;
181     if (isInMaze(x, y)) {
182         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
183             return makePosition(x, y);
184         }
185     }
186
187     x = pos.x;
188     y = pos.y + 1;
189     if (isInMaze(x, y)) {
190         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
191             return makePosition(x, y);
192         }
193     }
194
195     x = pos.x + 1;
196     y = pos.y;
197     if (isInMaze(x, y)) {
198         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
199             return makePosition(x, y);
200         }
201     }
202
203     x = pos.x;
204     y = pos.y - 1;
205     if (isInMaze(x, y)) {
206         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
207             return makePosition(x, y);
208         }
209     }
210
211     return NULL;
212 }
```

우수법, 좌수법 등의 방법을 사용하는게 일반적이지만 이번 과제에서는 과제pdf 6p에 나와있는 방식대로 탐색을 진행하였다.

input으로 받은 현재 위치 pos에서 $\leftarrow \rightarrow \uparrow \downarrow$ 좌하우상 순서대로 maze 값을 확인해봤을 때

MAZE_ROAD 혹은 MAZE_END와 같으면 이동할 수 있는 길이므로 해당 position 변수를 makePosition으로 만들고 이를 반환한다.

좌하우상 전부 확인해봤는데도 이동할 수 있는 경로가 없다면 NULL을 반환한다.

positionMove 함수

```
214 void positionMove(StackType *s, position *pos, position *nextPos) {
215     push(s, makePosition(pos->x, pos->y));
216     maze[pos->y][pos->x] = MAZE_VISITED;
217
218     pos->x = nextPos->x;
219     pos->y = nextPos->y;
220     maze[pos->y][pos->x] = MAZE_MOUSE;
221
222     free(nextPos);
223 }
```

이동하기 전에 먼저 stack에 과거 이동해온 경로를 순차적으로 저장하는 방식을 취하고 있으므로

현재 pos 값을 makePosition 함수를 이용해 copy하여 스택 s에 push한다.

(pos 직접 push에 넣지 않은 건, 프로그램 구조상 main에서도 공통으로 쓰이는 pos 포인터 변수가 스택 내에 그대로 저장되어 값이 계속 변화하기 때문이다.) 그리고, push한 pos의 maze 위치 값을 MAZE_VISITED로 바꿔준다.

이동할 위치를 갖고있는 nextPos 포인터 변수의 값을 현재 위치를 가리키는 pos 포인터 변수 값 x, y에 각각 할당해주고 nextPos에 의해 변경된 pos의 maze 위치 값을 MAZE_MOUSE로 바꿔준다.

nextPos는 동적할당된 변수이고, 더 이상 사용하지 않으므로 free로 메모리 해제하여 leak이 발생하지 않도록 한다.

positionBackMove 함수

