HANGMAN GAME

A C programming-based hangman game

Documentation Booklet

Release: v1.0

File Name: Hangman.c

Submitted On: 08 December 2025

INDEX

INTRODUCTION

The Hangman Game is a classic word-guessing puzzle in which the player attempts to discover a hidden word by guessing its letters one at a time. This project aims to implement a simple, terminal-based Hangman game using the C programming language. It demonstrates the use of basic programming concepts such as loops, conditional statements, arrays, strings, user input handling, and simple functions.

The project helps beginners understand how a real game can be built from fundamental principles, without relying on complex libraries or advanced programming techniques. It also strengthens logical thinking, algorithmic design, and the use of modular programming.

2.1 Overview of the Project

This project is designed to create a straightforward and interactive Hangman game that runs in a terminal environment. The game randomly selects a word from a predefined list. The player must guess the word by suggesting letters within a limited number of chances.

The program reveals correctly guessed letters in their respective positions and displays the number of attempts remaining after each guess. If all letters are guessed correctly, the player wins; otherwise, if the maximum number of wrong attempts is reached, the player loses.

The goal of this project is not just entertainment but learning—specifically how arrays, character operations, conditions, looping structures, and functions work together in C.

2.2 Purpose of the Hangman Game

The main purpose of this project is to apply basic C programming concepts to create a fully functioning game. It helps students understand:

- How to use arrays and strings
- How to process user input
- How to control program flow using loops and conditions
- How to divide a program into multiple small functions
- How to generate random output using the rand() function

The game also encourages problem-solving and critical thinking as players must predict letters and analyse patterns.

2.3 Scope of the Project

The scope of this Hangman Game project is intentionally kept simple to match the knowledge level of first-semester students. The project includes:

- A fixed list of words stored in an array
- Random selection of a word at runtime
- Display of masked letters and wrong guesses
- A limited number of attempts
- Basic input validation
- Modular programming with small, readable functions

FEASIBILITY STUDY

A feasibility study is carried out to determine whether the development of the Hangman Game in C is realistic, achievable, and practical within the constraints of time, knowledge, and available resources. This section evaluates the technical, operational, and economic aspects of the project to ensure that it can be developed successfully using basic programming tools and skills available to first-semester students.

3.1 Technical (Physical) Feasibility

Technical feasibility focuses on whether the project can be successfully implemented using the hardware and software tools available to the developer.

For this project, the technical requirements are very minimal. The Hangman Game is implemented entirely using the C programming language, and it can run on any basic system that supports a C compiler such as GCC, MinGW, or Turbo C. Since the program is text-based, it does not require advanced hardware like a graphics card or high processing power.

All programming concepts used—arrays, strings, loops, conditional statements, and simple functions—are part of the first-semester C syllabus. No external libraries or complex frameworks are required. Hence, the project is completely **technically feasible** with the available tools and student knowledge.

3.2 Operational (Behavioural) Feasibility

Operational feasibility determines whether the system will function correctly and be accepted by its intended users.

The Hangman Game is simple to use and requires only keyboard input from the player. The rules of the game are easy to understand: guess letters, avoid mistakes, and try to uncover the hidden word. The game displays clear instructions, masked words, wrong guesses, and remaining attempts, making it fully user-friendly.

Since the program runs in a terminal, there is no need for training or complex setup. Anyone familiar with using a computer can operate it easily, making the project **behaviourally feasible**.

3.3 Economic Feasibility

Economic feasibility examines whether the project can be completed within a reasonable cost or with no cost at all.

The Hangman Game does not require any financial investment. All tools used—C compiler, text editor, and terminal—are free or already available on most systems. There is no need for additional hardware, software licenses, or external resources.

Because the project is simple and developed using free tools, it is **economically feasible** for any first-semester student.

BACKGROUND STUDY

The Hangman Game has been a popular word-guessing puzzle for decades. Traditionally played with pen and paper, it challenges players to uncover a hidden word by guessing its letters within a limited number of attempts. Over time, Hangman has become a common beginner-level programming project used to teach logic, loops, condition handling, and string manipulation.

This project aims to recreate the classic Hangman Game using the C programming language. Before building the system, it is important to study existing versions of the game and understand the gaps or limitations that justify the development of a new, simplified version suitable for first-semester students.

4.1 Existing Systems

Several digital versions of Hangman exist today:

- Mobile apps and online Hangman games
  These come with large dictionaries, graphics, animations, sound effects, and multiple difficulty modes.
- Educational programming examples
  Many tutorials show basic Hangman logic written in languages like Python, Java, or JavaScript. Some provide advanced features such as ASCII art, GUI interfaces, or file-based dictionaries.
- Console-based Hangman programs
  Some versions use many advanced C concepts such as dynamic memory allocation, pointers to functions, linked lists, or external file handling.

Although these systems work well, they are often complex and use features beyond the scope of a first-semester C programming course.

This project focuses on building a simplified version that uses only core concepts such as arrays, functions, loops, and simple string handling.

