CSCI 3110 (Barbosa)
Project 5: Treasure Island

Due: **See course calendar** – may not be turned late.

Assignment ID: **proj5**
File(s) to be submitted: **proj5.cpp, treasuremap.h, treasuremap.cpp, makefile**

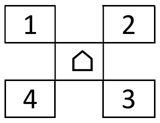
Objectives: 1) Use recursion; 2) Understand and apply backtracking to solve a problem.

Project description:
Write a C++ program that, given a starting point on an island, finds its way to a treasure, using recursion. The treasure map will be read from a file at the start of the program. Your code must work for all legal maps. The map is a rectangular grid represented as a 2D standard C++ array (not a vector or other STL container), composed of water cells (marked by **~**) and land cells (marked with **L**). The treasure (if there is one) is on a special land cell marked with the letter **T**. The program should run until the treasure is found or until it determines that there is no treasure (after exploring all reachable land cells). Exploration is done by recursively invoking a function and marking the cells visited with a special character (an electronic bread crumb to keep from reprocessing explored cells). The legal moves are to cells diagonal to the current cell. The order in which adjacent cells must be explored is read from the file. The treasure should be found by using recursive calls and backtracking, and **not by looking ahead** (you do not have the ability to "look" into neighboring cells before moving – you simply move and rely on backtracking if it's a "bad" move). If the specially marked treasure cell is encountered, the game should print a message that it was found. Otherwise, after exploring all reachable land cells, a message is output stating that there is no treasure on the island.

X	X	X	X	X	X	X	X	X	X
X	*	L	L	~	~	L	L	~	X
X	~	*	L	L	L	L	~	~	X
X	L	L	L	L	L	L	~	~	X
X	L	L	~	L	L	L	~	L	X
X	T	L	~	~	L	L	L	L	X
X	L	L	~	~	~	L	L	L	X
X	~	L	~	~	~	L	L	L	X
X	~	~	~	~	L	L	L	L	X
X	X	X	X	X	X	X	X	X	X

At left is an instance of a treasuremap. Note the attributes of a legal map:

- Cells are referred to by row # and column #. Row numbers increase in the downward direction, and column numbers increase to the right.
- The "play area" includes only the non-red cells.
- Cell 1,1 does not contain the treasure, and serves as the starting point for the treasure search.
- **X** (red) cells are **non-traversable** map boundary cells
    - The cells in red are not part of the treasure map read from the file, but must be constructed around it and represented in the array. The treasure map at left has an 8x8 play area and is shown in a 10x10 array).
- **~** (blue) cells are water cells and are also **non-traversable**.
- **L** (tan) cells are traversable land cells not yet visited.
- ***** (yellow) shows traversable land cells that have been previously visited.
- **T** (brown) cell is the treasure (if the map has one). This cell must be reachable from the starting point, using legal moves.
- The first line in the input file (a string made up of digits 1 – 4) is the order of exploration. Assuming the player is at the home plate cell (⌂), legal moves 1 – 4 are as shown. Any permutation of these digits is possible.



Requirements:

1. Your program must be split into 3 files (separate interface and implementation files, and a driver file):

   a) The *TreasureMap* class – This class represents a treasure map

   - Files must be named ***treasuremap.h*** and ***treasuremap.cpp***
   - Class must be named ***TreasureMap***
   - The interface (header file) is provided and you **CAN NOT** modify it.
     i. You should implement the provided interface file in a .cpp implementation file
     ii. All data members must be of the type and size specified in the header file
     iii. All member functions in the interface file must be implemented as declared – However you have flexibility in how you choose to implement the body of functions, including choice of local variables.
       - The **constructor** will accept an *ifstream* file object argument and will read the file and construct the map. The file contains a rectangular map no larger than 8 cells by 8 cells - your code must be able to handle all treasure maps from those having a minimum of two rows (2x1) or two columns (1x2) up to the maximum 8x8 (error checking is not required). The first line in the file is the exploration order: a **string** containing the digits 1,2,3,4 **in any order**. The next line holds the dimensions of the map (# of rows followed by # of columns, separated by a space). The remainder of the file is the treasure map play area. An example treasure file (4 rows by 5 columns) is shown below. Note: There are no spaces between cell legend symbols in the file.

```
1234
4 5
L~L~~
~L~~L
~LLLT
~~LL~
```