```
225 void positionBackMove(StackType *s, position *pos) {
226     maze[pos->y][pos->x] = MAZE_BACK_MOVED;
227
228     position* backPos = pop(s);
229     pos->x = backPos->x;
230     pos->y = backPos->y;
231     maze[pos->y][pos->x] = MAZE_MOUSE;
232
233     free(backPos);
234 }
```

현재 pos는 더이상 갈 길이 없는 막다른 길이므로, pos의 maze 위치 값을 MAZE_BACK_MOVED로 바꿔준다.

스택 s를 pop하여 스택에서도 최근 경로를 빼주면서, 되돌아가야할 position backPos를 반환값으로 넘겨 받는다.

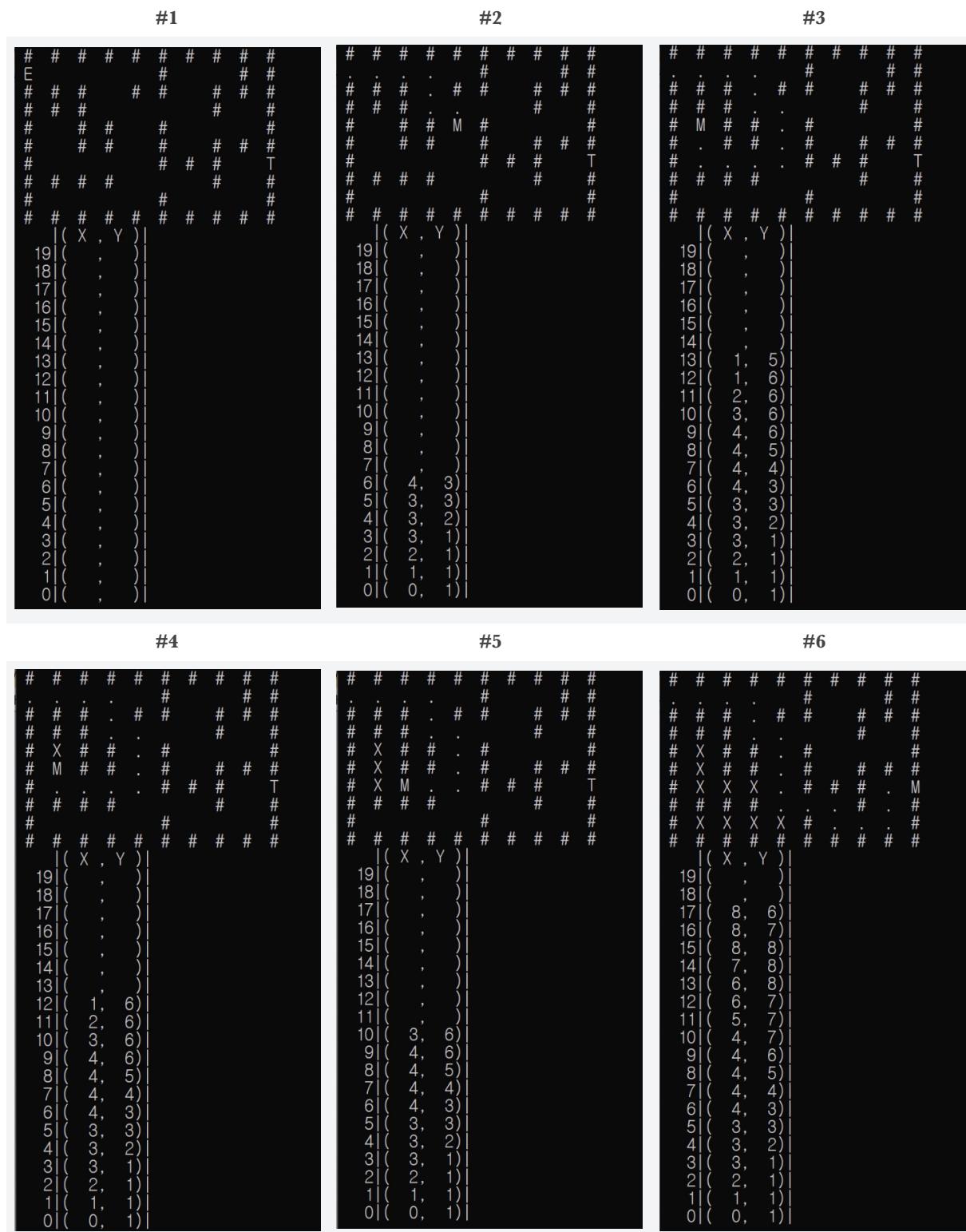
backPos 위치 값을 현재 위치를 가리키는 pos 포인터 변수 값 x, y에 각각 할당해주고

backPos에 의해 변경된 pos의 maze 위치 값을 MAZE_MOUSE로 바꿔준다.

backPos는 동적할당된 변수이고, 더 이상 사용하지 않으므로 free로 메모리 해제하여 leak이 발생하지 않도록 한다.

출력된 결과화면 캡쳐 및 결과에 대한 분석

10x10 미로 (MAX_STACK_SIZE = 20)



기본 제공된 10x10 미로(MAX_STACK_SIZE=20) 실행 콘솔 화면을 캡쳐하였다.

#2: 오른쪽 경로보다 아래 경로를 우선시하여 이동 (좌하우상 우선순위 적용)

#3, #4, #5: 막다른 길을 만나게 되면, stack을 pop하여 되돌아 이동

#6: 미로 탐색이 마무리 된 상태. 과제pdf 13p에 나와있는 결과와 동일한 것을 볼 수 있다.

12x12 미로 (MAX_STACK_SIZE = 30)

#1

```
# # # # # # # # # # # #
M # # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| .)|
13|(| .)|
12|(| .)|
11|(| .)|
10|(| .)|
9|(| .)|
8|(| .)|
7|(| .)|
6|(| .)|
5|(| .)|
4|(| .)|
3|(| .)|
2|(| .)|
1|(| .)|
0|(| .)|

