Project 2

<Hangman Extreme>

CSC-17C (42475)

Name: Kevin Vo

Date: 06/04/18

Introduction

Title: Hangman Extreme

Classic Hangman is an interactive word guessing game where the player has a certain amount of turns before the full image of a man hanging appears.

Each time a word unknown to the player will be selected with a graphical representation indicating the amount of letters this word has. The player is then requested to enter a letter that they think is present in the unknown word. Guess enough letters correctly that makes up the full word before the entire hanging man image appears. The appearance of the image of the hanging man is dependent on the amount of incorrect letters that were inputted as each time a part of his body reveals itself.

Hangman Extreme is based off the classic Hangman game but has some additional modes and features embedded into it.

Recently, the game is included with more features to setup the game. To start with, the game is introduced with the option of sorting the words in ascending or descending order using two different recursive sorts to which is it inserted into a virtual Tree-like structure so that the words can be displayed and selected for the tree for the Hangman game.

Moreover, the game also as an easier multi-round game mode where random words from the game's library are selected and with the use of a virtual graph so that the random words can be chosen based on the idea of minimizing the accumulated lengths of all those words (using Prim's Algorithm). In other words, from the first to the last round, the game will try to have you guess less.

The game is also just included with a Table of existing words for that round so that the player is able to search the chosen word for Hangman on the Table (with the use of hashing) for hints if the player finds him/herself stuck during the game.

There is a "Normal Mode" that is closest to the original version of the game that's mentioned above. Within this game mode you have features such as the "Guess Word" ability to guess the entire word to complete your turn/game quicker but in return you will lose the game if you guess the wrong word. The "Lifeline" feature that allows you to guess the word as well but is not a sudden lose if you were to guess wrongly; however, this feature is only available once per turn. You'll also have the ability to view your previous inputs for the current round (called "View Inputs").

The second game type is called "Ez-Mode". Within this mode of Hangman Extreme, there's only a list of potentially 25 randomly selected words from a list of over 800. On top of that, these words will display for you in alphabetical order for you to better guess the word. The features:

“Guess Word”, “Lifeline”, and “View Inputs” is also available for “Ez-Mode” along with the processing game modes that will be mentioned below.

The third game mode is potentially the longest out of all the game modes, (hence the name “Marathon Mode”) as the player is able to select the number rounds with a randomly selected word per each round. The player has to win each round as they have done before for all of the rounds to finish.

The last game mode is called “Custom Word Mode” where the player can choose the number of rounds for another person to play as they enter word(s) of their choice for each number of rounds that they have chosen. These words will them be collected and inserted into the library of the game so that they can later be called in the future.

Summary

Project Size: ~ 2460 lines

Number of Variables: 18+

Number of Functions/Methods: 35+

Number of Classes: 6

Types of Containers Used: 5

Besides going beyond of 750 lines of code, I have utilized Hash, Recursive Sorts, Tree, and Graph.

For **hashing**, I used a hash table along with the RS-Hash to allow the player to be able to utilized the hash’s O(1) search for a possible word during a round of Hangman within the Custom Word Mode *(a mode which the host of the game are able to have to game play with their own inserted words to which it will be sorted in the main library (words.txt) for further games)* if they have a clue on what the word of the round might be and choose to use this feature as a hint.

For **Recursive Sorts**, I used a Bubble recursive sort and a Selection recursive sort to be able to have the randomly chosen list of words to be in ascending order or descending order as a setup option for a game mode that utilizes a Tree.

For **Tree**, I used a binary tree that eventually takes in the sorted words through recursion (this was mentioned above) and displays the words in Pre-Order and Post-Order or In-Order to allow

the Host of the game to see their choices of sorts so that they will be able to confirm what the order they want the player to play in.

For **Graph**, I used Prim's algorithm to generate a minimum spanning tree of a graph of the random words' lengths then used the visited vertices to select which random words to use for the gameplay.

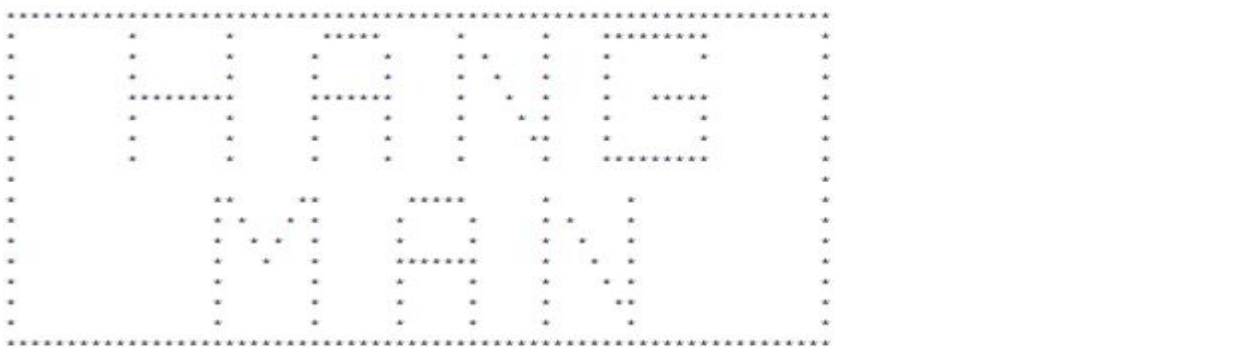
Description

The main point of this program is to use the hash, tree, recursive sorts, and graph.

The uses of hash, tree, recursive sorts, and graph are listed above under **Summary**.

Sample Input/Output

Start of Hangman Extreme (menu):



```
Enter 1 to play Hangman
Enter 2 to play EZ-Mode (choose the first letter of a word and that same letter will be given to you.)
Enter 3 to play Marathon Mode (choose number of rounds of Hangman to win.
Enter 4 to enter your own word for the round
Enter 5 to display the rules
Enter 6 to display scoreboard
Enter 7 to view 25 Free Words for EZ-Mode
Enter any other key to exit
User Input: 1
```

```
                                Try: 1/6

|------.
|
|
|
|
|
------------------------


-------
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
Enter a letter: e
```

Gameplay with selectable features:

```
                                Try: 4/6

|------.
|       O
|     --#
|
|
|
------------------------


-a--ou-
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
Enter a letter: 3

Previous Inputs (from Latest to Oldest):
------------------------------------------------
                          l
                          f
                          u
                          o
                          a
                          e
```

Endgame – Request to document score:

```
                                Try: 6/6

|------.
|       O
|     --#--
|      /
|
|
------------------------


-a-bou-
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
Enter a letter: m

|------.
|       O
|     --#--
|      / \
|
|
------------------------


*** You lose. Better luck next time! ***

Answer: harbour

You have a score of 0

Would you like to log your score? (y/n): 
```

Game Setup with Recursive Sorts then Tree – Request which Order the words would be played in then confirms:

```
You have chosen Tree Mode where you're able to sort the random words with the use of RECURSIVE SORTS and confirm your settings of the words from a TREE
to confirm your selection so that you'd be ready to offer the game to the player.

Let's begin...

Randomly chosen words: experience let start now take

Enter 1 to sort ABC Order.
Enter 2 to sort in Reverse.

User Input: 2

Sorted Reversed Order: take start now let experience

Inserting onto Tree...

Words in Pre-Order (Reversed Order):  take start now let experience

Words in In-Order (ABC Order):  experience let now start take

Do you want In-Order (ABC Order) instead? (y/n)
User Input: n



(for testing purposes) chosenWord = take


                              Try: 1/6

|------.
|
|
|
|
|
------------------------


----
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
```

