CSED101. Programming & Problem solving

Spring, 2014

Programming Assignment #4

(70 points)

마정현(doitnow0415@postech.ac.kr)

- ***Due***: **2014.06.01 23:59**
- ***Development Environment***: **Windows Visual Studio 2010**
- ***제출물***
  - **C Code files (*.c)**
    - **stack.h, stack.c, assn4_1.c, assn4_2.c**
    - 프로그램의 소스 코드를 이해하기 쉽도록 반드시 **주석**을 붙일 것.
  - **보고서 파일** (.doc(x) or .hwp) 예) assn4.doc(x) 또는 assn4.hwp
    - AssnReadMe.pdf 를 참조하여 작성할 것.
  - 소스코드와 보고서 파일을 LMS를 이용하여 제출한다.
- ***주의사항***
  - 문제의 요구사항을 반드시 지킬 것.
  - 모든 문제의 출력 형식은 아래의 예시들과 동일해야 하며, 같지 않을 시는 감점이 된다.
  - 각 문제에 제시되어 있는 파일이름으로 제출 할 것. 그 외의 다른 이름으로 제출하면 감점 또는 0점 처리된다.
  - 과제를 작성하는데 있어서 전역변수를 선언하여 사용할 수 없다.
  - 이번 과제에서는 추가 기능 구현에 대한 추가 점수는 없다
  - 컴파일 & 실행이 안되면 무조건 0점 처리된다.
  - **하루 late시 20%가 감점되며, 3일 이상 지나면 받지 않는다. (0점 처리)**
  - 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 'POSTECH 전자컴퓨터공학부 부정행위 정의'를 따른다. (LMS의 과목 공지사항의 제목 [document about cheating]의 첨부파일인 disciplinary.pdf를 참조할 것.)
  - Linked list로 구현하지 않는다.
  - 재귀(recursion) 함수를 사용할 수 없다.

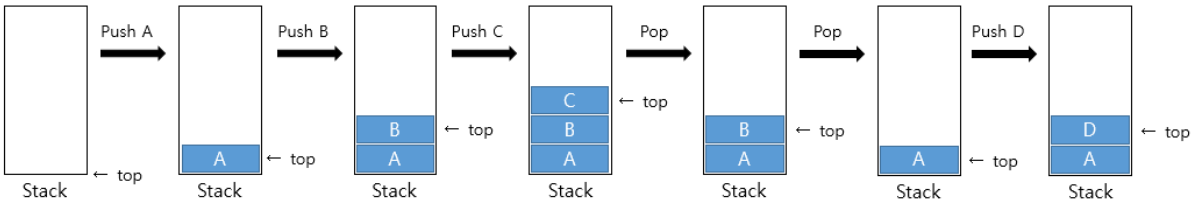
Problem 1: 스택 자료구조 구현 (20점)

(목적)

- 이 문제에서는 프로그래밍 시 사용되는 자료구조 중 하나인 스택을 직접 구현해봄으로써, 구조체와 포인터를 익힌다.

(스택의 정의)

- 스택은 쌓아 올린 더미를 의미하며 한 쪽 끝에서만 삽입과 삭제과 일어나는 자료구조이다. 즉, 스택 구조는 아이템을 Last In First Out (LIFO) 방식으로 저장하는 구조로써, 가장 마지막에 들어온 아이템(아래 그림상 가장 위쪽에 위치한)이 가장 먼저 나갈 수 있도록 정의한 구조를 의미한다.
- 스택 동작 그림



  - 그림은 (1)의 빈 스택에 아이템을 삽입(Push)하고 삭제(Pop)함에 따라 스택의 변화과정을 보여주는 그림이다.
  - 삽입과 삭제는 현재 저장된 최상위 항목이 위치한 꼭대기(top) 에서만 일어난다. 그림에서 top은 항상 가장 마지막 아이템을 가리킨다.
  - Stack 을 배열로 구현하면 배열 인덱스는 0에서 시작하므로 top의 초기값은 -1 로 설정한다.

(문제)

- 아래 stack.h 파일을 이용하여 실행예제와 같이 동작하는 스택을 구현한다.

(설명)

- 다운로드 받은 **assn4.zip**의 압축을 풀면 **stack.h** 와 **stack.c** 가 주어진다.
- 스택의 선언 (stack.h)

아래에서 Position type은 2차원 배열의 좌표 (row, col) 값을 저장하기 위한 자료형이다.
Stack 에 Push 또는 Pop 되는 아이템은 구조체 Position type 으로 위치 좌표 (row, col) 값을 저장한다. 그리고, 그 아래 스택 구현을 위한 함수들이 선언되어 있다.

```
#ifndef STACK_H
#define STACK_H

typedef struct
{
        int row;
        int col;
} Position; // stack 에 추가 또는 삭제할 아이템의 자료형 Position을 선언
```

```
typedef struct
{
	int top;	// stack의 마지막 아이템의 위치 저장
	int size;	// stack의 최대 사이즈를 저장
	Position *items_list; // stack 의 Position 타입의 정보들을 저장할 공간을 가리키는 포인터
} Stack;

void initStack(Stack *s, int stack_size);
int isEmpty(Stack *s);
int isFull(Stack *s);
void push(Stack *s, Position item);
Position pop(Stack *s);
Position top(Stack *s);
void freeStack(Stack *s);

#endif
```

- 아래의 설명을 참고하여 제공되는 stack.c 파일에 있는 함수들을 완성시키고, **assn4_1.c** 에는 **main 함수를 포함**하여 아래의 실행예제가 실행되도록 작성한다.

