Data Structure

(Looking for Maze)

제출일	2013.10.17
이 름	박 준 영
학 번	201020268

1. Algorithm

2. Program Structure

2.1 Data Structure

2.2 Structure Chart

3. Program Complexity

3.1 Space Complexity

3.2 Time Complexity

4. Reveiw

1. Algorithm

- 본 과제에서 가장 중심적으로 쓰인 Algorithm은 파일을 읽어오고, 파일로부터 읽어온 데이터를 기반으로 미로를 찾는 것이다. 먼저 파일을 입력하여 프로그램내의 변수에 저장하는 과정은 다음과 같다.

```
while(fileopen is NULL){
filename_rt = file consist data needed in maze program;
filename_+w = file writen output of program;
filename_rt = input;
filename_+w = input;
if( filename_rt or filename_+w = ‘q’ or ‘Q’)
      End program;
else
fileopen(fileopen_rt);
fileopen(fileopen_+w);
}
```

두 개의 파일명을 입력받는다. 두 개의 파일명중 하나라도 ‘q’ 혹은 ‘Q’가 입력되면 Program은 종료된다. ‘q’나 ‘Q’가 아니라면 filename_rt는 text형식으로 해당 폴더에 있는 파일을 열기 위한 스트림을 생성하고 해당 파일을 연다. filename_+w는 폴더에 파일명과 같은 파일이 있다면 그 파일을 열고, 없다면 입력된 파일명으로 파일을 생성 후 연다.

table 1 .Algorithm of file open

```
indata=fgets(fileopen_rt);
tok = strtok(indata,“x”);
x = tok‘s int type meaning;
tok = strtok(indata,“\n”);
y = tok’s int type meaning;
```

파일의 첫 줄을 읽어온다. 첫 줄을 indata라는 char형 배열(string)에 저장 후 token을 이용하여, “x”를 중심으로 미로의 크기를 읽어온다.

table 2. Maze map size

```
while(fgets(fileopen_rt)!=“-----”)
{
  tok = strtok(indata,“ ”);
  x = tok‘s int type meaning;
  tok = strtok(indata,“\n”);
  y = tok’s int type meaning;
  maze[x][y] = 0; // another points of maze = 1;
}
```

첫 줄과 마찬가지로 읽어와, (x, y)의 좌표를 maze map의 2차원 배열에 저장한다. 이는 미로 속에서 지나갈 수 있는 길을 나타내기 위함이다. 이와 같은 과정을 “-----”의 문자열이 나타날 때 까지 반복한다.

table 3. Possible point in Maze map

while(fileopen!=NULL)
{
fscanf(Starting point_x, y, Ending point_x, y and diretion)
}

“-----” 문자열 이후, 미로를 시작할 지점과 도착점, 먼저 scan할 방향을 입력받아 미리 설정해 놓은 변수에 저장한다.

table 5. starting point, end point and direction

위와 같은 Algorithm으로 파일을 불러왔다. 다음은 미로를 찾기 위한 Algorithm이다.

for(number of task)
{
<row,col,dir>
<nextrow, nextcol> = points that scanned to move at next turn
if(next turn is Possible point and no marking)
{
push(row, col)
nextrow, nextcol <- row, col
}
else if(next turn is no marking and impossible or possible and marking)
{
if(all direction in impossible)
<row,col>=pop
else
next turn of different direction.
}
if(row and col = end points)
{
end(this tast turn)
}
if(stack is empty and all around direction is blocked)
there is no path
}

위에서 입력받은 시작점, 도착점으로 총 시도 횟수를 알아와, 미로 찾기를 그 횟수만큼 시행한다. 먼저 next turn에 이동할 point를 direction 정보를 이용해 정한 뒤, 다음 시도에 이동할 좌표를 구한다. 해당좌표에 도달한 적이 있다면 marking 되어 있을 것이다. marking도 되어 있지 않고, 이동 가능한 경로라면 현재위치를 stack에 저장 후, 그 위치로 이동한다. 하지만 둘 중 하나라도 가능하지 않다면 방향을 바꾸어 다시 시도한다. 나아갈 수 있는 방향을 찾을 때까지 반복 시행하다가, 모든 경로가 막혀있다고(marking or impossible route) 판단이 되면 stack에 저장되어 있는 가장 최신 기록을 불러와 그 위치에서 다른 경로를 검색한다. 이 과정을 반복 시행하다가, 도착점에 도달 시, 다음 경로를 시도한다. 또한 어떤 위치에서 stack에서 불러올 정보도 없고, 사방이 막혀있다면 미로가 존재하지 않는 것으로 판단하고 메시지를 출력한다.

