Project 1

<Hangman>

CSC-5 41202
Name: Rothman, Alexander
Date: 02/02/16

Introduction

Title: Hangman

Hangman is a word or phrase guessing game.
The player is shown a menu allowing them to play the game, display their saved scores, alter options or quit. When the player chooses to play the game a word is selected based on the current difficulty setting (easy, medium, or hard) from the appropriate word list. The chosen word is then masked using the currently set masking character (defaults to underscore). The masked word is then displayed to the player indicating the number of letters in the word. The player then attempts to guess, one letter at a time, the word. As each character is guessed correctly it is unmasked. Each character guessed is also stored and displayed to the player each turn. If the player's guesses run out before they have completed the word they lose the game. Otherwise they win the game. Upon completing one round the player's score is recalculated and they are allowed to return to the menu.

Summary

Project Size: 592 Lines
Number of variables: 12 (in main) about 45 (all functions)
Number of functions: 16

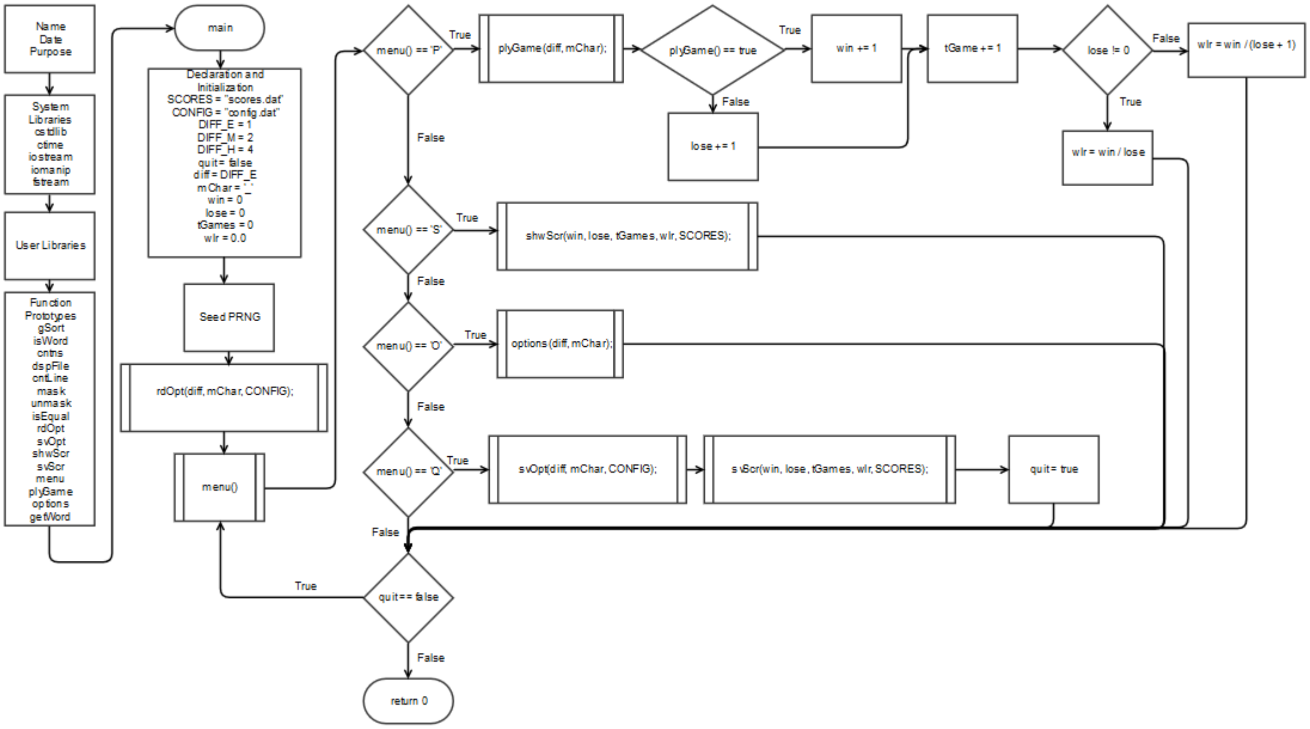
My goal was to write the game using only information already covered in the class. For this reason all string operations preformed in version 2 were replaced with character arrays in version 3. The game is fully functional, has an options menu, and saves scoring data. Scores are not ranked from highest to lowest and are merely appended to the end of the scores file. The game currently allows for only one player and has no computer opponents. Overall the project took around 3 days to become fully functional. When met with problems I primarily relied on the reference found at cplusplus.com and our text books.

