자료구조 실습과제 #7

미로 탈출 중 돌아온
총 횟수 출력 프로그램

5671144-강범창

1. 주어진 미로를 탈출하는 프로그램을 작성한다.

1.1 Stack 의 element 를 row , column 를 표현하는 구조체를 넣어 stack을 구현합니다. 이때 Stack은 이전에 사용해오던 stack 에서 element 의 타입만 수정을 하였습니다.

```
typedef struct
{
    /* data */
    short r;
    short c;
} mazeElement;
==========================================================================
typedef struct
{
    mazeElement *data;
    int capacity;
    int top;
} StackType;
```

1.2 미로에 사용할, 현재 위치를 표시할 here , 그리고 시작 지점을 표시할 entry , 2번에서 더 이상 움직일 수 없는 경우에 대해서 횟수와 없는 경우를 표시할 변수를 초기화 시켜줍니다. 전역으로 사용합니다. 그리고 미로도 초기화를 시켜줍니다.

```
mazeElement here = {1, 0}, entry = {1, 0};
int notPushCount = 0;
bool isPushed = false;
```

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

1.3 stack 을 초기화하고 현재(here) 의 위치가 출구인 x까지도는 반복문을 사용합니다. 그리고 지나온 길은 '.' 으로 표기를 해 온 곳을 확인합니다. 그후 미로를 mazePrint 함수로 출력합니다. mazePrint 함수는 현재 maze 에 있는 값들을 출력하는 함수입니다.

```
int row, column;
StackType *stack = create(MAX_STACK_SIZE);
here = entry;

while (maze[here.r][here.c] != 'x')
    {
        /* code */
        row = here.r;
        column = here.c;
        maze[row][column] = '.'; // 이미 지나온 곳은 . 으로 표기합니다.
        mazePrint(maze);

   …
==========================================================================

void mazePrint(char maze[MAZE_SIZE][MAZE_SIZE])
{
    printf("₩n");
    for (int row = 0; row < MAZE_SIZE; row++)
    {
        for (int column = 0; column < MAZE_SIZE; column++)
        {
            printf("%c ", maze[row][column]);
        }
        printf("₩n");
    }
}
```

1.4 pushLoc 이라는 함수를 통해 stack에 갈수 있는 곳(벽으로 표현이 된 1 이나 이미 지나온 길인 ' . ' 이 아닌 경우) 을 push 합니다. 그리고 2번을 해결하기 위해 길을 push 다는 의미인 isPushed 를 true로 값을 줍니다.

```
pushLoc(stack, row - 1, column);
pushLoc(stack, row + 1, column);
pushLoc(stack, row, column - 1);
pushLoc(stack, row, column + 1);
================================================================================


void pushLoc(StackType *stack, int row, int column)
{
    if (row < 0 || column < 0)
        return;
    // 막혀있거나 이미 지나온 길이 아니라면 push!
    if (maze[row][column] != '1' && maze[row][column] != '.')
    {
        mazeElement temp;
        temp.r = row;
        temp.c = column;
        push(stack, temp);
        isPushed = true;
    }
    return;
}
```

1.5 push 된 값에 따라 처리를 해줍니다. 만약 stack 에 값이 없다면 갈수 있는 길이 없다는 뜻이 되므로, 미로에는 나가는 곳이 없다는 메시지와 같이 프로그램을 종료(return 0) 합니다. 그런 경우가 아니라면 현재 위치를 나태내는 here 에 현재 위치를 pop 으로 stack 에서 가져와 넣어줍니다.

```
if (isEmpty(stack))
{
      printf("이 미로에는 나가는 곳이 없습니다.₩n");
      return 0;
}
here = pop(stack);
```

1.6 이후 계속 반복을 하여 x에 도달했을 경우 미로를 성공적으로 나갔다는 메시지를 출력합니다.

```
printf("미로를 성공적으로 나갔습니다!₩n");
```

2. 해당프로그램을 실행했을 때 더 이상 움직일 수 없는 경우 되돌아가는 총 횟수를 출력한다.

1.5 에서 처리를 해주듯 여기서도 처리를 해줍니다 만약 1.4 에서 push 를 하는 단계에서 왼쪽 오른쪽 아래 위를 보았을 때 갈수 있는 곳이 없다면, isPushed 가 false 로 남아있습니다. 이때, 움직일 수 있는 곳이 없었다고 보고, notPushCount(움직일 수 없었던 횟수.)를 증감시켜 줍니다. 이후 프로그램이 미로를 성공적으로 나갔을 때 움직일 수 없었던 경우의 횟수를 출력합니다.

```
if (!isPushed) // push 되지 않았나. 즉 더이상 움직일수 없엇던 경우!
        {
            notPushCount++;
        }
        else
        {
            isPushed = false; // 정상적으로 push 막힘없이 길을 찾았을경우 다시 push되었나를 확인하기위해 사용합니다.
        }
```

```
printf("더이상 움직일수 없엇던 경우: %d₩n", notPushCount);
```