# Assignment Summary

In this assignment we will have you create a program that will read in a file containing a maze and find an exit out of that maze by performing a Breadth First Search.   
  
This assignment is delivered in two parts:

1. The actual data structures to be used: Queue, Node, and Point.
2. The classes to process and solve the maze using the data structure

# Assignment Instructions

Choose a language that you will be coding in for the duration of the assignment:

C# or Java

For C#, use Visual Studios with NUnit.

For Java use JetBrains IntelliJ with JUnit.  
  
Each assignment project should be a console application.  
  
Satisfy the tests by creating the appropriate classes and methods. Note, you could uncomment all the tests and make method stubs that throw: NOT IMPLIMENTED YET EXCEPTIONS.   
  
**IMPORTANT NOTE: YOU ARE NOT ALLOWED TO MODIFY THE TEST CODE!**

Naming conventions

For the sake of simplicity C# naming conventions were used throughout this documentation.   
  
Please use appropriate Java conventions if you are using Java:  
  
1. Lowercase on method names: isEmpty() vs IsEmpty()

2. SMART properties were used in C#, you will need getter/setters: getName() and setName() vs just Name.

# Point Class – Milestone 1

* You can use the same Point class from the last assignment!

# Stack Class – Milestone 1

* You can use the same class from the last assignment!

|  |  |
| --- | --- |
|  |  |

# Node Class – Milestone 1

* You can use the same Node class you did in the first assignment!
* The only thing to consider is that we only need to use Next pointer for our Queue! No need to EVER call Previous in the Node class while using the Queue!

# Queue Class – Milestone 1

* Must handle any generic type as data (same as your Linked List)
* Maintains Head – Points to the front node in the Queue (or null if there are no nodes)
* Maintains Tail – Points to the last node in the Queue (or null if there are no nodes)
* Maintains Size – Count of the number of nodes in the list, zero when the list is empty

|  |  |
| --- | --- |
|  |  |
| Public Method Name | **Description** |
| Queue() | Constructor, initializes private fields for size, head, and tail. |
| void Enqueue(T element) | Creates a new Node with a new element and adds it to the tail of the queue. The old tail will now point to the new tail. |
| T Front() | Returns the front element in the queue (next to be removed) without removing it from the data structure |
| T Dequeue() | Returns the front element on the queue, removing it from the data structure. The new Head will point to the next person in line. |
| void Clear() | Empty all elements from the list |
| int GetSize() | Getter for the private size field. |
| Node<T> GetHead() | Returns the Node at the front of the queue (next out) |
| Node<T> GetTail() | Returns the Node at the end of the queue (last one in) |
| boolean IsEmpty() | Return true if the list is empty, else return false |

# Maze Class (updated from assignment 2) – Milestone 2

* Same class from last time, but with a method for BreadthFirst searching.
* A stack is created after the search with the top value being the first point and the bottom value being the exit.
* The returned stack should be a copy of the stack, this can be done easily by having your Stack class implement the Clonable interface. There are many solutions to this.
* You can choose to solve this problem using the maze copy with parent pointers on each point as discussed in the slides and in class, you are also free to be creative if you have another solution.

|  |  |
| --- | --- |
|  |  |
| Public Method Name | **Description** |
| string BreadthFirstSearch() | * Performs a breadth first search using the starting point values obtained in the constructor * The direction to take in the algorithm will be SOUTH, EAST, WEST, NORTH. (SEWN) * Marks visited Points as ‘V’ in the maze * The path taken from starting point to ending point will be indicated by changing the character from ‘V’ to ‘.’ * A Stack variable will be created and stored locally (obtained from calling the GetPathToFollow() method). The stack will contain the path of the points from start to end. The starting point should be on Top and the ending point on the bottom. In other words, the reverse of the Stack constructed to find the exit! Note that if no Exit was found, that Stack will be empty. * The returned value of this method will be a string result indicating the path to follow and a print out of the maze after exploring (see details in the below pages for formatting) |
| Stack<Point> GetPathToFollow() | * Returns a cloned copy of the stack created from the search. |

# Other Methods and Classes

Extra methods and classes can be coded, however the documentation for that method or class must contain a justification of why it is necessary. You likely will want to make some extra methods if you notice the same functionality happening between many methods.