The program contains a few concepts not covered prior such as converting a time value to text and sorting a character array.

Description

The major point of this program is to utilize all the major concepts we have learned up to this point and use them to create a game.

Flowchart



Pseudocode

```
Declare SCORES and set it to "scores.dat"
Declare CONFIG and set it to "config.dat"
Declare DIFF_E and set it to 1
Declare DIFF_M and set it to 2
Declare DIFF_H and set it to 4
Declare quit and set it to false
Declare diff and set it to DIFF_E
Declare mChar and set it to '_'
Declare win, lose, and tGames and set them to 0
Declare wlr and set it to 0.0
Seed the PRNG by calling srand() and feeding in
time(0)
Call rdOpt() and pass in diff, mChar, and CONFIG
*Call menu()
If menu() returns 'P'
    Call plyGame() and pass in diff and mChar
    If plyGame() returns true
        Increment win
    Else
```

```
        Increment lose
    Increment tGames
    If lose is not 0
        Calculate wlr by dividing win by lose
    Else
        Calculate wlr by dividing win by lose + 1
Else if menu() returns 'S'
    Call shwScr() and pass in win, lose, tGames, wlr,
    and SCORES
Else if menu() returns 'O'
    Call options() and pass in diff and mChar
Else if menu() returns 'Q'
    Call svOpt() and pass in diff, mChar, and config
    Call svScr() and pass in win, lose, tGames,wlr,
    and SCORES
    Set quit equal to true
If quit is not true
    loop back to calling menu() (*)
Return 0
```

Major Variables

Type	Variable Name	Description	Location
bool	quit	Flag controlling whether or not to quit the game	main()
char	SCORES[]	Path to file containing game scoring data	main()
	CONFIG[]	Path to file containing game configuration data	main()
	DIFF_E	A numerical value representing easy mode	main()
	DIFF_M	A numerical value representing medium mode	main()
	DIFF_H	A numerical value representing hard mode	main()
	diff	The current game difficulty	main()
	mChar	The current character used to mask hidden words	main()
	GMAX	The hardcoded maximum number of guesses a player may have	plyGame()
	WDLNGTH	The length of a word in the game	plyGame()
	ALPHA	The length of the used character array	plyGame()

	gCount	The current number of guesses remaining	plyGame()
	usedPos	The position of the next character to be inserted in the array of used characters	plyGame()
	guess	The players most recent guess	plyGame()
	word[]	The actual word to guess	plyGame()
	mWord[]	The word to guess masked using the masking character	plyGame()
	used[]	An array containing each character already used by the player	plyGame()
short	win	The number of games won	main()
	lose	The number of games lost	main()
	tGames	The number of games played	main()
int	length	The length of the selected word	plyGame()
float	wlr	The ratio of wins to losses	main()

C++ Constructs

Chapter	New Syntax and Keywords	Location

2	cout	cout << "HARD" << endl;
		cout << "CURRENT GAME:" << endl;

	#include directives	#include <cstdlib>
		#include <ctime>

	Variables and Literals	win = 0;
		lose = 0;

	Identifiers and keywords	const char SCORES[] break;

	Integer data types	unsigned short win = 0;
		int lnCount;

	Character data types	unsigned char diff const char E_LIST[] = "wdleasy.txt";
	Floating point data types	float wlr = 0.0f;
	Boolean data types	bool quit = false;
	Arithmetic operators	static_cast<float>(win) / (lose + 1);
	Comments	//Scores file
	Named Constants	const unsigned char DIFF_E = 1
3	cin	cin >> input;
	Type conversion	static_cast<float>(win) / lose
	Formatting output	cout << "Options:" << setw(20) << "(D)ifficulty" << setw(20) << "(M)ask Character" << setw(10) << "(H)elp" << endl;
4	Relational operators	if(diff == 1)