table 5. Algorithm of maze process

2.Program structure

2.1 Data structure

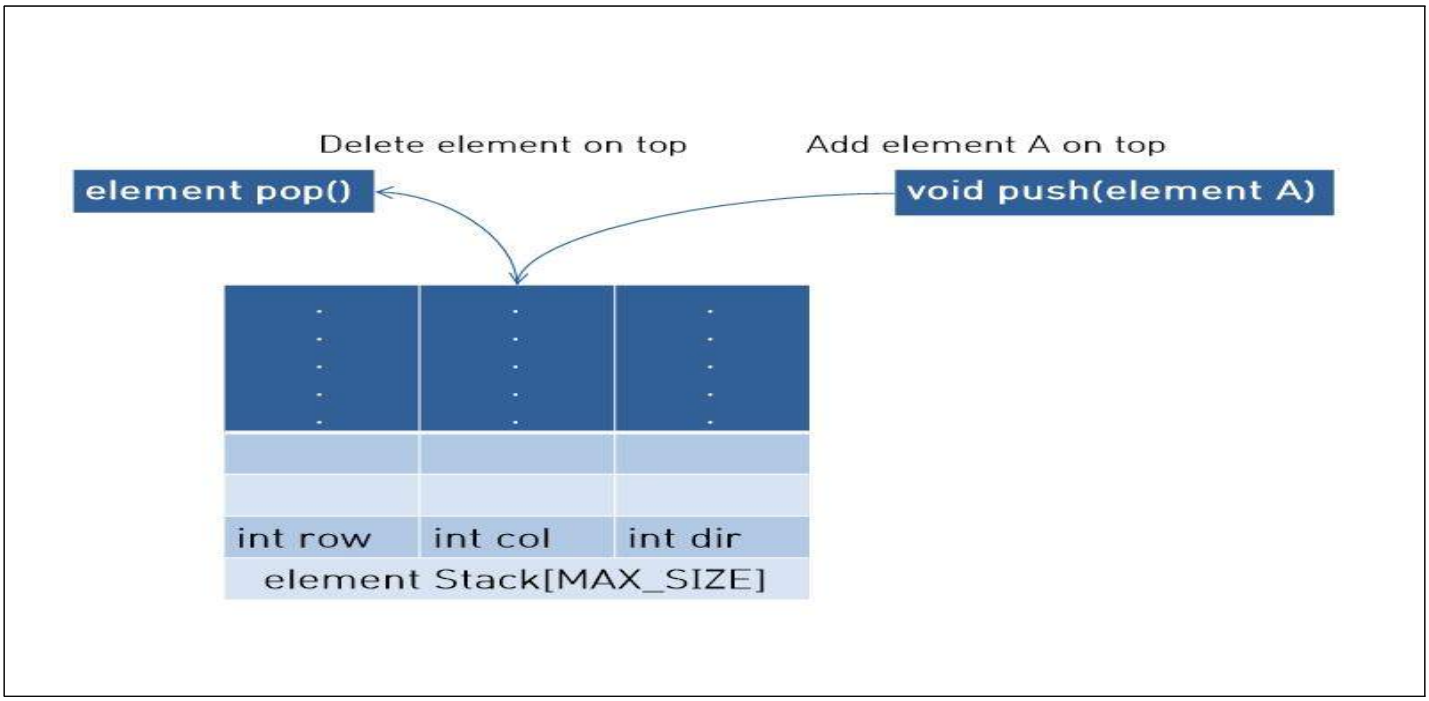

figure 1. Stack

- Stack이라는 Data는 maze process에서 경로를 이동할 때, 지나가는 경로를 저장하고 다음 경로에서 이동할 경로가 없을 때, 직전에 있었던 경로로 재이동하기 위해 사용된다. 현재 경로에서 이동할 다음 경로가 존재할 때, push를 통해 현재경로를 stack에 저장하고, 다음경로가 존재하지 않을 때 pop함수를 이용해 직전 data를 읽어온다.

