ECE 321 - Introduction to Software Engineering Homework

Part 1 - Impact of C in modern operating system

Research the history of C programming language along with all the other programming languages it has affected (e.g., C++, python, Rust). Prepare a small PowerPoint presentation listing and explaining the achieved breakthroughs and the important novelties that C introduced. Make sure to provide (cite) all references for the information that you find.

Part 2 - Analysis of user management software

The following code is used by a company for account management. Study the code and answer the following questions:

```c
#include <stdio.h>
#include <string.h>

#define MAX_USERS 10
#define MAX_NAME_LEN 20
#define MAX_PASS_LEN 20

typedef struct {
    char username[MAX_NAME_LEN];
    char password[MAX_PASS_LEN];
} User;

User users[MAX_USERS];
int user_count = 0;

void addUser() {
    char username[100]; // Unsafe: buffer larger than the struct allows
    char password[100]; // Unsafe: buffer larger than the struct allows

    printf("Enter username: ");
    gets(username); // Unsafe: allows buffer overflow

    printf("Enter password: ");
    gets(password); // Unsafe: allows buffer overflow

    // Unsafe: no bounds checking when copying to struct
    strcpy(users[user_count].username, username);
    strcpy(users[user_count].password, password);

    user_count++;
    printf("User added successfully!\n");
}

void loginUser() {
    char username[100];
    char password[100];

    printf("Enter username: ");
    gets(username); // Unsafe: allows buffer overflow

    printf("Enter password: ");
    gets(password); // Unsafe: allows buffer overflow

    // Unsafe: linear search without bounds checking
    for (int i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0 &&
            strcmp(users[i].password, password) == 0) {
            printf("Login successful!\n");
            return;
        }
    }
    printf("Invalid username or password.\n");
}

void menu() {
    int choice;

    while (1) {
        printf("\n1. Add User\n2. Login\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        getchar(); // Unsafe: ignores possible input issues

        switch (choice) {
            case 1:
                addUser();
                break;
            case 2:
                loginUser();
                break;
            case 3:
                printf("Exiting...\n");
                return;
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}

int main() {
    printf("Simple Password Management System (Unsafe Version)\n");
    menu();
    return 0;
}
```

- Ethical considerations are crucial in software engineering, especially when designing systems that handle sensitive data. Answer the following questions based on your analysis of the provided code:
  - How does the use of unsafe coding practices (e.g., buffer overflows, storing plaintext passwords) violate the trust users place in software systems?
  - If this code were used in a real-world application, how might it affect the users' privacy and security?
  - If a developer knowingly writes insecure code, what ethical principles might they be violating?
  - Refer to a code of ethics for software engineers (e.g., IEEE Code of Ethics). Identify at least two principles from the code and explain how the provided program violates them.
- Software systems can have significant societal and technical impacts. Analyze the potential consequences of deploying the provided code in the real world:
  - Describe how specific vulnerabilities in the code (e.g., buffer overflows, lack of password encryption) could be exploited by malicious users.
  - What types of attacks could be performed, and what might be the consequences for the affected users and organizations?

    - If this code were used in a commercial product, what legal consequences might the developers face due to its vulnerabilities?

Part 3 - Dungeon crawler game (team)

For this part, students will work in a team of 2 or 3 students.

Game overview

The dungeon crawler game simulates a player moving through a series of rooms in a dungeon. Each room can contain a monster, treasure, or be empty. The player's objective is to navigate through the dungeon and survive encounters with monsters. The game concludes when the player either exits the dungeon or succumbs to their injuries.

Code description

The provided C code (`dungeon.c`) sets up the basic framework for the game. The code does the following:

- Dungeon creation: When the game starts, it dynamically generates a dungeon with a specified number of rooms. Each room has a randomized characteristic—it could be an empty room, contain a monster, or house a treasure.
- Player structure: The game creates a player character with predefined attributes such as name, strength, and health. These attributes affect the outcomes of encounters within the game.
- Gameplay mechanics:
    - As the player enters each room, the game describes the contents of the room to the player.
    - If a room contains a monster, a fight ensues. The player's strength is compared to the monster's strength to determine the outcome. Winning increases the player's score, while losing results in a reduction of health.
    - Finding treasure in a room increases the player's score based on the treasure's value.
    - Empty rooms have no effect but serve as a buffer between encounters.
- Room navigation: The dungeon is structured as a linked list, where each node represents a room. The player moves from the start to the end of the list as they progress through the game.
- Game loop: The core game loop handles the player's progress through the dungeon, room descriptions, and encounters until the game ends.

Deliverables

You are asked to enhance the realism and interactivity of the game by introducing progressive difficulty and player choices with consequences. The game's randomness will be tempered with a sense of progression, as both risks and rewards increase the deeper the player ventures into the dungeon.

Objectives:

- Progressive monster strength:
  - Modify the ‘createRoom’ function so that the strength of the monsters encountered in each room increases with the depth of the room in the dungeon. This means a monster in room N should generally be stronger than a monster in room N-1.
  - The progression can be linear or follow another reasonable mathematical model that ensures a steady increase in difficulty.
- Increasing treasure value:
  - Similar to monster strength, the amount of treasure (coins) found in each room should increase with the room’s depth in the dungeon.
  - Implement a formula that increases the value of the treasure as the player progresses. This could be a simple linear increase or a more complex function to reflect the growing risk.
- Coin collection mechanism:
  - Add a ‘coins’ attribute to the ‘Player’ structure to keep track of the player’s wealth.
  - Modify the ‘playerAction’ function to update the player’s ‘coins’ attribute whenever they discover treasure.
- Fleeing from monsters:
  - When a player encounters a monster, provide the option to either fight or flee.
  - Implement a decision-making process where the player inputs their choice.
  - If fleeing, the player must pay a fee in coins equivalent to the room level multiplied by 10. This simulates a penalty for avoiding combat.
  - If the player cannot afford the fee, they are forced to fight the monster.
- Consider a project team tasked with enhancing this game with graphics, run-time dungeon generation, and enemies that adapt to user choices. Describe how teams with different compositions would approach the task. Specifically, address the following:
  - How would a team composed solely of computer engineering students approach the project? Consider their strengths in understanding user interface and graphics.
  - With the addition of network engineers, how does the team’s capability expand?