	If statement	if(diff == 1)

	If/Else and If/Else if	if(diff == 1){
		cout << "EASY" << endl;
		}
		else if (diff == 2){
		cout << "MEDIUM" << endl;
		}
		else{
		cout << "HARD" << endl;
		}

	Logical operators	temp != 'E' && temp != 'M' && temp

	Input validation	do{
		cout << "> ";
		cin >> input;
		temp = toupper(input[0]); //get choice from input
		}
		while(temp != 'E' && temp != 'M' && temp != 'H');

	Ternary operator	wlr = lose != 0 ? static_cast<float>(win) / lose //Calculate win loss ratio
		: static_cast<float>(win) / (lose + 1);

	Switch statement	switch(temp){ //Choose based on second choice
		case 'E': //Set difficulty to easy
		{
		diff = 1;
		break;

		}
		case 'M': //Set difficulty to medium
		{
		diff = 2;
		break;
		}
		case 'H': //Set difficulty to hard
		{
		diff = 4;
		break;
		}

5	Increment and Decrement operators	++tGames;

	While loop	while (iFile.getline(next, 256, '\n')) { //read each line
		cout << next << endl; //output line
		}

	Do while loop	do{ //Input validation
		cout << "> ";
		cin >> input;
		choice = toupper(input[0]); //get choice from input's first character
		} while(choice != 'D' && choice != 'M' && choice != 'H');

	For loop	for(int i = 0; i < length; ++i){ //loop through all characters
		if(word1[i] != word2[i] && word1[i] != '\0'){ //compare characters unless they are null

		return false;
		}
		}

	File I/O	ifile.open(path); //Open the file
		while (ifile >> temp) { //read while the stream is good
		++counter; //increment counter
		}
		ifile.close(); //close file

Reference

1. C++ form Control Structures through Objects 8th Ed.
2. http://www.cplusplus.com/reference/
3. https://en.wikipedia.org/wiki/Gnome_sort

