**Maze Solver**

**Homework Nr. 5**

**Gabriel Kuka 10/12/2018**

**Libraries**

The first part of the source code is including the libraries necessary for the program to work properly. These libraries include:

1. **iostream**: needed for the basic input and output commands within the program
2. **queue**: queue library is needed in the BFS algorithm
3. **vector**: needed to create collections of different objects

After including libraries, we also add the “using namespace std” line which makes it easier for the coder to output or input data (with “cin” and “cout”).

**Class Node**

Each element in the maze is a node. In this sense, the source, the exit, a wall, a path is a node. Every node is represented by the class node. This class acts as a template for each element in the maze. Nodes can be of 4 types: wall, path, source, or exit. A wall is a 0. A path is a 1. The source is the character ‘s’ and the exit is the character ‘e’. In order to go from ‘s’ to ‘e’, the algorithm travels from one path to another, namely: from 1 to 1 in the maze.

Each node has its coordinates: x and y. Another information is the distance of this node from the source.

**BFS Algorithm**

The algorithm used to solve the maze is the Breadth-First-Search algorithm. The search for the path starts from the source. For every node of whatever type, the program scans its neighbors. The node has only 4 neighbors: the upper node, the lower node, the right node, and the left node. While scanning the program makes sure not to go beyond the maze itself. To sum up, since there are 4 neighboring nodes, I use a for loop to scan every one of them. If the neighboring node is a 1, then we jump to it to follow the path. The algorithm uses a queue list. The queue is used to keep the all the nodes that could lead the to exit. Each of these is scanned with a while loop. After the following node has been validated as a possible path, then the distance is incremented. The algorithm will scan every for its neighbors until it reaches the ‘e’. If all the possible routes don’t lead to the ‘e’, then we say that there is no path to the exit of the maze.

**Main function**

In the main function we assign a 2D vector to represent the whole maze. Then, we enter the maze. For each element, the program validates the input to check whether the input is valid or not. For example, we cannot have 2 sources or 2 exits. After entering the maze, the algorithm is executed to solve it and the program prints the solved maze.