**Algorithms and Data Structures – Maze Project**

**Purpose of the Maze Project.**

The purpose of this project is to create a two-dimensional array of objects. Each position of the maze will be represented by an object. Each object will be represented by a 3x3 output dependent on the type of object.

Ex. XXX

XXX

XXX

The goal will be to navigate from the starting position to the goal position. For each step, the object can only move in one of four directions: Up, Right, Down, Left. The object can only move to open positions. The challenge is to find a path from the start position to the goal position.

**Approach.**

My approach to this project was to define each object with an array of strings that would be instantiated differently by what type the object was (Open, Blocked (Wall), Start, and goal (Finish)). I chose to represent object with string because each row of the object array is the same except for the Start and Finish squares. I used an Enum to allow the constructor of the mazeObject to determine the appropriate content of each string in the object array.

To generate the maze, I created a class called MazeGenerator. The constructor of this class will take two arguments (Number of rows, and number of columns) and construct the two-dimensional array with mazeSquare objects. The type of each object is the default type, Open. I chose to initialize the maze this way to allow the input\_file to only must handle creating the start, finish, and wall type objects.

Each line of the input\_file contains two numbers. The first is assigned to a global variable called “rows” and the second to a global variable called “columns”. These variables are then passed to the MazeGenerator constructor. When the constructor is called it passes the variables again to the generateMaze() function. This function is what creates the two-dimensional array of mazeSquare objects. This function iterates through a nested for-loop and inputs the objects into each index depending on the size of the maze.

After the maze has been fully instantiated, the maze is printed to the console and to the output\_file. To print to the console, the function printMaze() is called. This function accepts two arguments; int rows and int columns. This function has two nested for loops. The reason two are needed is because each object has three indexes that cannot be printed at once. If all three were printed, the format of the maze would be unreadable. To avoid this issue, the function iterates through only the first index of the object array for each object in the maze row. After each iteration it shifts to the next object array index. After it reaches the bottom object array index, it shifts to the next row of objects. This allows the output to be formatted as a complete maze.

The same process is followed to print the maze to the output\_file except for outputting to the file vise the console.

**Class Skeletons.**

MazeSquare:

#include <string>

using namespace std;

enum squareType {START = 1, OPEN = 2, WALL = 3, FINISH = 4};

MazeSquare (squareType c);

MazeSquare();

string charArray [3];

void generateObjectArray (enum squareType);

MazeGenerator:

#include “MazeSquare.h”

#include <fstream>

MazeGenerator (int rows, int columns);

void generateMaze (int rows, int columns);

MazeSquare \*\*myMaze = new MazeSquare\*[rows];

void printMaze (int rows, int columns);

Main:

#include <iostream>

#include <fstream>

#include “MazeSquare.h”

#include “MazeGenerator.h”

int rows;

int columns;

int rowCount;

string line