Program

```
/*
 * File:   main.cpp
 * Author: Alexander Rothman
 * Purpose: Hangman game
 * Created on January 30, 2016, 4:12 AM
 */

//System Libraries
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace std;

//User Libraries

//Global Constants

//Function Prototypes
//Gnome Sort
void gSort(char[], int);
//Is Word
bool isWord(const char[], int);
//Contains
bool cntns(const char[], int, char);
//Display File
void dspFile(const char[]);
//Count Lines
int cntLine(const char[]);
//Mask
void mask(int, const char[], char[], char);
//Unmask
void unmask(int, const char[], char[], char);
//Is Equal
bool isEqual(const char[], const char[], int);
//Game Functions
//Read Options
void rdOpt(unsigned char&, unsigned char&, const char[]);
//Save Options
void svOpt(unsigned char, unsigned char, const char[]);
//Show Scores
void shwScr(short, short, short, float, const char[]);
//Save Scores
void svScr(short, short, short, float, const char[]);
//Menu
char menu(void);
```

```
//Play Game
bool plyGame(unsigned char, unsigned char);
//Options
void options(unsigned char&, unsigned char&);
//Get Word
void getWord(unsigned char, char[], int&);

//Begin Execution
int main(int argc, char** argv) {
    //Declaration and Initialization
    const char SCORES[] = "scores.dat", //Scores file
               CONFIG[] = "config.dat"; //Configuration file
    const unsigned char DIFF_E = 1, //Easy Difficulty
                        DIFF_M = 2, //Medium Difficulty
                        DIFF_H = 4; //Hard Difficulty
    bool quit = false; //Determine whether or not to quit the game
    unsigned char diff = DIFF_E, //Current difficulty
                  mChar = '_'; //Masking character
    unsigned short win = 0, //Number of wins
                   lose = 0, //Number of losses
                   tGames = 0; //Total number of games
    float wlr = 0.0f; //Win loss ratio

    srand(static_cast<int>(time(0))); //Seed PRNG
    rdOpt(diff, mChar, CONFIG); //Read options and map them to settings

    //Game Loop
    do{
        switch(menu()){ //Choose the input from the menu display
            case 'P': //Play the game
            {
                plyGame(diff, mChar) ? ++win : ++lose; //Play a game and increment 
scores
                ++tGames; //Increment number of games
                wlr =  lose != 0 ? static_cast<float>(win) / lose //Calculate win 
loss ratio
                                 : static_cast<float>(win) / (lose + 1); 
//Calculate ratio if denominator is 0
                break;
            }
            case 'S': //Display Scores
            {
                shwScr(win, lose, tGames, wlr, SCORES); //Display score list
                break;
            }
            case 'O': //Display Options
            {
                options(diff, mChar); //Display options menu
                break;
            }
            case 'Q': //Quit
            {
                svOpt(diff, mChar, CONFIG); //Write config data to file
                svScr(win, lose, tGames, wlr, SCORES); //Write score data to file
                quit = true; //Set quitting flag
            }
        }
    } while(!quit);
    //Exit
    return 0;
}

/******************************************************************************/
/********************************Read Options**********************************/
/******************************************************************************/
//  Read the current game configuration from a file
//Inputs
//  &diff : The current difficulty setting
//  &mChar : The current masking character
//  path : The location of the file you're reading from
void rdOpt(unsigned char &diff, unsigned char &mChar, const char path[]){
    char input[3] = {0}; //input array for file reading
    ifstream iFile; //input file stream

    iFile.open(path); //open the file
    iFile.getline(input, 3, '\n'); //read the first value
    diff = input[0]; //grab the difficulty out of the first value
    iFile.getline(input, 3, '\n'); //read the second value
    mChar = input[0]; //grab the masking character
    iFile.close(); //close the file
}

/******************************************************************************/
/********************************Save Options**********************************/
/******************************************************************************/
//  Write the current game configuration to a file
//Inputs
//  diff : The current difficulty setting
//  mChar : The current masking character
//  path : The location of the file you're writing to
void svOpt(unsigned char diff, unsigned char mChar, const char path[]){
    ofstream oFile; //Output file stream

    oFile.open(path); //open file
    oFile << diff << endl; //Output difficulty
    oFile << mChar << endl; //Output masking character
    oFile.close(); //close file
}

/******************************************************************************/
/********************************Options***************************************/
/******************************************************************************/
//  Display and handle options menu
//Inputs
//  &diff : The current difficulty setting
//  &mChar : The current masking character
void options(unsigned char &diff, unsigned char &mChar){
    char choice; //The chosen option
    char input[15] = {0}; //The input string to get the choice from

    //Output menu
    cout << "Options:" << setw(20) << "(D)ifficulty"
         << setw(20) << "(M)ask Character" << setw(10) << "(H)elp" << endl;

    //Input data
    do{ //Input validation
        cout << "> ";
        cin >> input;
        choice = toupper(input[0]); //get choice from input's first character