- **스택 오퍼레이션의 정의**
  - push: 아이템을 스택에 추가
  - pop: 스택의 가장 마지막에 들어온 아이템을 제거하고 반환
  - top: 스택의 가장 마지막에 들어온 아이템을 반환
  - isEmpty: 스택이 비었는지 확인
  - isFull: 스택이 가득차 있는지 확인

- **아래의 실행 예제를 참고하여 스택을 구현하자.**
  - **필요하면 string.h 에 정의되어 있는 함수를 사용할 수 있다.**

  (1) 명령어

  - **create:** 스택을 생성하고 초기화한다. 스택은 메모리를 동적할당 받아서 사용하도록 한다. **모든 명령어는 create 명령어 수행 후에 이루어진다고 가정한다.**
    create 명령어를 입력하면, 스택의 크기(stack_size)를 입력받아 스택을 초기화한다.

```
Stack *stack;
stack = (Stack *) malloc (sizeof(Stack) );
initStack(stack); // 초기화 하기 위해 함수 호출
```

void initStack(Stack *s, int stack_size);

스택을 초기화하는 함수로 top 을 -1, size를 stack_size,
item_llist = (Position *)calloc(stack_size, sizeof(Position)); 로 설정한다.

- **exit:** 동적할당 받은 메모리를 모두 free 시키고 프로그램을 종료한다.
  함수 freeStack 을 작성해서 사용하자.
  void freeStack(Stack *s);
  스택 생성시 동적할당 받은 메모리를 free 시킨다.
  - **free를 수행해야 할 동적할당 받은 메모리 목록**
    - 스택의 item_list
    - 스택

- **push**: 아이템을 스택에 추가하는 명령어로 그림(1)의 빈 스택에서 Push A 를 수행하면 스택에 A 가 삽입 되어 그림(2)의 상태가 된다.
  void push(Stack *s, Position item);

- **pop**: 스택의 가장 마지막에 들어온 아이템을 스택에서 제거하고 그 값을 반환하는 명령어로 그림(4)의 상태에서 Pop 을 수행하면 그림(5)의 상태가 된다.
  Position pop(Stack *s);

- **top**: 스택의 가장 마지막에 들어온 아이템을 반환하는 명령어로 그림(7)의 상태에서 top 을 수행하면 아이템 D를 읽어온다. top 수행 후, 스택 상태의 변화는 없다.
  Position top(Stack *s);

- **isEmpty**: 스택이 비었는지 확인하는 명령어로 비어있으면 1, 그렇지 않으면 0을 반환한다.
  int isEmpty(Stack *s);

- **isFull**: 스택에 아이템이 가득 차있는지 확인하는 명령어로 가득 차있으면 1, 그렇지 않으면 0을 반환한다.
  int isFull(Stack *s)

(2) 그 외 함수

void showStack(Stack *s);
스택에 아이템을 삽입한 순서대로 출력한다.
아래 실행 예제 중 "[stack]: |(1, 0)|(1, 1)|(2, 2)|" 과 같이 현재 스택에 존재하는 아이템을 출력하기 위해 사용한다

(3) 스택 오버플로우 (Overflow) 와 언더플로우 (Underflow)

- 배열로 구현하는 스택 사이즈는 제한이 있으므로 스택이 가득 차 있을 때, push 하고자 하면, overflow가 발생한다.
- 스택이 비어 있을 때, pop과 top 을 하고자 하면, underflow가 발생한다.

실행예제) 빨간색 글씨는 사용자 입력이다.

모든 명령어는 create 명령어 수행 후에 이루어진다고 가정하며 틀린 입력은 없다고 가정한다.

예제1)

```
create
Input stack size: 3
[stack]:
push 1 0
[stack]: |(1, 0)|
push 1 1
[stack]: |(1, 0)|(1, 1)|
push 2 2
[stack]: |(1, 0)|(1, 1)|(2, 2)|
push 3 3
Stack Overflow
[stack]: |(1, 0)|(1, 1)|(2, 2)|
pop
item: (2, 2)
[stack]: |(1, 0)|(1, 1)|
top
item: (1, 1)
[stack]: |(1, 0)|(1, 1)|
pop
item: (1, 1)
[stack]: |(1, 0)|
pop
item: (1, 0)
[stack]:
pop
Stack Underflow
[stack]:
exit
계속하려면 아무 키나 누르십시오...
```

예제2)

```
create
Input stack size: 2
[stack]:
isEmpty
1
[stack]:
push 1 0
[stack]: |(1, 0)|
isEmpty
0
[stack]: |(1, 0)|
isFull
0
[stack]: |(1, 0)|
push 3 3
[stack]: |(1, 0)|(3, 3)|
isFull
1
[stack]: |(1, 0)|(3, 3)|
pop
item: (3, 3)
[stack]: |(1, 0)|
exit
계속하려면 아무 키나 누르십시오...
```

(주의사항)

- 제출해야 파일은 **stack.h**, **stack.c**, **assn4_1.c** 이다.
- 보고서는 **"assn4.doc"** or **"assn4.hwp"**로 저장 한다. (보고서는 통합하여 작성)
- 화면 출력은 반드시 실행 예와 같아야 한다.

Problem 2: 스택 자료구조를 이용한 미로 찾기 (50점)

(문제)

- Problem1에서 구현한 스택을 이용하여 미로 찾기를 구현하자. 반드시 스택을 이용하여 구현한다. (재귀함수로 구현하지 말 것)

(문제 정의)

- **Problem1에서 작성한 stack.c와 stack.h를 include 하여 사용한다.**
- 파일로부터 입력 받은 미로에서 스택 자료구조를 이용하여 길을 찾는다.
- 미로는 N x M 행렬로 구성된다. 시작점의 좌표는 (1, 0)이며, 출구의 좌표는 (N-2, M-1)으로 고정된다.
- 한 지점에서 움직일 수 있는 방향은 동, 서, 남, 북 네 방향이다.