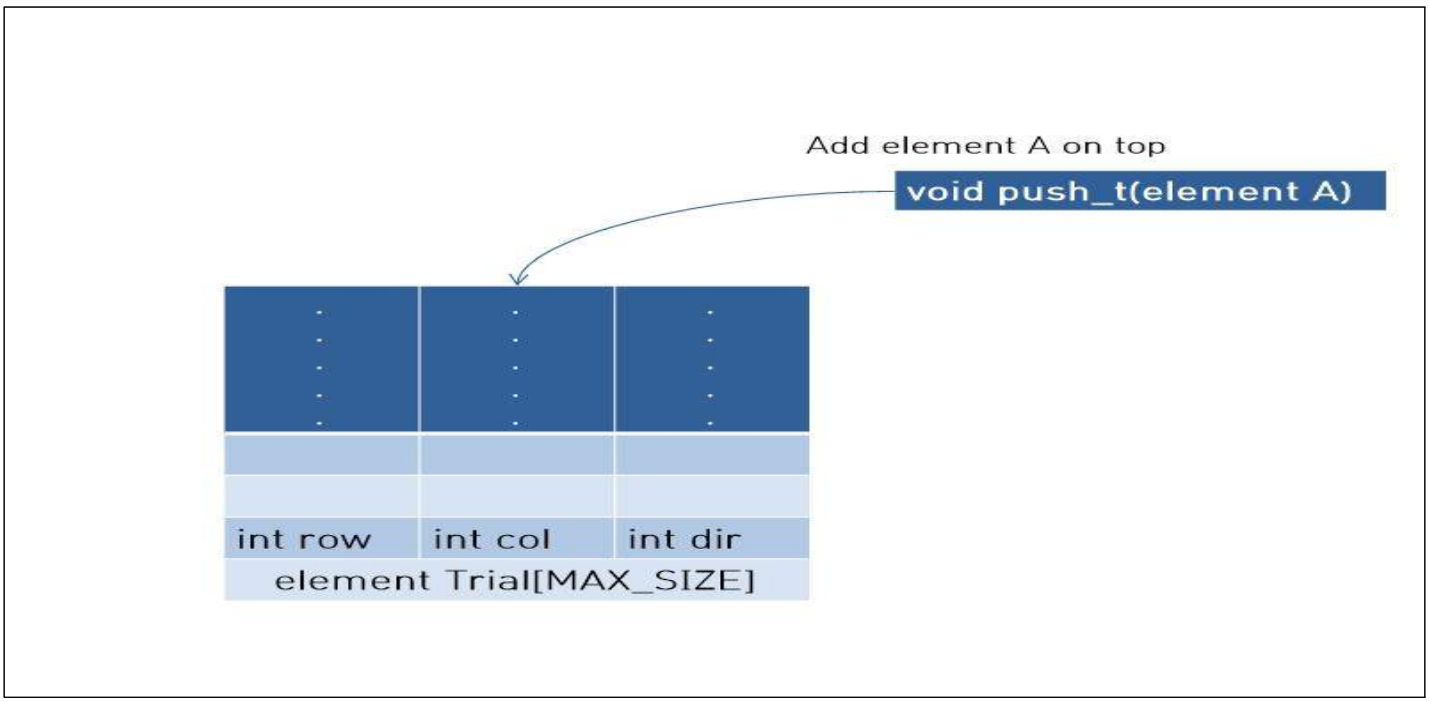

figure 2. Trial

-Trial이라는 data는 과제에서 요구하는 조건이외에 임의로 만든 data이다. Stack이라는 방식을 사용하고 있으나, path가 순차적으로 저장되는 stack이라는 data와는 다르게 pop과 같은 함수처럼 저장되어 있는 data를 꺼내는 함수는 존재하지 않는다. 미로를 찾는 과정에서 거쳤던 모든 경로를 출력하는 trial의 특성 때문에 저장은 하지만 제거는 하지 않는다.