       - The ***PrintMap()*** function outputs the map's current state. It must only be output in unvisited traversable cells. See sample output.
       - The ***FindExit()*** function is a void recursive function that explores the map. It has two *int* parameters indicating the player's row and column (in that order), and a *bool* reference variable that represents whether or not the treasure has been found. The exploration of the treasure will always begin at row 1, column 1 (which must be a land cell). This function must rely on recursion to explore the map and to backtrack. You MAY NOT use a *return* or *break* statement in it. Write the code so that it backtracks naturally. Hint: To keep code streamlined, do not write special case code to handle boundary cells. Simply backtrack out of them.
       - The ***getMove()*** function, has a single *char* parameter (an element of the *xplor* string). It returns a *std::pair* of two ints. The first is the row offset [-1 (go up one row), or 1 (go down one row)], and the second the column offset [-1 (go left one column), or 1 (go right one column)]. Here's an example of how to declare and set a *pair*:

```
int row_offset = -1, col_offset = 1;                    // row_offset and col_offset  make up the pair
std::pair<int,int> move = std::make_pair(row_offset, col_offset);    // move is the variable name
```

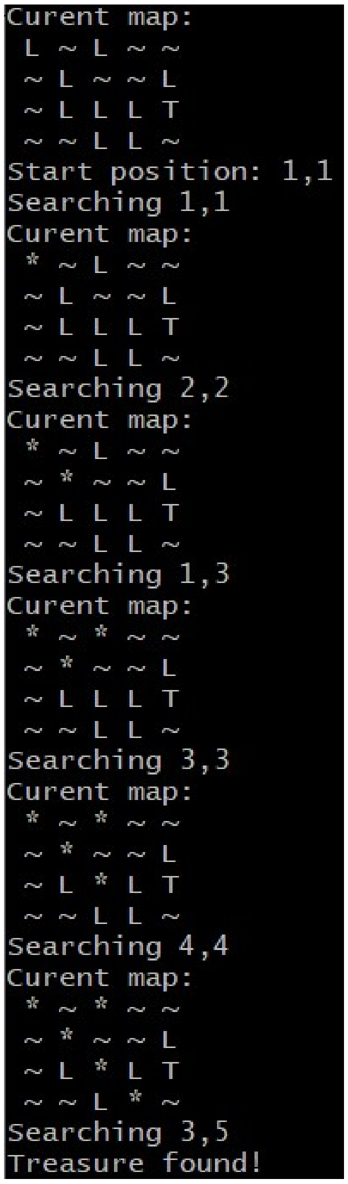
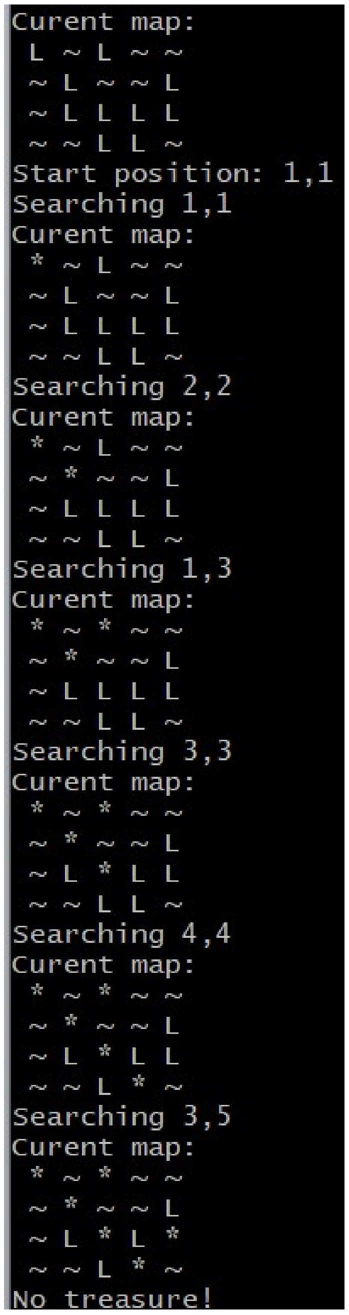
         Here's an example of how to use *move*: if the player is currently in cell 3,4 and *getMove* returns the pair (-1, 1), the player will proceed to cell 2,5.

