Project 1 - Hangman

Miguel Juarez

February 2024

1 Introduction

This program was written to play a concise version of the game Hangman. Without the figure being drawn, players can still experience the game with lives in place of the well-known hanging man. This project took a lot more planning than coding but during the development process, I learned what I needed to accomplish but was not able to implement at the time of the final version. I have, to the best of my ability, completed a sort of working game that allows for player interaction and how the final product will result.

2 How To Use the Program

This program has very simple steps that the user can also follow along with easy readability and user-friendliness. Begin by running the code, the user will be displayed with a welcome message and a list of rules to follow about how to play the game. The user is then asked to pick a number from 1 - 10 that will determine the word they will try to guess. A prompt then appears for the user to input their name that will hold their name in a file to add points upon completion of the word later.

3 Program Design

The design for this program requires several C++ libraries to be included for the game to operate and include arrays, vectors, and implementation of functions for better visibility and game play action for the players. Function prototypes and local constants are declared outside of the main program to be manipulated and worked around without disrupting the flow of the purpose of the program. Lets begin by taking apart the function prototypes and the constants outside of the main function and breaking down their individual components to see how they all come together to make this game run for the player.

4 Choosing Word Function

```
//Function Prototypes
void choosingword(int&, string&, string[], vector<int>&);
```

Beginning with **void choosingword**, this is the first function called inside the do-while loop, that is responsible for selecting a word for the player to guess on the chosen difficulty level. This function has three parameters that all constructively bring the rest of the game together and are geld inside this function for easy access. The **chWrd** is passed by reference and it holds the difficulty level word that is chosen by the player when prompted to choose a level of difficulty. It is modified within the function and it is used to make sure that it is within the range of the srand of (1-10).

```
// Choose random number from 1 to 10 & validate
cout << "Choose a level from 1 - 10:";
cin >> chWrd;
```

The next parameter is **gsWrd** which is a string data type being passed by reference that represents the word that the player is playing to guess. This works with chWrd as it is the chosen word based on the number chosen by the player within the chWrd variable. Following that is a string array that is passed into the function that will store the chosen word. This array will be used for tracking the words that have and will be chosen throughout all gameplay and will be updated and reset when the player wants to play again.

```
// Set word to string
string words[10] = {"CAR", "TREE", "LINUX", "MONKEY", "COMPUTER",
        "WEREWOLF", "INFLATION", "AUTOMOBILE", "RELATIONSHIP",
        "QUINTESSENTIAL"};
if (chWrd >= 1 && chWrd <= 10) {
    gsWrd = words[chWrd - 1]; // Adjust index
} else {
    // Handle invalid input
    while (chWrd > 10 || chWrd <= 0 || chWrd != '0' ||
    chWrd != '9') {
        cout << "Invalid number, try again:" << setw(2);
        cin >> chWrd;
    }
}
```

Finally, the vector integer type that is passed is called **scores** which will be used to track the scores for each of the words being chosen. This works with later functions that can be used for the files that the game writes to.

5 Jumping inside Main

After this function is called, moving inside the main function is another action that takes place once the previous function is called and executed. A file called **nameNpoints.txt** is opened to read in the player's name to the file. The following prompt is next in the gameplay allowing for tracking of the user's points and name.

```
// Name to keep score
    cout << "Enter your name: ";
    cin >> usrNm;

    // Check if user inputs a name
    ofstream wrdBnk("nameNpoints.txt");//declared for writing
    while (usrNm.empty()) {
        cout << "Invalid, please input your name again: ";
        cin >> usrNm;
    }
    // Write to the file
    wrdBnk << usrNm;
    wrdBnk.close();
```

The player is instructed to input their name and is checked to make sure that what they input is correct. If the player fails to input anything into the **usrNm** variable, they are instructed to input their name again as that is what they will be saved as. After the player name is set to the variable, it is written into a file and the file is closed to prevent any issues with the file after this execution.

6 Gameplay Function

```
void gameplay(string& gsWrd, int chWrd,int& lives,char usrLtr
, char ltrsGs[], int& numGs)
```

This function is the biggest function in the game but for a valid reason as it is responsible for managing the actual gameplay loop from updating words and user validation. The function passes in 6 parameters showing its size and impact on the program. Inside the function, there is an initialization for the variable that will be responsible for keeping track of the word state as the player plays the game. The variable **wrdSt** is declared as a 2D array for better formatting, where each element of this array represents a character inside of the word passed through the string as a reference to gsWrd. The array replaces the character for a **'-'** (dash) that represents unrevealed letters when playing the game which looks as follows:

```
Current word: - - - - - - - -
Guessed letters:
Guess a letter:
```

