



---

## Programming 연습문제 1

---

과목	자료구조.
제출 일자	2019.05.06
담당 교수	강현철 교수님
학 과	컴퓨터공학과
학 번	20172129
이 름	박예슬

## 1번 문제

(a) 제시된 프로그램은 교재에서 공부한 프로그램과 어떤 차이점이 있는지 설명하시오.

가장 주된 차이점은 책과 다르게 미로의 경로를 출력할 때, `print_record()` 함수를 통해 `stack`에 저장된 `dir`의 값과 이동 방향까지 출력해주었다는 점이다. `print_record()` 함수 내부의 코드 중에서 `switch (dir - 1)`를 보면 -1을 해주는 것을 볼 수 있다. 그 이유는 `dir`에 저장된 값이 내가 이동한 방향이 아니라, Backtrack 했을 때 이어서 실행해야 되는 방향이기에 현재 내가 이동한 방향보다 값이 1만큼 크다. 따라서 `dir - 1`을 하고 `switch` 구문을 실행하여, 올바른 이동 방향을 알파벳으로 출력해주었다.

그 외의 차이점은 `offsets move[9];`로 선언하여 방향 이동을 `move[0]`부터 `move[8]`까지가 아니라, `move[1]`부터 `move[9]`까지로 정의하였다. 따라서 처음에 시작점의 데이터를 `stack`에 저장할 때, `stack[0].dir = 2`로 선언한다. (N 방향이 1이고, NE방향이 2이다. 시작점에서는 N방향으로 갈 수 없다) 이 밖에도, `while` 문에서 `dir < 8`이 아니라 `dir <= 8`으로 바꾸는 등의 차이점들이 있다.

(b) 제시된 프로그램에서 발견한 `path`를 출력할 때 이동 방향을 출력하고 있는데 단, 마지막 이동인 `EXIT`이로의 이동 방향은 출력하지 않는다. 이를 출력하려면 제시된 프로그램을 어떻게 수정해야 하는지 설명하시오.

현재 코드에서는 `if (next_row == NUM_ROWS && next_col == NUM_COLS)`에서 `EXIT`가 있다면 `found`를 `TRUE`로 바꿔버리고 조건문을 빠져나간다. 따라서 `else if` 조건문을 실행하지 못하므로, `EXIT`이로의 이동 방향에 대한 정보가 `stack`에 저장되지 못한다.

이를 수정하려면 밑의 'else if' 조건문을 'if' 조건문으로 바꾸어, 위의 if 조건문을 실행하고서도 `stack`에 add될 수 있도록 해주어야 한다. 따라서 코드는 다음과 같이 바뀐다.

```
if (next_row == NUM_ROWS && next_col == NUM_COLS) found = TRUE;

if (!maze[next_row][next_col] && !mark[next_row][next_col]) {
    mark[next_row][next_col] = 1;
    position.row = row;    position.col = col;
    position.dir = ++dir;
    add(position);
    row = next_row;        col = next_col; dir = 1;
}
else ++dir;
```

`EXIT`에 도달하기 전 마지막 좌표가 `stack`에 저장되었으므로 미로 결과를 print할 때, 기존에 있던 `printf("%2d%5d\n", row, col);` 코드는 빼 주어야 한다. 왜냐하면, 바로 위에서 출력되는

```
for (i = 0; i <= top; i++)    print_record(stack[i].row, stack[i].col, stack[i].dir);
```

코드에서 `row`와 `col` 값은 이미 `stack top`으로 출력되었기 때문이다.

## 2번 문제

(a) eclass에 제시된 프로그램에서 path() 함수의 빈 곳 '가', '나'에 들어갈 C코드를 쓰시오.

가)

```
//stack top 정보 불러옴
row = position.row; col = position.col; dir_vector = position.dir_vector;
dir = get_next_dir(dir_vector); //다음 방향 어디로 갈 지 랜덤하게 불러옴
if (dir != 8) { // Backtrack 하고 갈 수 있는 방향이 있을 때
    position.dir_vector += (int)pow(2.0, (double)dir);
    push(position);
}
```

나)

```
row = nextRow; col = nextCol;
maze[row][col] = -1;
dir_vector = init_dir_vector(row, col);

if (dir_vector == -1) found = TRUE; // to report that a path is found
else { //stack에 push하고, 다음 방향 어디로 갈 지 랜덤하게 불러옴
    position.row = row; position.col = col; position.dir_vector = dir_vector;
    push(position);
    dir = get_next_dir(dir_vector);
}
```

