컴퓨터공학실험1 12주차 결과보고서

전공: 컴퓨터공학과 학년: 2 학번: 20191559 이름: 강상원

1. 실험시간에 작성한 프로그램에서 자료구조와 구성한 자료구조를 화면에 그리는 방법들을 설명한다. 완성한 자료구조를 이용한 그래픽 전환 작업의 시간 및 공간복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술한다.

`vector<vector<char>> maze;` 를 사용하였다. #include <vector>를 하여, char형 이차원 vector인 maze를 선언할 수 있게끔 하였다. 이 이차원 배열(vector)에는 입력한 .maz 파일 내용을 그대로 집어넣는다.

```
while (getline(input_file, line)) {
	realWIDTH = line.length();
	WIDTH = (line.length()-1)/2;
	realHEIGHT++;
}

HEIGHT = (realHEIGHT-1)/2;
cout << "WIDTH : " << WIDTH << "  HEIGHT : " << HEIGHT << endl;
```

먼저 getline 함수를 이용해 미로의 폭, 높이의 사이즈를 잰다. while문이 반복되는 횟수가 미로 데이터 파일의 높이가 되고, 한 줄의 길이가 미로의 폭이 된다.

이후 WIDTH, HEIGHT 변수에 (폭, 길이 – 1)/2를 저장하는데, 이는 실제 미로의 가로, 세로 칸 수를 의미한다.

cout으로 폭과 높이를 확인해준다.

```
maze.resize(realHEIGHT, vector<char>(realWIDTH));

string tmp;

for(int i=0; i<realHEIGHT; i++){
	getline(input_file2, tmp);
	for(int j=0; j<realWIDTH; j++)
		maze[i][j]=tmp[j];
}
```

maze.resize를 통해 이차원 배열의 크기를 조정해준다.

string형 변수 tmp를 선언하고, .maz 파일에서 한 줄씩 데이터를 받은 후 for문으로 이차원 배열에 그대로 저장해준다.

```
MazeLines line_tmp;

for(int i=0; i<realHEIGHT; i++){
	for(int j=0; j<realWIDTH; j++){
		if(maze[i][j] == '-') {
			line_tmp.startX=j-0.5; line_tmp.startY=i+0.5;
			line_tmp.endX=j+1.5; line_tmp.endY=i+0.5;
			this->LINES.push_back(line_tmp);
		}

		else if (maze[i][j] == '|') {
			line_tmp.startX=j+0.5; line_tmp.startY=i-0.5;
			line_tmp.endX=j+0.5; line_tmp.endY=i+1.5;
			this->LINES.push_back(line_tmp);

		}
	}
}
```

한 줄씩의 정보를 MazeLines tmp 구조체에 저장하고 push_back 시킨다. 이차원 배열 안의 문자가 '-'거나 '|'일 때만 시작 x,y좌표와 끝 x,y좌표를 저장한다. 시작, 끝 좌표의 계산 방식은 다음과 같다. 가로 선과 세로 선이 만나서 ㄱ, ㄴ, 모양을 만들기 위해서는, 선을 정해진 길이보다 일정하게 더 길게 그려야 한다. '-'이 maze[i][j]에 있는 경우에는 (i+0.5, j-0.5) ~ (i+0.5, j+1.5)까지 선을 그리고, '|'인 경우에는 (i-0.5, j+0.5) ~ (i+1.5, j+0.5)까지 선을 그린다. 선은 이후에 MazeLine 함수로 그린다.

```
void ofApp::MazeLine(MazeLines lines) {
    ofSetColor(100);
    ofSetLineWidth(5);
    ofDrawLine(20*lines.startX, 20*lines.startY, 20*lines.endX,
20*lines.endY);
}

void ofApp::freeMemory(){
    this->maze.clear();
    vector<vector<char>>().swap(this->maze);

    this->LINES.clear();
    vector<MazeLines>().swap(this->LINES);
}
```

MazeLine 함수는 주어진 시작 좌표, 끝 좌표를 이용해 선을 그린다. 미로 그림의 사이즈를 적당히 맞추기 위해 각 좌표값에 20씩 곱해준다.

freeMemory 함수는 사용한 메모리를 다시 반환해주는 역할을 한다. this->LINES.clear()로 구조체 배열의 길이를 0으로 설정해주고, swap method를 이용해 빈 메모리와 swap해준다.

완성한 자료구조를 이용한 그래픽 전환 작업의 시간복잡도는, 총 칸의 수만큼 걸릴 것이다. 따라서 O(WIDTH*HEIGHT)이다. 공간복잡도도 마찬가지로 O(WIFTH*HEIGHT)이다.

실험 전에는 미로 데이터를 포함할 수 있는 충분히 큰 2차원 배열을 만들고, 그 자료구조에 입력하는 식으로 진행하려 하였으나 복잡도 면에서 비효율적이고 예외 경우도 발생할 수 있기에 먼저 미로의 너비와 높이를 정한 후, 입력받는 방식으로 바꾸었다. 더하여 전역변수 형식으로 먼저 maze 배열을 선언하고 사용자 정의 함수 내에서 다뤄야 다른 사용자 정의 함수에서도 다루기 편해질 것이다. 또한 일반적인 이차원 배열보다는, vector를 이용한 방법이 resize나 기타 입, 출력 면에서 더 다루기가 쉬워 vector를 사용하게 되었다. 2차원 vector 배열에 저장하게 되면, 다음 실습 시간에 DFS/BFS를 이용한 미로 경로 탐색 알고리즘을 구현할 때 경로 좌표를 다루기 쉬워질 것이다.

2. **본 실험 및 숙제를 통해 습득한 내용을 기술하시오.**

미로의 크기를 알기 위해 .maz 파일을 읽게 되면, 다시 데이터 값을 알기 위해 입력을 받아야 해서 입력 버퍼를 복제하는 방식으로 해결하였다. 또한 이전에는 vector 일차원 배열의 크기 조정 방법은 알았으나 2차원 배열의 크기 조정은 익숙치 않았다. 이번 실습을 통해 확실히 습득하게 되었다. 또한 C++에서 파일 줄 끝까지 입력 받기 위해서 while(getline(input_file, line) 방법을 쓰는 것도 앞으로 유용히 쓰일 듯 하다. 기타 구현하면서 세세하게 느낀 점은 상기되어 있다.