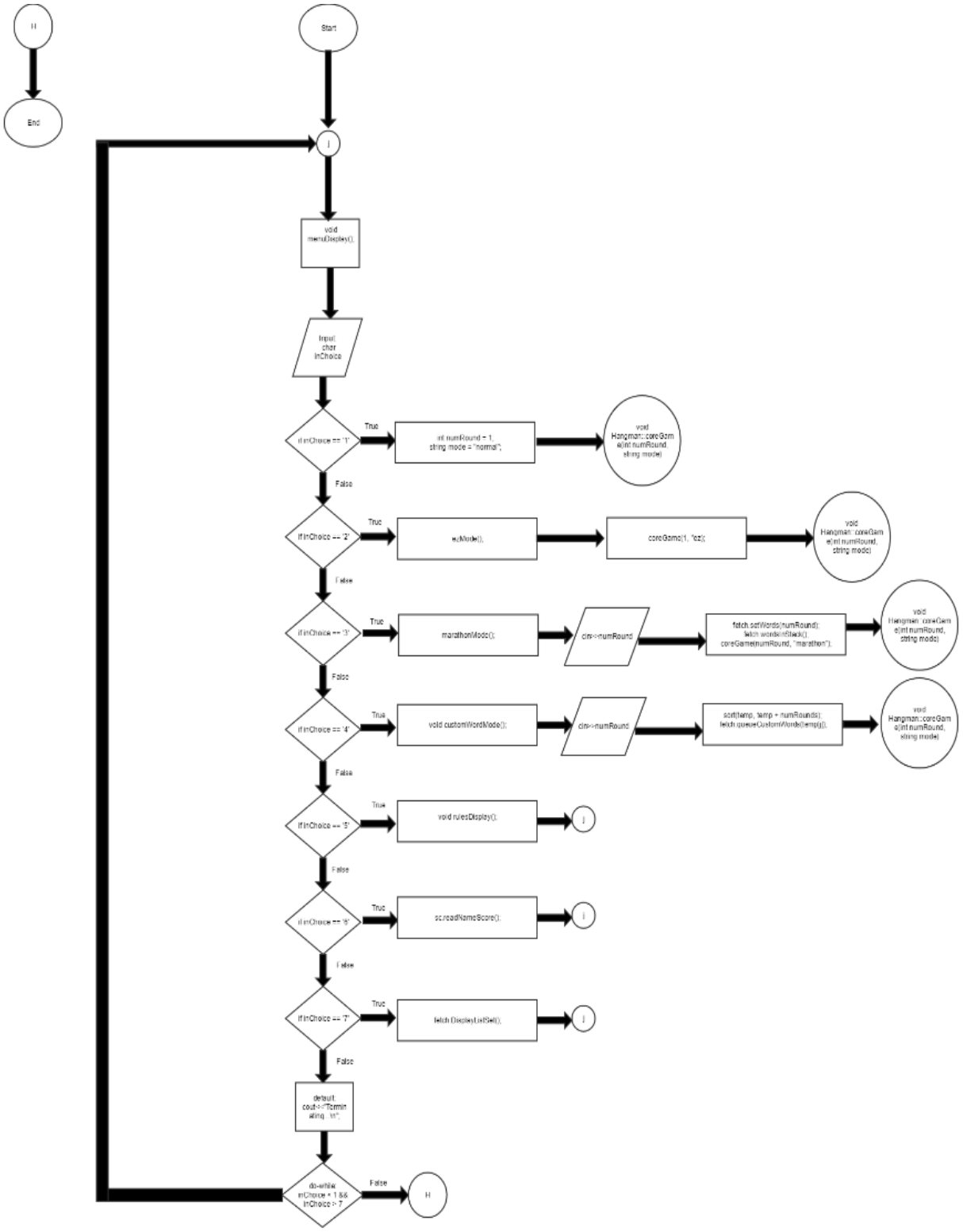
Uses Prim's Algorithm of each word's length in a graph to find minimum spanning tree to select words for game (in Graph – Option 9 on the main menu)

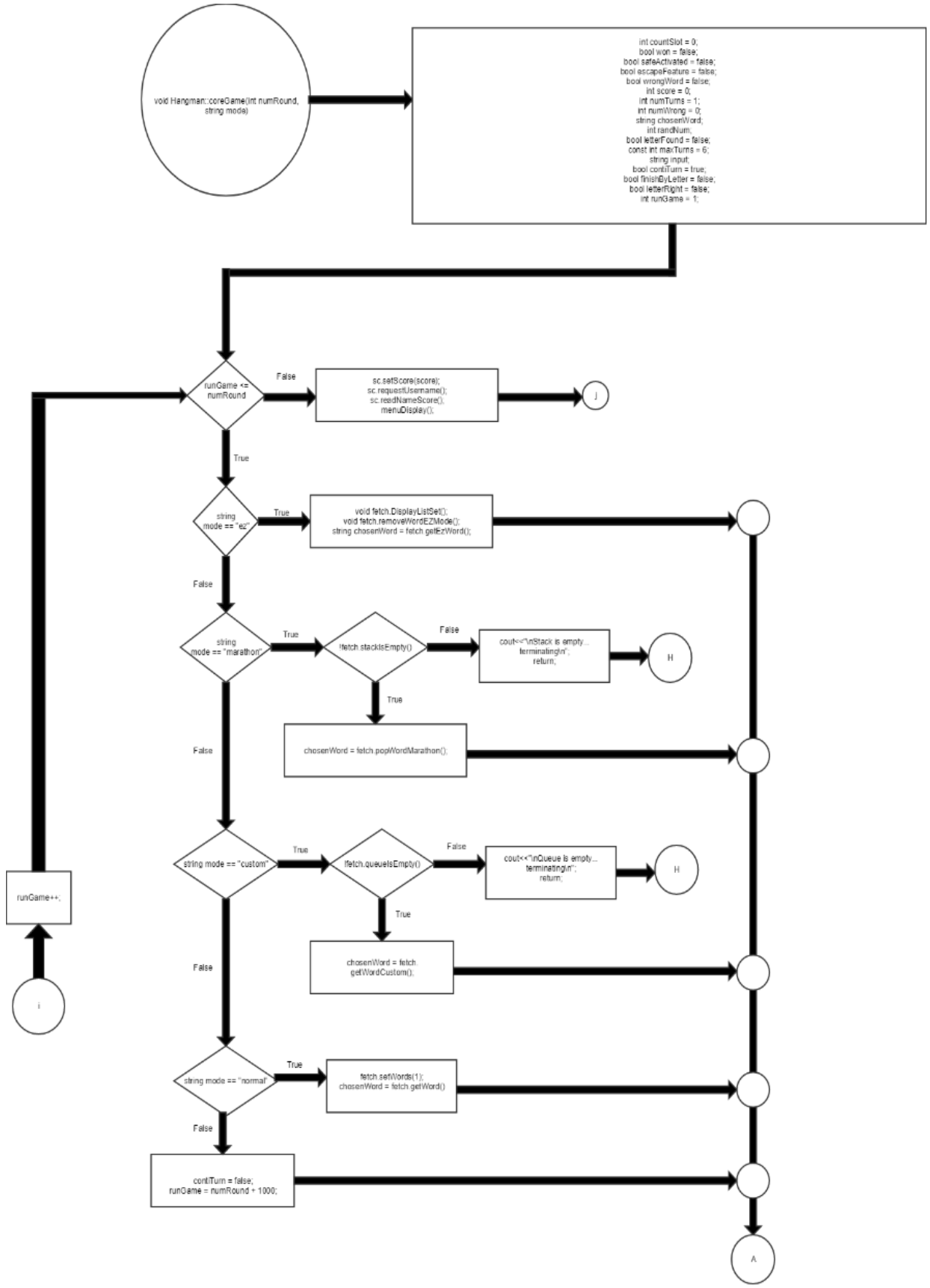
```
*****************************************************************************
* This game mode inserts the lengths of random words into a Graph *
* then find the graph's minimum spanning tree and use the visited *
* vertices to select which random words to use for the game.       *
*****************************************************************************


Using Minimum Spanning Tree to select words...

Number of Rounds for Graph Mode Left: 6


(for testing purposes) chosenWord = shake


                                    Try: 1/6

|-------.
|
|
|
|
|
--------------------------


-----
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
Enter a letter:
```

