Project 1

Title:

Wordle

Utilizing the Standard Template Library

Class:

CIS-17C

Due Date:

May 8, 2022

Author:

Colby Brunk

Introduction

For this project I decided to code my own version of the game Wordle. Wordle is already an online game that is designed to test your knowledge on English vocabulary and your ability to recognize patterns. I chose Wordle as m project because it is a game that I regularly play and have always been interested in the concept behind the game. The game rapidly grew in popularity and received many copycat applications on the App Store and Google Play, which received millions of dollars of advertisement revenue. When this project was assigned, I wanted to test myself to see if I too could create a version of this game. The project took me around 40-50 hours in total. The first four to five hours were spent figuring out how I was going to create the concept of the game with my knowledge of the STL library. About 10 hours were spent doing research on different ways to utilize the STL containers, and the last 20-30 hours were spent developing the game. My project reached a manageable 850+ lines, with 500 lines belonging to: functions, variables, and container data. The last 350 lines belong to the main function.

Approach to Development

When designing the game I had three main focuses, creating a randomly generated list of words, designing a keyboard display that is altered from user input, and an alphabetical list. All of these ideas were easily implemented with the STL containers. I used a List of four, five, and six letter words to be randomly generated. I used a Map for the alphabetical list. To do this I set the key to be the letter in the alphabet and the value to be a true or false value representing whether or not the letter is in the word. The keyboard display used these true or false values to display whether the letter was in the word, then moved the values into a Stack and Queue in the order of a real keyboard. The mainframe of the game relies around these containers, using functions and other containers to manipulate and store data from them.

Game Rules

Wordle is an online game where you have 6 attempts to guess a randomly generated word. The objective of the game is to guess the word in as little attempts as possible. When you guess a word, if any of the letters are incorrect it will be removed from the list of available of letters to guess. When you guess a letter correctly, it will be added to the list of correct letters, and when you guess the correct location of a letter in the word, it will appear in the spot the letter is located.

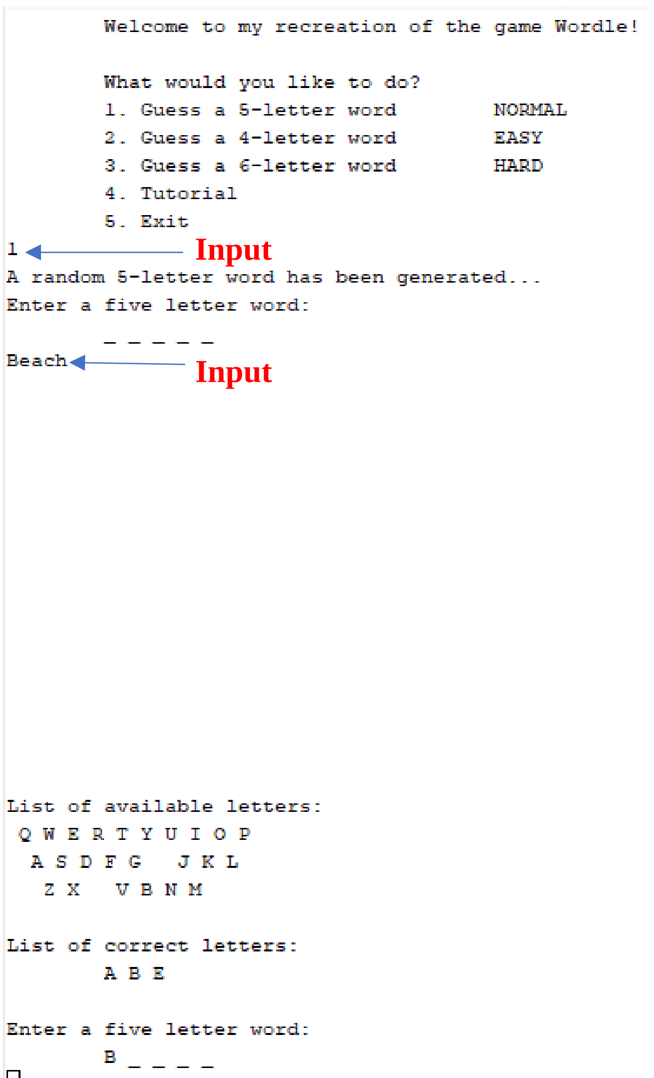
Example:

If the letter you are guessing is BEACH. If you guessed the word BENCH, the letter N would be removed from the list of correct letters, and the letters BECH would all be added to the list of correct letters. The word that you are attempting to guess will show BE_CH. The underscore represents a letter that has not been guessed correctly.

