Appointment Scheduling using Queue Data Structure

MINOR PROJECT REPORT

By

RUFAS KANIKANTI (RA2211026010557)
SUDHEER CHIDIPOTHU (RA2211026010558)
KEERTHI PAVAN S (RA2211026010561)

Under the guidance of

Dr.MEENAKSHI

In partial fulfilment for the Course

of

21CSC201J – DATA STRUCTURES AND ALGORITHMS
in CINTEL

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSC201J – DATA STRUCTURES AND ALGORITHMS** entitled in "**Apointment scheduling using queue data structure** " is the bonafide work of **Rufas kanikanti (RA2211026010557) ,Sudheer chidipothu (RA2211026010558)** and **Keerthi pavan s (RA2211026010561)** who carried out the work under my supervision.

SIGNATURE

Dr.Meenakshi
Assistant professor
CINTEL
SRM Institute of Science and Technology
Kattankulathur

ABSTRACT

The code implements an appointment scheduling system using a queue. It defines structures for Appointment and Queue. An appointment includes fields for name, date, and time. The program provides functionalities to create a new appointment, add it to the queue, display all appointments, and exit the system.

Structures:
Appointment: Contains information about a scheduled appointment, such as the name of the individual, the date, the time, and a pointer to the next appointment.

Queue:
Manages the appointments using a queue data structure, maintaining pointers to the front and rear of the queue.

Functions:
createAppointment(): Creates a new appointment by prompting the user to input name, date (in MM/DD/YYYY format), and time. Returns the created appointment.

createQueue(): Initializes the appointment queue, allocating memory for the queue structure and setting the front and rear pointers to NULL.

enqueue(): Adds a new appointment to the queue, adjusting pointers to maintain the queue structure.

displayAppointments(): Prints all the appointments currently in the queue with their details.

Main Function:-
The main function initiates the program flow, presenting a menu to the user with options to add an appointment, display all appointments, or exit the system. It continuously loops, prompting the user for their choice and executing the corresponding functionality (creating/displaying appointments or exiting the program).

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty Dr.Meenakshi **, assistant professor , CINTEL,** for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. annie uthra ,Head of computer intelligence, CINTEL** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

1. INTRODUCTION

Include the necessary header files, <stdio.h> and <stdlib.h>. Define a structure for an appointment that includes fields for name, date, time, and a pointer to the next appointment.
Define a structure for the queue, which includes pointers to the front and rear of the queue.
Implement a function to create a new appointment (createAppointment) and initialize it with user input.
Implement a function to initialize a queue (createQueue) and set the front and rear pointers to NULL.

Main Function:

In the main function, create a queue called appointmentQueue using createQueue.
Declare an integer variable choice to store the user's menu choice.

Menu Loop:

Enter a continuous loop with a while (1) statement to display the menu and handle user choices. Menu

Options:

Add Appointment:

When the user selects option 1, call createAppointment to create a appointment allolocate memory for it.
Enqueue the new appointment into the queue using the enqueue function.
Display a message indicating that the appointment has been added to the queue.

Display Appointments:

When the user selects option 2, call displayAppointments to display all the appointments in the queue.
If the queue is empty, indicate that there are no appointments in the queue.

Exit:

When the user selects option 3, display a message indicating that the program is exiting.
Terminate the program with exit(0).

User Input and Error Handling:

Prompt the user to enter a choice, and use scanf to read the choice into the choice variable.
Implement a switch statement to handle the user's choice and provide feedback for invalid choices.

Display Appointments:

Within the displayAppointments function, check if the queue is empty (front is NULL).
If the queue is empty, display a message indicating that there are no appointments.
If the queue is not empty, traverse the linked list, displaying appointment details, such as name, date, and time, for each appointment.
This algorithm outlines the main structure and functionality of the code, providing a clear sequence of actions for creating appointments, managing the queue, and displaying appointment information based on user choices.