(문제의 개요)

- 다음의 N x M 행렬은 주어진 미로를 나타낸다. '#'는 벽을 의미하며 공란(Blank)은 통로를 의미한다. 'O'는 시작점의 위치를 나타낸다. 시작점의 좌표는 (1, 0)으로 고정되고, 'G'는 출구를 나타내며, 좌표는 (N-2, M-1)으로 고정된다. 시작점으로부터 출구까지의 Path가 존재하지 않을 수도 있다. 출구까지의 Path가 존재하지 않을 때는 출구를 찾을 수 없다는 문구를 출력해준다.
  - **가정**: 입, 출구에 해당하는 좌표에 벽'#'은 존재하지 않는다. 즉, 미로 맵을 만들 때 이를 고려할 필요가 없다.

```
############################
O #     #####  ### # ## ###
#   ###  ##### ### # ## ###
# ######          #      #
#  ##### ## ######## ## # #
##   ##  ##          ## # #
####  ######### ####### # G
############################
```

- 미로는 파일에 정해진 양식을 따른다. 오른쪽 그림은 위의 8 x 27 미로 행렬에 대한 파일의 예이다. 오른쪽 그림에서처럼 통로는 blank로 벽은 #으로 표현된다. 입구와 출구의 위치가 정해져 있기 때문에, 입구와 출구는 파일에 따로 표현하지 않는다. (maze1.dat 파일 참고) 미로 파일의 첫 행에는 행(N)과 열(M)이 반드시 명시되어 있으며, 파일로부터 이 정보를 읽어 N x M 크기의 배열 형태를 가지도록 동적 할당을 하여 미로를 메모리에 저장한다. (아래의 문제힌트 부분 참고)

```
8 27
############################
  #     #####  ### # ## ###
#   ###  ##### ### # ## ###
# ######          #      #
#  ##### ## ######## ## # #
##   ##  ##          ## # #
####  ######### ####### #
############################
```

  - **가정1**: 미로 파일이 위의 정해진 양식을 벗어나는 경우는 없다.
  - **가정2**: 미로 맵에 대한 파일 첫 행의 행과 열 정보가 틀린 경우는 없다.

- 현재 위치에서 가능한 진행방향을 조사할 때, 동->서->남->북 순서대로 조사하여 벽이 아니고 이미 방문한 곳이 아니라면 그 쪽으로 진행하도록 한다. 방향을 조사하는 순서는 위의 순서를 반드시 지켜야 한다. (대각선 방향으로 이동하는 경우는 없다)
  - 다음 진행방향으로의 이동에 대한 연산 및 미로 맵에서 현재 위치의 출력은 0.1초 간격으로 진행한다. (Sleep함수 이용, windows.h에 정의).
  - 사용방법: Sleep(100); // Sleep함수에 들어가는 인자는 밀리초를 나타냄.

- 현재 위치에서 다음 위치로 진행하는 경우, 현재 위치에서 진행 가능한 방향 중 하나를 스택에 저장하게 된다. 이 때, 아래의 구조체를 사용하도록 한다. (Problem 1에서 사용함)
  Stack 에 push 또는 pop 되는 item type은 구조체 Position type 으로 위치 좌표 (row, col)를 나타낸다. 구조체 Stack type 은 top 과 size(**스택의 최대 크기: M*N**)를 멤버로 가지고 있다.

```
typedef struct
{
    int row;
    int col;
} Position; // stack 에 추가 또는 삭제할 아이템의 자료형 Position을 선언

typedef struct
{
    int top;      // stack의 마지막 아이템의 위치 저장
    int size;     // stack의 최대 사이즈를 저장
    Position *items_list; // stack 의 Position 타입의 정보들을 저장할 공간을 가리키는 포인터
} Stack;
```

- **Position을 Stack 에 저장하게 되는 경우는** 현재 위치에서 이동 가능한 지점(동서남북 중 하나)이 벽이 아니고 아직 방문을 하지 않은 경우이다. Stack에는 현재까지 방문한 Position 들 중에서 되돌아 나오지 않은 것들이 저장된다. 즉, 현재까지 알아낸 입구에서 출구까지의 길이 저장된다.
- **Stack 에서 Position 를 꺼내게 되는 경우는** 현재 위치를 기준으로 조사한 네 방향 모두가 벽 혹은 이미 방문한 곳에 해당되는 경우이다. 아래의 문제힌트의 알고리즘을 참고하라.

