HW #4 미로찾기 UI 만들기

자료구조 01분반

전자전기공학부 20196891 이민영

1. 주어진 미로에서 자동으로 출구를 찾는 c 코드

미로찾기here.c

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

#define MAX_SIZE 100
#define MAZE_SIZE 10

typedef struct Position {
	short x;
	short y;
}Position;

typedef struct Stack
{
	Position data[MAX_SIZE];
	int top;
}Stack;

char maze[MAZE_SIZE][MAZE_SIZE] = {
	{'0','1','1','1','1','1','1','1','1','1'},
	{'0','0','0','0','1','0','0','0','0','1'},
	{'1','0','1','0','1','0','0','1','0','1'},
	{'1','0','1','1','1','0','0','1','0','1'},
	{'1','0','0','0','1','0','1','1','0','1'},
	{'1','0','1','0','1','0','1','1','0','1'},
	{'1','0','1','0','1','0','1','1','0','1'},
	{'1','0','1','0','1','0','0','1','0','1'},
	{'1','0','1','0','0','0','0','1','0','x'},
	{'1','1','1','1','1','1','1','1','1','1'} };

void Init(Stack* p)
{
	p->top = -1;
}

int Is_full(Stack* p)
{
	return (p->top == MAX_SIZE - 1);
}

int Is_empty(Stack* p)
```

```
{
	return (p->top == -1);
}

void Push(Stack* p, Position data)
{
	if (Is_full(p))
	{
		printf("스택이 꽉찼습니다₩n"); return;
	}
	else
	{
		p->top++;
		p->data[p->top].x = data.x;
		p->data[p->top].y = data.y;
	}
}

Position Pop(Stack* p)
{
	if (Is_empty(p))
	{
		printf("스택이 비어있습니다₩n");
		exit(1);
	}
	return p->data[(p->top)--];
}

void Push_Loc(Stack* s, int x, int y)
{
	if (x < 0 || y < 0 || x > MAZE_SIZE || y > MAZE_SIZE) return;

	if (maze[x][y] != '1' && maze[x][y] != '2')
	{
		Position next;
		next.x = x;
		next.y = y;

		Push(s, next);
	}
}

int main()
{
	Stack s;
	Position cur;
	int i, j, x, y;

	Init(&s);
	// 미로 출력
	for (i = 0; i < MAZE_SIZE; i++) {
		for (j = 0; j < MAZE_SIZE; j++) {
			printf("%c ", maze[i][j]);
		}
		printf("₩n");
```

```
        }

        // 시작점 (0,0)
        cur.x = 0;
        cur.y = 0;
        printf("시작 점 (%d,%d) ₩n", cur.x, cur.y);

        while (maze[cur.x][cur.y] != 'x')
        {
                x = cur.x;
                y = cur.y;

                maze[x][y] = '2'; // 방문한 곳을 표시

                // 좌,우,위,아래중 이동 가능한 곳을 탐색
                Push_Loc(&s, x + 1, y);
                Push_Loc(&s, x - 1, y);
                Push_Loc(&s, x, y + 1);
                Push_Loc(&s, x, y - 1);

                if (Is_empty(&s))  // 공백함수
                {
                        printf("실패₩n");
                        return 0;
                }
                else
                {
                        cur = Pop(&s);// 현재 좌표를 변경
                        printf("(%d,%d) -> ", cur.x, cur.y);
                }
        }

        printf("도착 점 (%d,%d)₩n", cur.x, cur.y);
        printf("탐색 성공₩n");

        for (i = 0; i < MAZE_SIZE; i++) {
                for (j = 0; j < MAZE_SIZE; j++) {
                        printf("%c ", maze[i][j]);
                }
                printf("₩n");
        }
        return 0;
}
```

출력 결과 :