2. PROBLEM STATEMENT

The provided C code offers a simple framework for an appointment scheduling system, yet it presents certain challenges and shortcomings. While it allows users to add and display appointments, it lacks a comprehensive mechanism for error handling, input validation, and data persistence. These gaps could potentially lead to unexpected issues, such as buffer overflows when users input more data than expected, memory leaks due to missing memory deallocation, and a lack of safeguards against erroneous user input.

Additionally, the code is single-threaded, meaning it can't handle multiple users or concurrent scheduling requests effectively. For a more robust and scalable system, multi-threading or other concurrency solutions might be necessary.

Moreover, the code doesn't store appointments persistently. When the program exits, all appointments are lost. A data persistence solution, such as file I/O or database integration, would be essential for retaining appointments between program runs.

Documentation is another concern; the code lacks comments and explanations, making it less accessible for other developers or for future maintenance. To enhance the code's reliability and userfriendliness, addressing these issues by incorporating input validation, error handling, memory management, and data persistence would be prudent, while also providing detailed comments for clarity.

3. ANALYSIS

Time Complexity:

Creating an Appointment (createAppointment): This function allocates memory for an appointment structure and reads input from the user. The time complexity for memory allocation is O(1), and the time complexity for reading input is generally O(n), where n is the size of the input. In this case, n is the maximum length of the strings (50 characters for 'name' and 20 characters for 'date' and 'time'). Therefore, the time complexity for creating an appointment is O(1).

Initializing a Queue (createQueue): Initializing a queue involves memory allocation, which is O(1) in terms of time complexity.

Enqueueing an Appointment (enqueue): Enqueueing an appointment involves modifying the pointers of the queue. This operation is generally O(1) in terms of time complexity because it involves updating a few pointers regardless of the size of the queue.

Displaying Appointments (displayAppointments): Displaying appointments requires traversing the entire queue. In the worst case, when all appointments are to be displayed, this operation has a time complexity of O(n), where n is the number of appointments in the queue.

Main Loop (main): The main loop runs until the user chooses to exit, so the number of iterations depends on the user's interactions, but each iteration is relatively quick. Therefore, the main loop's time complexity is primarily driven by the choices the user makes.

Space Complexity:

Creating an Appointment (createAppointment): This function allocates memory for an appointment structure. The space complexity is O(1) because it allocates a fixed amount of memory for each appointment.

Initializing a Queue (createQueue): Initializing a queue involves allocating memory for the queue structure. The space complexity is O(1).

Enqueueing an Appointment (enqueue): This operation doesn't allocate additional memory for the appointment itself but involves changing pointers in the queue structure. The space complexity is O(1).

Displaying Appointments (displayAppointments): This function uses a constant amount of memory for temporary variables. The space complexity is O(1).

Main Function (main): The space complexity for the main function is primarily determined by the space required for the appointment queue. If there are 'n' appointments in the queue, the space complexity is O(n) because each appointment occupies a fixed amount of memory.

In summary, the time complexity of the operations in this code is generally quite efficient. The most time-consuming operation is displaying appointments, which has a linear time complexity with respect to the number of appointments. The space complexity is also reasonable, mainly dependent on the number of appointments in the queue.

4. EXPLANATION

The provided C code represents a basic appointment scheduling system. It is organized around the concepts of appointments and a queue data structure. In the first paragraph, the code defines two primary structures: Appointment and Queue. The Appointment structure represents an individual appointment, with fields for the name, date, time, and a pointer to the next appointment. The Queue structure acts as a linked list, allowing the storage of multiple appointments with pointers to the front and rear of the queue. The code also includes functions to create new appointments, initialize a queue, enqueue appointments into the queue, and display the appointments stored in the queue.

The second paragraph focuses on the core functionality of the code. The createAppointment function dynamically allocates memory for a new appointment, collects input from the user (name, date, and time), and then returns the newly created appointment. The createQueue function initializes a queue by allocating memory and setting the front and rear pointers to NULL. The enqueue function adds appointments to the queue by modifying the rear pointer and connecting the appointments in a linked list fashion. The displayAppointments function is used to display the details of all appointments in the queue. The main function is responsible for managing the program's execution flow, presenting a user-friendly menu, and handling user choices.