(문제 힌트)

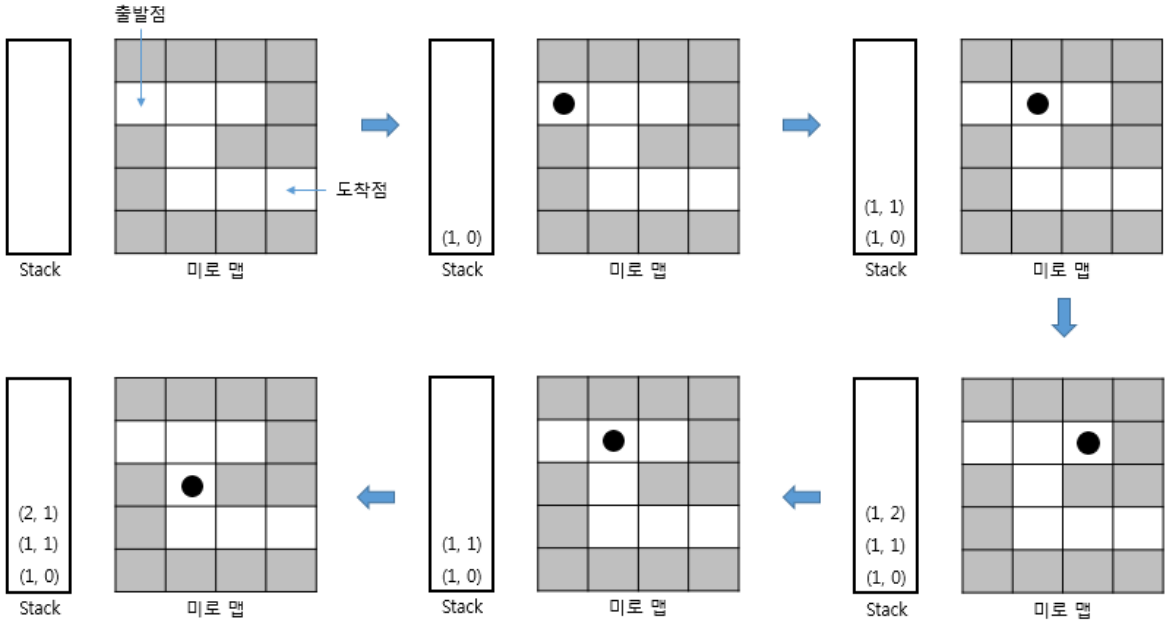
- **스택을 이용한 미로 찾기 알고리즘**
  - 알고리즘
    1. 스택을 초기화한다.
    2. 출발지점을 스택에 push한다.
    3. 아래 Description을 따라 미로를 이동한다.

    3-1. 이동 가능한 지점 중 한 곳을 stack에 push 하고 이동한다.
    이동 가능한 지점은 동->서->남->북 순서로 조사한다.

    3-2. 벽 혹은 이미 지나온 점으로 사방이 둘러 쌓여있다면 stack pop을 수행하여 이전 위치로 이동한다.

    4. 도착점을 찾을 때까지 3을 반복한다.

■ 모식도



- **미로 생성을 위한 2차원 배열 동적 할당 방법**

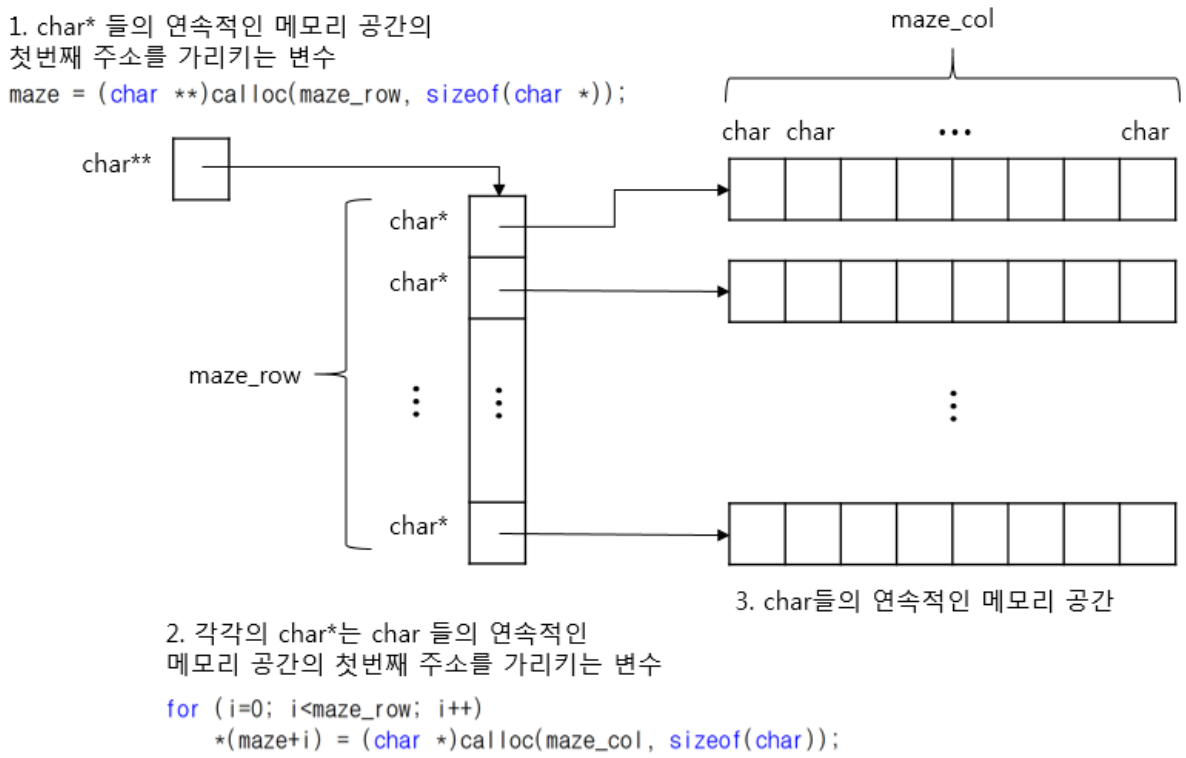
  미로 맵 파일은 character로 되어 있다. 미로 맵은 2차원 char 배열로 구현 하도록 한다.
  현재 위치 (i, j)가 벽이면 a[i][j] 는 '#', 방문하지 않은 길이면 a[i][j] 는 ' ', 방문한 위치라면 적당한 문자를 할당하면 된다.
  미로 맵을 위한 2차원 char 배열 생성은 아래를 참조한다.

```
char **maze;
int maze_row = N, maze_col = M, i;

// 2차원 char 배열 메모리 할당
maze = (char **)calloc(maze_row, sizeof(char *));
for (i=0; i<maze_row; i++)
    *(maze+i) = (char *)calloc(maze_col, sizeof(char));
```

  ■ char **maze는 2차원 배열을 동적 할당 받기 위한 포인터 변수이다.
  ■ 위의 예에서, maze_row와 maze_col에는 파일로부터 읽은 값인 N값과 M값이 할당된다고 생각하자.