4.2 Gaps & Limitations

While existing Hangman implementations are functional, they contain certain limitations when considered for beginner-level academic use:

1. Too complicated for beginners
   Many versions rely on advanced concepts not yet taught in the first semester, making them hard to understand for new programmers.
2. Dependence on external files or graphics
   Some implementations require external word lists or graphical resources, which increases complexity and setup time.
3. Lack of modularity
   Many beginner tutorials write the entire game inside the main() function, making the code harder to read and maintain.
4. Not tailored for academic learning
   Most online versions focus on gameplay features rather than explaining the programming concepts behind them.
5. Overuse of platform-specific functions
   Some versions use system-dependent commands (like clearing the screen) which may not work consistently across different operating systems.

The Hangman Game developed in this project aims to address these issues by providing:

- A clean, understandable program structure
- Modular functions
- Basic terminal-based gameplay
- No external files
- No OS-dependent commands
- No advanced C concepts beyond the first-semester syllabus

OBJECTIVES

The main objective of this project is to develop a simple and interactive Hangman Game using the C programming language. The game should function smoothly in a terminal environment and help beginners understand core programming concepts through practical implementation. This section outlines the primary and secondary objectives that guide the development of the project.

5.1 Primary Objective

The primary objective of this project is:

✓ **To develop a functional and user-friendly Hangman Game using basic C programming concepts.**

This involves:

- Using arrays and strings to store words and track guesses
- Implementing loops and conditionals to control the game flow
- Using functions to organize the logic into manageable modules
- Handling user input and validating entered characters
- Allowing the player to guess letters and attempt to solve the hidden word
- Displaying the number of remaining attempts and the progress of the game

The primary goal is that the game should be easy to understand, simple to run, and fully playable without any advanced programming features.

5.2 Secondary Objectives (Performance & Control)

The secondary objectives support the main goal by improving playability, structure, and learning value.

✓ **To ensure clear program structure using modular functions**

Splitting the code into smaller functions (like word selection, checking guesses, updating the mask) helps beginners understand how a program is built in parts.

✓ **To provide smooth gameplay through basic error handling**

The game should guide the user with friendly messages, such as when they enter:

- numbers

- symbols
- repeated letters

✓ **To use simple, readable, and beginner-level C code**

The project avoids complexities such as:

- File handling
- Dynamic memory allocation
- Pointers beyond basic usage
- OS-dependent commands
- Complex data structures

✓ **To encourage logical thinking and problem-solving**

The game helps players think strategically by analyzing patterns and guessing letters intelligently.

✓ **To make the game easily expandable for future improvements**

The code is kept clean so additional features (difficulty levels, ASCII art, bigger word list) can be added later.

SYSTEM DESIGN & CODING

The Hangman Game is a simple terminal-based application created using the C programming language. The design follows a modular approach where the entire logic is divided into smaller functions. This makes the program easier to understand, debug, and maintain—especially for beginners.
This section explains the system architecture, data structures used, and the functions that form the core of the game's logic.

6.1 System Architecture

The system is designed using a **modular functional architecture**. Instead of writing the entire program inside main(), the project is divided into several functions, each handling a specific task such as selecting a word, updating the mask, checking guesses, and printing game status.

Main Components

1. **Word Selection Module**
   Responsible for choosing a random word from a predefined list.
2. **Mask Generation Module**
   Converts the chosen word into an underscore mask (e.g., "____").
3. **Input Processing Module**
   Reads and validates user input (letters).
4. **Guess Checking Module**
   Compares user guesses to the secret word and updates the mask.
5. **Game Loop**
   Controls the gameplay until the player wins or runs out of attempts.

This modular structure makes the program easier to read, enhances clarity, and follows good programming practices taught in the first semester.

6.2 Data Structures Used

The Hangman game uses only simple data structures suitable for first-semester students:

1. Character Arrays (Strings)

Used to store:

- The secret word
- The masked word (e.g., "____")
- Wrong letters guessed by the player

Example:

```
char mask[50];
char wrong[27];
```

2. Integer Variables

Used for:

- Loop counters
- Tracking number of wrong guesses
- Storing function return values

3. Constant Arrays for Word List

A static array of strings is used for selecting a random word.

Example:

```
const char* WORDS[] = {"code", "pointer", "variable"};
```

These simple structures avoid complexity while still demonstrating useful real-world programming patterns.

6.3 Function Descriptions

This section explains the core functions used in the Hangman Game, describing what each function does and why it is needed.