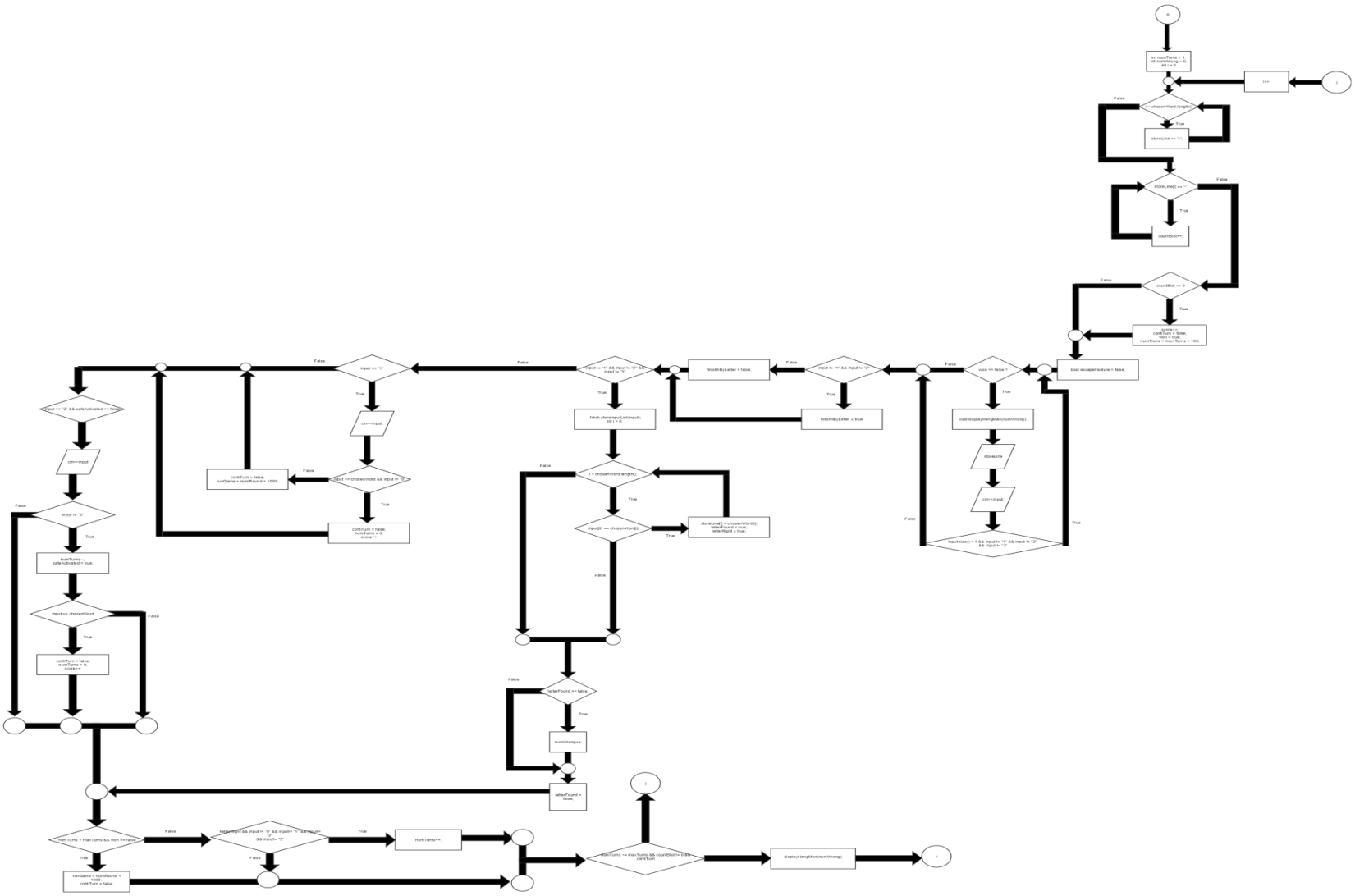
Uses RS-Hash Table to look up a predicted word as a hint without losing a turn (ONLY IN CUSTOM WORD MODE)

```
                                Try: 6/6

|------.
|      O
|    --#--
|     /
|
|
------------------------


pi--l-
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
Enter "4" to search Hash Table for hints on the word.
Enter a letter: c

                                Try: 6/6

|------.
|      O
|    --#--
|     /
|
|
------------------------


pic-l-
Enter "1" to guess word. (Will lose instantly if guessed wrong.)
Enter "2" to guess word without losing. (Once per round)
Enter "3" to see numbers you have previously entered
Enter "4" to search Hash Table for hints on the word.
Enter a letter: 4

You have chosen the Custom Word Mode.
Would you like to search the Hash Table for possible words the host might have might have entered previously? (y/n): y

Enter a word to be searched: pickle

Your hinted word(s) may be:
pickle


Would you like to search again? (y/n):
```

Flowchart


H
End
Start
I
void menuDisplay();
Input: char inChoice
If inChoice == '1'
True
int numRound = 1;
string mode = "normal";
void Hangman::coreGame(int numRound, string mode)
False
If inChoice == '2'
True
eLMode();
coreGame(1, "ez");
void Hangman::coreGame(int numRound, string mode)
False
If inChoice == '3'
True
marathonMode();
cin>>numRound
fetch.setWords(numRound);
fetch.wordsInStack();
coreGame(numRound, "marathon");
void Hangman::coreGame(int numRound, string mode)
False
If inChoice == '4'
True
void customWordMode();
cin>>numRound
sort(temp, temp + numRounds);
fetch.queueCustomWords(temp[j]);
void Hangman::coreGame(int numRound, string mode)
False
If inChoice == '5'
True
void rulesDisplay();
I
False
If inChoice == '6'
True
sc.readNameScore();
I
False
If inChoice == '7'
True
fetch.DisplayListSet();
I
False
default: cout<<"Terminating...\n";
do-while inChoice < 1 && inChoice > 7
False
H


void Hangman::coreGame(int numRound, string mode)
int countSlot = 0;
bool won = false;
bool safeActivated = false;
bool escapeFeature = false;
bool wrongWord = false;
int score = 0;
int numTurns = 1;
int numWrong = 0;
string chosenWord;
int randNum;
bool letterFound = false;
const int maxTurns = 6;
string input;
bool contiTurn = true;
bool finishByLetter = false;
bool letterRight = false;
int runGame = 1;
runGame <= numRound
False
sc.setScore(score);
sc.requestUsername();
sc.readNameScore();
menuDisplay();
J
True
string mode == "ez"
True
void fetch.DisplayListSet();
void fetch.removeWordEZMode();
string chosenWord = fetch.getEzWord();
False
string mode == "marathon"
True
!fetch.stackIsEmpty()
False
cout<<"\nStack is empty... terminating\n";
return;
H
True
chosenWord = fetch.popWordMarathon();
False
string mode == "custom"
True
!fetch.queueIsEmpty()
False
cout<<"\nQueue is empty... terminating\n";
return;
H
True
chosenWord = fetch.getWordCustom();
False
string mode == "normal"
True
fetch.setWords(1);
chosenWord = fetch.getWord()
False
contiTurn = false;
runGame = numRound + 1000;
runGame++;
i
A