■ **위의 2차원 배열 동적 할당에 대한 모식도**



- 구현을 위해 Problem1 에서 구현한 함수와 아래와 같은 함수를 사용하게 된다. 반드시 아래의 함수를 구현해야 하는 것은 아니다.

char ** readMaze(char *file_name, int *, int *);
파일로부터 미로의 모양을 읽어 들여 char ** 타입의 행렬을 반환한다. fopen(), fclose() 모두 이 함수 안에서 이루어지도록 한다.

void printMaze(charmaze, int, int);**
미로 맵을 출력한다.

Stack *findPath(char **maze, int, int);
주어진 미로 맵에서 시작점부터 출구까지의 길을 Stack에 담아 돌려준다.

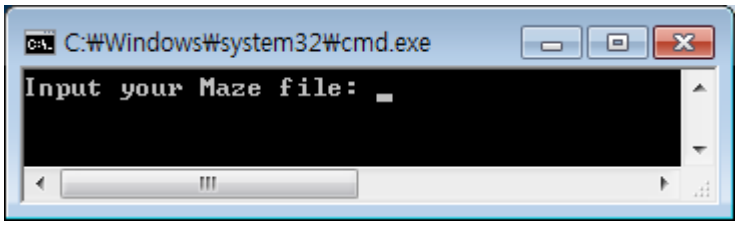
void printPath(char **maze, int, int, Stack *);
미로의 길을 좌표로 출력하는데, 반드시 시작점에서부터 출구로 표현되어야 한다.

void freeMaze(char **maze, int);
미로 맵을 생성하기 위해 동적할당 된 메모리를 free 시킨다.

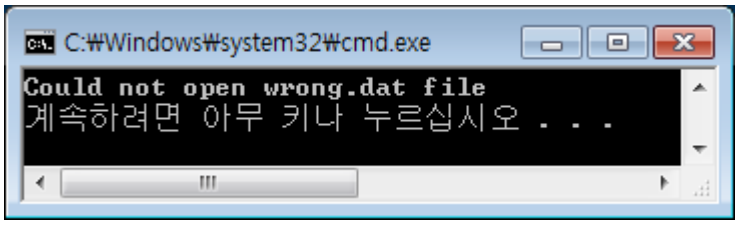
(실행 예)

(1) 길이 있는 경우 – maze0.dat

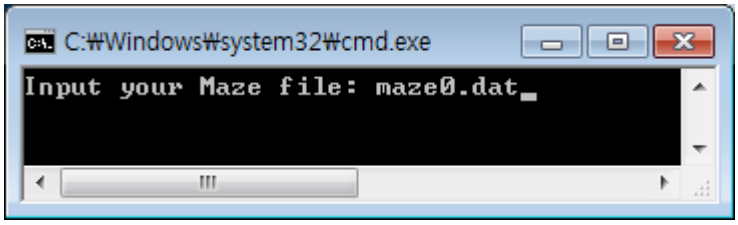
프로그램을 실행하면 아래와 같이 미로 맵을 생성하기 위해 읽어야 할 파일명을 요구한다. 입력 받는 파일명에는 공백이 없다고 가정한다.



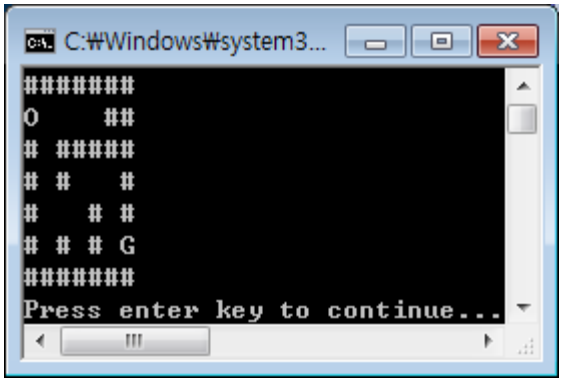
존재하지 않는 파일명 wrong.dat 를 입력한 경우에는 아래와 같은 에러 메시지를 출력 후, 프로그램을 종료한다.



존재하는 파일의 파일명 maze0.dat 를 아래와 같이 입력 후 엔터키를 누르면, 입력파일을 읽어서 미로 맵의 초기 상태를 출력한다.