```
0 1 1 1 1 1 1 1 1 1
0 0 0 0 1 0 0 0 0 1
1 0 1 0 1 0 0 1 0 1
1 0 1 1 1 0 0 1 0 1
1 0 0 0 1 0 1 1 0 1
1 0 1 0 1 0 1 1 0 1
1 0 1 0 1 0 1 1 0 1
1 0 1 0 1 0 0 1 0 1
1 0 1 0 0 0 0 1 0 x
1 1 1 1 1 1 1 1 1 1
시작 점 (0,0)
(1,0) -> (1,1) -> (1,2) -> (1,3) -> (2,3) -> (2,1) -> (3,1) -> (4,1) -> (4,2) -> (4,3) -> (5,3)
-> (6,3) -> (7,3) -> (8,3) -> (8,4) -> (8,5) -> (8,6) -> (7,6) -> (7,5) -> (6,5) -> (5,5) ->
(4,5) -> (3,5) -> (3,6) -> (2,6) -> (2,5) -> (1,5) -> (1,6) -> (1,7) -> (1,8) -> (2,8) -> (3,8)
-> (4,8) -> (5,8) -> (6,8) -> (7,8) -> (8,8) -> (8,9) -> 도착 점 (8,9)
탐색 성공
2 1 1 1 1 1 1 1 1 1
2 2 2 2 1 2 2 2 2 1
1 2 1 2 1 2 2 1 2 1
1 2 1 1 1 2 2 1 2 1
1 2 2 2 1 2 1 1 2 1
1 0 1 2 1 2 1 1 2 1
1 0 1 2 1 2 1 1 2 1
1 0 1 2 1 2 2 1 2 1
1 0 1 2 2 2 2 1 2 x
1 1 1 1 1 1 1 1 1 1
```

미로를 2차원 배열로 생성하고 출구를 x로 표시할 때, 출구의 x는 문자형이므로 자료형을 char로 설정했습니다. 스택 구조체와 위치 구조체를 생성하여 position의 위치 (x,y)를 스택에 입력하도록 설계하고, 스택의 포화와 공백을 검사하고 push와 pop을 실행할 수 있는 함수를 생성했습니다. 메인 함수에서 스택 s에 위치를 입력하고 현재 위치를 cur이라 하였습니다. 출발점부터 시작해 한번 지나친 위치는 모두 숫자 2로 표현할 수 있도록 while문의 초반에 maze[x][y] = '2' 를 넣었습니다. 숫자 1은 막힌 부분, 숫자 0은 이동 가능한 부분으로 Push_Lock 함수를 이용해 이동가능 여부를 판단한 후, 이동 가능하다면 push, 이동이 불가능하다면 pop을 실햄함으로써 뒤로 돌아가 새로운 길을 시도해볼 수 있도록 했습니다. 매 이동의 경로를 추적할 수 있도록 한번의 push나 pop마다 위치를 출력하도록 설계했고, X에 도달하면 탐색 성공을 출력하고 다시 maze 함수를 출력해 프로그램이 지나온 길을 확인할 수 있습니다. 길이 막혀 다시 돌아왔을 때 3으로 숫자를 변환하고 싶었으나 성공하지 못했습니다.
Maze 함수를 char형이 아닌 int형으로 바꾸어 한번 지나갈 때마다 값을 1씩 증가시키도록 프로그램을 수정하면 프로그램이 개선될 것으로 판단되지만 아쉽게도 계속되는 오류로 성공시키지 못해 기존의 소스를 첨부합니다.

2. 사용자 지정 방향으로 이동하며 출구를 찾는 c 코드

미로찾기UI.c

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

#define _CRT_SECURE_NO_WARNINGS

#define MAX_SIZE 100
#define MAZE_SIZE 10
#define PATH 0
#define WALL 1
#define VISITED 2
#define BACKTRACKED 3
int sum; int input;

typedef struct Position
{
        short x;
        short y;
}Position;

typedef struct Stack
{
        Position data[MAX_SIZE];
        int top;
}Stack;

char maze[MAZE_SIZE][MAZE_SIZE] = {
        {'0','1','1','1','1','1','1','1','1','1'},
        {'0','0','0','0','1','0','0','0','0','1'},
        {'1','0','1','0','1','0','0','1','0','1'},
        {'1','0','1','1','1','0','0','1','0','1'},
        {'1','0','0','0','1','0','1','1','0','1'},
        {'1','0','1','0','1','0','1','1','0','1'},
        {'1','0','1','0','1','0','1','1','0','1'},
        {'1','0','1','0','1','0','0','1','0','1'},
        {'1','0','1','0','0','0','0','1','0','0'},
        {'1','1','1','1','1','1','1','1','1','0'} };

void Init(Stack* p) {               // 스택 초기화
        p->top = -1;
}

int Is_full(Stack* p) {             // 스택 포화 검사
        return (p->top == MAZE_SIZE - 1);
}

int Is_empty(Stack* p) {            //스택 공백 검사
        return (p->top == -1);
}

void push(Stack* p, Position data)       // push
```

```
{
    if (Is_full(p))    {
        printf("스택이 꽉찼습니다₩n"); return;
    }
    else    {
        p->top++;
        p->data[p->top].x = data.x;
        p->data[p->top].y = data.y;
    }
}

Position pop(Stack* p)              // pop
{
    if (Is_empty(p))    {
        printf("스택이 비어있습니다₩n");
        exit(1);
    }
    return p->data[(p->top)--];
}

int offset[4][2] = { {-1,0}, {1,0}, {0,-1}, {0,1} };

int movable(Position pos, int dir) {    // 지정한 방향으로 이동이 가능하면 1, 불가능하면 0을
return하는 함수
    Position next;
    next.x = pos.x + offset[dir][0];
    next.y = pos.y + offset[dir][1];
    if (maze[next.x][next.y] == 0)
        return 1;
    else
        return 0;
};

// 지정한 방향으로 이동 move_to
Position move_to(Position pos, int dir) {
    Position next;
    maze[pos.x][pos.y] = 2;
    next.x = pos.x + offset[dir][0];
    next.y = pos.y + offset[dir][1];
    return next;
};

int main() {

    int sum; int input;
    Stack s;
    Position cur;
    cur.x = 0;
    cur.y = 0;
    printf("시작 점 (%d,%d) ₩n", cur.x, cur.y);

    while (cur.x < MAZE_SIZE - 1 && cur.y < MAZE_SIZE - 1) {     // 현재 위치가 출구가 아닌
동안
        maze[cur.x][cur.y] = '2';  // 현재 위치의 값 2로 변경
```

```
        // 사용자 지정 방향 입력
        int input;
        printf("이동방향(0:상 1:하 2:좌 3:우) = ");
        scanf("%d", &input);

        // sum의 값 = 이동 가능한 방향의 수
        int sum;
        for (int i = 0; i < 4; i++) {
            sum = sum + movable(cur, i);
        }

        if (sum == 0) { // 이동 가능한 방향이 없다면  값을 3 으로 바꾸며 마지막 분기점까지 pop
            while (sum == 0) {
                maze[cur.x][cur.y] = '3';
                pop(&s);
            }
        }
        else if (movable(cur, input) == 1) {  // movable == 1 이면 이동
            push(&s, cur);
            cur = move_to(cur, input);
            printf("(%d,%d) ₩n", cur.x, cur.y);
        }
        else {  // 이동 가능한 방향이 있으나 입력한 방향은 불가능한 경우엔 제자리에서 다시 입력
받기
            cur = cur;
        }
    }

        if (cur.x == MAZE_SIZE - 1 && cur.y == MAZE_SIZE - 1) {       // 현재 위치가 출구일 때
            printf("도착점 (%d,%d)₩n", cur.x, cur.y);
            printf("Found the path.₩n");

        for (i = 0; i < MAZE_SIZE; i++) {
            for (j = 0; j < MAZE_SIZE; j++) {
                printf("%c ", maze[i][j]);
            }
            printf("₩n");
        }

        return 0;
}
```