A
int numTurns = 1;
int numWrong = 0;
int i = 0;
i++;
J
i < chosenWord.length();
True
False
storeLine += "-";
storeLine[i] == '-'
countBlot++;
countBlot == 6
score++;
contTurn = false;
numGame = numRound + 1000;
numTurns = max Turns + 100;
bool escapeFeature = false;
userInput == false ?
void displayHangMan(numWrong);
storeLine
cin>>input;
input.size() > 1 && input != "-" && input != "2" && input != "3"
input != "-" && input != "2"
finishedByLetter = true;
finishedByLetter = false;
input != "-" && input != "2" && input != "3"
fetch storeLine[i].toUpper(input);
int i = 0;
i < chosenWord.length();
input[i] == chosenWord[i]
storeLine[i] = chosenWord[i];
letterFound = true;
letterRight = true;
letterFound == false
numWrong++;
letterFound = false;
input == "-"
cin>>input;
input == chosenWord && input != "-"
contTurn = false;
numGame = numRound + 1000;
input == "2" && safeActivated == false
cin>>input;
input != "0"
numTurns--;
safeActivated = true;
input == chosenWord
contTurn = false;
numGame = numRound + 1000;
score++;
contTurn = false;
numGame = numRound + 1000;
numGame = numRound + 1000;
contTurn = false;
numTurns == max Turns && userInput == false
letterFound && input != "2" && input != "-" && input != "3"
numTurns++;
numTurns == max Turns && countBlot == 6 ||
contTurn
displayHangMan(numWrong);
J
I

Pseudo Code

Initialize

Display menu

If '1' is entered from the menu (Normal Mode has been selected)

Select obtain random word from words.txt

Set numRound = 1;

Set maxRound = 6;

Display characterized graphics

Request Input

If input == "1"

Request player to guess word

If input == chosenWord

Then win game

Else

Then lose game

Else if input == "2"

Request player to guess word

If input == chosenWord

Then win game

Else

Continue turn

Else if input == "3"

Then display sorted inputs from STL's List

Else if input == letter in chosenWord

Then continue turn

Else if '2' is entered from the menu (Ez-Mode has been selected)

Obtain 25 words from words.txt

Insert those 25 words into STL's Set

Calls function to the core of the game

Use Iterator to display all elements of STL's Set and pick a random word from it

Display characterized graphics

Request Input

If input == "1"

Request player to guess word

If input == chosenWord

Then win game

Else

Then lose game

Else if input == "2"

Request player to guess word

If input == chosenWord

Then win game

Else

Continue turn

Else if input == "3"

Then display sorted inputs from STL's List

Else if input == letter in chosenWord

Then continue turn

Else if '3' is entered from menu (Marathon Mode has been selected)

Run Marathon Mode

Request for number of rounds

Obtain words based on number of rounds

Push words on to STL's Stack

Pop a word from Stack from for each round

Checks if letter is in words

Else if '4' is entered from menu (Custom Word Mode has been selected)

Run Custom Word Mode

Requests number of rounds

Requests a word to be entered for each round

Push word on to STL's Queue

Pop word from STL's Queue for each round

Run core game

Else if '5' is entered from menu (Show the rules)

Then display the rules

Else if '6' is entered from menu (Show the Score Board)

Then display the names and scores from STL's Map

Else if '7' is entered from the menu (Show the Ez-Mode free words that has been chosen)

Use Iterator to go through STL's Set to display the list of words chosen only for Ez-Mode

Else if '8' is entered from the menu setup game with the use of TREE and RECURSIVE SORTS

Else if '9' is entered from the menu setup game with the use of Graph and Prim's Algorithm

Major Variables

Type	Variable Name	Description	Location
Integer	numTurns	Stores the number of rounds	void coreGame(int, string)
	numWrong	Stores the number of wrong inputs	void coreGame(int, string)
	countSlot	Counts for amount of open slots left	void coreGame(int, string)
	score	Stores the score for each word correct	void coreGame(int, string)
	newScore	Obtains the score for class Scoreboard	Class Scoreboard
string	input	Stores player's input	void coreGame(int, string)
	chosenWord	Stores the word chosen for the round	void coreGame(int, string)
	storeLine	Displays progress of unknown word	void coreGame(int, string)
	*wordList	Stores the string for class WordFetch	class WordFetch
char	input	Store yes/no choice for confirmations with player	void Scoreboard::requestUsername();
bool	won	Holds winning status	void coreGame(int, string)
	safeActivated	Holds if player has activated their lifeline	void coreGame(int, string)
	escapeFeature	Holds if player has exited guess word mode	void coreGame(int, string)
	wrongWord	Holds if the wrong word has been guessed	void coreGame(int, string)
	letterFound	Alerts game if letter has been found	void coreGame(int, string)
	contiTurn	Holds if turn should continue after guessing word mode	void coreGame(int, string)
	finishByLetter	Holds if player previous turn was Letter Mode	void coreGame(int, string)
	letterRight	Holds if guessed letter is correct	void coreGame(int, string)

Concepts

---------- (New For Project 2) ----------

Hash: Located in ***Hash.cpp*** (Lines: 18 - 138)

Object Declaration of Hash --> Located in Hangman.cpp (Line 27)

Tree: Located in ***BalancedTree.cpp*** (Lines: 18 - 264)

Object Declaration of BalancedTree --> Located in Hangman.h (Line 33)

Recursive Sorts: Located in ***RecursiveSorts.cpp*** (Lines: 4 - 51)

Object Declaration of RecursiveSorts --> Located in Hangman.cpp (Line 30)

Graph: Located in ***Hangman.cpp*** (Lines: 834 – 948)

---- (Old From Project 1) ---

Maps: Located in ***Scoreboard.cpp*** (Lines: 88 -189)

Sets: Located in ***WordFetch.cpp*** (Lines: 90 – 201)

Lists: Located in ***WordFetch.cpp*** (Lines 203-230)

Stacks: Located in ***WordFetch.cpp*** (Lines 233-261)

Queues: Located in ***WordFetch.cpp*** (Lines 264-300)

Iterators: Located in ***Scoreboard.cpp*** (lines 103-105), ***WordFetch.cpp*** (lines: 117 & 189-194 & 221-223)

Algorithms: Located in ***WordFetch.cpp*** (lines: 117), ***Hangman.cpp*** (lines: 437), ***Scoreboard.cpp*** (Line 143)

Reference

1) The C++ Standard Library by Nicolai M. Josuttis
2) www.cplusplus.com
3) Introduction to Algorithms (Third Edition) by Charles E. Leiserson

Program (Not entire code --- CODE FOR HASH, TREE, RECURSIVE SORTS, AND GRAPH are on pages 41 – 53)

//filename: main.cpp

```
#include <ctime>

#include <cstdlib>

#include <iostream> //For input output

#include <fstream>  //Use to obtain words from words.txt

#include "WordFetch.h"

#include "Hangman.h"
```

```
using namespace std;

int main(int argc, char** argv) {
    srand(time(NULL));

    //Declares object for Hangman class
    Hangman h;
    //Display the Hangman title on the screen
    h.titleDisplay();
    //Displays the menu to select game mode
    h.menuDisplay();

    return 0;
}
```