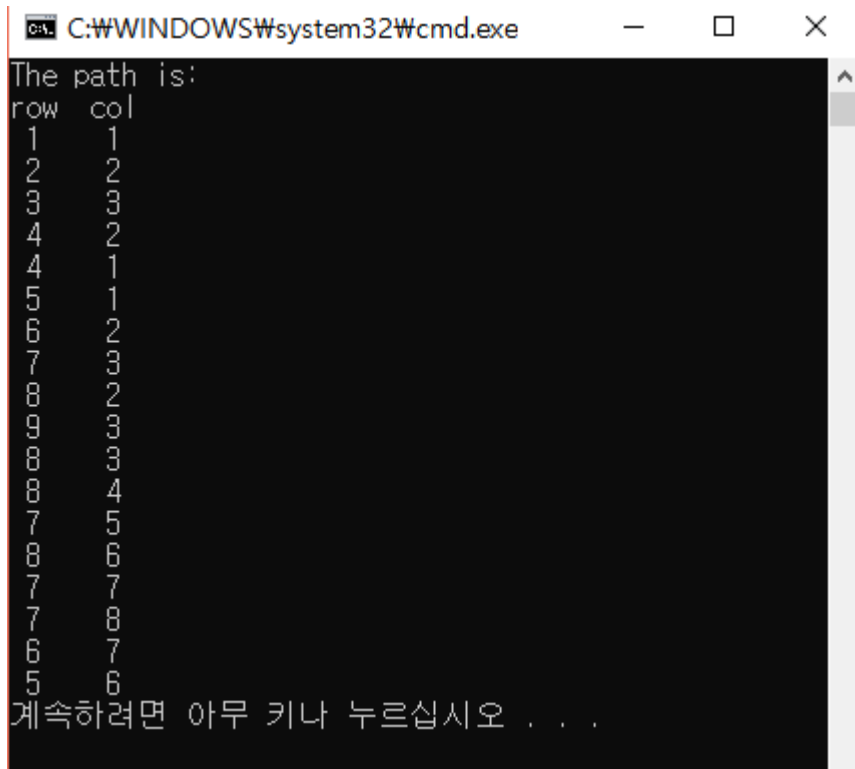
(b) '가', '나'에 작성한 C코드에 대해 왜 그렇게 작성하였는지 이유, 불필요한 연산을 수행한 것은 없는 지, 기타 설명이 필요한 부분에 대해 설명하시오.

'가'의 코드는 stack top 데이터가 pop() 된 이후의 코드다. 즉, 가의 코드가 실행되는 경우는 미로의 첫 시작과 갈 수 있는 길이 없어서 Backtrack이 될 때 두 가지 경우로 나뉜다. 따라서, pop함수를 통해 이전 위치로 돌아왔을 때 다시 다른 곳으로 이동할 수 있다면(dir이 8이 아닐 때), 기존에 저장되어 있던 dir\_vector의 데이터를 수정해서 push 해줘야 한다. 새로 이동할 방향이 나타내는 번호를 N (dir = N)이라고 가정하면, dir\_vector 값에서  $2^N$  만큼 더해 줘야한다. 왜냐하면 이진수로 고쳤을 때, N번째 bit가 이제 지나간 자리가 되므로 0에서 1으로 바뀌기 때문이다.

'나'의 코드는 Backtrack없이 계속해서 stack에 데이터를 쌓아 나가는 while문 안의 코드이다. 랜덤으로 정해진 방향으로 이동한 위치를 row, col 변수에 저장하고, 이동한 위치를 체크해주기 위해 maze값을 -1로 바꿔준다. 그 후, stack에 저장하기 전에 우선 주변에 목적지가 있는지 확인해준다. init\_dir\_vector() 함수를 살펴보면 주변에 목적지가 있을 시, -1을 return하도록 코드가 짜여 있다. 따라서 init\_dir\_vector() 함수를 실행했을 때 -1을 return하면 found를 TRUE로 바꿔주고, 아니면 현재 위치의 주변 방향에 대한 정보가 return된 것이므로 stack에 저장(push)해준다. 그리고 다시 새로 이동할 수 있는 방향을 랜덤으로 불러온다.