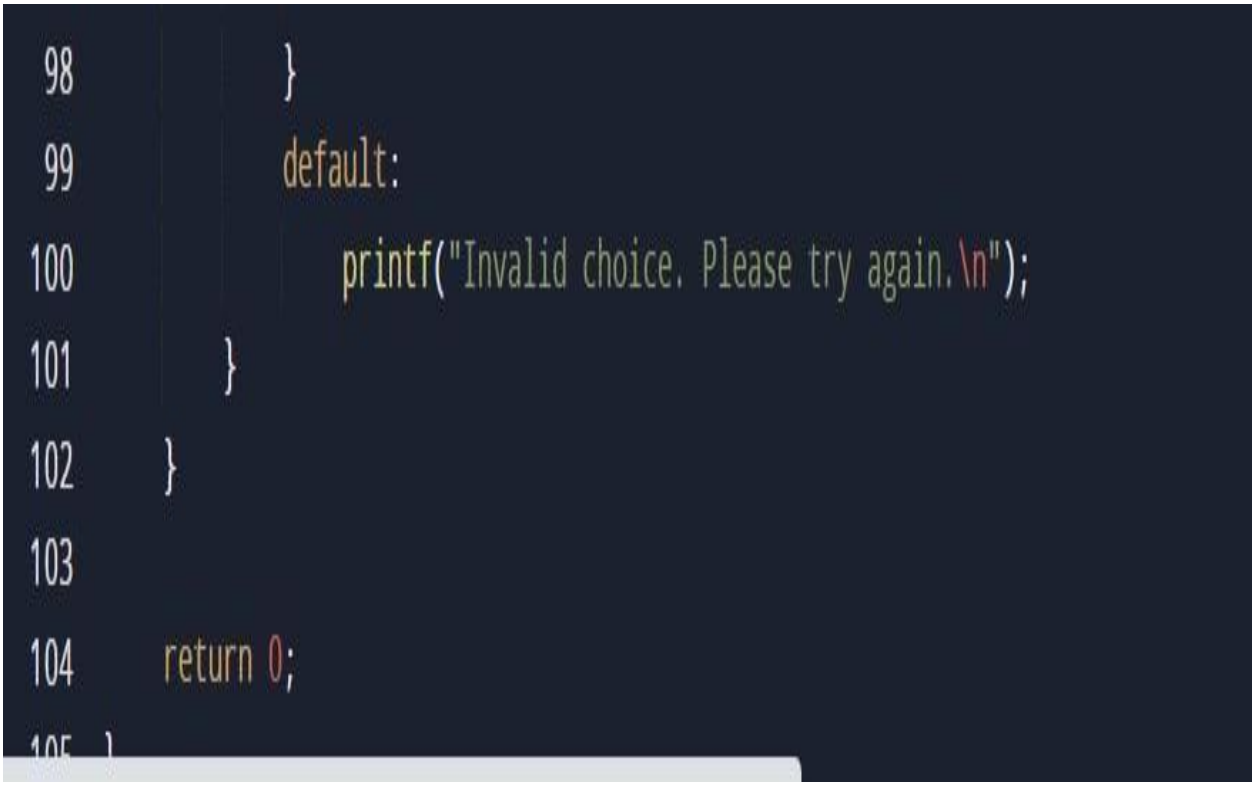
The third paragraph highlights both the strengths and limitations of the code. Some of its advantages include a simple user interface, modularity through well-defined functions, dynamic memory allocation for flexible appointment storage, basic error handling for memory allocation failures, and a straightforward example of implementing a queue using structures and linked lists. However, the code also has notable disadvantages, such as a lack of input validation that may lead to buffer overflows, no memory deallocation mechanisms that can result in memory leaks, and the absence of data persistence, which means appointments are lost when the program exits. Additionally, the code is single-threaded and doesn't handle concurrent scheduling requests effectively, limiting its real-world applicability. Addressing these limitations would be necessary to develop a more robust and practical appointment scheduling system.