    } while(choice != 'D' && choice != 'M' && choice != 'H');

    switch(choice){ //choose based on the value of choice
        case 'D': //Change the difficulty
        {
            char temp; //The choice for the second switch

            cout << "CURRENT DIFFICULTY: ";
            //Covert difficulty to text
            if(diff == 1){
                cout << "EASY" << endl;
            }
            else if (diff == 2){
                cout << "MEDIUM" << endl;
            }
            else{
                cout << "HARD" << endl;
            }
            //Output availaible modes
            cout << "Modes: " << "(E)asy (M)edium (H)ard" << endl;
            //Input data
            do{
                cout << "> ";
                cin >> input;
                temp = toupper(input[0]); //get choice from input
            }while(temp != 'E' && temp != 'M' && temp != 'H');

            switch(temp){ //Choose based on second choice
                case 'E': //Set difficulty to easy
                {
                    diff = 1;
                    break;
                }
                case 'M': //Set difficulty to medium
                {
                    diff = 2;
                    break;
                }
                case 'H': //Set difficulty to hard
                {
                    diff = 4;
                    break;
                }
            }
            cout << "CURRENT DIFFICULTY: ";
            if(diff == 1){
                cout << "EASY" << endl;
            }
            else if (diff == 2){
                cout << "MEDIUM" << endl;
            }
            else{
                cout << "HARD" << endl;
            }
            break;
        }
        case 'M': //Change the masking character
        {
            cout << "CURRENT MASK CHARACTER: " << mChar << endl;
            //Input Data
            do{ //Input Validation
                cout << "> ";
                cin >> input;
                if((input[0] > ' ' && input[0] < 'A') ||
                   (input[0] > 'Z' && input[0] < 'a') ||
                   (input[0] > 'z' && input[0] < 127)){ //if the input character is 
printable but not alphabetic
                    mChar = input[0]; //set masking character
                }
                else{ //Otherwise output error message
                    cout << "> INVALID INPUT" << endl;
                }
            } while((input[0] < ' ' && input[0] > 'A') ||
                    (input[0] < 'Z' && input[0] > 'a') ||
                    (input[0] < 'z' && input[0] > 127));
            cout << "CURRENT MASK CHARACTER: " << mChar << endl;
            break;
        }
        case 'H': //Output help information
        {
            dspFile("help.dat");
            break;
        }
    }
}

/******************************************************************************/
/*******************************Save Scores************************************/
/******************************************************************************/
//  Save scoring data to a file
//Inputs
//  win : The current number of wins
//  lose : The current number of losses
//  tGames : The current total number of games played
//  wlr : The current win loss ratio
//  path : A file path to write to
void svScr(short win, short lose, short tGames, float wlr,
                const char path[]){
    ofstream oFile; //Output file stream
    time_t rawTime; //The current time as a time_t object

    if(tGames > 0){ //If a game has been played
        time(&rawTime); //Feed current time into rawTime

        oFile.open(path, fstream::app); //Open file and seek to the end
        //Output Data to file
        oFile << ctime(&rawTime) << "SCORES: " << win << " : " << lose << endl;
        oFile << "TOTAL GAMES: " << tGames << endl;
        oFile << "RATIO: " << wlr << endl;
        oFile.close(); //Close output
    }
}

/******************************************************************************/
/*******************************Show Scores************************************/
/******************************************************************************/
```

```
//  Display scoring data
//Inputs
//  win : The current number of wins
//  lose : The current number of losses
//  tGames : The current total number of games played
//  wlr : The current win loss ratio
//  path : A file path to read scores from
void shwScr(short win, short lose, short tGames, float wlr,
                const char path[]){
    //Output current data
    cout << "CURRENT GAME:" << endl;
    cout << "SCORES: " << win << " : " << lose <<  endl;
    cout << "TOTAL GAMES: " << tGames << endl;
    cout << fixed << setprecision(2) << showpoint;
    cout << "RATIO: " << wlr << endl;
    //Output data from file
    dspFile(path);
}

/******************************************************************************/
/*******************************Is Equal***************************************/
/******************************************************************************/
//  Test whether or not two words of equal length are the same
//Inputs
//  word1 : the first word to compare
//  word2 : the second word to compare
//  length : the length of the words
//Outputs
//  true if all characters match
//  false if there is a single character difference
bool isEqual(const char word1[], const char word2[], int length){
    for(int i = 0; i < length; ++i){ //loop through all characters
        if(word1[i] != word2[i] && word1[i] != '\0'){ //compare characters unless
they are null
            return false;
        }
    }
    return true;
}

