Algorithmic Complexity and Review

1 Review

Analyze the C code snippet provided, considering aspects like data structures, memory management, and control flow, then answer the subsequent questions to exhibit your understanding of C programming concepts.

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

#define MAX_NAME_LEN 50
#define GREETING "Hello, %s! (Greeting number %d)\n"

typedef struct User {
    char name[MAX_NAME_LEN];
    int times_to_greet;
    struct User *next;
} User;

void greet(const User *u);
typedef void (*GreetFunction)(const User*);

void write_greeting_to_file(const char* name, int number) {
    FILE* file = fopen("greetings.txt", "a");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }
    fprintf(file, GREETING, name, number);
    fclose(file);
}


int main() {
    User* users = NULL;
    GreetFunction my_greet_function = greet;

    User* new_user = (User*) malloc(sizeof(User));
    if(new_user == NULL) {
        perror("Unable to allocate memory for new user");
        return 1;
    }

    printf("Enter your name: ");
    scanf("%49s", new_user->name);
    printf("How many times should I greet you? ");
    scanf("%d", &(new_user->times_to_greet));

    if(new_user->times_to_greet < 1) {
        printf("Invalid number of greetings. Exiting.\n");
        free(new_user);
        return 1;
    }

    new_user->next = users;
```

```
    users = new_user;

    my_greet_function(users);

    switch(new_user->times_to_greet) {
        case 1:
            printf("You got greeted once.\n");
            break;
        case 2:
            printf("Twice! Nice to greet you, %s!\n", new_user->
    name);
            break;
        default:
            printf("%s, you got greeted %d times!\n", new_user->
    name, new_user->times_to_greet);
            break;
    }

    while(users) {
        User* to_free = users;
        users = users->next;
        free(to_free);
    }

    return 0;
}
void greet(const User *u) {
    while(u) {
        for(int i = 0; i < u->times_to_greet; i++) {
            printf(GREETING, u->name, i+1);
            write_greeting_to_file(u->name, i+1);
        }
        u = u->next;
    }
}
```

Listing 1: C code for analysis

Questions:

Section 1: Basic Code Understanding

Q1: Can you identify the purpose of using `#define` for `MAX_NAME_LEN` and `GREETING` in the code?

Q2: Explain the functionality and usage of the `next` pointer within the `User` structure in the code.

Q3: Explain why the code checks if `new_user` is `NULL` after calling `malloc` in the main function.

Section 2: Memory Management and Pointers

Q4: What is the role and significance of pointers in the C language, providing examples from the code?

Q5: What are the potential risks of using pointers and how can they be mitigated?

Q6: Discuss the concept of function pointers and how it is employed in the code.

Q7: How can pointer arithmetic be applied for array traversal, and what considerations should be made about memory boundaries?

Section 3: User Input and File I/O Operations

Q8: Suggest an alternative approach to handle names with spaces during user input and explain why the proposed solution might be a preferable function for reading strings.

Q9: Why might it be necessary to handle invalid `times_to_greet` input and how might you clear invalid input from the input buffer?

Q10: Propose a feature that allows a user to specify the filename where greetings will be saved and provide a relevant code snippet.

Section 4: Data Structures (Stack and Queue) Implementation

Q11: Implement a stack for storing user data, describe push and pop operations, and discuss the impact on the user greeting order.

Q12: Implement a queue for user data storage, elucidate enqueue and dequeue operations, and discuss their impact on the greeting order of users.

Q13: In the context of data structure implementations (both stack and queue), discuss how two users added sequentially would be greeted and justify your answers.