SECRET Game Report

1. Introduction

This report provides a detailed explanation of the **SECRET** game developed as part of the CSE 241 Programming Assignment 1. The report covers the methodology used, the operational logic of the code, test results, and the obtained outputs. It also includes placeholders for screenshots where you can insert images of the code and program output.

2. Objective of the Assignment

The main goal of this assignment is to create an interactive game where the player attempts to guess a secret word. The key features are:

- **Secret Word**: It is composed of unique characters read from a file named "alphabet.txt".
- **Two Modes**:
  - `r N`: A random secret word of length N is generated with distinct characters.
  - `u word`: The provided word is used as the secret word.
- **Guessing Process**: For each attempt, the player inputs a word and the program provides feedback.
- **Hints Provided**:
  - **Cexact**: The count of characters that exactly match the secret word in the correct position.
  - **Cmisplaced**: The count of characters present in the secret word but in a different position.
- **Win/Lose Condition**:
  - If the secret word is guessed correctly, the program outputs "FOUND \<guessCount\>".
  - If the player fails to guess the word within 100 attempts, the output is "FAILED".

3. Methodology

3.1. Input and Parameter Validation

- **Command-Line Arguments**: The program accepts two arguments on execution. The first argument specifies the mode (`r` or `u`), and the second argument provides either the number (for random mode) or the specific secret word.
- **Alphabet File**: The "alphabet.txt" file is read, and its characters are filtered to eliminate spaces, commas, newline characters, and duplicate characters.

3.2. Secret Word Generation

- **Random Word Generation (-r)**: A secret word comprising a specified number of unique characters is randomly generated from the alphabet.
- **User-Defined Word (-u)**: The provided word is validated to ensure it only contains characters from the alphabet and that there are no duplicate characters.

3.3. Game Loop and Hint Calculation

- **Guess Attempts**: Each guess entered by the user is validated. If it is invalid, an "INPUT ERROR" message is displayed.
- **Hint Calculation**:
  - *Cexact*: Increments for every character in the guess that matches the secret word at the same position.
  - *Cmisplaced*: Counts characters that exist in the secret word but are placed in a different position than in the guess.
- **Result**: When the correct word is guessed, the game prints "FOUND \<attemptCount\>" and terminates. If 100 guesses are exceeded, the game outputs "FAILED".

4. Code Details

The code consists of the following core components:

- `readAlphabet()`: Reads characters from "alphabet.txt" and stores them in an array after filtering out unwanted characters.

- `gecerliKelimeMi()`: Validates the guessed word by checking its length, ensuring it contains only allowed characters, and confirming there are no duplicate characters.
- `rastgeleKelimeUret()`: Generates a random secret word using non-repeating characters.
- `ipucuHesapla()`: Calculates the Cexact and Cmisplaced hints for each guess.

5. Test Scenarios and Outputs

5.1. Running with Valid Command-Line Arguments (Random Mode)

- **Scenario**: `SECRET -r 6`
  - **Expected Output**: A random secret word of 6 characters is generated, and for each guess, the Cexact and Cmisplaced values are properly computed and displayed.

5.2. Running in User-Defined Word Mode

- **Scenario**: `SECRET -u abc`
  - **Expected Output**: The word "abc" is used as the secret word after validating its correctness, with subsequent guesses receiving accurate hints.

5.3. Error Handling

- **Scenario**: When invalid parameters or inputs are provided, the program outputs "INPUT ERROR".

6. Additional Notes

- **Code Readability**: The code is well-commented in detail, with variable names and comments in English making the logic easy to follow.
- **Adherence to Requirements**: C-style arrays are used, and STL containers such as vector have been avoided as required. The program complies with the assignment specifications.
- **Compilation**: The code can be compiled with:
  It has been developed and tested in a Linux environment using the GCC

compiler.

```
g++ -std=c++11 <filename>.cpp -o SECRET
```

- **Testing Environment**: The program was tested under Ubuntu, ensuring it behaves as expected under typical usage scenarios.