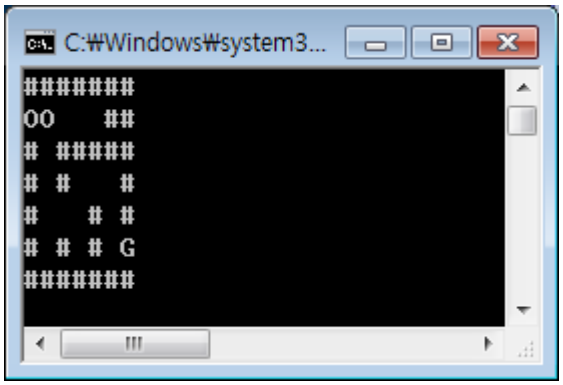
0. 초기상태



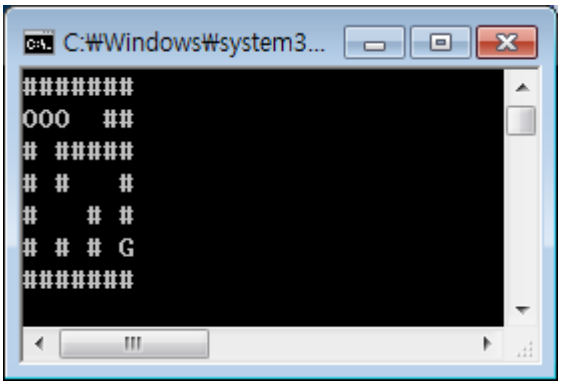
초기 상태에서 엔터를 입력하면 길 찾기가 진행된다. 미로 맵에서 위치를 이동 할 때마다 화면을 아래와 같이 갱신해준다. 화면을 갱신하기 전에 system("cls")를 이용하여 화면을 지운 후 미로 맵에서 현재까지 알아낸 입구에서 출구까지의 길을 찾은 경로를 출력한다.

아래에서 O는 경로이며 . 은 방문은 했지만 경로는 아니며, 공백은 방문을 하지 않은 경우이다.

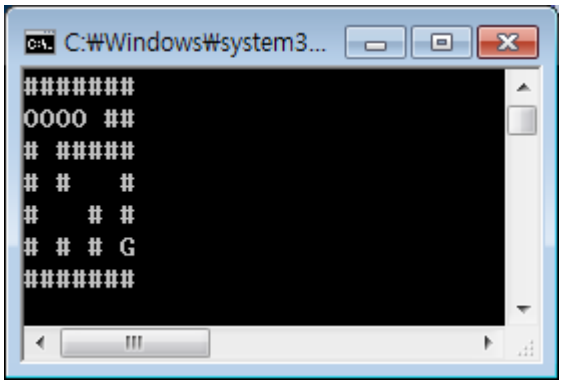
1.



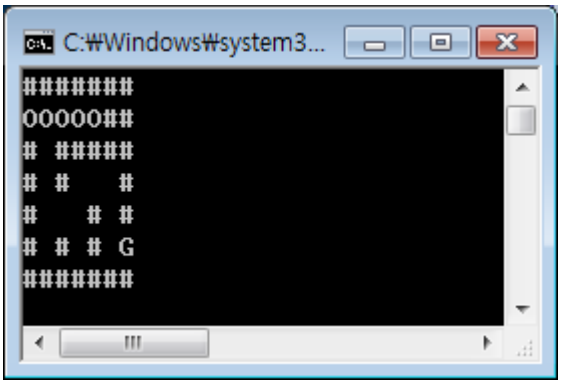
2.



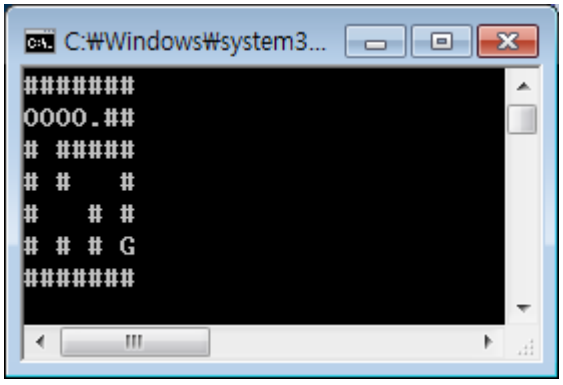
3.



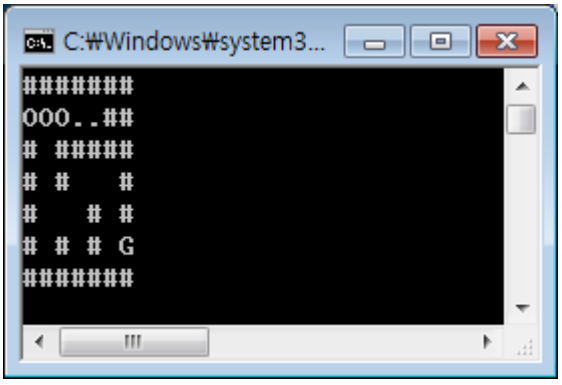
4.



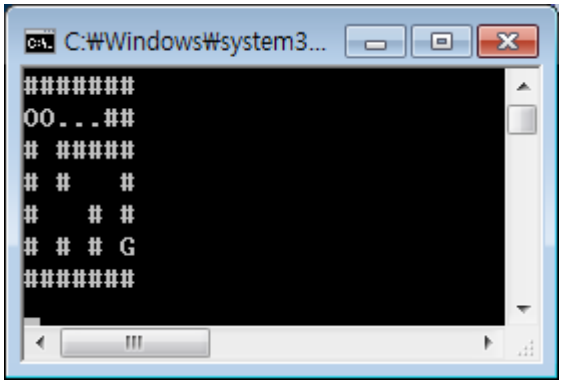
5.