#2

```
# # # # # # # # # # #
# . # . # # # # # #
# # # . M # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| .)|
13|(| .)|
12|(| .)|
11|(| .)|
10|(| .)|
9|(| .)|
8|(| .)|
7|(| .)|
6|(| .)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

#3

```
# # # # # # # # # # #
# . # . # # # # # #
# # # . M # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| .)|
13|(| 1, 5)|
12|(| 1, 6)|
11|(| 2, 6)|
10|(| 3, 6)|
9|(| 4, 6)|
8|(| 4, 5)|
7|(| 4, 4)|
6|(| 4, 3)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

#4

```
# # # # # # # # # # #
# . # . # # # # # #
# # X # # . # # # #
# X X # # . # # # #
# X X X # . # # # #
# X X X X # . # # #
# X X X X X # . # #
# X X X X X X # . #
# X X X X X X X # #
# X X X X X X X X #
# X X X X X X X X X #
# X X X X X X X X X X #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| 10, 6)|
24|(| 10, 7)|
23|(| 9, 7)|
22|(| 8, 7)|
21|(| 8, 8)|
20|(| 8, 9)|
19|(| 9, 9)|
18|(| 9, 10)|
17|(| 8, 10)|
16|(| 7, 10)|
15|(| 6, 10)|
14|(| 6, 9)|
13|(| 5, 9)|
12|(| 4, 9)|
11|(| 4, 8)|
10|(| 4, 7)|
9|(| 4, 6)|
8|(| 4, 5)|
7|(| 4, 4)|
6|(| 4, 3)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

#5

```
# # # # # # # # # # #
. # . # # # # # #
# # # . # # # # #
# X # # . # # # #
# X X # # . # # #
# X X X # # . # #
# X X X X # # . #
# X X X X X # # #
# X X X X X X # #
# X X X X X X X #
# X X X X X X X X #
# X X X X X X X X X #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| .)|
13|(| 5, 9)|
12|(| 4, 9)|
11|(| 4, 8)|
10|(| 4, 7)|
9|(| 4, 6)|
8|(| 4, 5)|
7|(| 4, 4)|
6|(| 4, 3)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

#6

```
# # # # # # # # # # #
. # . # # # # # #
# # # . M # # # #
# X # # X # # # #
# X X # # X # # #
# X X X # # X # #
# X X X X # # X #
# X X X X X # # X #
# X X X X X X # #
# X X X X X X X #
# X X X X X X X X #
# X X X X X X X X X #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| .)|
13|(| .)|
12|(| .)|
11|(| 8, 4)|
10|(| 7, 4)|
9|(| 6, 4)|
8|(| 6, 3)|
7|(| 5, 3)|
6|(| 4, 3)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

#7

```
# # # # # # # # # # #
. # . # # # # # #
# # # . M X X # #
# X # # X # # # #
# X X # # X # # #
# X X X # # X # #
# X X X X # # X #
# X X X X X # # X #
# X X X X X X # #
# X X X X X X X #
# X X X X X X X X #
# X X X X X X X X X #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| .)|
13|(| .)|
12|(| .)|
11|(| 8, 4)|
10|(| 7, 4)|
9|(| 6, 4)|
8|(| 6, 3)|
7|(| 5, 3)|
6|(| 4, 3)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

#8

```
# # # # # # # # # # #
. # . # . # . # #
# # # # X # X # X #
# X # # X # X # X #
# X X # # X # X # X #
# X X X # # X # X #
# X X X X # # X X #
# X X X X X # # X X #
# X X X X X X # # X X #
# X X X X X X X # #
# X X X X X X X X #
# X X X X X X X X X #
# X X X X X X X X X X #
|( X , Y )|
```

29|(| .)|
28|(| .)|
27|(| .)|
26|(| .)|
25|(| .)|
24|(| .)|
23|(| .)|
22|(| .)|
21|(| .)|
20|(| .)|
19|(| .)|
18|(| .)|
17|(| .)|
16|(| .)|
15|(| .)|
14|(| 10, 1)|
13|(| 9, 1)|
12|(| 8, 1)|
11|(| 7, 1)|
10|(| 6, 1)|
9|(| 6, 2)|
8|(| 6, 3)|
7|(| 5, 3)|
6|(| 4, 3)|
5|(| 3, 3)|
4|(| 3, 2)|
3|(| 3, 1)|
2|(| 2, 1)|
1|(| 1, 1)|
0|(| 0, 1)|

커스텀으로 만든 12x12 미로(MAX_STACK_SIZE=30) 실행 콘솔 화면을 캡쳐하였다.