Description of Code

The first 500 lines of my code are organized into first functions, the STL containers, and lastly variables. The next 350 lines is the main function, mainly focused on character output explaining your progress in the game. I chose not to use classes in my program because it was not necessarily useful in my case. Most of my data needed to be accessed within program and it would have caused more of a hassle to implement it rather than leave it out. Each function is labeled as exactly what it does, and each container is labeled as the data it contains. For example, the function to randomly generate a word is called randomWord, the function to delete all data from stack is called clearStack, and the function to display the keyboard on the users screen is displayKeyboard. Some of my lists names are Words, which holds a list of five letter words, theGuess, which holds the data of the users guessed word, and alphabet which contains the letters of the alphabet.

Sample Input/Output



```
        Welcome to my recreation of the game Wordle!

        What would you like to do?
        1. Guess a 5-letter word        NORMAL
        2. Guess a 4-letter word        EASY
        3. Guess a 6-letter word        HARD
        4. Tutorial
        5. Exit
1
A random 5-letter word has been generated...
Enter a five letter word:
        _ _ _ _ _
Beach













List of available letters:
 Q W E R T Y U I O P
  A S D F G   J K L
   Z X   V B N M

List of correct letters:
        A B E

Enter a five letter word:
        B _ _ _ _
```

In this input/output demonstration, you can see how a random word is generated, it is checked with the user's input, and then outputs a keyboard display with incorrect letters missing (C and H). A list of correct letters is displayed below that. Also, since B was the first letter of the randomly generated word, the word being guessed shows up as

"B _ _ _ _ "

Checkoff Sheet

1. Container classes (Where in code did you put each of these Concepts and how were they used?

    1. Sequences (At least 1)

        1. ~~list~~ (Implemented with a list of words to be randomly generated)

        2. slist

        3. bit_vector

    2. Associative Containers (At least 2)

        1. ~~set~~ (Declared with no values, values are initialized during gameplay when an incorrect letter is chosen. The set is then displayed to the user.)

        2. ~~map~~ (Initialized with the key being the letter of the alphabet, and the value representing a true or false value. Used to display a keyboard, and check for already used letters.)

        3. hash

    3. Container adaptors (At least 2)

        1. ~~stack~~ (Stack and queue are both declared with no values, and are used to output the keyboard display.)

        2. ~~queue~~

        3. priority_queue

2. Iterators

    1. Concepts (Describe the iterators utilized for each Container)

        1. Trivial Iterator

        2. ~~Input Iterator~~ (I found it useful to use input iterators to output the letters from a map and set. I implemented this by setting the iterator equal to the designated value, and returning the value the iterator points to.)

3. Output Iterator (I used output iterators to change the value of maps. I needed to change the value from true to false and from false to true. The iterator simply checked if the key was equal to a string, and then would change the keys value.)

4. ~~Forward Iterator~~ (I used forward iterators to iterate through my containers, such as my maps, multimaps, and sets. I used these just to compare each value in a container, and to output the value stored in each iteration. Input was added to the values of maps.)

5. ~~Bidirectional Iterator~~ (Bidirectional iterators were used in my maps, to iterate through the map from the front and from the back.

6. Random Access Iterator

3. Algorithms (Choose at least 1 from each category)

1. Non-mutating algorithms

1. for_each

2. ~~find~~ (I used the find.() algorithm in my map and multimap containers to find the key and display the value related to the key.)

3. count

4. equal

5. search

2. Mutating algorithms

1. copy

2. ~~Swap~~ (I used the swap.() function for stacks and queue to swap the values stored inside of them. I did this to reset the values to the original values I wanted stored in them when data needs to be cleared.)

3. Transform

4. Replace

5. fill

    6. Remove

    7. Random_Shuffle

3. Organization

    1. ~~Sort~~ (I used the sort.() algorithm to organize the words in my list in alphabetical order, since a list is unordered.)

