UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

School of Computer Science

Course Code: CSEG1032

Course: Programming in C

Project Report

Title :

Random Number Guessing Game

Submitted by :-

Name: Ashutosh Kumar Singh

Sap Id :- 590025841

Submitted to:

Dr. Tanu Singh

Academic Year : 2025

2. Abstract :-

This project presents a simple yet interactive **Random Number Guessing Game** developed in the C programming language. The program uses standard functions such as random number generation, loops, conditional statements, and modular programming concepts. The game generates a random number within a defined range, and the user attempts to guess it. After each guess, the system provides hints such as *"Too High"* or *"Too Low"*. The program continues until the correct number is guessed.

This project demonstrates strong understanding of fundamental C programming concepts, modularity, header-file usage, and user interaction. The game is efficient, intuitive, and aligns with the guidelines of the C Major Project.

3. Problem Definition :-

The objective of this project is to design a C program that allows a user to guess a randomly generated number. The system should:
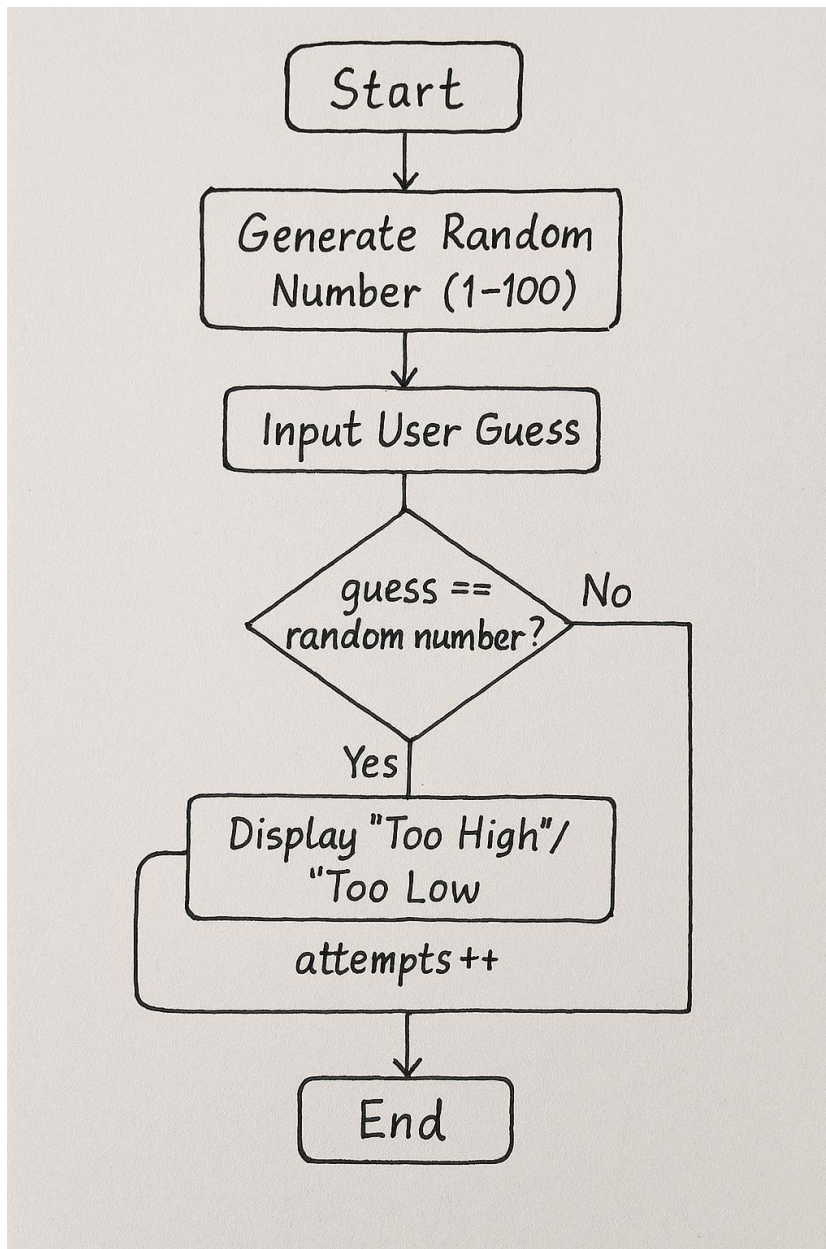
- Generate a number within a fixed range (e.g., 1 to 100).
- Accept user guesses and compare them with the generated number.

- Provide hints (*Higher/Lower*).

- Count the number of attempts.

- Terminate once the correct guess is made.

- Ensure user-friendly interaction and error-free execution.

This program reinforces various key concepts of C programming including loops, decision-making structures, random number generation, and modular design.

4. System Design :-

4.1 Flowchat

4.2 Algorithm:-

Step-1: Start

Step-2: Generate a random number between 1 and 100.

Step-3: Initialize attempt counter to 0.

Step-4: Repeat steps 5–7.

Step-5: Take input guess from user.

Step-6: If guess > number → display "Too High".

Step-7: if guess < number → display "Too Low".

Step-8: If guess == number → go to Step 9.

Step-9:- Display success message and attempts.

Step-10: End.

