Lab Assignment 1

Due Tuesday by 11:59pm **Points** 24 **Submitting** a file upload
Available Sep 8 at 8am - Sep 16 at 11:59pm 9 days

Recursive Thinking

Traversing a Maze

In this week's lecture on recursion discussed ***thinking recursively*** to solve a problem. This lab will provide an opportunity to do just that as you will use recursive thinking to write a program to traverse a maze.

Every programming assignment is as much a test of **English language comprehension** as it is a test of programming or comprehension skills. This week, the explanation of the task below is given clearly in plain English if you read carefully. However, if there is ***any*** question about what is being asked, you are urged to ask for clarification in the public forums.

Make sure that you have read and understood

- ***Unit modules 1 and 2*** *and*
- ***Ch. 11 Recursion*** *and*
- ***C++ Coding Style Guidelines*** (https://ohlone.instructure.com/courses/18987/files/3097756/download?download_frd=1)

before submitting this assignment. Hand in only one submission.

Lab Assignment Objectives

1. Be able to recognize a situation in which a subtask is a simpler version of the larger problem and design a recursive solution to solve this problem.
2. Safeguard against infinite recursion by defining a valid variant expression and threshold on a problem solution.

Understand the Application

Suppose that you have a friend, Ann **Ohlone**, that has a maze in her backyard. One day, Ann discloses the following two key facts about the maze:

1. Somewhere within the maze is a magic tapestry that contains the secret of happiness.
2. The finder of the tapestry gets to enjoy this secret of happiness provided that they can return to the maze's entrance after entry. (***Aside***: So far, unfortunately, many a seeker of happiness have entered, but none have returned successfully).

The maze is built on a rectangular grid. At each point of the grid, there are four directions to move: north, east, south, or west. Some directions, however, may be blocked by an impenetrable wall.

Your task is to accept the challenge of providing a program that will guide a user to enter the maze, find the treasure and back out to successfully return to the maze's entrance. In short, you will write a recursive function that can be executed on a laptop that your friend can carry through the maze. The function will give directions and ask the person traversing the maze questions to take them to the magic tapestry and back out to the entrance.

The Program Specification

Write an interactive program to lead the user into a maze and back out again while searching for a magic tapestry. User input will dictate search options available.

Traversing a Maze

In order to traverse the maze, your program needs to detect either a ***dead end*** condition (*i.e. user facing wall*) or an unpursued ***path*** *(i.e. name is amiss on an unexplored option)*.

Implementation Subtasks

The **dead_end()** function will return a boolean value to indicate whether or not the direction directly ahead can or cannot contain the tapestry.

The **traverse_maze()** recursive function will return a **boolean** value. A function that returns a value can be recursive in a situation such as this where the answer to a small problem will help to solve the complete problem. In this case, searching part of the maze can help determine whether the tapestry is present or not in the entire maze.

```
bool dead_end();
// Postcondition:The return value is true if the direction directly in front
// is a deadend (i.e., a direction that cannot contain the tapestry).

bool traverse_maze();
// Precondition: The user of the program is facing an unblocked spot in the
// maze which has not previously been visited by the user.
// Postcondition: The function has asked a series of questions and provided
// various directions to the user. The questions and directions have led the
// user through the maze and back to the exact same position that the user
// started at. The return value of the function is a true/false value
// indicating whether or not the user found a magic tapestry in the maze.
```

The Program Design

Since this is our first lab assignment for this course, I will walk you through the design.

traverse_maze

When this function starts, all that is known is that "Ann", the user of the program, is facing some spot in the maze that has not been previously visited. This function needs to take Ann into the maze, eventually heading her back to the exact starting spot. The ***crown jewel*** is for her to find the magic tapestry along the way.

Thinking recursively

Always ask Ann to take a step onto an unvisited spot, writing her name on the ground to tag whether or not she has been to a spot before. Once these two recordings are in place, ask Ann whether she has found the tapestry at this new spot; record her answer.

Cases to consider:

- Tapestry is found at this first spot, Ann can pick up the tapestry, step back backward to her starting spot. This is the stopping case.
- Tapestry is not found at this first spot. In this case, there are 3 uncharted possible directions for Ann to explore: forward, left or right. ***Note:*** Sometimes, not all 3 directions need be explored (i.e. the direction is a dead end if either a wall is blocking that direction ***or*** Ann's already been there).

Remember, anytime that Ann has found the tapestry not to explore new directions.

Steps (Ann has not found tapestry):

1. Designate left as first direction to explore.

2. for each of the 3 possible directions

{

(Test if the tapestry is found and test that the direction being faced is not a dead end).

{

Explore direction selected returning to this spot and setting found to true if the tapestry is found.

}

Have Ann turn 90 degrees to the right (next direction).

}

3. Ann is now facing the direction she came from, so ask her to step forward and turn around (so that she is facing this spot, as she was before this function was called).

Implementation

To implement your solution write the functions to carry out subtasks:

bool dead_end(): This function determines whether the direction in front of the user is a dead end. Dead ends are caused by a wall or a direction that the user has already explored. The function returns true (for a dead end) or false (for no dead end).

bool traverse_maze(): This function explores the three directions available at a given spot in the maze. The function returns true (tapestry found) or false (tapestry not found).

main(): This function provides the test driver to traverse the maze and report whether or not the tapestry is found.