//filename: Hangman.cpp

```
#include <cstdlib>
#include <iostream>
#include <ctime>
#include "Hangman.h"
#include "WordFetch.h"
#include "Scoreboard.h"
#include <algorithm>    //To use STL's sort algorithm

using namespace std;
```

```
void Hangman::coreGame(int numRound, string mode){
    int countSlot = 0;          //counts the value for open slots left
    bool won = false;           //holds winning status
    bool safeActivated = false; //Checks if player has activated their lifeline
    bool escapeFeature = false; //Checks if player has exited guess word mode
    bool wrongWord = false;     //Checks if the wrong word has been guessed
    int score = 0;              //Stores the number points
    int numTurns = 1;           //Stores the number of turns
    int numWrong = 0;           //Store the number of wrong guesses for letters
    string chosenWord;          //Stores the word chosen for the round
    int randNum;                //Stores a random number for choosing word
    bool letterFound = false;   //Alerts game if letter has been found
    const int maxTurns = 6;     //Holds number of max turns per round
    string input;               //Stores player's input
    bool contiTurn = true;      //Checks if turn should continue after guessing word mode
    bool finishByLetter = false;  //Checks if player previous turn was Letter Mode
    bool letterRight = false;   //Checks if guessed letter is correct

    //Holds the status of guessed characters and open slots left
    string storeLine = "";

    srand(time(0));



    //Do-Whole loops the number of rounds a game mode offers
    //Also increments the number of rounds in its condition
    for(int runGame = 1; runGame <= numRound; runGame++){

        /**********************************
```

```
 *        Setting Up Round       *
 ************************************/
fetch.deleteInputHist();
contiTurn = true;
won = false;
safeActivated = false;
finishByLetter = false;
letterFound = false;

//Determines how chosenWord is selected depending on game mode
if(mode == "ez"){
   fetch.DisplayListSet();
   fetch.removeWordEZMode();
   chosenWord = fetch.getEzWord();
}else if(mode == "marathon"){
   cout<<"\nNOTE: Unless you've guessed the wrong word under challenge"
      <<" mode \"Word Mode\" (Option \"1\") you can still continue "
      <<"and play.\n You'll just not get a point for that word.\n";
   if(!fetch.stackIsEmpty())
      chosenWord = fetch.popWordMarathon();
}else if(mode == "custom"){
   if(!fetch.queueIsEmpty()){
      chosenWord = fetch.getWordCustom();
   }
}else if(mode == "normal"){
   fetch.setWords(1);
   chosenWord = fetch.getWord();
}else{
   cout<<endl<<"Error: Game Mode has not been selected correctly.";
   cout<<" Now terminating...\n";
```

```
    contiTurn = false;
    runGame = numRound + 1000;
}

cout<<endl<<"(for testing purposes) chosenWord = "<<chosenWord<<endl;

//Holds the status of guessed characters and open slots left
storeLine = "";

//For-loop that checks the number of open slots base on
//the length of the chosen word for the round
for(int i = 0; i < chosenWord.length(); i++){
    storeLine+="-";
}

cout<<endl;

//Reset for new round
numTurns = 1;
numWrong = 0;

/**********************************
 *        Starting Game          *
 **********************************/

//Do-while loop: Checks the winning conditions
do{
    cout<<"\n\t\t\t\t Try: "<<numTurns<<"/"
        <<maxTurns<<endl;
    letterRight = false;      //Resets if player guessed letter right
```

```
//Resets countSlot to 0 for every round
//checks if player has won
countSlot = 0;

//For-loop that assigns the number of open slots base on
//the length of the chosen word for the round
for(int i = 0; i < chosenWord.length(); i++){

   //Increments countSlot for every open slots avaliable
   if(storeLine[i] == '-'){
      countSlot++;    //counts the amount of open slots left
                      //if count == 0 then player wins
   }

}

//If there are no open slots left then word has been guess
//Therefore player has won
if(countSlot == 0){
   score++;
   contiTurn = false;
   won = true;
   numTurns = maxTurns + 100;    //Terminates do-while loop round
                                 //if word has been guessed in
                                 //Letter Mode
}

escapeFeature = false;

if(won == false){
```

```
//inner while loop: Prompts the player to enter guessed
//character or select one of the following features
//Feature #1: Guess word and instantly lose
//Feature #2: A once per round life line where incorrectly guessed
//word will not cause the player to lose
do{
    //Displays the appropriate hangman image based on amount of
    //wrong guesses
    displayHangMan(numWrong);

    //Displays status of guessing the word
    cout<<endl<<storeLine<<endl;

    cout<<"Enter \"1\" to guess word. (Will lose instantly if "
        <<"guessed wrong.)\n";

    //Opens up game feature to allow a free mode where guessing
    //a word wrong will not harm player's status of winning
    if(safeActivated == false)
        cout<<"Enter \"2\" to guess word without losing. "
            <<"(Once per round)\n";

    cout<<"Enter \"3\" to see numbers you have previously entered\n";

    //Prompts player for input
    cout<<"Enter a letter: ";
    cin>>input;

    if(input == "3")
```

```
        fetch.displayInputList();


    if(input == "2" && safeActivated == true)
        cout<<"\nSorry. You have already used up your life line"
            <<" (Option 2). You must wait for the next round "
            <<"to use it again.\n";
    //Warns the player if they've enter too many letters
    if(input.size() > 1)
        cout<<"\nInvalid Input: You must only enter 1 character. "
            <<"Please enter \"1\" or if you haven't \"2\" to"
            <<" guess the entire word.\n";


    //do-while loop checks for activation of features and size of
    //input
    //If player chooses one of the two options above, it will then
    //terminate this do-while loop to proceed them to the next


}while(input.size() > 1 && input != "1" && input != "2"
        && input != "3");


//Checks what the previous mode was
if(input != "1" && input != "2")
    finishByLetter = true;
else
    finishByLetter = false;


//Stores the guessed values per turn
if(input != "1" && input != "2" && input != "3"){
    fetch.storeInputList(input);
}
```

```
//Checks if inputed character is present in the chosen word
if(input != "1" && input != "2" && input != "3"){
    for(int i = 0; i < chosenWord.length(); i++){
        for(int j = 0; j < chosenWord.length();j++){
            if(input[0] == chosenWord[i] ){

                storeLine[i] = chosenWord[i];
                letterFound = true;
                letterRight = true;
            }
        }

    }

    //Keeps track of the number of wrong guesses
    if(letterFound == false)
        numWrong++;



    letterFound = false;   //resets found letter to false for
                           //the next turn
  //  numTurns++; //Incrementor for number of rounds

//Unlocks word guessing mode
}else if(input == "1"){
    cout<<endl;
    cout<<"BEWARE: Guess the word correctly and you will automatic"
        <<"ally win the game.\nGuess the wrong word and you will"
        <<" instantly lose.\n";
```

```
    cout<<"If you're not up for it, enter \"0\" to return to "
       <<"guessing only letters.\n";
    cout<<"Enter the word: ";
    cin>>input;

    if(input != "0"){
       if(input == chosenWord){
          cout<<"*** You've guessed the correct word! "
             <<"You have WON! ***"<<endl;
          contiTurn = false;
          numTurns = 0;
          score++;
       }else{
          cout<<"\nYour guess was wrong. You Lose\n";
          contiTurn = false;
          runGame = numRound + 1000;
       }

    }



//Unlocks the safe feature of the game
}else if(input == "2" && safeActivated == false){
    cout<<endl;
    cout<<"Warning: You'll only have safety mode once per turn.\n";
    cout<<"Safe Mode --- You will not lose a guess if word is "
       <<"guessed incorrectly.\n";
    cout<<"Enter \"0\" if you do not wish to use up this feature.\n";
    cout<<"Enter the word: ";
    cin>>input;
```

```
            if(input != "0"){
                numTurns--; //Does not count turn if life mode
                safeActivated = true;
                if(input == chosenWord){
                    cout<<"*** You've guessed the correct word! "
                        <<"You have WON! ***"<<endl;
                    contiTurn = false;
                    numTurns = 0;
                    score++;

                }
            }
        }
    }

    //Terminates the for-loop if player loses a turn in letter mode
    //In other words, GAMEOVER
    if(numTurns > maxTurns && won == false){
        runGame = numRound + 1000;
        contiTurn = false;
    }

    if(!letterRight && input != "0" && input!= "1" && input!= "2"
            && input!= "3")
        numTurns++;

}while(numTurns <= maxTurns && countSlot != 0 && contiTurn);

//Redisplay man's status
```

```
        displayHangMan(numWrong);

        //Informs player of the game's outcome before game ends
        if(finishByLetter == true){
            if(won == true){
                cout<<"\n*** YOU'VE WON ***\n";
            }else
                cout<<"\n*** You lose. Better luck next time! ***"<<endl;
        }
        //Displays the word at the end of the game
        cout<<endl<<"Answer: "<<chosenWord<<endl;


    }   //End of for-loop


    //Takes in score for scoreboard
    sc.setScore(score);

    //Requests if player wants to enter a username and have score be stored
    sc.requestUsername();

    //Displays scoreboard of previous players
    sc.readNameScore();

    menuDisplay();


}
```