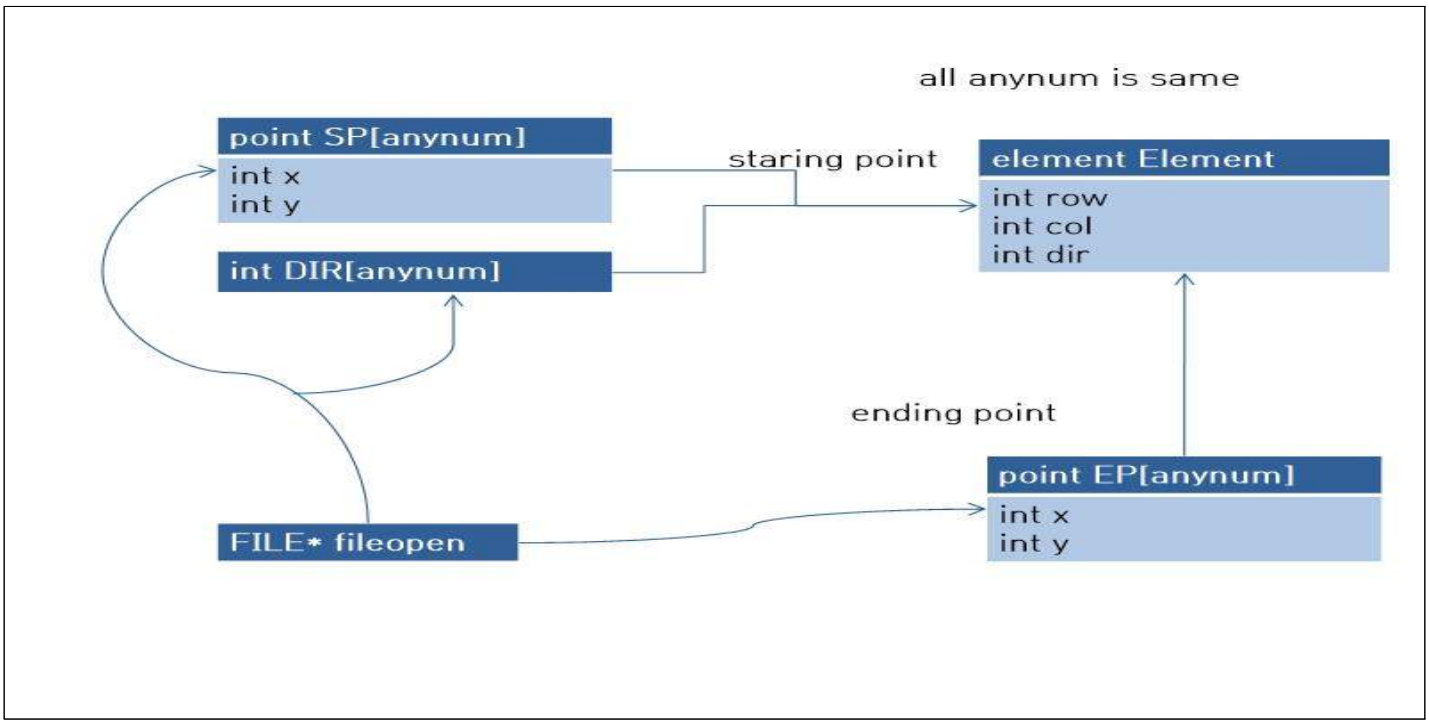

figure 3. Point

- point라는 구조를 가진 SP, EP배열과 int형의 DIR 배열은 file로부터 data를 가지고 온다. I번째 data에 I번째 시도할 start point와 end point, 방향이 저장된다. 이는 미로를 찾는 과정에서 element형 구조체 Element의 초기값을 설정해주고, 끝내야할 지점을 설정해준다. 또 DIR은 다음 경로를 확인하기 위해 사용된다.

