Course: CSE115, Section 4 · Group: 3
MD Labib Daiyan Digonto (2521761642), Rahath Imran (2522916042)
Abdullah Al Mamun (2522141), MD Shahed Ahmed (2523286042)

Hangman Game – A C-Based Console Application

***Abstract*—This project is a console-based version of the hangman game using C code. Within a fixed number of chances, the player must guess the hidden word. The input mechanism for accepting a legitimate secret word has been put into place. This report will discuss the development processes and key features finished so far.**

***Index Terms*—C programming, string validation, hangman, input handling, game logic.**

I. Introduction

The classic game of Hangman, while simple in concept, serves as an excellent tool for beginner programmers. This game compels the learner to master the concepts of state management, securing user input/output (I/O), and string manipulation, all within the constraints of a command-line interface (CLI). This program aims to develop a completely functional hangman game, while at the same time ensuring it is safe and secured. This report documents the successful completion of the project's initial phase, that is, laying down the foundations of accepting and validating a secret word from the first player.

II. System Design

During the inception phase, a clear design was established to guide the development process. Our intended design focused on building a working application that made use of appropriate data types and viable logic.

A. Development Environment

The C programming language was the foundation for the entirety of this project. GNU Compiler Collection (GCC) was used to compile our source code. Being a team project, collaborative editing and progress tracking was achieved by hosting the source code in a GitHub repository.

B. Core Data Structures

To begin with, the significant data structure that we used was an one-dimensional character array declared as `char word[MAX]`. The constant `MAX` is set to be at a size of 51, which was chosen to house a generous 50-character word plus the null terminator (`\0`). Such an allocation strategy bypasses the complexities that involve memory management, whilst providing the user with a fixed buffer for their input.

C. Program Logic Flow

The program uses a linear, sequential logic centered around a validation loop. This design ensures that no invalid data can proceed to the main gameplay logic, thereby preventing potential crashes or unexpected behavior. The logical flow of this input and validation process is illustrated in Fig. 1.

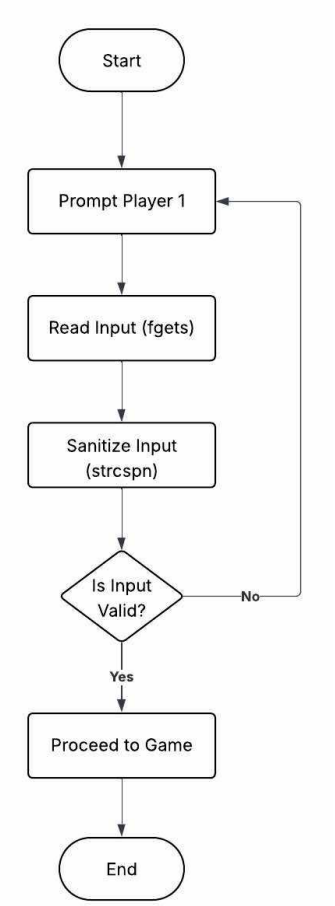

Fig. 1.

III. Implementation of Key Features

A. Secure Input Handling and Buffer Sanitization

Early in the project a crucial decision was made to secure an input. While `scanf()` is the go-to option, it is infamous for being prone to buffer overflow attacks. TO resolve this problem, we used `fgets()`. It prevents users inputting data exceeding the memory b by restricting input to a specified number of characters. However, one common problem with `fgets()` is that it does not eliminate the trailing newline character (`\n`), which can be very problematic for the code. This issue was prevented using strcsp(), which can easily locate the index in which the newline character was stored, that was subsequently replaced by the null terminator (`\0`).

B. Multi-Tiered Input Validation

To guarantee the integrity of the game state, a multi-layered filtration system was developed. The first step is simply checking whether an empty string has been passed, this was done using `strlen()`. An empty string is, if inputted, is rejected, and the user is re-prompted. The next stage of the verification checks for the presence of special characters or numbers in the string. A for loop equipped with the `isaplha()` function from the `<ctype.h>` library iterates on each character. If anything other than an alphabet is detected, an error message is diplayed and the user is re-prompted for a word.

C. Command-Line User Experience (UX)

To improve the user experience necessary steps have been taken. After successfully inputting a secret word, a series of newline characters are printed to clear the screen of previous clutter. The program then dynamically displays a single underscore (_) for each letter in the secret word.

IV. Challenges and Solutions

One of the biggest hurdles of the initial phase was finding an input system that was secured and provided clean data. The decision to use (`fgets()`) brought forth another challenge of sanitising the data, due to the extra trailing newine character.As mentioned previously with the help of AI tools, (`strcspn()`) was adopted, which solved the problem in a single line.

AI Help Used:

- Assistance taken for correct usage of `fgets()` and `strcspn()` for safe input handling.

V. Next Steps

The next steps include the most important part of the game, that is player 2's guessing logic. The guessed letters must be stored to prevent duplicate entries. A logic to keep track of the number of chances will be added, alongside win/loss messages.

VI. Conclusion

Steady progress has been made so far, with input handling completed. Upcoming iterations will implement game logic and player interactions to complete the game mechanics.

References

[1] C Programming Documentation, GNU Project.

[2] AI Assistance from ChatGPT for `fgets()` and `scanf()` usage and validation logic.