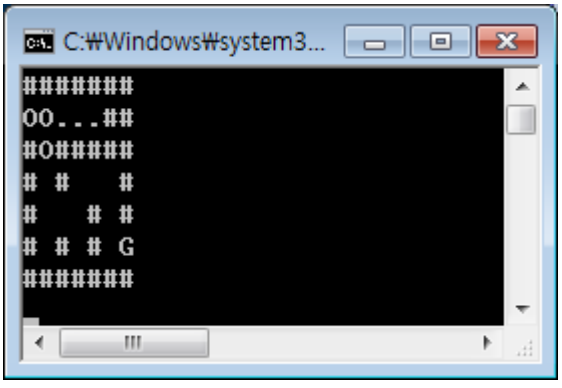
6.



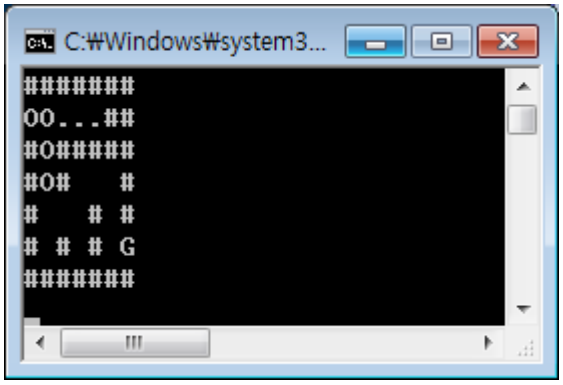
7.



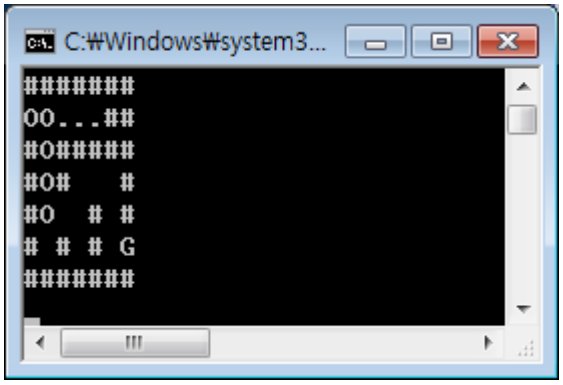
8.



9.



10.



중간 과정 생략

출구를 찾은 경우, 아래와 같이 시작점부터 출구까지 좌표로 출력하고 프로그램은 종료된다.

```
########
OO...##
#O######
#O#OOO#
#OOO#O#
# #.#OG
########
[ 1,  0]->[ 1,  1]->[ 2,  1]->[ 3,  1]->[ 4,  1]->[ 4,  2]->[ 4,  3]->[ 3,  3]->
[ 3,  4]->[ 3,  5]->[ 4,  5]->[ 5,  5]->[ 5,  6]
계속하려면 아무 키나 누르십시오 . . .
```

경로 출력 형식은 **printf("[%2d, %2d]", row, col);** 문을 사용한다.

(2) 길이 있는 경우 – maze1.dat

- **초기화면**

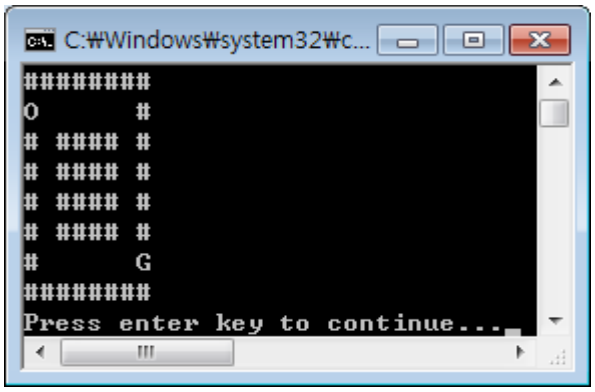
```
##############################
O #      ######  ### # ## ###
#   ###  ###### ### # ## ###
# ######           #      #
#  ##### ## ######## ## # #
##   ##  ##          ## # #
####  ########## ######## # G
##############################
Press enter key to continue...
```

- **마지막 결과화면**

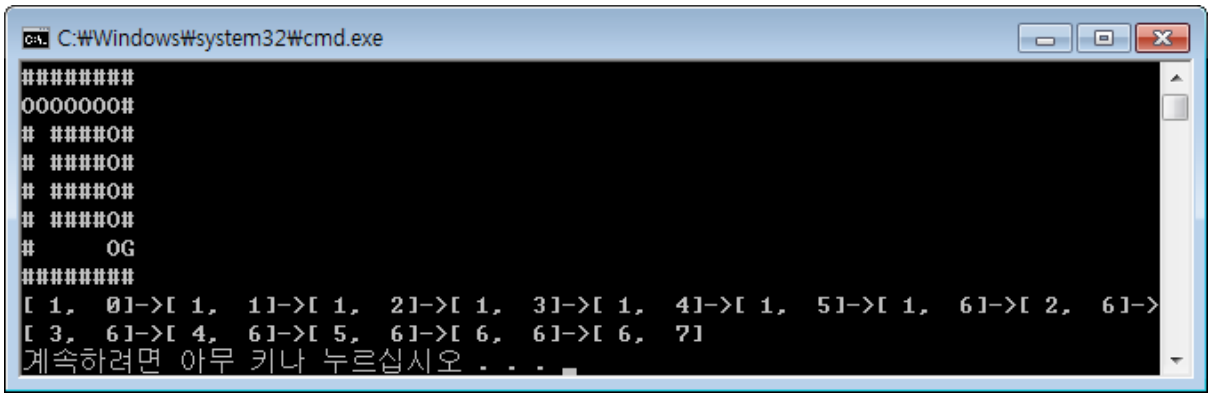
```
##############################
OO#OOOOO#####..###.# ## ###
#OOO###OO#####.###.# ## ###
# ######OOOO.......#OOOOOO#
#  ##### ##O##########O## #O#
##   ##  ##OOOOOOOOOO## #O#
####  ######### ######## #OG
##############################
[ 1,  0]->[ 1,  1]->[ 2,  1]->[ 2,  2]->[ 2,  3]->[ 1,  3]->[ 1,  4]->[ 1,  5]->
[ 1,  6]->[ 1,  7]->[ 2,  7]->[ 2,  8]->[ 3,  8]->[ 3,  9]->[ 3, 10]->[ 3, 11]->
[ 4, 11]->[ 5, 11]->[ 5, 12]->[ 5, 13]->[ 5, 14]->[ 5, 15]->[ 5, 16]->[ 5, 17]->
[ 5, 18]->[ 5, 19]->[ 5, 20]->[ 4, 20]->[ 3, 20]->[ 3, 21]->[ 3, 22]->[ 3, 23]->
[ 3, 24]->[ 3, 25]->[ 4, 25]->[ 5, 25]->[ 6, 25]->[ 6, 26]
계속하려면 아무 키나 누르십시오 . . .
```

