Project 5 Design Document

**Program Requirements:**

This program is designed to run a game of “Hangman”. The program will open a text file that has a random selection of words in all caps, one word per line. We know that the text file only contains 30 words. We know that the average size of a word is 4 letters, so we will use 40 as a maximum word size (10 times the average). We know there are 26 words in the English alphabet. We know that the player is allowed a maximum number of 20 guesses.

**Program Inputs:**

* The guessed letter
  + Char guess
  + Must be a character.
  + Holds the char value of the next letter the user has guessed.
* The maximum amount of words
  + Const int MAX\_WORDS = 100
  + Will always be 100.
  + Declared in the global scope
* The maximum size of any word
  + Const int MAX\_WORDS\_SIZE = 40
  + Will always be 40.
  + Declared in the global scope
* The number of letters in the English alphabet
  + Const int NUM\_LETTERS = 26
  + Will always be 26.
  + Declared in the global scope
* The maximum amount of guesses
  + Const int MAX\_GUESSES = 20
  + Will always be 20
  + Declared in the global scope

**Program Outputs:**

* Capital letter of the English alphabet
  + string alpha = “ABCDEFGHIJKLMNOPQRSTUVWXYZ”
  + Contains every capital letter of the alphabet
  + Will replace letters with “ “ as they are used
* List of possible words
  + string words[]
  + An array containing each word from the text file
  + The size of this array is equal to MAX\_WORDS
* Word number variable
  + int wordNum = 0
  + Will hold a randomly assigned value between 0 and 30
  + Value will be an index value for words[]
* Number of letters variable
  + Int letters=0
  + Will hold the number of letters in the chosen word
  + Will be a positive non-zero integer value no greater than 11
* Number of misses
  + int misses = 0
  + Keeps track of how many incorrect guesses the user has made.
  + Can be any positive integer value between 0 and 20
* Number of words
  + int numWords = 0
  + Stores how many words are avaliable
  + Must be a positive non-zero integer between 20 and 100
* Letters guessed
  + int guessed[]
  + An array that stores a bool index value.
  + At each index, a value of true represents a correct guess of the corresponding letter.
  + A value of false represents an incorrect guess.
  + The array is the same size as MAX\_WORD\_SIZE

**Test Plan:**

The tests will verify that the program can handle different amounts of words between 20 and 100. The test will also ensure that the program will work with a variety of different word lengths. The program should keep track and print any amount of misses. The program should be able to keep track of which letters have been guessed. The program should correctly determine if the user wins or loses. By the nature of the program, the output cannot be predicted unless the user immediately knew the word. Play testing should include at least 10 trials. Each trial will use a different “puzzles.txt” file. To avoid bias, I have recruited my little cousin as game tester.

**Solution Overview:**

We will be using the ‘iostream’ library to define our I/O stream objects. We will also be using the ‘fstream’ library to access the needed file. The ‘string’ library will be necessary to incorporate string objects. The ‘ctime’ and ‘cstdlib’ will be needed to generate random numbers. We start with declaring our variables. There will be 4 constants declared in the global scope (MAX\_WORDS, MAX\_WORDS\_SIZE, NUM\_LETTERS, and MAX\_GUESSES). The string alpha will be declared in the scope of the main function along with the string array words[]. 4 integers will also be declared in the scope off the main function: wordNum, letters, misses, numWords. In addition, we will have a char variable guess, and a bool array guessed[].

Start by calling the read\_puzzles(string words[], int &wordCount) function. The function creates a ifstream object to open and read values from the file “puzzles.txt”. If the file is open, the program should read each of the words in the text file and store them in the words[] array. This can be accomplished by using a while loop with the conditions of wordcount<=MAX\_WORDS && the end of file has not been reached. The pass by reference variable wordcount should increment by 1 with each iteration. If the function is unable to open the file, and error message should appear. Lastly, the function should close the file.

Now we will generate a random number and use that number to determine what word we will be using in the game. The srand() function can be used to set the seed of the rand() function. Seed the rand() function with the time() function to ensure an effectively random number every time the program is compiled. Next, set the variable letters equal to the length of the word located at the randomly generated index value. Then, use a for loop to set all the values in the bool array guessed[] to “false”.

This section requires a do-while loop. The code block should start by calling the display\_puzzle(string word, bool guessed[]) function. The function should use a for loop to check the values of the bool array guessed[] for any “true” values. If a value is true, the corresponding letter in the string data structure should be printed to the screen. If a value is false, an underscore will be printed instead.

The code block of the do-while loop will continue by printing the available letters left to choose (alpha[]) and prompting the user to enter their next guess. All values entered into guess should be made uppercase. Use a linear search algorithm to find the letter the user chose in the array alpha[] and replace the letter with a space. Use an if statement to determine if found(string word, bool guessed[], char guess) function returned a false value. If the found() function returns a value of false, increment misses by 1. This concludes the code block for the do while loop.

The found() function should be implemented in the following way. So far, all other functions have not returned any values. The found function will return a bool value. This means we need a bool variable to return. This variable should be initialized to a value of “false”. Use a for loop to move through the word[] array. If a value of the word[] array is equal to the user’s guess, then the value in the same index of the guessed[] array should be set to true as well as the function’s return variable. After exiting the for loop, use an if statement to determine if the return variable was set to true or false and return the appropriate value.

We have yet to discuss the condition of the do-while loop. The loop should run if bool winner(bool guessed[], int letters) function returns a value of false AND the number of misses is less than the MAX\_GUESSES. The winner() function can be implemented in the following way. Use a linear search algorithm to look at each value stored in the guessed[] array. If there are no “false” values in this array, the function should return a value of true.

We now will check to see if the number of misses is equal to MAX\_WORDS. If it is, the program will inform the user that they have lost and what the word was. If it is not, then the program will inform the user that they have won. The program will end when the main function returns a value of 0.

Diagram

Description automatically generated