//filename: Scoreboard.cpp (not entire file)

```
*********************************************************
 *                    Map -                                *
 * Used to store and display both usernames & score from   *
 * usernames.txt & scores.txt                              *
 * ********************************************************/


//Displays the list of previous names & scores
void Scoreboard::readNameScore(){


    updateNameScoreMap();



    map <string, string> :: iterator itr;


    cout<<endl<<endl;
    cout<<"*********************************************"<<endl;
    cout<<"*              Scoreboard                   *"<<endl;
    cout<<"*********************************************"<<endl;
    cout<<endl;
    cout<<"Players:\n";
    cout<<"--------\n";
    for (itr = m.begin(); itr != m.end(); ++itr){
        cout << itr->first<< endl << "\tScore ----> " << itr->second<<endl;
    }
    cout << endl;
```

```
}

//Takes in score
void Scoreboard::setScore(int score){
    newScore = score;
}



//Checks if username already exists
bool Scoreboard::checkUsername(string username){
    updateNameScoreMap();

    bool foundName;     //Returns if username already exist or not

    //Checks to see if the name on already on the map
    if(m.count(username) == 0)
        foundName = false;
    else
        foundName = true;

    //cout<<endl<<endl<<"foundName = "<<foundName<<endl;
    //cout<<endl<<endl<<"m.count = "<<m.count(username)<<endl;

return foundName;}

//Checks if backup username already exists too
bool Scoreboard::checkUserCode(string username){
    bool foundCode = false;     //Returns if username already exist or not

    std::hash<std::string> str_hash;
```

```
    string line;
    ifstream myfile ("backup_usernames.txt");
    if (myfile.is_open()){
        stringstream ss;    //create a stringstream
        while ( getline (myfile,line) && foundCode != true){

            ss << str_hash(username);     //add integer to the stream
            cout<<"\n\t\t\t\tss.str() = "<<ss.str()<<endl;
            if(ss.str() == line)
                foundCode = true;
        }
        myfile.close();
    }else cout <<"\nReading error: Unable to open backup_usernames.txt for "
              <<"cross-referencing with hash.\n";


    //cout<<endl<<endl<<"foundCode = "<<foundCode<<endl;

return foundCode;}


//Updates the map dealing with the scoreboard
void Scoreboard::updateNameScoreMap(){
    string lineName;
    string lineScore;       //temporary string for each line in .txt

    int countLine = 0;  //Counts the number of lines (elements) in .txt
    int i = 0;          //incrementer
```

```
    ifstream myfileCount ("usernames.txt");


    while(getline (myfileCount, lineName) ){
      countLine++;
    };


    myfileCount.close();



    //Reading files usernames.txt & scores.txt to insert into map
    ifstream myfileName ("usernames.txt");
    ifstream myfileScore ("scores.txt");
    if (myfileName.is_open() && myfileScore.is_open()){


        while(getline (myfileName, lineName) && getline (myfileScore, lineScore)){


            p.first = lineName;
            p.second = lineScore;
            m.insert(p);
        };


        myfileName.close();
    } else
        cout << "\nReading Error: Cannot open usernames.txt or scores.txt";



}
```

