1. Before starting implementation, design your code base. Design a class that will be in charge of reading the maze file.
   The list of methods and member variables:
   //constructor
   MazeReader(string file) throw(MazeCreationException);

   //destructor
   ~MazeReader();

```
//Maze is read correctlly, return pointer to the Maze
    const char* const* getMaze() const;

    //if maze read correctlly, return how many rows in maze
    int getRows() const;

    //if maze read correctlly, return how many collumns in maze
    int getCols() const;

    //return starting position
    int getStartRow() const;

    int getStartCol() const;
```

   Member variables:

```
    int m_row;   //rows
    int m_col;  //collums
    const char* const* M; //maze
    int ms_row;  //starting position
    int ms_col;
    char** m_maze;
```

2. Design a class that, given a valid maze, will traverse it.
   The list of member variables and methods:

   //constructor
   MazeRunner(const char* const* mazePtr, int startRow, int startCol, int rows, int cols);

   //destructor

~MazeRunner();

//print the array
void print();

//The maze is traversed until it exited
bool runMaze();

```
//return a which has been visited in array
    const int* const* getVisited() const;

    //return how many row has been visited
    int getVisitedRow() const;

    //return how many collumn has been visited
    int getVisitedCol() const;

    //return a const pointer to maze
    const char* const* getMaze() const;

    //try in order of up/right/down/left
    bool solveMaze(const char* const* s_M, int r, int c,int** sol);

    //check this is step if can be counted
    bool isSafe(const char* const* maze, int r, int c,int** solv);
```

Variables:

```
    const char* const* m_maze;
    int** m_visited;
    int m_rows, m_cols;
    int m_curRow, m_curCol;
    int m_StartRow, m_StartCol;
    int m_curStep;  //current step
```

3. Discuss how you plan to detect the need for backtracking should you reach a dead-end in the maze. What will variables/objects/arrays will need to be updated when you backtrack?

   I planned to use recursive methods for backtracking, the maze check for possible moves in a clockwise order (up, right, down, left). If the step is correct, it moves to the new position and the current step add one,

and return false. Otherwise, it moves back to the last step, current step -1, and returns false.

4. Discuss how you plan to detect and handle finding an exit or running out of places to traverse.
   The Exit of the given maze is in the first row, usually, the exit would be in the edge, which are the first row or the first column. The char ‘E’ stands for Exit, so I checked:

```
else if (s_M[r][c]=='E'&& sol[r][c]==0)
    {
        m_curRow=r;
        m_curCol=c;
        m_curStep++;
        sol[r][c]=m_curStep;
        return true;
    }
```

if the step meets the ‘E’, that means we escaped. Otherwise, the up, right, down, and left. If none of them works, return false.