(c) (a)에서 완성한 프로그램을 실행한 화면, 사용한 10 by 10 maze 데이터 및 목적지 좌표를 제시하시오. 설명이 필요한 부분이 있으면 설명하시오.

<프로그램 실행 화면>

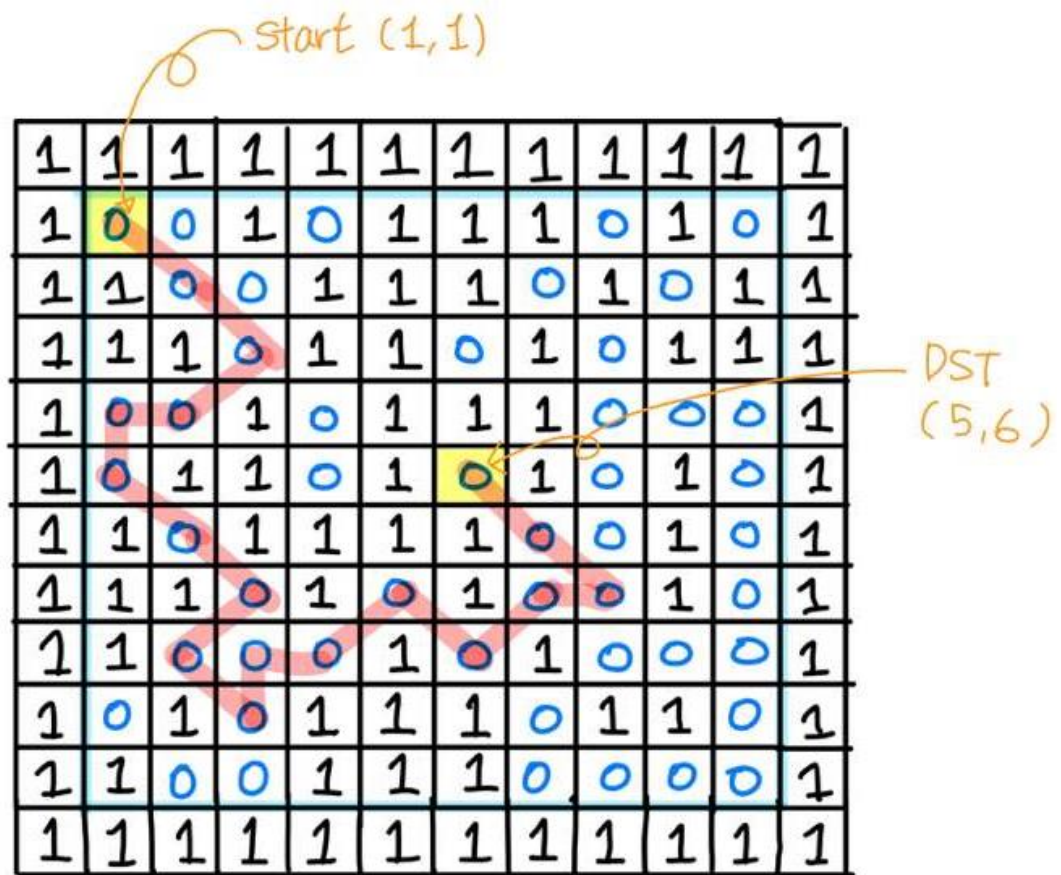


```
C:\WINDOWS\system32\cmd.exe
The path is:
row  col
1    1
2    2
3    3
4    2
4    1
5    1
6    2
7    3
8    2
9    3
8    3
8    4
7    5
8    6
7    7
7    8
6    7
5    6
계속하려면 아무 키나 누르십시오 . . .
```

<10 by 10 maze 데이터>

```
{ 0,0,1,0,1,1,1,0,1,0 },
{ 1,0,0,1,1,1,0,1,0,1 },
{ 1,1,0,1,1,0,1,0,1,1 },
{ 0,0,1,0,1,1,1,0,0,0 },
{ 0,1,1,0,1,0,1,0,1,0 },
{ 1,0,1,1,1,1,0,0,1,0 },
{ 1,1,0,1,0,1,0,0,1,0 },
{ 1,0,0,0,1,0,1,0,0,0 },
{ 0,1,0,1,1,1,0,1,1,0 },
{ 1,0,0,1,1,1,0,0,0,0 }
```

<목적지 좌표 : (5,6) >



▲ 직접 그린 프로그램 실행 경로