Include the provided library tools:

bool inquire(const char query[ ]): This function reads characters from standard input until a newline character is encountered.

bool inquire(const char query[ ]): This function asks the user a yes/no question; it returns true if the user answers "yes", and returns false if the user answers "no".

The Program Design

Write a complete C++ program implementing the functions dead_end(), traverse_maze() and main().

Testing Requirements

Use the provided files to eliminate the need to validate user input.

- **auxa1.h (https://ohlone.instructure.com/courses/18987/files/3142502/download?download_frd=1)**
- **auxa1.cpp (https://ohlone.instructure.com/courses/18987/files/3142503/download?download_frd=1)**

Provide a commented out copy of your program test run after the source statements in your a1.cpp test driver file.

Example Test Run

Your program display should look something like these *sample* runs (although the values will differ for each student):

```
$ ./a1
Step forward & write your name on the ground.
Have you found the tapestry? [Yes or No]
Yes
Pick up the tapestry and take a step backward.
You found it!

$ ./a1
Have you found the tapestry? [Yes or No]
No
Please turn left 90 degrees.
Are you facing a wall? [Yes or No]
No
Is your name written in front of you? [Yes or No]
No
Step forward & write your name on the ground.
Have you found the tapestry? [Yes or No]
No
Please turn left 90 degrees.
Are you facing a wall? [Yes or No]
Yes
Please turn right 90 degrees.
Are you facing a wall? [Yes or No]
No
Is your name written in front of you? [Yes or No]
No
Step forward & write your name on the ground.
Have you found the tapestry? [Yes or No]
Yes
Pick up the tapestry and take a step backward.
Please turn right 90 degrees.
Please turn right 90 degrees.
Please step forward, then turn 180 degrees.
Please turn right 90 degrees.
Please turn right 90 degrees.
Please turn right 90 degrees.
Please step forward, then turn 180 degrees.
You found it!
```

What to Turn In

Hand in **5** files:

- auxa1.h

- auxa1.cpp - provides eat_line(), inquire()
- maze.h
- maze.cpp - provides dead_end(), traverse_maze()
- a1.cpp - provides main test driver

No zip files.

Tips and Requirements

1. Multiple file compilation required.
2. Ensure that your solution is well organized. Provide a program header and comments to document and organize your source code.
3. Provide a commented out copy of your program run. Enclose the run inside of multi-line comment delimiters so that your program will run in the grader test bed. Place the run after your program source code in the a1.cpp file.
4. Title your test driver in the format: **lastname**a1.cpp (for lastname **YOUR** lastname).
5. Personalize the namespace on the provided auxiliary files (i.e. change AUXA1_H to lastnameA1_H for lastname **YOUR** lastname).

Submission Resources

For more information on how to submit your assignment, please visit:

- **How do I submit an online assignment? Canvas Student Guide (https://community.canvaslms.com/docs/DOC-9539)**
- **Assignments Overview Canvas Video Guide (https://community.canvaslms.com/videos/1122-assignments-overview-students)**
- **Assignments Submissions Canvas Video Guide (https://community.canvaslms.com/videos/1121-assignment-submissions-students)**

Submitting multiple files to an assignment

Your lab 1 assignment requires uploading more than one file; you should upload these 5 files as one submission. To add these files, the **Add Another File button** is clicked to **upload the two files one by one. Check to make sure that both files uploaded okay.** When finished click **Submit Assignment.**

Questions?

Feel free to **ask** in the forum!

File Upload | Google Doc | Studio | Dropbox | Office 365

Upload a file, or choose a file you've already uploaded.

Upload File Use Webcam

+ Add Another File

Click here to find a file you've already uploaded

Comments...

Cancel Submit Assignment

Lab 1 Rubric

Criteria	Ratings			Pts
On time submission Submitted on time 4 points One day late -2 points Two days late -4 points	**4 pts** **On time**	**2 pts** **One day late**	**0 pts** **Two days late**	4 pts
Header file Satisfies the specifications as stated in the assignment. auxa1.h, included in multiple file compilation; guard namespace is customized to include student name, maze.h includes prototypes of functions used in maze.cpp.	**2 pts** **Full Marks**		**0 pts** **No Marks**	2 pts
Source file Satisfies the cpp source statements as specified in the assignment. Auxiliary source (auxa1.cpp) and maze traversal source (maze.cpp) files are provided.	**8 pts** **Full Marks**		**0 pts** **No Marks**	8 pts
Driver file Satisfies the main.cpp source statements as specified in the assignment. This is your submission test driver. Include in THIS file a commented out copy of your program test run.	**4 pts** **Full Marks**		**0 pts** **No Marks**	4 pts
Commented out test validation run included A copy of the program test validation run output is attached after the source code in the main.cpp test driver file (enclosed within comment delimiters). The run matches the source code submitted. The test run demonstrates validation of user input, pass by value and pass by reference.	**4 pts** **Full Marks**		**0 pts** **No Marks**	4 pts
Coding Style Includes a program header. Is correctly formatted (i.e. follows code style rules). User defined functions are documented.	**2 pts** **Full Marks**		**0 pts** **No Marks**	2 pts
				Total Points: 24