6.3.1 Word Selection Module

Function: choose_word()
Purpose: Selects a random word from the predefined list.
Concepts Used: Arrays, random numbers.

```
const char* choose_word() {
   return WORDS[rand() % WORD_COUNT];
}
```

6.3.2 Mask Generation Module

Function: make_mask()
Purpose: Converts the secret word into a mask of underscores.
Example: "code" → "____"

```
void make_mask(const char* word, char mask[]) {
   int len = strlen(word);
   for (int i = 0; i < len; i++)
      mask[i] = '_';
   mask[len] = '\0';
}
```

6.3.3 Input Processing Module

Function: get_guess()
Purpose: Reads a single letter from the user and converts it to lowercase.

```
char get_guess() {
   char c;
   scanf(" %c", &c);
   return tolower(c);
}
```

6.3.4 Guess Checking Module

Function: apply_guess()
Purpose: Checks if the guessed letter appears in the secret word and updates the mask.

```
int apply_guess(const char* word, char mask[], char g) {
   int hits = 0;
   for (int i = 0; word[i]; i++) {
```

```
        if (word[i] == g) {
            mask[i] = word[i];
            hits++;
        }
    }
    return hits;
}
```

This function returns the number of positions updated.
If hits == 0, the letter was wrong.

6.3.5 Game Loop & Control Flow

Role:
The *main game loop* controls the entire gameplay:

- showing masked word
- reading guesses
- updating wrong attempts
- checking win/lose conditions

The loop continues until:

- all letters of the word are guessed (**win**)
- wrong guess limit is reached (**lose**)

PROGRAM OUTPUT / SAMPLE RUN

This section provides a sample execution of the Hangman Game. It demonstrates how the game appears to the player, how input is taken, and how the game responds to correct and incorrect guesses. The output shown below is based on actual terminal interaction.

7.1 Sample Gameplay Output

```
=== Hangman (Simple) ===
The word has 4 letters.

Word: _ _ _ _
Wrong letters:
Chances left: 6
Enter a letter: a
Nope! 'a' is not in the word.

Word: _ _ _ _
Wrong letters: a
Chances left: 5
Enter a letter: o
Nice! 'o' occurs 1 time(s).

Word: _ o _ _
Wrong letters: a
Chances left: 5
Enter a letter: d
Nice! 'd' occurs 1 time(s).

Word: d o _ _
Wrong letters: a
Chances left: 5
Enter a letter: c
Nice! 'c' occurs 1 time(s).

Word: d o c _
Wrong letters: a
Chances left: 5
Enter a letter: e
Nice! 'e' occurs 1 time(s).

Word: d o c e
Wrong letters: a
Chances left: 5

You WIN! The word was: doce
```

Explanation of Sample Run

- The program begins by showing the number of letters in the secret word.
- The user guesses letters one by one.
- Wrong guesses are recorded and reduce the number of remaining chances.
- Correct guesses reveal letters in the masked word.
- The game ends when:
    - The word is guessed completely (**win**)
    - The chances run out (**lose**)

FUTURE ENHANCEMENT

Although the current version of the Hangman Game is simple and fully functional, there are several ways it can be improved in the future. These enhancements can increase the game's usability, visual appeal, and educational value. The following subtopics describe potential upgrades that can be implemented as the developer gains more programming experience.

8.1 Adding Difficulty Levels

The game can be enhanced by introducing multiple difficulty modes such as:

- **Easy:** shorter words and more chances
- **Medium:** slightly longer words
- **Hard:** long or uncommon words with limited attempts

This would make the game more engaging and allow players of different skill levels to enjoy it.

8.2 Implementing ASCII Art Hangman Stages

An improved version of the game can include a visual representation of the Hangman figure.
Each wrong guess can reveal another part of the drawing, such as:

- head
- body
- arms
- legs

This adds visual feedback and makes the game more interactive and enjoyable.

8.3 File-Based Word Dictionary

Instead of using a fixed list of words inside the program, the game can read words from an external text file, such as: words.txt

This allows:

- A larger word list
- Easy updates without modifying code
- Custom word lists for difficulty levels

It also helps students practice file handling—an important programming concept.

8.4 Graphical User Interface (GUI) Version

In the future, the Hangman Game can be upgraded from a terminal-based program to a graphical interface using technologies such as:

- C++ with Qt
- Java Swing
- Python Tkinter
- HTML/CSS/JavaScript (Web-based version)

A GUI would allow features like:

- buttons for selecting letters
- animations
- colorful visuals
- progress bars

CONCLUSION

The Hangman Game project successfully demonstrates how fundamental programming concepts in the C language can be combined to create an interactive and enjoyable application. Through this project, basic topics such as loops, conditional statements, arrays, strings, functions, and user input handling were applied in a practical manner.

The modular structure of the program makes it easy to understand, maintain, and improve. The game runs smoothly in a terminal environment and fulfills its purpose of providing a beginner-friendly learning experience. It also encourages logical thinking and problem-solving skills, both of which are essential in computer science.

Overall, the project is simple yet effective and serves as a strong foundation for future improvements such as ASCII visuals, difficulty levels, file-based word storage, or even a graphical interface. This project highlights the importance of applying theoretical knowledge to real-world applications and helps build confidence for more advanced programming tasks in the future.

SUBMITTION

Submitted by:

Name: Krishna

Course: B.Tech CSE Core

Roll No: GF202565408

Submitted To:

Mr. Anurag Rana Sir

Submitted On:

08 December 2025