//filename:WordFetch.cpp

```
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <fstream>  //Use to obtain words from words.txt
#include <iterator>
#include <string>
#include <algorithm>
#include <set>  //For using sets
#include <list> //For using lists
#include <stack>
#include <queue>

#include "WordFetch.h"



using namespace std;

//Access a random line in word.txt to obtain a word for Normal Mode
//Some elements of this function was borrowed from cplusplus.com
string WordFetch::getRandWords(){
    ifstream myfile("words.txt"); //Access .txt for list of words
    string line = "";
    int numOfWords = 0;
    string newword = "";
    string word = "";
    int wordlength = 0;
    int k = 0;
    string alphabet="a b c d e f g h i j k l m n o p q r s t u v w x y z"; //used for validation
    while(getline(myfile, line)){
        numOfWords++;
```

```
    };

    int index = rand()%numOfWords + 1;

    //Reset curser to first line
    myfile.clear();
    myfile.seekg(0, ios::beg);
    //Grab line from file
    for (int lineno = 0; getline (myfile,newword) && lineno < index; lineno++);
    wordlength=string(newword).size();
    word="";
    for (int i = 0; wordlength>i; i++){
       for (k = 0; k < 51; k+=2) //Alphabetic Checks
          if (alphabet[k]==newword[i]) //Check if letter is in alphabet
             word=word+newword[i];
    }

return word;}


//Obtains number of words based on number of rounds
void WordFetch::setWords(int nRounds){

    numRounds = nRounds;
    string tempWord = "";
    int count = 0;
    wordList = new string[numRounds];
    for(int numWords = 0; numWords < numRounds; numWords++){
       tempWord = getRandWords();
       wordList[numWords] = tempWord;
```

```
    }
     //for(int m = 0; m < numRounds; m++)
          //cout<<endl<<"word = "<<wordList[m]<<endl;
}

//Destructor deallocates memory
//Update word list if new custom words have been added
WordFetch::~WordFetch(){
   //If queue is not empty then it will write its existing words onto the list
   if(!q.empty()){
      //****** Updating library ******
      //Cleans duplicate entries
      //Organizes words in ABC order
      fstream fileUpdateWords;
      fileUpdateWords.open("words.txt", fstream::app);

         if (fileUpdateWords.is_open()){
            while(!q.empty()){
               fileUpdateWords<<q.front();
               q.pop();
            };
            fileUpdateWords.close();
         }else
            cout<<"\nWriting Error: Cannot open words.txt to update library.\n";
   }
   delete[] wordList;
}

/*********************************************************
 *                  Set -                                *
```

```
 * Use to single insert words into words.txt and display                *
 * entire list in alphabetical order with built in sort                 *
 * ******************************************************/

void WordFetch::wordsInSet(){
    string line;
    const int numFreeWords = 25;
    ifstream myfileWords ("words.txt");
    if (myfileWords.is_open()){
        for(int i = 0; i < numFreeWords; i++){
            s.insert(getRandWords());
        };

        myfileWords.close();
    } else
        cout << "\nReading Error: Cannot open words.txt";
}


//Displays 25 free words from the list of possible words in Alphabetical order
//(utilizing STL set's built-in sort)if player wishes to see it in the menu
void WordFetch::DisplayListSet(){
    cout<<"\nList of 25 Possible Free Words (Alphabetical Order):\n";
    cout<<"----------------------------------------------------\n";

    for(set<string>::iterator it = s.begin(); it != s.end(); it++){
        cout<<*it<<endl;
    }
    cout<<endl<<endl<<"\n*Note: Unless playing under EZ-Mode, there is not a "
        <<"definitive chance that these words will appear\nin other"
```

```
        <<" game modes.\n";

}

//Removes a word from STL's Set
void WordFetch::removeWordEZMode(){
    char choice;    //Asks if player wants to correct their input mistake
    string input;   //Holds word to be removed from Set

    do{
        cout<<endl<<"From the list of words above, would you like to remove any "
            <<"one word to improve your odds? (y/n): ";
        cin>>choice;

        if(choice != 'n' && choice != 'N' && choice != 'y' && choice != 'Y')
            cout<<"\nSorry your has to be \"y\" for Yes or \"n\" for No.\n";
    }while(choice != 'n' && choice != 'N' && choice != 'y' && choice != 'Y');

    if(choice == 'y' || choice == 'Y'){
        DisplayListSet();   //Redisplay list from set for player to select

        //Requests player to select a word from set
        cout<<"\nEnter a word you'd like to remove from the list above: ";
        cin>>input;

        int n = 3; //number of attempts
        //Confirms the inputted word exists and removes it
        while(choice != 'n' && choice != 'N'  && s.count(input) == 0 && n > 0){
            cout<<"\t\t\t\t\t\t"<<n--<<" Attempt(s) Left\n\n";
```

```
        if(s.count(input) == 0){


            cout<<"The word \""<<input<<"\" does not exist.\n";
            cout<<"\nWould you like to try again? (y/n): ";
            cin>>choice;



            if(choice == 'Y')
                choice = 'y';
            else if(choice == 'N')
                choice = 'n';


            if(choice != 'n' && choice != 'y')
                cout<<"\nSorry your input has to be \"y\" for Yes or \"n\""
                    <<"for No.\n";
            else if (choice == 'y'){
                cout<<"\nEnter another word to be removed: ";
                cin>>input;
            }



        }
      };


      if(s.count(input) > 0)
         s.erase(input);
   }
}

//Obtain a word for Ez-Mode with the use of STL's Set
```

```
string WordFetch::getEzWord() const{
    string ezWord;

    int randomNum = rand()%s.size() + 1;
    int i = 0;

    for(set<string>::iterator it = s.begin(); it != s.end(); it++){
        i++;
        if(i == randomNum)
            ezWord = *it;

    }

    return ezWord;
}

//Clears the set
void WordFetch::clearWordListSet(){
    s.clear();
}




/*****************************************
 *              List -                   *
 * Used store previous inputs of a round *
 * ***************************************/

//Pushes inputed value onto the list
void WordFetch::storeInputList(string input){
```

```
    l.push_front(input);
}

//Use to display player's input for the round so that they could see
//the letters they've already guessed
void WordFetch::displayInputList(){

    cout<<"\nPrevious Inputs (from Latest to Oldest) RESETS FOR EVERY NEW WORD:\n";
    cout<<"-------------------------------------------------------------------\n";

    for(list<string>::iterator it = l.begin(); it != l.end(); it++){
        cout<<"\t\t\t"<<*it<<endl;
    }

}



//Deletes input history list (after every round)
void WordFetch::deleteInputHist(){
    l.clear();
}




/*****************************************
 *              Stacks (LIFO)-           *
 * Used store strings for marathon mode  *
 * ***************************************/
```

```
//Pushes all the words onto the stack when Marathon Mode is selected
void WordFetch::wordsInStack(){
    for(int i = 0; i < numRounds; i++)
        st.push(wordList[i]);

}

//Returns a popped value from the stack for Marathon Mode
string WordFetch::popWordMarathon(){
    //Pops the list when STL's stack is not empty
    if(st.empty() != true){
        string temp = st.top();
        st.pop();
        return temp;
    }else
        //Warns player stack is empty
        cout<<"\nThere are no longer any elements on the stack.\n";
return 0;}

//Informs if stack is empty for outside classes
bool WordFetch::stackIsEmpty(){
    return st.empty();
}



/*****************************************
 *          Queues (FIFO) -              *
 * Uses Queue to display custom words    *
 *****************************************/
```

```
//Pushes custom words onto queue for Custom Word Mode while updating library
void WordFetch::queueCustomWords(string customWord){
    q.push(customWord);

    /****** Updates library ******/

    fstream fileUpdateWords;
    fileUpdateWords.open("words.txt", fstream::app);

        if (fileUpdateWords.is_open()){
              fileUpdateWords<<customWord<<endl;
           fileUpdateWords.close();
        }else
           cout<<"\nWriting Error: Cannot open words.txt to update library.\n";

}

//Returns custom words from queue
string WordFetch::getWordCustom(){
    if(!q.empty()){
        string temp = q.front();
        q.pop();
        return temp;
    }else
        cout<<"\nThe queue of custom words is empty.\n";
    return 0;}
```

//Filename: BalancedTree.cpp

```
#include <iostream>

#include "BalancedTree.h"

#include <queue>


using namespace std;


BalancedBT::BalancedBT(){

   root = NULL;

   cntPost = 0;

   cntIn = 0;

   cntPre = 0;

}


int BalancedBT::getLength(Node *node){

   int maxLength = 0;


   if (node != NULL)

      maxLength = max(getLength(node->left), getLength(node->right));


   return maxLength + 1;

}




/*********************************************

* Functions with different types of rotations *

*********************************************/
```

```
BalancedBT::Node *BalancedBT::rightToRight(Node *node){
    Node *temp;
    temp = node->right;
    node->right = temp->left;
    temp->left = node;
    return temp;
}

BalancedBT::Node *BalancedBT::leftToLeft(Node *node){
    Node *temp;
    temp = node->left;
    node->left = temp->right;
    temp->right = node;
    return temp;
}

BalancedBT::Node *BalancedBT::leftToRight(Node *node){
    Node *temp;
    temp = node->left;
    node->left = rightToRight (temp);
    return leftToLeft (node);
}

BalancedBT::Node *BalancedBT::rightToLeft(Node *node){
    Node *temp;
    temp = node->right;
    node->right = leftToLeft (temp);
```

```
    return rightToRight(node);
}

//Balances Tree with rotating node
BalancedBT::Node *BalancedBT::balance(Node *node){

    if ((getLength(node) - getLength(node)) > 1){

        if ((getLength(node->left) - getLength(node->left)) > 0)
            node = leftToLeft (node);
        else
            node = leftToRight(node);

    }else if ((getLength(node) - getLength(node)) < -1){
        if ((getLength(temp->right) - getLength(temp->right)) > 0)
            node = rightToLeft (node);
        else
            node = rightToRight(node);
    }

    return node;
}

//Insert data into Tree
BalancedBT::Node *BalancedBT::insert(Node *root, string data){
    if (root == NULL){
        root = new Node;
        root->data = data;
        root->left = NULL;
        root->right = NULL;
```

```
        return root;
    }
    else if (data < root->data){
        root->left = insert(root->left, data);
        root = balance (root);
    }else if (data >= root->data){
        root->right = insert(root->right, data);
        root = balance (root);
    }

    return root;
}


void BalancedBT::runInsert(string inVal){
    root = insert(root, inVal);
}


void BalancedBT::print_PreOrder(Node *root){

    if(root != NULL){
        string t = root->data;
        wordPre[cntPre++] = t;
        qPre.push(t);
        cout<<t<<" ";
        print_PreOrder(root->left);
        print_PreOrder(root->right);
    }
}
```