# Maze File Format (Same as Assignment 2):

* A file that contains the maze that the program must navigate through
  + Give the file an extension of ‘.maze’
* Your file must contain the following information (in this order on separate lines)
  + Size of the maze (rowLength, columnLength)
  + Starting position (rowStart, columnStart)
* Each spot in the maze is represented by a single character value
  + one line for each row
  + one value per for each column, with the following meanings:
    - A ‘ ‘ indicates that this point in the maze can be traversed
    - A ‘W’ indicates a wall, the space cannot be traversed
    - An ‘E’ marks the maze exit
* Notes:
  + Make sure the border of the maze is surrounded by walls

# Maze File Format Example:

Example File Maze (11 rows x 13 columns. Starting Point [row 1, column 1], where the starting point is the top left empty space in the below maze):  
  
11 13

1 1

WWWWWWWWWWWWW

W W W

W WWW W WWW W

W W W W

W WWWWWWW WWW

W W W W

WWW W WWW W

W W WEW

W WWWWW W WWW

W W W  
WWWWWWWWWWWWW

Maze Output after exploring (with an exit) - Example:

**Path to follow from Start [1,1] to Exit [7,11] - 17 steps:**

**[1, 1]**

**[1, 2]**

**[1, 3]**

**[1, 4]**

**[1, 5]**

**[2, 5]**

**[3, 5]**

**[3, 6]**

**[3, 7]**

**[3, 8]**

**[3, 9]**

**[4, 9]**

**[5, 9]**

**[6, 9]**

**[6, 10]**

**[6, 11]**

**[7, 11]**

**WWWWWWWWWWWWW**

**W.....WVVVVVW**

**WVWWW.WVWWWVW**

**WVWVV.....W W**

**WVWWWWWWW.WWW**

**WVVVWVVVW.VVW**

**WWWVWVWWW...W**

**WVVVVVWVVVWEW**

**WVWWWWW WVWWW**

**WVVVVV WV W**

**WWWWWWWWWWWWW**

Maze Output after exploring a maze without an (E)xit - Example:

There is no exit out of the maze.

WWWWWWWWWWWWW

WvvvvvWvvvvvW

WvWWWvWvWWWVW

WvWvvvvvvvWVW

WvWWWWWWWvWWW

WvvvWvvvWvvvW

WWWvWvWWWvvvW

WvvvvvWvvvWvW

WvWWWWWvWvWWW

WvvvvvvvWvvvW

WWWWWWWWWWWWW

# Deliverable Hand In

* Create an archive of the entire project folder.
* The file must be a zip archive (which ends with the suffix ‘.zip’)
* Submit the zip file directly to the appropriate LEARN dropbox.

# Marking Guidelines

If a milestone is not handed in before the due date and/or the student has not met with the instructor for a code review, a mark of 0 will be given for that milestone. You must meet in person (online meet up) with your instructor BEFORE the milestone deadline.  
  
25% will be deducted for each test that cannot be passed with a minimum grade of 0 being assigned.  
  
Another marking criteria is optimization. Because this is subjective, it will be determined on a case-by-case basis during the time the student sits with the instructor. These suggestions will often relate to the following:

* **Refactoring**: Should be able to reuse duplicate code through a private method
* Simplification of code, if it is deemed that the student has written too much code, where less would have sufficed
* Renaming of private variables or functions

When an optimization suggestion is made, the student must fix this within an arranged timeline based on the complexity and to be determined by the instructor.  
  
All optimization suggestions will be default weighted to 25%, unless the instructor deems it is sufficiently important enough to be worth more.