   b) A driver, or client, file

   - Must be named ***proj5.cpp***
   - Declares the *ifstream* file object containing the map. The map is read from a file named ***treasuremap.txt***.
   - Must instantiate the TreasureMap object.
   - Must invoke the ***FindTreasure*** function, beginning at cell 1,1.
   - Outputs an appropriate message "***Treasure Found!***" or "***No treasure!***", based on the outcome of map exploration.

2. Sample output:

a) This output show possible program executions with the player starting at cell 1,1 – The ***Treasure Found*** case is from the 4x5 map file shown above. The ***No Treasure*** case was built by converting the ‘T’ in the treasure map file to an ‘L’.

Case with Treasure	Case with No Treasure
	

```
Curent map:
 L ~ L ~ ~
 ~ L ~ ~ L
 ~ L L L T
 ~ ~ L L ~
Start position: 1,1
Searching 1,1
Curent map:
 * ~ L ~ ~
 ~ L ~ ~ L
 ~ L L L T
 ~ ~ L L ~
Searching 2,2
Curent map:
 * ~ L ~ ~
 ~ * ~ ~ L
 ~ L L L T
 ~ ~ L L ~
Searching 1,3
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L L L T
 ~ ~ L L ~
Searching 3,3
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L * L T
 ~ ~ L L ~
Searching 4,4
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L * L T
 ~ ~ L * ~
Searching 3,5
Treasure found!
```

```
Curent map:
 L ~ L ~ ~
 ~ L ~ ~ L
 ~ L L L L
 ~ ~ L L ~
Start position: 1,1
Searching 1,1
Curent map:
 * ~ L ~ ~
 ~ L ~ ~ L
 ~ L L L L
 ~ ~ L L ~
Searching 2,2
Curent map:
 * ~ L ~ ~
 ~ * ~ ~ L
 ~ L L L L
 ~ ~ L L ~
Searching 1,3
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L L L L
 ~ ~ L L ~
Searching 3,3
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L * L L
 ~ ~ L L ~
Searching 4,4
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L * L L
 ~ ~ L * ~
Searching 3,5
Curent map:
 * ~ * ~ ~
 ~ * ~ ~ L
 ~ L * L *
 ~ ~ L * ~
No treasure!
```

3. Test your program - Use different maps, with and without treasures.

4. Code comments - Add the following comments to your code:

- A section at the top of the source file(s) with the following identifying information:

  Your Name
  CSCI 3110-00X (your section #)
  Project #X
  Due: mm/dd/yy

- Below your name add comments in each file that give an overview of the program or class.
- Place a one or two-line comment above each function that summarizes the workings of the function.

5. Rubric

Requirement	Points Off
TreasureMap Class	
Class name	-5
Include guards	-3
Constructor – number, types, and order of parameters	-5 (per item)
Constructor – file correctly read and treasure map correctly built (including outer boundary)	-15
FindTreasure – correct function signature	-10
FindTreasure – correctly handles boundary cells	-5
FindTreasure – does not traverse water cells	-5
FindTreasure – correctly handles previously visited cells	-5
FindTreasure – avoids look-ahead	-10
FindTreasure – explores treasure map in specified order	-10
FindTreasure – utilizes recursion and backtracking to find the treasure	-15
FindTreasure – invokes *PrintMap* function only in appropriate cells	-5
FindTreasure – correctly terminates exploration (does not *overexplore*)	-10
FindTreasure – correctly sets/uses *bool* reference variable	-5
FindTreasure – does not use *return* or *break* statements	-10
PrintMap – correct output	-10
Driver/Client	
main – input filename	-5
main – correctly opens input file	-5
main – correctly instantiates *TreasureMap* object	-10
main – correctly invokes the treasure map's *FindTreasure* function	-5
main – correctly outputs the outcome of exploring the map	-5
general – Compiles (on *ranger* using Linux g++ compiler, and the provided makefile)	-25
general – Runtime crash	-35
general – Other	TBD