Depending on the letter that the user inputs, it is passed through the array **ltrsGs** which is where all previous letters that were guessed are stored. This array checks if the guessed letter is found inside of the array and if it is true -¿ it flags **ltrAgn** to be true which lets the player know that they have already guessed that letter. The code is then passed through a while loop that will continually check for the number of lives the player has left. inside this loop, the inputted letters are converted to capital and are displayed which means the user can input lowercase and uppercase letters as well. Inside this loop, the **selection sort algorithm** is used to sort the letters that have been guessed and prints them on the screen after they are sorted through the array of letters guessed.

```
//Select Sort for letters guessed
    for (int i = 0; i < numGs - 1; ++i) {
        int minIdx = i;
        for (int j = i + 1; j < numGs; ++j) {
            if (ltrsGs[j] < ltrsGs[minIdx]) {
                minIdx = j;
            }
        }
        if (minIdx != i) {
            // Swap ltrsGs[i] and ltsGs[minIdx]
            char temp = ltrsGs[i];
            ltrsGs[i] = ltrsGs[minIdx];
            ltrsGs[minIdx] = temp;
        }
    }
```

After each guess, the current state of the word is updated and then displayed for the player to see. The loop checks after every guess if the array **wrdSt** to see if it has been correctly guessed, where no dashed are left to guess. This follows a bool statement to be flagged as true where **compGs** is set true for a completed guess. This loop is executed until the player either guessed the word or has 0 lives left which makes the condition false and is exited.

When the loop ends a congratulations message appears congratulating the player for solving the word, or a game over screen is printed where the correct word is also displayed. This function is the core of gameplay for the hangman logic which has everything the game needs to operate and function properly.

```
//Game over screen
Game over! You ran out of lives. The word was: CAR

//Winner Screen
Game over! You ran out of lives. The word was: CAR
```

7 Jumping back into Main

Moving back into the main function after the gameplay function as called, there are some lines that operate before the end of the game is completed and for the end game. The lines of code leading to the end of the do-while loop manage the points eaerrned by the players during the game and it updates the file that their names were originally written to. Beggining with the ternary operator

```
    int pts = (lives == 0) ? 0 : WIN_POINTS;
```

THis initlaizes the variable pts to 0 on the condition that the player has run out of lives and makes it so that no points are added, but if the lives are not 0, pts is initialized to WIN POINTS for the total points earned from the game. The conditional statement follows:

```
    if (lives == 0) {
        cout << "Game over! You ran out of lives. The word was: " << gsWrd
        << endl;
        pts += 0;

        while (rdWrds >> usrNm) {
            cout << usrNm << " has won no points." << endl;
            cout<<endl;
        }

        // Update file
        wrdBnk << usrNm << " " << pts << " has points." << endl;
        }
```

where the game over screen is printed if the condition is met and also updates the file with the total points earned and player name.

8 Again or Exit

The function following the gameplay function is a boolean return type that will set the condition inside the do-while loop to be true or false.

```
    bool againORexit()
```

This function's functionality is simple as it prompts the player a question asking if they would like to play again or exit the program. The functions stores the users choice of **(y/n)** as a variable **choice** that will determine if the character is a 'y' it will loop back making the condition true. If the user inputs 'n' as their choice, the program exits the do while loop that is followed by the return 0; to close the main program.

```
    bool againORexit() {
    char choice;
    cout << "Do you want to play again? (y/n): ";
    cin >> choice;
    return (choice == 'Y' || choice == 'y');
}
```

9 Results

The resulting gameplay should look something like this when the player wants to begin playing:

```
Welcome to HangMan!
Rules:
1. Do not repeat the same letter
2. Solve the mystery word
3. Have fun!
--------------------------------
Choose a level from 1 - 10: 3
```

The player chooses level **3**. The word to guess is **LINUX**.

```
Enter your name: John
```

The player enters **John** as their name and it is stored in the **nameNpoints** text file.

```
Current word: - - -
Guessed letters:
Guess a letter: i
```

The player enters the letter **i** which is not in the word CAR. The program detects that the letter is not in the array with the word and is deducted a life for the incorrect letter. The game continues until the player solves the word.

```
Incorrect. Lives remaining: 4
Current word: C A R
Guessed letters: I C A R
Guess a letter:
```

The player guesses the word correctly and sees the letters he guessed already, including the correct letters. The player wins and is prompted with this screen.

```
Congratulations! You guessed the word!
John has won 10 points!

Do you want to play again? (y/n): n
Thank you for playing!
```

The player's name is read back in from the file and prompted with their total points and the option to play again. The player in this case did not want to play again and was given a thank you screen to end the program. If the player wanted to play again, this cycle would repeat until they no longer want to play.

10 References

1. https://cplusplus.com/
2. Gaddis, Tony, et al. Starting out with C++. Pearson Education, Inc., 2020.