/******************************************************************************/
/*********************************Unmask***************************************/
/******************************************************************************/
//  Unmask a single character in a masked string based on an unmasked string
//Inputs
//  length : the length of the strings to use
//  word : the unmasked string
//  mWord : the masked string
//  guess : the character to search for
void unmask(int length, const char word[], char mWord[], char guess){
    for(int i = 0; i < length; ++i){ //loop through all characters
        if(word[i] == guess){ //compare each character to the guess character
            mWord[i] = guess; //Unmask the character if they match
        }
    }
}

/******************************************************************************/
/*********************************Count Lines**********************************/
```

```
/*******************************************************************************/
//  Count the number of lines in a file
//Inputs
//  path : the path to the file to count
//Outputs
//  counter : the number of lines counted in a file
int cntLine(const char path[]){
    int counter = 0; //The line count
    char temp[256]; //A temporary string to read into
    ifstream ifile; //The input file stream

    ifile.open(path); //Open the file
    while (ifile >> temp) { //read while the stream is good
        ++counter; //increment counter
    }
    ifile.close(); //close file

    return counter;
}

/*******************************************************************************/
/*************************************Mask**************************************/
/*******************************************************************************/
//  Puts a string on equal length to an input string into a new array
//  Fills the new string with the given character
//Inputs
//  length : the length of the input string
//  word : the input string
//  mWord : the return string
//  mChar : The masking character
void mask(int length, const char word[], char mWord[], char mChar){
    for(int i = 0; i < length; ++i){ //loop through the words
        if(word[i] != '\0'){ //if the character is not null
            mWord[i] = mChar; //set to a masking character
        }
    }
}

/*******************************************************************************/
/*********************************Get Word**************************************/
/*******************************************************************************/
//  Gets a word from the word lists
//Inputs
//  diff : The current game difficulty
//  buffer : the array to return the word in
// &length : the length of the word to be returned
void getWord(unsigned char diff, char buffer[], int &length){
    const unsigned char E_LNGTH = 7, //Easy word length
                        M_LNGTH = 12, //Medium word length
                        H_LNGTH = 14; //Hard word length
    const char E_LIST[] = "wdleasy.txt", //Easy word file path
               M_LIST[] = "wdlmedium.txt", //Medium word file path
               H_LIST[] = "wdlhard.txt"; //Hard word file path
    int lnCount; //The line count
    ifstream iFile; //input file stream

    if (diff == 1){ //If Easy
        iFile.open(E_LIST); //Open file
        lnCount = cntLine(E_LIST); //count file lines
        int target = rand() % lnCount; //pick a random line from the file
        for(int i = 0; i < target; ++i){ //read to that line
            iFile >> buffer;
        }
        iFile.close(); //close file
        length = E_LNGTH; //set length
    }
    else if (diff == 2){ //If Medium
        iFile.open(M_LIST);
        lnCount = cntLine(M_LIST);
        int target = rand() % lnCount;
        for(int i = 0; i < target; ++i){
            iFile >> buffer;
        }
        iFile.close();
        length = M_LNGTH;
    }
    else{ //If Hard
        iFile.open(H_LIST);
        lnCount = cntLine(H_LIST);
        int target = rand() % lnCount;
        for(int i = 0; i < target; ++i){
            iFile >> buffer;
        }
        iFile.close();
        length = H_LNGTH;
    }
}

/******************************************************************************/
/*********************************Play Game***********************************/
/******************************************************************************/
//  Main game processing function. Handle one game of hangman and then return
//Inputs
//  diff : the current difficulty
//  mChar : the current masking character
//Outputs
//  true if you win
//  false if you lose
bool plyGame(unsigned char diff, unsigned char mChar){
    const unsigned char GMAX = 24, //The maximum number of guesses
                        WDLNGTH = 20, //word length
                        ALPHA = 28; //alphabet length
    char gCount = GMAX / diff, //The actual guess count
         usedPos = 0, //The number of used characters
         guess; //The current guess
    int length = 0; //The length of the chosen word
    char word[WDLNGTH] = {0}, //The actual word to guess
         mWord[WDLNGTH] = {0}, //The masked word
         used[ALPHA] = {0}; //The list of used characters

    getWord(diff, word, length); //Get a word
    mask(length, word, mWord, mChar); //Mask the word

    do{ //Turn loop
        char input[2]; //input buffer
        cout << mWord << endl;
        cout << "REMAINING GUESSES:" << static_cast<int>(gCount) << endl;
        cout << "USED CHARACTERS: " << used << endl;
        cout << "> ";
        //Input Data
        cin >> input;
        guess = tolower(input[0]); //convert input buffer to single guess character