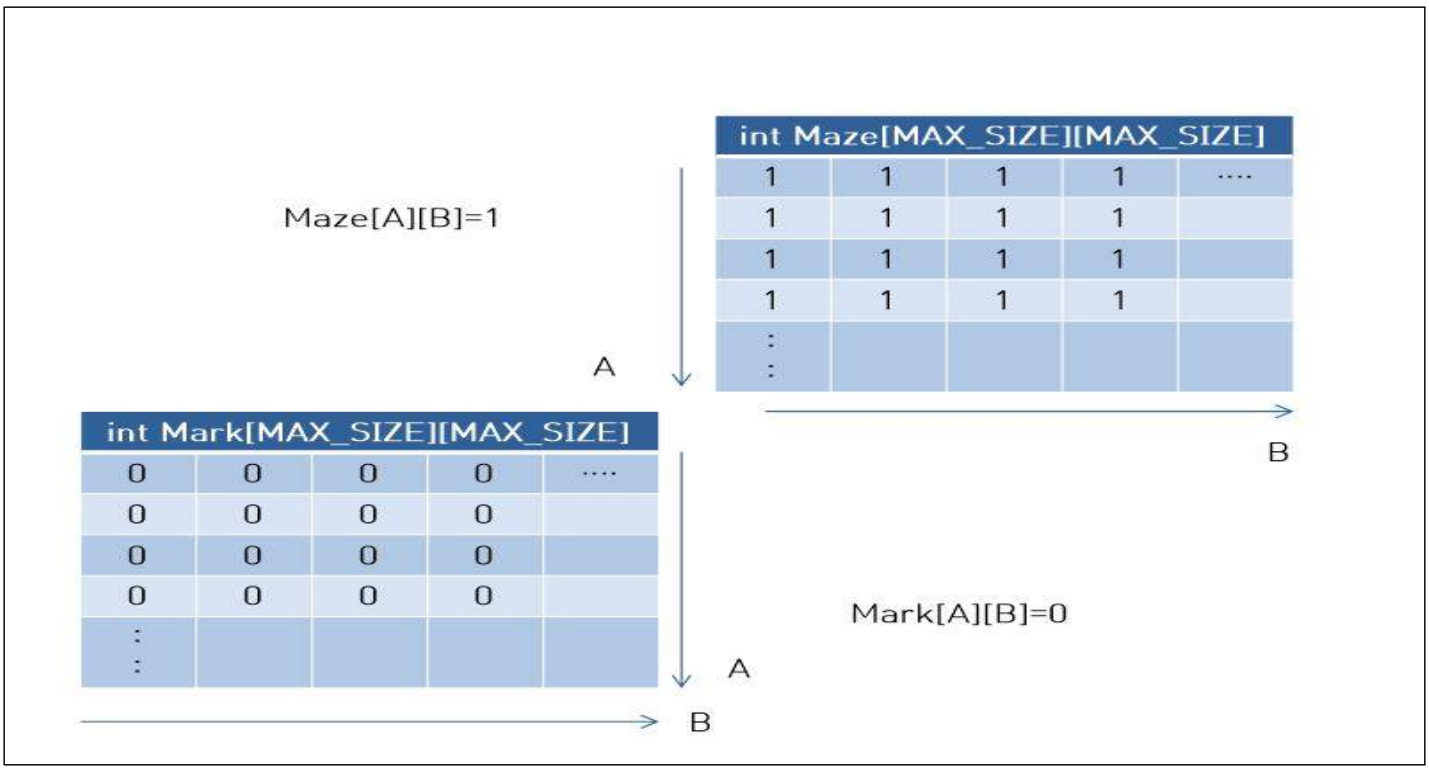

figure 4. Maze and Mark

- maze 2차원 배열은 최초에 1으로 모두 초기화한다. 파일에서 읽어온 이동가능 한 좌표에 0을 입력한다. mark 2차원 배열은 반대로 0로 초기화한다. 또한 미로 찾기 과정 중, 지나온 경로의 mark는 1로 입력해, 지나왔던 경로임을 표시한다. 이 2개의 2차원 배열은 파일에서 입력받은 map의 크기보다 +2된 값으로 선언된다. map이외의 값을 1로 만들어 벽으로 인식하게 하기 위함이다.