#2: 오른쪽 경로보다 아래 경로를 우선시하여 이동 (좌하우상 우선순위 적용)

#3 ~ #7: 막다른 길을 만나게 되면, stack을 pop하여 되돌아 이동

#8: 미로 탐색이 마무리 된 상태, 수 많은 위치를 이동하였지만 결국 스택에 남는 것은 15개의 위치 값이다.

배열을 이용하는 스택을 만들었기에 탐색에 많은 경로 저장이 필요한 미로를 적용할 때에는 MAX_STACK_SIZE 크기를 변경해줘야하는 번거로움이 있었다.
연결 리스트를 이용한 스택을 만들어 적용하면, 보다 편리하고 안전하게 미로를 변경하여 탐색을 진행할 수 있을 것이다.

전체 소스코드

```
1 #include <stdio.h>
2 #include <Windows.h>
3
4 #define SLEEP_TIME 500
5
6 #define MAX_STACK_SIZE 20
7 #define MAZE_WIDTH 10
8 #define MAZE_HEIGHT 10
9
10 #define MAZE_ROAD      ' '
11 #define MAZE_WALL      '#'
12 #define MAZE_VISITED   '.'
13 #define MAZE_BACK_MOVED 'X'
14 #define MAZE_MOUSE     'M'
15 #define MAZE_START     'E'
16 #define MAZE_END       'T'
17
18 typedef struct position {
19     int x;
20     int y;
21 }position;
22
23 typedef struct {
24     position *stack[MAX_STACK_SIZE];
25     int top;
26 }StackType;
27
28 char maze[MAZE_HEIGHT][MAZE_WIDTH] = {
29     { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' },
30 { 'E', ' ', ' ', ' ', ' ', '#', ' ', ' ', '#', '#' },
31 { '#', '#', '#', ' ', '#', '#', ' ', '#', '#', '#' },
32 { '#', '#', '#', ' ', ' ', ' ', '#', ' ', '#', '#' },
33 { '#', ' ', '#', '#', ' ', '#', ' ', ' ', '#', '#' },
34 { '#', ' ', '#', '#', ' ', '#', ' ', '#', '#', '#' },
35 { '#', ' ', ' ', ' ', ' ', '#', '#', '#', ' ', 'T' },
36 { '#', '#', '#', '#', ' ', ' ', '#', ' ', '#', '#' },
37 { '#', ' ', ' ', ' ', ' ', '#', ' ', ' ', '#', '#' },
38 { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' }
39 };
40
41 void init(StackType *s);
42 int isEmpty(StackType *s);
43 int isFull(StackType *s);
44 void push(StackType *s, position *item);
45 position* pop(StackType *s);
46 position* peek(StackType *s);
47
48 position* makePosition(int x, int y);
49
50 void printMaze();
51 void printStack(StackType *s);
52 position* positionSearch(position pos);
53 void positionMove(StackType *s, position *pos, position *nextPos);
54 void positionBackMove(StackType *s, position *pos);
55 int isInMaze(int x, int y);
56
57 int main(void)
58 {
59     int itrX, itrY;
60     position pos, endPos;
61     StackType stack;
62     position *tempPos;
63
64     init(&stack);
65
66     printMaze();
67     printStack(&stack);
```

```

68
69     for (itrY = 0; itrY < MAZE_HEIGHT; itrY++) {
70         for (itrX = 0; itrX < MAZE_WIDTH; itrX++) {
71             if (maze[itrY][itrX] == MAZE_START) {
72                 pos.x = itrX;
73                 pos.y = itrY;
74                 maze[itrY][itrX] = MAZE_MOUSE;
75             }
76             if (maze[itrY][itrX] == MAZE_END) {
77                 endPos.x = itrX;
78                 endPos.y = itrY;
79             }
80         }
81     }
82
83     while (pos.x != endPos.x || pos.y != endPos.y) {
84         Sleep(SLEEP_TIME);
85         printMaze();
86         printStack(&stack);
87
88         tempPos = positionSearch(pos);
89         if (tempPos)
90             positionMove(&stack, &pos, tempPos);
91         else
92             positionBackMove(&stack, &pos);
93     }
94     Sleep(SLEEP_TIME);
95     printMaze();
96     printStack(&stack);
97     Sleep(SLEEP_TIME);
98
99     return 0;
100 }
101
102
103
104 void init(StackType *s) {
105     s->top = -1;
106 }
107
108 int is_empty(StackType *s) {
109     return (s->top == -1);
110 }
111
112 int is_full(StackType *s) {
113     return (s->top == (MAX_STACK_SIZE - 1));
114 }
115
116 void push(StackType *s, position *item) {
117     if (is_full(s)) {
118         fprintf(stderr, "스택 포화 에러\n");
119         return;
120     }
121     else s->stack[++(s->top)] = item;
122 }
123
124 position* pop(StackType *s) {
125     if (is_empty(s)) {
126         fprintf(stderr, "스택 공백 에러\n");
127         exit(1);
128     }
129     else return s->stack[(s->top)--];
130 }
131
132 position* peek(StackType *s) {
133     if (is_empty(s)) {
134         fprintf(stderr, "스택 공백 에러\n");
135         exit(1);
136     }
137     else return s->stack[s->top];
138 }
139

