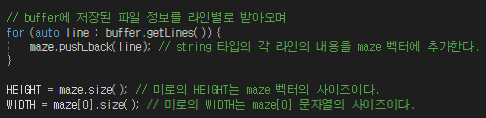
12주차 결과보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181288 이름: 윤성호

1. <이용한 자료구조>



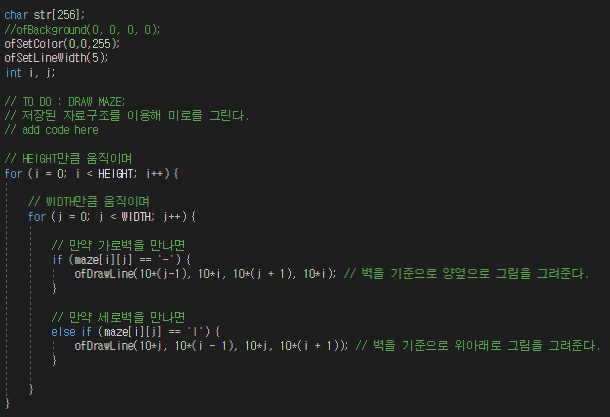
[사진 1]



[사진 2]

이번 실습시간에는 .maz 확장자로 만들어진 텍스트 파일을 읽어온 후, 안에 저장되어 있는 미로정보를 가지고 오픈프레임워크상에서 미로를 그리는 작업을 진행하였다. 미로 정보를 가져온 후 저장을 해야하는데, 3주차에서 진행할 dfs, bfs 탐색을 위해서 2차원 배열의 형태로 저장하였다. 2차원 배열은 c++ 프로그래밍에서 자주 사용하는 vector 자료구조를 사용하였다. [사진 1]과 같이 미로를 저장할 벡터를 vector<string> maze; 로 선언한다. 이는 벡터 자료구조의 원소로 문자열을 넣어준다는 것을 의미한다. [사진 2]에서처럼 buffer라는 변수에 파일을 받아온 후, 반복문을 통해 buffer의 모든 줄을 탐색하며 각 줄을 maze에 추가해주었다. push\_back 매서드는 vector 자료구조에서 많이 쓰는 매서드로, 원소를 추가해주는 기능을 한다. 파이썬의 append와 같은 기능을 한다. 동적할당을 하지 않고 파이썬과 같이 동작하게 할 수 있기 때문에 편리하다. 이렇게 저장된 maze 벡터는 행은 미로의 세로사이즈, maze 벡터의 열은 미로의 가로사이즈이므로, 이를 각각 HEIGHT와 WIDTH 라는 변수에 넣어주었다. Vector 자료구조는 사이즈가 일정하지 않은 자료구조이지만, 배열과 같이 인덱스로 접근할 수 있기 때문에 미로를 여러 번 받을 수도 있는 현재 실습에서 이용하기 적합한 자료구조이다.

<자료구조를 화면에 그리는 방법>



[사진 3]

Maze 라는 변수명을 가진 vector에 미로의 정보를 모두 저장해준 다음에는, 이를 오픈프레임워크를 이용해서 화면에 출력해줄 수 있다. 먼저, 미로의 색은 ofSetColor(0,0,256) 으로 설정하여 파란색으로 그려줄 수 있도록 하였다. 그리고 미로를 그릴 선의 굵기는 ofSetLineWidth(5); 를 통해 5로 설정해주었다. 기본적으로 미로를 그릴 준비를 마친 후에는 maze 자료구조를 이차원 배열을 탐색하듯이 탐색하며 미로를 실제로 그려준다. 이중 반복문을 이용하여 vector의 정보에 접근한다. HEIGHT는 i라는 변수로, WIDTH는 j라는 변수를 이용해 순회한다. 만약 maze[i][j]가 “-“이라면, 이는 현재 미로의 정보가 가로벽이라는 것을 의미한다. 따라서, x축과 평행한 선분을 현재 위치에 그려주어야 한다. 이 때, 위치는 세로위치를 의미하는 i는 그대로 두고, j-1부터 j+1의 위치까지 선분을 그려준다. 그런데 이렇게 그리면 너무 작게 그려지므로 모든 좌표들을 10배 해주어 확대를 하여 그려주었다. 즉, ofDrawLine(10\*(j-1), 10\*i, 10\*(j+1), 10\*i)라고 써서 그려주었다. 만약 maze[i][j]가 “|“이라면, 이는 현재 미로의 정보가 세로벽이라는 것을 의미한다. 따라서, y축과 평행한 선분을 현재 위치에 그려주어야 한다. 이 때, 위치는 가로위치를 의미하는 j는 그대로 두고, i-1부터 i+1의 위치까지 선분을 그려준다. 그런데 이렇게 그리면 너무 작게 그려지므로 모든 좌표들을 10배 해주어 확대를 하여 그려주었다. 즉, ofDrawLine(10\*j, 10\*(i-1), 10\*j, 10\*(i+1))라고 써서 그려주었다. 만약 maze[i][j] 가 빈칸이라면, 빈방을 의미하므로 아무것도 그려주지 않아야 한다. 따라서, 코드상에는 따로 조건문을 써주지 않았다.