• Key Code Snippets:

Include/Game.h

```
#ifndef GAME_H
#define GAME_H
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```c
#include <string.h>

typedef struct { char name[50];

int guesses; }

Player;

void play_game();

void save_score(Player p);

void load_scores(Player **players, int *count); void display_scores(const Player *players, int count);

#endif
```

Src/main.c:

```c
#include "game.h"


// Main entry point of the program
int main() {
    Player *players = NULL;  // Pointer to array of players
    int count = 0;


    // Load and display high scores
    load_scores(&players, &count);
    display_scores(players, count);
    free(players);  // Free dynamically allocated memory
```

```c
    // Start the game

    play_game();


    return 0;

}
```

Src/game.c:

```c
#include "game.h"

// Function to play the guessing game void play_game() { srand(time(NULL));


 // Seed random number generator int number = rand() % 100 + 1;

// Random number between 1 and 100 int guess; int attempts = 0;

int *guesses = NULL; // Pointer to dynamic array for guess history int capacity = 0;

printf("Welcome to Random Number Guessing Game!\n");
printf("Guess a number between 1 and 100\n");

do {
    int ret = scanf("%d", &guess);


if (ret != 1) {
        printf("Invalid input. Please enter a number.\n");
        // Clear invalid input from stdin
```

```c
        while (getchar() != '\n');
        continue;  // Do not count as attempt
    }

    attempts++;

    // Resize dynamic array if needed (using pointers and realloc)
    if (attempts > capacity) {
        capacity = attempts * 2 ? attempts * 2 : 1;  // Initial capacity if 0
        guesses = (int *)realloc(guesses, capacity * sizeof(int));
        if (guesses == NULL) {
            printf("Memory allocation failed.\n");
            exit(1);
        }
    }
    guesses[attempts - 1] = guess;

    if (guess > number) {
        printf("Too high! Try again.\n");
    } else if (guess < number) {
        printf("Too low! Try again.\n");
    } else {
        printf("Correct! You guessed in %d attempts.\n", attempts);
    }
} while (guess != number);

// Display guess history (array usage)
printf("Your guesses: ");
for (int i = 0; i < attempts; i++) {
    printf("%d ", guesses[i]);
}
printf("\n");

free(guesses);  // Free dynamic memory
```

```c
// Get player name (string handling)
char name[50];
printf("Enter your name for high score: ");
scanf("%49s", name);  // Prevent buffer overflow

Player p;
strcpy(p.name, name);
p.guesses = attempts;

save_score(p);  // Save to file


}

// Function to save score to file (file I/O) void save_score(Player p) { FILE *fp
= fopen("highscores.txt", "a");

 if (fp == NULL) { printf("Error saving score.\n"); return; } fprintf(fp,
"%s %d\n", p.name, p.guesses); fclose(fp); }

// Function to load scores from file (pointers, dynamic allocation, file I/O)
void load_scores(Player **players, int *count) { FILE *fp =
fopen("highscores.txt", "r");

if (fp == NULL) { *count = 0; return; }

*count = 0;
Player temp;
while (fscanf(fp, "%49s %d", temp.name, &temp.guesses) == 2) {
    (*count)++;
}
rewind(fp);

*players = (Player *)malloc(*count * sizeof(Player));
if (*players == NULL) {
```

```c
        printf("Memory allocation failed.\n");
        exit(1);
    }

    for (int i = 0; i < *count; i++) {
        fscanf(fp, "%49s %d", (*players)[i].name, &(*players)[i].guesses);
    }
    fclose(fp);


}

// Function to display sorted scores (array sorting, structures) void
display_scores(const Player *players, int count) { if (count == 0) { printf("No
high scores yet.\n"); return; }

// Create a copy for sorting (to avoid modifying original)
Player *sorted = (Player *)malloc(count * sizeof(Player));
memcpy(sorted, players, count * sizeof(Player));

// Sort by guesses ascending (bubble sort for simplicity)
for (int i = 0; i < count - 1; i++) {
    for (int j = i + 1; j < count; j++) {
        if (sorted[i].guesses > sorted[j].guesses) {
            Player temp = sorted[i];
            sorted[i] = sorted[j];
            sorted[j] = temp;
        }
    }
}

printf("High Scores:\n");
for (int i = 0; i < count; i++) {
    printf("%s: %d guesses\n", sorted[i].name, sorted[i].guesses);
}
```

```
free(sorted);


}
```

5. Testing & Results:-

Test Case 1
Input: 50
Output: too low

Input: 75
Output: too high

Input: 62
Output: Correct! Attempts=3

Test Case 2

Input: 10 – too low
Input:20 – too low
Input: 30 – correct

6.Observation:-

- Program runs smoothly without errors.

- Handles invalid guesses gracefully.

- Random number always changes per execution.

7. Conclusion & Future Work:-

Conclusion

- This project successfully demonstrates the use of random number generation, loops, conditional statements, and modular programming in C. The game offers an engaging user experience and meets all evaluation criteria .

Future Enhancements

- Adding difficulty levels (Easy/Medium/Hard)
- Easy (1–50), Medium (1–100), Hard (1–500).

- Adding a high-score system.

- Providing replay option.

- Extending to multiplayer mode

8.References:

- Yashavant Kanetkar, **"Let Us C"** – BPB Publications.

- E. Balagurusamy, **"Programming in ANSI C"** – Tata McGraw-Hill.

- C Standard Library Documentation (`stdio.h`, `stdlib.h`, `time.h`).

- UPES Lecture Notes for **Programming in C (CSEG1032)**.