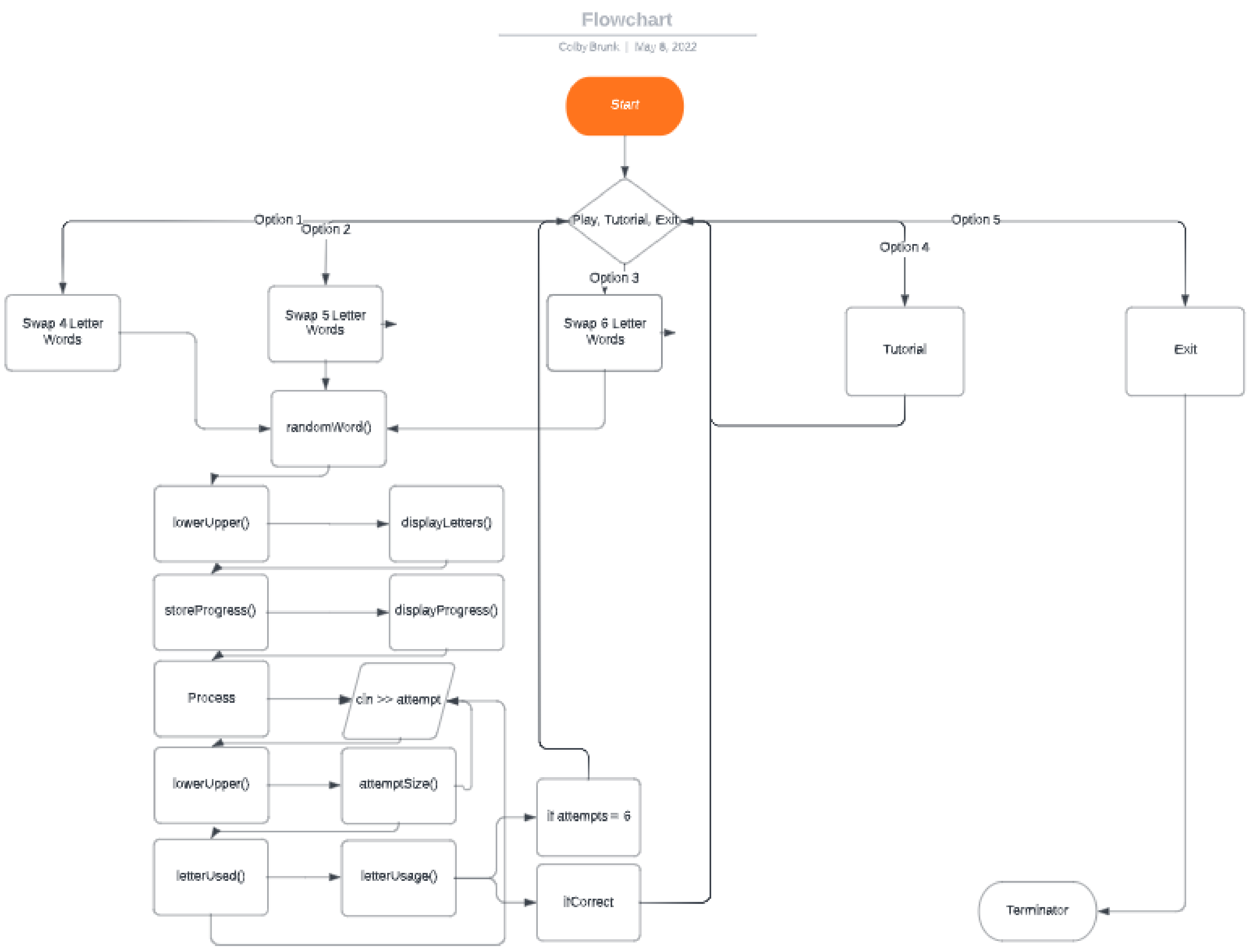
    2. Binary search

    3. merge

    4. inplace_merge

    5. Minimum and maximum

Flowchart


Flowchart
Colby Brunk | May 8, 2022
Start
Play, Tutorial, Exit
Option 1
Option 2
Option 3
Option 4
Option 5
Swap 4 Letter Words
Swap 5 Letter Words
Swap 6 Letter Words
Tutorial
Exit
randomWord()
lowerUpper()
displayLetters()
storeProgress()
displayProgress()
Process
cin >> attempt
lowerUpper()
attemptSize()
letterUsed()
letterUsage()
if attempts = 6
ifCorrect
Terminator

Pseudo-Code

```
Initialize

While selection doesn't equal 5
    Input selection
        Case 1:
            Swap letters in alphabet map
            Swap letters in letters map
            While attempts is less than 6
                Generate random word
                Change word being guessed to capital characters
                Display letters being used
                Store values from last guess
                Display values from last guess
                User inputs a guess
                Change users input to capital letters
                While input size is not equal to 5
                    User inputs a guess
                If input contains false letter
                    User inputs a guess
                Changing non-used letters to false
                Order the keyboard display
                If guess is correct, go back to Input Selection
            If attempts = go back to Input Selection
        Case 2:
            Swap letters in alphabet map
            Swap letters in letters map
            While attempts is less than 6
                Generate random word
                Change word being guessed to capital characters
                Display letters being used
                Store values from last guess
                Display values from last guess
                User inputs a guess
                Change users input to capital letters
                While input size is not equal to 5
                    User inputs a guess
                If input contains false letter
                    User inputs a guess
                Changing non-used letters to false
                Order the keyboard display
                If guess is correct, go back to Input Selection
            If attempts = go back to Input Selection
        Case 3:
            Swap letters in alphabet map
            Swap letters in letters map
            While attempts is less than 6
                Generate random word
                Change word being guessed to capital characters
                Display letters being used
                Store values from last guess
                Display values from last guess
                User inputs a guess
                Change users input to capital letters
                While input size is not equal to 5
                    User inputs a guess
                If input contains false letter
                    User inputs a guess
                Changing non-used letters to false
```

```
            Order the keyboard display
            If guess is correct, go back to Input Selection
        If attempts = go back to Input Selection
    Case 4:
        Display Tutorial
        Return to Input Selection
    Case 5:
        Exit Game
```