```

```

140 position* makePosition(int x, int y) {
141     position* pos;
142     pos = (position*)malloc(sizeof(position));
143     if (!pos)
144     {
145         fprintf(stderr, "position 할당 에러\n");
146         exit(1);
147     }
148
149     pos->x = x;
150     pos->y = y;
151
152     return pos;
153 }
154
155 void printMaze() {
156     int itrY, itrX;
157
158     system("cls");
159     for (itrY = 0; itrY < MAZE_HEIGHT; itrY++) {
160         for (itrX = 0; itrX < MAZE_WIDTH; itrX++) {
161             printf(" %c ", maze[itrY][itrX]);
162         }
163         printf("\n");
164     }
165 }
166
167 void printStack(StackType *s) {
168     int itr;
169     printf("%4s|(%3s,%3s)|\n", "", " X ", " Y ");
170     for (itr = MAX_STACK_SIZE - 1; itr > s->top; itr--)
171         printf("%4d|(%3s,%3s)|\n", itr, "", "");
172     for (; itr >= 0; itr--)
173         printf("%4d|(%3d,%3d)|\n", itr, s->stack[itr]->x, s->stack[itr]->y);
174 }
175
176 position* positionSearch(position pos) {
177     int x, y;
178
179     x = pos.x - 1;
180     y = pos.y;
181     if (isInMaze(x, y)) {
182         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
183             return makePosition(x, y);
184         }
185     }
186
187     x = pos.x;
188     y = pos.y + 1;
189     if (isInMaze(x, y)) {
190         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
191             return makePosition(x, y);
192         }
193     }
194
195     x = pos.x + 1;
196     y = pos.y;
197     if (isInMaze(x, y)) {
198         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
199             return makePosition(x, y);
200         }
201     }
202
203     x = pos.x;
204     y = pos.y - 1;
205     if (isInMaze(x, y)) {
206         if (maze[y][x] == MAZE_ROAD || maze[y][x] == MAZE_END) {
207             return makePosition(x, y);
208         }
209     }
210
211     return NULL;
212 }

```

```

213
214 void positionMove(StackType *s, position *pos, position *nextPos) {
215     push(s, makePosition(pos->x, pos->y));
216     maze[pos->y][pos->x] = MAZE_VISITED;
217
218     pos->x = nextPos->x;
219     pos->y = nextPos->y;
220     maze[pos->y][pos->x] = MAZE_MOUSE;
221
222     free(nextPos);
223 }
224
225 void positionBackMove(StackType *s, position *pos) {
226     maze[pos->y][pos->x] = MAZE_BACK_MOVED;
227
228     position* backPos = pop(s);
229     pos->x = backPos->x;
230     pos->y = backPos->y;
231     maze[pos->y][pos->x] = MAZE_MOUSE;
232
233     free(backPos);
234 }
235
236 int isInMaze(int x, int y) {
237     if (x < 0 || x > MAZE_WIDTH)
238         return 0;
239     if (y < 0 || y > MAZE_HEIGHT)
240         return 0;
241     return 1;
242 }
243
244

```

carbon
carbon.now.sh

참고 자료

- 자료구조및알고리즘이해, 과제3 PDF
- 자료구조및알고리즘이해, 강의노트 5장 Stack
- source code image formatter, <https://carbon.now.sh/>