Filename: Hash.cpp

```
#include "Hash.h"
#include <fstream>  //Use to obtain words from words.txt
#include <string>

//Constant string variable to alert that a word DNE in library
const string alertDNE = "WORD-DOES-NOT-EXIST";

Hash::Hash(){
    numItem = 0;
    for(int i = 0; i < tableSize; i++){
        HashTable[i] = new Item;
        HashTable[i]->initial = alertDNE;
        HashTable[i]->next = NULL;
    }




}

void Hash::storeWordInHash(){
    string line;
    ifstream rFileHash ("words.txt");
    if(rFileHash.is_open()){
        while( getline (rFileHash,line) ){
                //cout<<endl<<line;
            insert(line);
        }
```

```
        rFileHash.close();
    }
    else cout<<"Unable to open words.txt for hashing.\n";
}
unsigned int Hash::RSHash(string str){

    unsigned int b = 378551;
    unsigned int a = 63689;
    unsigned int hash = 0;
    unsigned int i = 0;
    unsigned int len = str.length();

    for (i = 0; i < len; i++)
    {
        hash = hash * a + (str[i]);
        a *= b;
    }

    return hash;
}



void Hash::insert(string initial){
    numItem++;
    unsigned int index = RSHash(initial) % 1000; //get an integer index from hash with name as the key

    //Check if that index of the Hash Table array is empty
    if(HashTable[index]->initial == alertDNE){
        //if it is then go ahead an assign the values accordingly
```

```
        HashTable[index]->initial = initial;
    }else{


        //if not then create a pointer pointing to the index of the array for CHAINING
        Item *ptrHashTable = HashTable[index];
        Item *n = new Item; //Creating new Item to store info if HashTable[index] is filled already


        //Assign values to the new node
        n->initial = initial;
        n->next = NULL;


        //Traverse until you reach the end of this chain
        while(ptrHashTable->next != NULL){
            ptrHashTable = ptrHashTable->next;


        };


        //Have the last Item on that chain point to our newest item
        ptrHashTable->next = n;


    }
}
```

//Filename: RecursiveSorts.cpp

```
#include "RecursiveSorts.h"


//Finds min index for recursive selection sort
int RecursiveSorts::recursiveMinIndex(string w[], int indx1, int indx2){
```

```
    if (indx1 == indx2)
        return indx1;

    //Locates min. index for leftovers
    int indx3 = recursiveMinIndex(w, indx1 + 1, indx2);

    //returns min index base on values
    if(w[indx1] > w[indx3])
        return indx1;
    else
        return indx3;
}

//Has recursive bubble sort sort the words backwards
void RecursiveSorts::recurSelectionSort(string w[], int size, int indx){

    //base
    if (indx == size)
      return;

    //stores index of min value
    int minValIndx = recursiveMinIndex(w, indx, size-1);

    //Sorts array of words
    if (minValIndx != indx)
      swap(w[minValIndx], w[indx]);

    //Recursion
    recurSelectionSort(w, size, indx + 1);
}
```

```
//Has recursive bubble sort sort the words in order
void RecursiveSorts::recursiveBubbleSort(string w[], int n){
    //base
    if (n == 1)
        return;

    //sorts array by swapping base on values
    for (int i = 0; i < n-1; i++)
        if (w[i] > w[i+1])
            swap(w[i], w[i+1]);

    //Recursion
    recursiveBubbleSort(w, n-1);
}
```

//Code for Graph inside Hangman.cpp (lines 847 – 960)

```
void Hangman::selectWordsGraph(){
    cout<<endl;
    cout<<"\nUsing Minimum Spanning Tree to select words...\n";



    for(int i = 0; i < nWordGraph; i++){
        wordsWGraph[i] = fetch.getRandWords();
    }
    //graph input
```

```
        int lngth[nWordGraph];

        for(int i = 0; i < nWordGraph; i++)
            lngth[i] = wordsWGraph[i].length();

    int graph[sizeVert][sizeVert] = {

        {0       , lngth[0],     0   , lngth[1], 0   , lngth[2] , lngth[3]},
        {lngth[0],     0   , lngth[4] , lngth[5] , 0   , 0   , lngth[7] },
        {0   , lngth[4] , 0   , 0   , lngth[6], 0   , lngth[8] },
        {lngth[1], lngth[5] , 0   , 0   , 0, 0, lngth[9]   },
        {0   , 0   , lngth[6], 0, 0   , 0, 0},
        {lngth[2] , 0   , 0   , 0, 0   , 0   , 0},
        {lngth[3], lngth[7] , lngth[8]   , 0 , lngth[9], 0   , 0   }

     };



     //uses prim to find the minimum spanning tree of random words to be selected
     //for gameplay
     prim(graph);
}


//Selects new vertex
int Hangman::minVal(int arrVal[], bool vertStat[]){
```

```
  // Initialize min value
  int min = maxValInt, min_index;

  for (int v = 0; v < sizeVert; v++)
    if (vertStat[v] == false && arrVal[v] < min)
        min = arrVal[v], min_index = v;

  return min_index;
}

//Displays results or minimum spanning tree
int Hangman::setWordsGraph(int parent[], int n, int graph[sizeVert][sizeVert]){

   //cout<<"   Edge    \tWeight"<<endl;
   //int weighted = 0;
   for (int i = 1; i < sizeVert; i++){
      //int temp = graph[i][parent[i]];
      //weighted += temp;
      gToQ.push(wordsWGraph[i]);
   }
   //cout<<"\nWeighted = "<<weighted<<endl;

   /*
   cout << "\nThe set gToQ is : ";
   while(!gToQ.empty()){
      cout<<gToQ.front()<<" ";
      gToQ.pop();
   };
   cout << endl;
    */
```

```
}

//Using prim to find the minimum spanning tree of graph
void Hangman::prim(int graph[sizeVert][sizeVert]){
    int parent[sizeVert];
    int val[sizeVert];
    bool vertStat[sizeVert];

    for(int i = 0; i < sizeVert; i++)
        val[i] = maxValInt, vertStat[i] = false;

    val[0] = 0;
    parent[0]--; //The root of min. spanning tree

    for(int count = 0; count < sizeVert-1; count++){
        int newMin = minVal(val, vertStat);

        //set used vertices
        vertStat[newMin] = true;

        for (int cVert = 0; cVert < sizeVert; cVert++)
            if (graph[newMin][cVert] && vertStat[cVert] == false &&
                graph[newMin][cVert] <  val[cVert])
            parent[cVert] = newMin, val[cVert] = graph[newMin][cVert];
     }

     //Shows minimum spanning tree
     setWordsGraph(parent, sizeVert, graph);
}
```

```
void Hangman::graphGameSetup(){
    cout<<endl<<endl;
    cout<<"*************************************************************************"<<endl;
    cout<<"* This game mode inserts the lengths of random words into a Graph *"<<endl
        <<"* then find the graph's minimum spanning tree and use the visited *"<<endl
        <<"* vertices to select which random words to use for the game.      *"<<endl;
    cout<<"*************************************************************************"<<endl;
    cout<<endl;

    selectWordsGraph();

    coreGame(gToQ.size(), "graph");

}
```