Please refer to **RecursiveWordChecker.hpp** for this document

I decided to purse the recursive algorithm extra credit for this assignment.

The project said we should try to see if their guess was a part of the phrase, either a letter or a word, and do so recursively.
To accomplish this I attempted to write out (in English) what my algorithm would be, and sought for the solution that involved repeating the same process/comparison in smaller and smaller chunks.

So the algorithm to check if the user either guessed the phrase, guessed a word in the phrase, or guessed a letter in the phrase is as follows

"Is this guess the secret phrase?, No, Is it one of the words in this phrase?, No, Is it one of the letters in this phrase?"

This is an OR , OR , style decision, only one condition, against one word or letter must be true for this to return true, and due to short circuting the algorithm will cease upon the first true conditional (first part of the phrase matching the guess)

so the guess needed to be compared against the whole phrase, then against words in the phrase one at a time, and then against the letters in a word one at a time. This proved to be the thorniest part of the algorithm, how to reduce the size of the phrase to the next logical piece, a special helper function was written for this

list<string> GuessChecker::NextSmallestSubSets (string mySet) Line#136

As the calls go on, the phrase that the guess is compared to gets smaller, but the guess persists throughout all calls.

Quite enjoyably this was my first time writing a mutual recursive function. The public method that compared one string to another returns true or false, but if the phrase is more than one letter (not yet at its sentinal case) then it returns its own truth value || the boolean return of the method

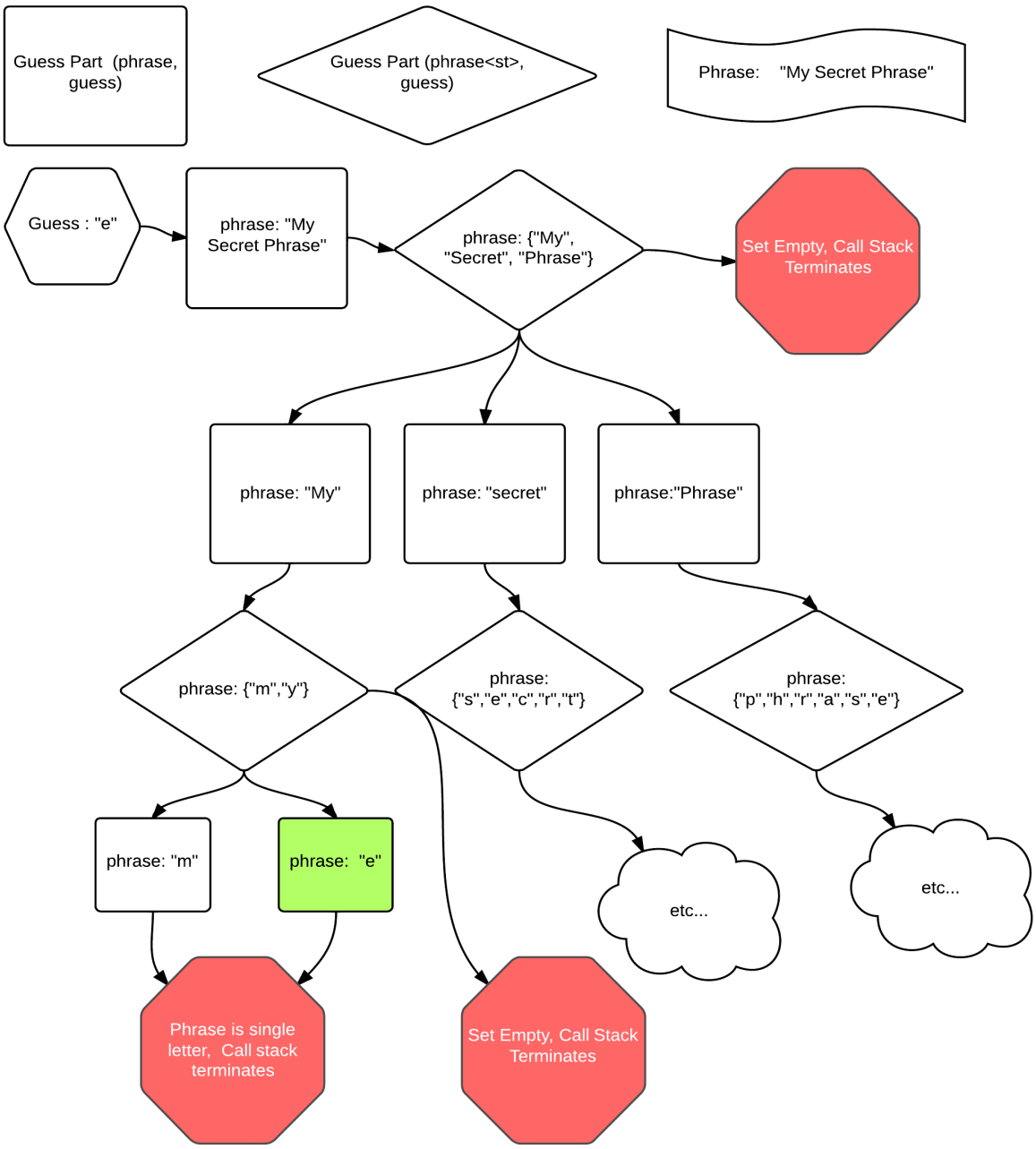
bool GuessChecker::GuessIsPart (string element , list<string> setList)

which returns the value of the the other function one at a time until the list is empty.
A diagram of this call stack attached below.

This algorithm proved to be useful for other checks, such as is this guess a valid guess based on my dictionary (this was an && check conversely). But as an alternative method I used an iterative approach for a method designed to add words that were guessed one letter at a time to the record of guesses. See Line #48. It demonstrates the call stack of the recursive algorithm.

Reflection.

This entire class of recursive guess checker could have easily been replaced by set<string> containing the phrase, its words, and letters, and then a set.count(element) call. But that wasnt the extra credit.
But if I learned one thing from it it was that many of my functions are in fact a really long conditional statement comprised of a series of && or || conditions. The Tic-Tac-Toe has a player won the game is an example of one of these, It returns true as soon any of the possible conditions are met, and returns false if it has reached it to the end of the function without returning true. But this could as well be written as each of the conditions concatenated to a long chain of || statements (or recursive calls to a stack of conditions).


Guess Part (phrase, guess)
Guess Part (phrase<st>, guess)
Phrase: "My Secret Phrase"
Guess : "e"
phrase: "My Secret Phrase"
phrase: {"My", "Secret", "Phrase"}
Set Empty, Call Stack Terminates
phrase: "My"
phrase: "secret"
phrase:"Phrase"
phrase: {"m","y"}
phrase: {"s","e","c","r","t"}
phrase: {"p","h","r","a","s","e"}
phrase: "m"
phrase: "e"
etc...
etc...
Phrase is single letter, Call stack terminates
Set Empty, Call Stack Terminates