(3) 길이 있는 경우 – maze2.dat

- **초기화면**



- **마지막 결과화면**



(4) 길이 없는 경우 – maze3.dat

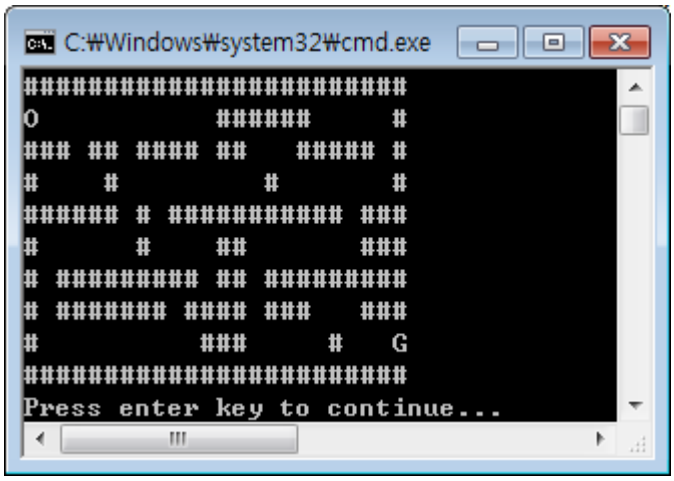
- **초기화면**

- **마지막 결과화면:**

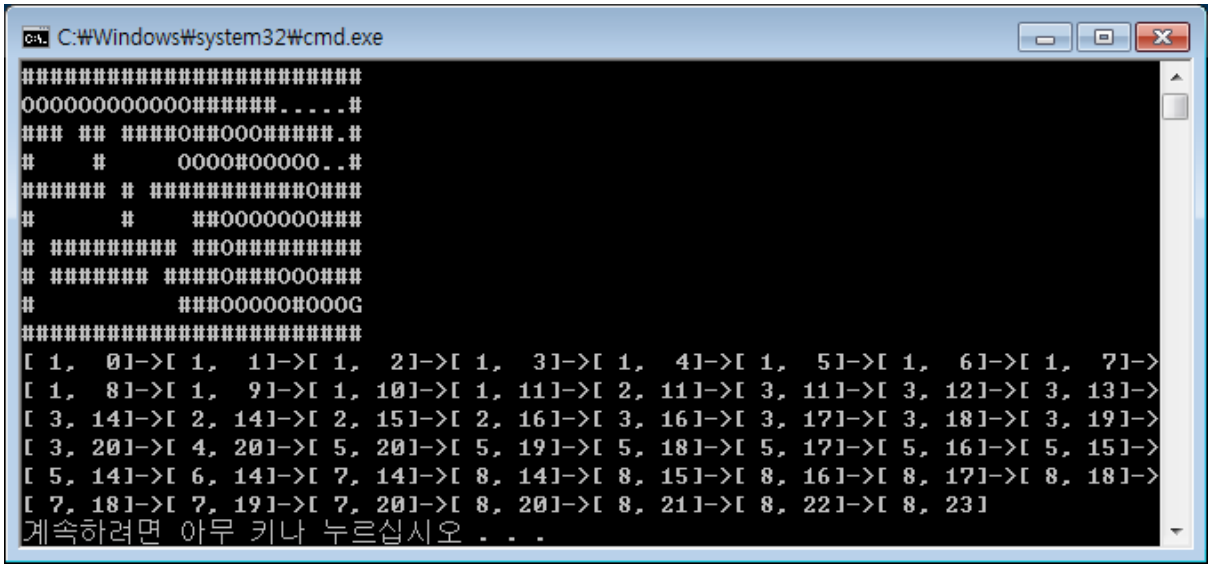
시작점부터 출구까지의 경로를 찾지 못했을 경우엔 아래와 같이 출력한다.

(5) 길이 있는 경우 – maze4.dat

- **초기화면**



- **마지막 출력화면**



(주의사항)

- 파일 이름은 **assn4_2.c** 로 저장한다.
- 보고서는 **"assn4.doc"** or **"assn4.hwp"**로 저장 한다. (보고서는 통합하여 작성)
- 화면 출력은 반드시 실행 예와 같아야 한다.
- 프로그램은 최소한 첨부된 sample data (maze1.dat, maze2.dat, maze3.dat, maze4.dat)에 대해서 올바른 결과를 출력해야 한다. 그 결과를 파일에 포함시킬 것.
- 동적할당 받은 메모리는 프로그램 종료시 free 시키도록 한다.