5. IMPLEMENTATION

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

// Define the structure for an appointment
struct Appointment {
    char name[50];
    char date[20];
    char time[20];
    struct Appointment* next;
};

// Define a structure for the queue
struct Queue {
    struct Appointment* front;
    struct Appointment* rear;
};

// Function to create a new appointment
struct Appointment* createAppointment() {
    struct Appointment* newAppointment = (struct Appointment*)malloc(sizeof
        (struct Appointment));
    if (newAppointment == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }

```

```
25
26      printf("Enter Name: ");
27      scanf("%s", newAppointment->name);
28      printf("Enter Date (MM/DD/YYYY): ");
29      scanf("%s", newAppointment->date);
30      printf("Enter Time: ");
31      scanf("%s", newAppointment->time);
32      newAppointment->next = NULL;
33
34      return newAppointment;
35  }
36
37  // Function to initialize a queue
38  struct Queue* createQueue() {
39      struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
40      if (queue == NULL) {
41          printf("Memory allocation failed.\n");
42          exit(1);
43      }
44      queue->front = queue->rear = NULL;
45      return queue;
46  }
47
48  // Function to enqueue an appointment
49  void enqueue(struct Queue* queue, struct Appointment* newAppointment) {
50      if (queue->rear == NULL) {
```

```
74      int choice;
75
76      while (1) {
77          printf("\nAppointment Scheduling System\n");
78          printf("1. Add Appointment\n");
79          printf("2. Display Appointments\n");
80          printf("3. Exit\n");
81          printf("Enter your choice: ");
82          scanf("%d", &choice);
83
84          switch (choice) {
85              case 1: {
86                  struct Appointment* newAppointment = createAppointment();
87                  enqueue(appointmentQueue, newAppointment);
88                  printf("Appointment added to the queue.\n");
89                  break;
90              }
91              case 2: {
92                  displayAppointments(appointmentQueue);
93                  break;
94              }
95              case 3: {
96                  printf("Exiting the program.\n");
97                  exit(0);
98              }
99              default:
```

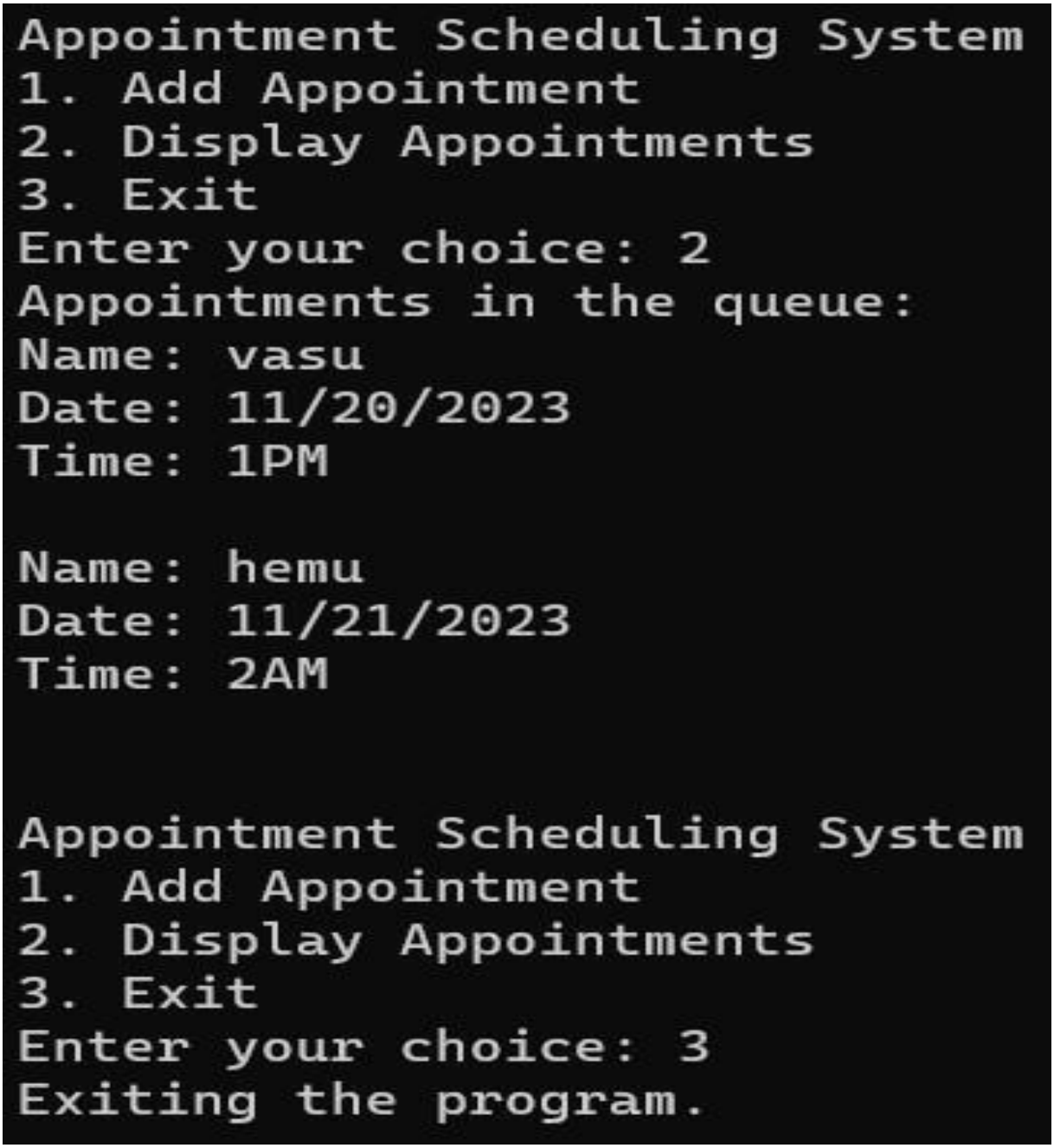