2.2 Structure chart

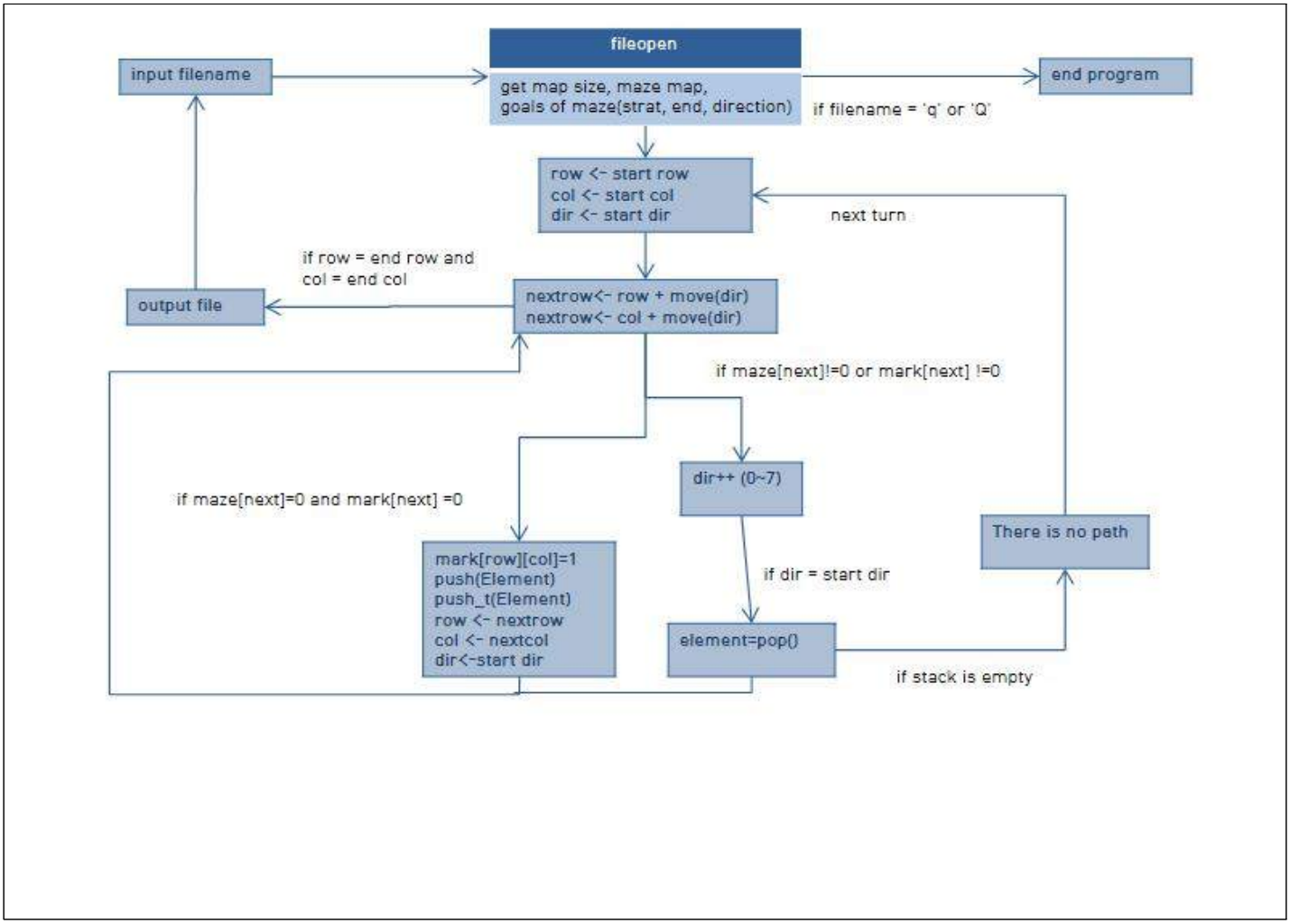

figure 5. flow chart

- figure 5는 해당 program의 flowchar이고, figure 6은 structure chart이다. 먼저 크게 file에서 입력받는 부분과 미로를 찾는 부분으로 나뉜다. file에서 maze map, size, goal(start point, end poinr and direction)을 입력받는다. 입력받은 변수들로 Maze 부분을 진행한다. 먼저 size에 맞는 maze 2차원 배열, mark 2차원 배열을 생성하고 이를 초기화한다. 시작점과 입력받은 방향을 기점으로 다음에 진행할 좌표를 nextcol, nextrow에 입력한다. next변수들에 의한 다음 좌표가 진행 가능한 좌표일 때, stack과 trial에 현 좌표를 입력하고, 다음 경로로 이동한다. 진행 가능하지 않는 좌표라면 방향을 바꿔 다시 한 번 진행가능 여부를 판단한다. 모든 방향이 진행불가 할 때는, stack에 있는 가장 최근경로를 불러온다. 이를 반복하다가 end point에 도달하면 출력파일에 stack에 쌓여있는 경로를 역순으로 출력한다. 모든 경로가 막혀있고 stack에서 빼올 data도 없다면, 사용자에게 불가능하다는 메시지를 출력하고 maze part를 종료한다. 그리고 다시 파일명을 입력받아 위의 과정을 반복한다. 도중에 파일명으로 q 혹은 Q가 입력된다면 프로그램을 종료한다.