        if(cntns(word, WDLNGTH, guess)){ //If word contains guess
            unmask(length, word, mWord, guess); //unmask all characters that match 
guess
        }
        else{ //If word does not contain guess
            --gCount; //Subtract one guess
        }
        if(!cntns(used, ALPHA, guess)){ //If used characters doesn't contain guess
            used[usedPos++] = guess; //add guess to used characters and increment 
position
            gSort(used, ALPHA); //Sort used characters
        }
    } while(!isEqual(word, mWord, WDLNGTH) && gCount > 0);
    //Game end processing
    cout << "ANSWER: " << word << endl; //output the actual answer
    if(isEqual(word, mWord, WDLNGTH)){ //If the guessed word matches the actual 
word
        cout << "YOU WIN!" << endl;
        return true;
    }
    else{ //If the guessed word is incomplete
        cout << "YOU LOSE!" << endl;
        return false;
    }
}

/******************************************************************************/
/********************************Display File**********************************/
/******************************************************************************/
//  Write the text from a file to standard out one line at a time
//Inputs
//  path : the path to the file to display
void dspFile(const char path[]){
    char next[256]; //input buffer
    ifstream iFile; //input file stream

    iFile.open(path); //open file
    while (iFile.getline(next, 256, '\n')) { //read each line
        cout << next << endl; //output line
    }
    iFile.close(); //close file
}

/******************************************************************************/
/************************************Menu**************************************/
/******************************************************************************/
//  Main menu processing function. Returns the player's choice as a character
//Outputs
//  The character representing the player's choice of action
char menu(){
    const char TITLE[] = "gametitle.txt"; //File containing game title text
    char choice; //The user's choice
    char input[8] = {0}; //The string input by the user to cut the choice from

    dspFile(TITLE); //Display the title file
    //Display the game menu
    cout << "Menu:" << setw(15) << "(P)lay" << setw(15) << "(S)cores"
         << setw(15) << "(O)ptions" << setw(10) << "(Q)uit" << endl;

    do{ //Input validation
        //Read in input
        cout << "> ";
        cin >> input;
        //Chop choice from input
        choice = toupper(input[0]);
        //Display error message
        if(choice != 'P' && choice != 'S' && choice != 'O' && choice != 'Q'){
            cout << "> Invalid Input. Please input (P), (S), (O), or (Q) without 
parentheses"
                 << endl;
        }
    } while(choice != 'P' && choice != 'S' && choice != 'O' && choice != 'Q');


    return choice;
}

/******************************************************************************/
/**********************************Contains************************************/
/******************************************************************************/
//  A function to determine if a string contains a character
//Inputs
//  word : the string to compare to
//  length : the length of the string
//  key : the character to search for
//Outputs
//  true if the character is found in the string
//  false if the character is not found in the string
bool cntns(const char word[], int length, char key){
    for(int i = 0; i < length; ++i){ //loop through word
        if(word[i] == key){ return true; } //if a character matches the key return 
true
    }
    //Otherwise return false
    return false;
}

/******************************************************************************/
/***********************************Is Word************************************/
/******************************************************************************/
//  Checks if a string only contains alphabetic characters
//Inputs
//  word : the string to check
//  length : the length of the input string
//Outputs
//  true if the string only contains alphabetic characters
//  false if the string contains non alphabetic characters
bool isWord(const char word[], int length){
    for(int i = 0; i < length; ++i){ //loop through every character in word
        if(word[i] > 126 || word[i] < 33) { return false; } //if word only contains 
alphabetic characters
    }
    return true;
}
```

```
/******************************************************************************/
/*********************************Gnome Sort***********************************/
/******************************************************************************/
//  Sort a character array. Weights null characters as higher than all others
//Inputs
//  cArr : the character array to sort
//  length : the length of the array to sort
void gSort(char cArr[], int length){
    for(int pos = 1; pos < length;){ //Gnome Sort modified to handle null
terminators
        if(cArr[pos] >= cArr[pos - 1] || cArr[pos] == '\0'){ //if the current
character is greater than the previous one or null
            ++pos; //move one position forward
        }
        else if (cArr[pos] <= cArr[pos - 1]){ //If the current character is less
than or equal to the previous
            //Swap the current character and the last
            cArr[pos] = cArr[pos] ^ cArr[pos - 1];
            cArr[pos - 1] = cArr[pos] ^ cArr[pos - 1];
            cArr[pos] = cArr[pos] ^ cArr[pos - 1];
            if(pos > 1){ //If the position is greater than one
                --pos; //Move back one position
            }
        }
    }
}
```