```
98              }
99              default:
100                 printf("Invalid choice. Please try again.\n");
101         }
102     }
103
104     return 0;
105 }
```

6. EXPERIMENTAL RESULTS

OUTPUT:

```
Appointment Scheduling System
1. Add Appointment
2. Display Appointments
3. Exit
Enter your choice: 1
Enter Name: vasu
Enter Date (MM/DD/YYYY): 11/20/2023
Enter Time: 1PM
Appointment added to the queue.

Appointment Scheduling System
1. Add Appointment
2. Display Appointments
3. Exit
Enter your choice: 1
Enter Name: hemu
Enter Date (MM/DD/YYYY): 11/21/2023
Enter Time: 2AM
Appointment added to the queue.
```


Appointment Scheduling System
1. Add Appointment
2. Display Appointments
3. Exit
Enter your choice: 2
Appointments in the queue:
Name: vasu
Date: 11/20/2023
Time: 1PM
Name: hemu
Date: 11/21/2023
Time: 2AM
Appointment Scheduling System
1. Add Appointment
2. Display Appointments
3. Exit
Enter your choice: 3
Exiting the program.

7. CONCLUSION

In conclusion, the provided code offers a basic but functional implementation of an appointment scheduling system. It leverages C data structures and dynamic memory allocation to manage a queue of appointments. Users can add appointments with names, dates, and times, and then display the existing appointments. This code serves as a useful starting point for understanding fundamental concepts such as structures, linked lists, and memory management in C programming.

However, there are several important limitations to consider. First and foremost, the code lacks input validation, which makes it vulnerable to buffer overflows if users input data exceeding the allocated buffer sizes. Additionally, there is no provision for memory deallocation, leading to potential memory leaks. Furthermore, the code does not persist appointment data, meaning that all appointments are lost when the program exits. Lastly, the code is single-threaded, making it less suitable for handling concurrent scheduling requests. To create a more robust and practical appointment scheduling system, these limitations need to be addressed, and additional features such as data storage, data validation, and concurrent access support should be considered.