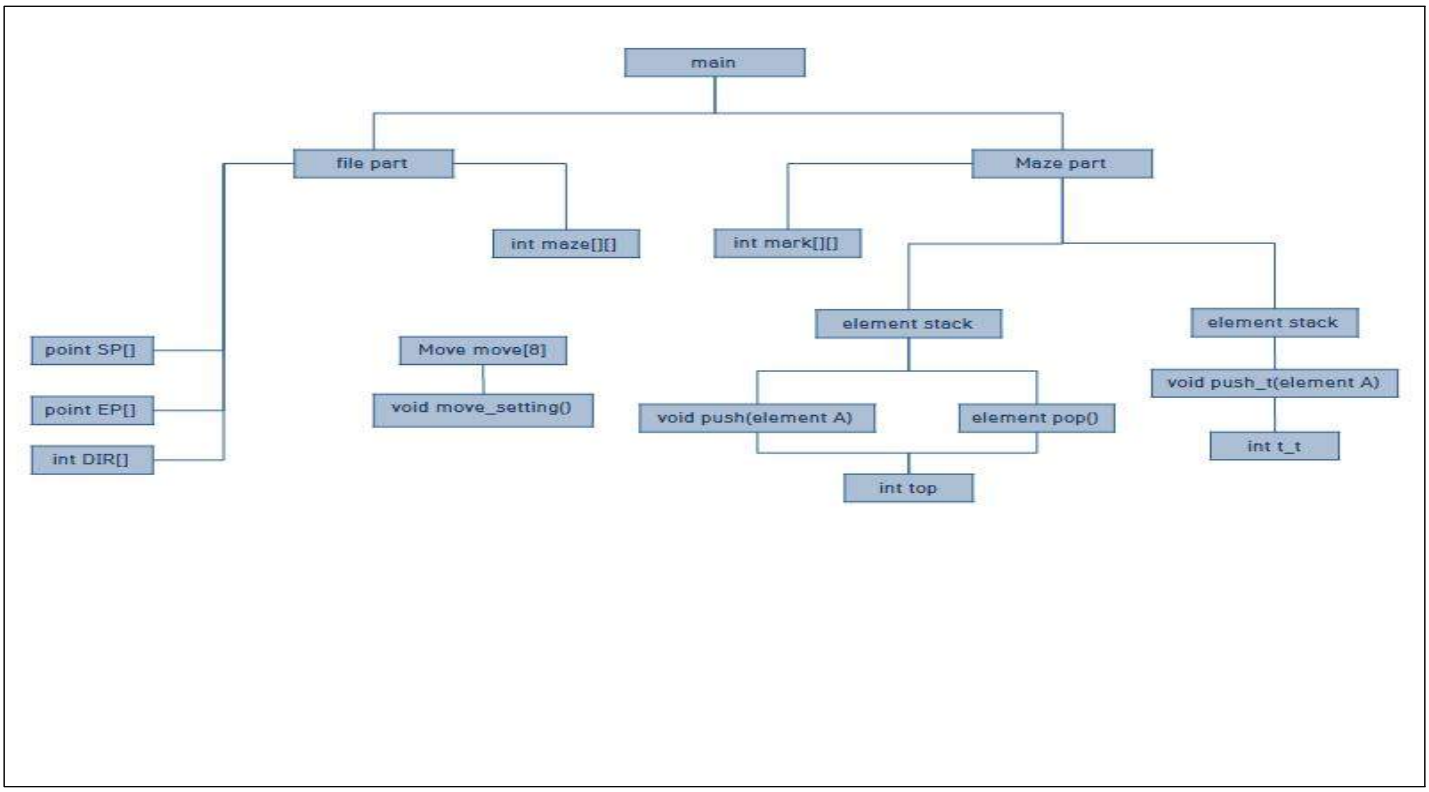

figure 6. structure chart

3. Program Complexity Analysis

3.1 Space Complexity

- 위 프로그램의 Space Complexity는 메모리에 대해 분석된다.