<그래픽 전환 작업의 시간 및 공간복잡도>

그래픽 전환과정은 maze라는 전역변수 vector에 저장된 미로의 정보를 이중 반복문을 순회하며 maze[i][j]가 “-“ 이면 ofDrawLine(10\*(j-1), 10\*i, 10\*(j+1), 10\*i)를 통해 가로선분을 그려주었고, maze[i][j]가 “|“ 이면 ofDrawLine(10\*j, 10\*(i-1), 10\*j, 10\*(i+1))를 통해 세로선분을 그려주었다. 이중 반복문은 미로의 정보의 가로 및 세로를 전부 순회하였다. 따라서, 현재 미로의 가로를 WIDTH, 미로의 세로를 HEIGHT라고 저장해놓았으므로, 그래픽 전환 작업의 시간복잡도는 O(WIDTH\*HEIGHT)이다. 또한, maze라는 vector 자료구조에 모든 미로의 정보를 미리 저장해두고, 이를 이용하여 미로를 그려주었다. 따라서 그래픽 전환 작업에는 maze라는 vector가 차지하는 공간 또한 필요하다. Vector의 특성을 이용하여 메모리는 미로의 사이즈에 딱 맞게끔 설정해주었다. 따라서, 그래픽 전환 작업의 공간복잡도는 O(WIDTH\*HEIGHT)이다.

<실험 전에 생각한 방법과 실제 실습방법의 차이>

Eller’s algorithm으로 미로를 생성하고 난 이후, dfs와 bfs 탐색이 가능하게 하는 자료구조를 만들라는 요구사항이 있었다. 따라서, 이차원 배열을 동적할당하여 생성하여 생성하려는 생각을 하였다. Open을 여러 번 누르는 것을 염두에 두고 프로그램을 짜기로 생각하였기 때문이다. 매번 open을 누를 때마다 새로운 미로를 자료구조에 저장해야 하기에, 미로의 크기가 다르면 미로를 저장하는 자료구조 또한 크기를 새롭게 지정해주어야 하기 때문에 이차원 배열을 동적할당하여 사용할 계획을 했었다. 그러나, c++ 프로그래밍에서는 vector를 자료의 빈번한 삽입이나, 삭제 등 동적인 작업에서 많이 사용한다. 따라서, c언어에서는 동적할당을 하는 연습을 여러 번 해보았기에, 새로운 시도를 해보고자 vector 자료구조를 이용하여 미로를 저장하였다. 비록, 메모리적인 측면에서 보면 array가 더 효율적이지만 동적인 작업이 빈번한 상황에 대비하여 vector 자료구조를 이용하였다. Vector 자료구조 또한 array와 마찬가지로 인덱스로 접근이 가능하기에 배열과 같은 방식으로 값을 참조하여 이용할 수 있었다. 그러한 차이 이외에는 readFile의 다른 부분이나, freeMemory, draw 모두 실험 전에 설계했던 방식으로 실습을 진행하였고, 큰 문제없이 프로그램이 실행되었다.

1. 이번 실습에서는 .maz 파일을 읽어와서 오픈프레임워크를 이용하여 그래픽으로 전환하는 작업을 진행하였다. 가장 먼저, array에서는 자료의 빈번한 추가와 삭제과정에서 불리하다는 점과, 이를 보완하기 위해 vector를 이용할 수 있다는 사실을 알게 되었다. 자료의 추가와 삭제는 stack 및 queue에서는 빈번하게 진행될 때 사용되는 것을 알고 있었지만, fifo, filo와 같은 방식으로 추가와 삭제가 진행되고, 인덱스로는 접근이 불가능하지만, vector는 인덱스 접근또한 가능하므로 array 대신 이용할 수 있다는 점 또한 알게 되었다. 그리고, ofSystemLoadDialog 함수를 이용해 파일을 불러왔는데, 여기서 불러올 파일의 형시을 제한해줄 수 있다는 것을 알게 되었다. Maze project ppt 3페이지에서 요구한 사항을 보면, 기본확장자를 설정해주라는 것이었다. 이를 찾기위해 여러날을 구글링하였고, ofSystemLoadDialog 함수가 저장되어 있는 ofSystemUtils.cpp 파일에서 lpstrfilter 부분에서 원하는 형식을 써주면 된다는 것을 알게 되었다. Default는 All이다. 즉, 모든 파일을 불러올 수 있다는 이야기이다. 그러나, 실습에서의 요구사항은 Example File(\*.maz)라는 문구와 함께 maz 확장자의 파일만 불러올 수 있도록 설정하는 것이었다. 이를 위해서 lpstrfilter의 값을 L“Exaple File(\*.maz)\0\*.maz\0All\0\*.\*\0”로 변경해주었고, 이렇게 해주었더니 불러올 파일을 제한 할 수 있는 칸이 두개 생겼다. 하나는 Example File(\*.maz)라는 문구와 함께 maz 확장자의 파일만 불러올 수 있도록 하였고, 다른 하나는 All이라는 문구와 함께 모든 파일을 불러올 수 있도록 하였다.