빌드 오류 목록 : Input과 sum의 변수 범위 문제

Scanf 반환값 무시

프로그램을 성공적으로 빌드하지 못한 관계로 미완성 코드를 첨부합니다.

코드를 작성하며 각각의 과정에서 본래 의도했던 실행과정을 주석으로 상세히 설명하려 노력했습니다. 앞서 작성했던 자동 미로찾기와 같이 주어진 미로 배열을 int형으로 설정해 프로그램을 작성하면 더욱 깔끔하게 개선된 미로찾기 프로그램을 만들 수 있을 것이라 생각됩니다.

함수를 작성하면서 배열을 다루는 법과 스택을 활용하는 법에 대해 많이 공부할 수 있었습니다. 배열을 통해 미로를 생성하고 미로에서의 각 위치를 정확하게 짚어낼 수 있고, 막다른 길에 도착했을 때, 스택에 지금까지 거쳐온 경로가 순서대로 저장되어 있으므로 pop을 이용해 쉽게 전 마지막 분기점으로 돌아가게 만들 수 있다는 것을 공부했습니다.

코드를 작성하며 계속해서 변수의 정의와 초기화 등에서 오류가 발생함을 느꼈습니다. 좀 더 수월하게 정돈된 코드를 작성하기 위해서는 변수의 범위를 다루는 공부를 더 많이 해야 함을 느꼈습니다. 또한 배열과 스택을 다룸에 있어서도 많이 서투르고 더욱 공부가 필요함을 느꼈습니다. 자료구조는 컴퓨터 코드를 프로그래밍함에 있어 가장 기초적인 구조틀이라는 느낌이 들었습니다. 처음에는 한없이 막막했던 코드 작성이었지만, 자료구조를 다시 공부하며 무작정 코드부터 써내려가는 것이 아닌 코드의 틀을 먼저 세우고 그에 맞추어 코드를 작성하는 것을 연습했습니다. 그 덕분에 미완성이지만 어느정도 틀이 잡힌 코드를 작성할 수 있었습니다. 자료구조에 있어 지나온 과정과 앞으로 공부할 과정을 더욱 공부하면 완성된 코드를 작성할 수 있을 것이라고 생각합니다.