int Direction(char* c)			
Category	Name	Type	Number of bytes
input parameter	C	char *	4
return address			4
Total per recursive call			8
Equation		$S_{Direxion}(n) = 8n$	

table 6. Space compelxity of function Direction

void push(element item)				
Category	Name	Type		Number of bytes
input parameter	item	element	int x	4
			int y	4
return address				0
Total per recursive call				8
Equation		$S_{push}(n) = 8n$		

table 7. Space compelxity of function push

void push_t(element item)				
Category	Name	Type		Number of bytes
input parameter	item	element	int x	4
			int y	4
return address				0
Total per recursive call				8
Equation		$S_{push\_t}(n) = 8n$		

table 8. Space compelxity of function push_t

element pop()				
Category	Name	Type		Number of bytes
return address	stack[top--]	element	int row	4
			int col	4
			int dir	4
Total per recursive call				12
Equation		$S_{pop}(n) = 12n$		

table 9. Space compelxity of function pop

void move_setting()			
Category	Name	Type	Number of bytes
local parameter	i	int	4
Total per recursive call			4
Equation		$S_{move\_setting}(n) = 4n$	

table 10. Space compelxity of function move_setting

function	Equation	Asymtotic Notation		
		Big-Oh	Big-Omega	Big-Theta
int Direction(char* C)	$8n$	$O(n^2)$	$\Omega(n)$	$\theta(n^2)$
void move_setting()	$4n$	$O(n^2)$	$\Omega(n)$	$\theta(n^2)$
void push (element item)	$8n$	$O(n^2)$	$\Omega(n)$	$\theta(n^2)$
void push_t (element item)	$8n$	$O(n^2)$	$\Omega(n)$	$\theta(n^2)$
element pop ()	$12n$	$O(n^2)$	$\Omega(n)$	$\theta(n^2)$

table 12. Space compelxity of function

3.2 Time Complexity

function	total	Asymtotic Notation		
		Big-Oh	Big-Omega	Big-Theta
int Direction(char* C)	22	$O(n)$	$\Omega(1)$	$\theta(1)$
void move_setting()	25	$O(n)$	$\Omega(1)$	$\theta(1)$
void push (element item)	3	$O(n)$	$\Omega(1)$	$\theta(1)$
void push_t (element item)	3	$O(n)$	$\Omega(1)$	$\theta(1)$
element pop ()	3	$O(n)$	$\Omega(1)$	$\theta(1)$

table 12. Time compelxity of function

4. Reveiw

- ‘자료구조’라는 강의명에서 느낄 수 있듯이, 단지 프로그래밍만 하는 것이 아니라 효율적인 자료사용과 구조적으로 사용하는 가를 많이 생각하며 했던 과제였다. 간신히 코딩을 마치고, 보고서를 쓰면서 내가 만든 프로그램이 자료구조학의 관점으로 바라봤을 때, 얼마나 비효율적이고 난장판인 프로그램인지 반성하게 되었다.
  본 과제를 진행하면서 파일을 입력해오는 과정이 가장 어려웠다. 파일 읽어와, 필요한 데이터만 추출하는 과정에서 난관을 만났지만, 인터넷과 1학년 때 수강한 컴퓨터프로그래밍 교재를 살펴보며 fgets, fgetc, fscanf의 의미를 다시 한번 재정의했다. 또, 최초 코딩시 동적 할당이라는 개념을 이용했는데, 이를 이용하는 과정에서 할당된 메모리외의 다른 메모리에 접근하게 되어, 오류가 자꾸 발생하여 사용하지 않았다. 과제는 끝났지만 다시 한 번 동적 할당이라는 개념을 공부해볼 것이다. 또 하나의 아쉬운 점으로는 Linked list의 개념을 접목시켜보지 못한 부분이다.
  과제는 끝났지만, 분명 반성하고 보완해야 할 부분 투성이다. 이를 다시 생각하고, 다시 공부하면서 반성할 것이다.