

# Tugas Akhir SDA Semester 2



- 24040700107  
M.Daffa
- 24040700110  
Muhammad Alif Razaqi
- 24040700105  
Nurul Hakim

---

## Journal: Restaurant Waitlist Management System

**Project Title:** Restaurant Waitlist Management Using Queue (C Language)  
Memory for each node is dynamically allocated using malloc().

- Once a customer is served, the node is removed and its memory is released using free().

This design ensures flexibility and scalability: the number of customers can grow and shrink dynamically without predefined limits.

---

### Key and Problem

One of the key issues in restaurant management is maintaining an organized and fair waitlist system, especially during peak hours when customer flow exceeds seating capacity. Traditional approaches—such as using paper lists or verbal confirmations—are inefficient, prone to human error, and difficult to monitor in real time.

The core problem addressed by this project is the absence of a simple, reliable, and automated system for managing customer queues in small to medium-sized restaurants. Many establishments lack access to advanced reservation systems due to cost or complexity. As a result, they often struggle with inconsistent queue handling, miscommunication, and customer dissatisfaction.

This project focuses on solving the problem by providing:

- A digital queue system based on the First-In-First-Out (FIFO) principle.
- A console-based interface that is easy to use and implement.

- Dynamic memory handling to adapt to varying queue sizes.
- A foundation for future enhancements like prioritization, data persistence, and GUI integration.

By implementing this solution in the C programming language, the project not only addresses a real-world service gap but also reinforces important programming concepts in data structures and memory management.

---

### Key Features of the Program:

- **Add Customer to Waitlist:** Customers can be added to the end of the queue using dynamic memory allocation.
- **Serve Customer:** The system serves customers in First-In-First-Out (FIFO) order by removing them from the front of the queue.
- **View Waitlist:** Displays the names of customers currently in the waitlist in their order of arrival.
- Avoiding memory leaks and dangling pointers.

The ultimate aim is to make queue handling more efficient and reliable through basic programming principles.

---

### System Design

The system revolves around a **queue**, a linear data structure where elements are added at the rear and removed from the front. This matches the behavior expected in a restaurant waiting scenario.

Each **customer is represented by a node** in a singly linked list. The node contains:

- The customer's name (a string),

- A pointer to the next customer.

Two global pointers—front and rear—keep track of the beginning and end of the queue.

#### Memory Handling:

- Memory for each node is dynamically allocated using malloc().
- Once a customer is served, the node is removed and its memory is released using free().

This design ensures flexibility and scalability: the number of customers can grow and shrink dynamically without predefined limits.

---

#### Key and Problem

One of the key issues in restaurant management is maintaining an organized and fair waitlist system, especially during peak hours when customer flow exceeds seating capacity. Traditional approaches—such as using paper lists or verbal confirmations—are inefficient, prone to human error, and difficult to monitor in real time.

The core problem addressed by this project is the absence of a simple, reliable, and automated system for managing customer queues in small to medium-sized restaurants. Many establishments lack access to advanced reservation systems due to cost or complexity. As a result, they often struggle with inconsistent queue handling, miscommunication, and customer dissatisfaction.

This project focuses on solving the problem by providing:

- A digital queue system based on the First-In-First-Out (FIFO) principle.
- A console-based interface that is easy to use and implement.
- Dynamic memory handling to adapt to varying queue sizes.

- A foundation for future enhancements like prioritization, data persistence, and GUI integration.

By implementing this solution in the C programming language, the project not only addresses a real-world service gap but also reinforces important programming concepts in data structures and memory management.

---

#### Key Features of the Program:

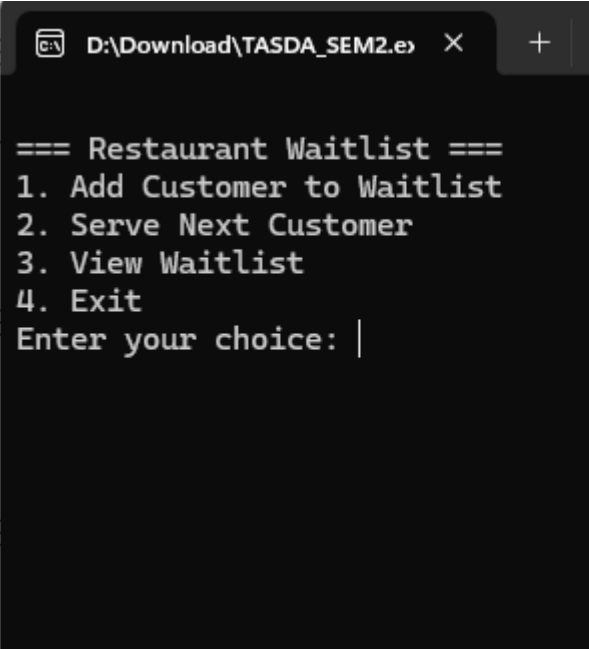
- **Add Customer to Waitlist:** Customers can be added to the end of the queue using dynamic memory allocation.
- **Serve Customer:** The system serves customers in First-In-First-Out (FIFO) order by removing them from the front of the queue.
- **View Waitlist:** Displays the names of customers currently in the waitlist in their order of arrival.
- **Menu Driven:** User interacts via a simple console menu with options for each functionality.

---

### Data Structure Used:

- **Queue (Linked List Implementation):**
    - Struct Node represents each customer.
    - Pointers front and rear manage the start and end of the queue.
    - malloc() is used for dynamic node creation; free() for memory deallocation.
- 

### User Interface (Sample):



```
D:\Download\TASDA_SEM2.e>

=== Restaurant Waitlist ===
1. Add Customer to Waitlist
2. Serve Next Customer
3. View Waitlist
4. Exit
Enter your choice: |
```

---

### Challenges Faced:

- Ensuring memory was properly allocated and freed.
- Managing string input safely with fgets() and removing trailing newlines.
- Avoiding memory leaks and dangling pointers.

---

### Function Descriptions

The program is composed of modular functions, each responsible for a specific task. Below are the main functions used:

- **addCustomer(char name[])**  
Adds a customer to the rear of the queue. If the queue is empty, the new node becomes both the front and rear.
- **serveCustomer(char servedName[])**  
Removes the customer at the front of the queue, returns the name for display, and frees the associated memory.
- **getWaitlist(char list[][MAX\_NAME\_LEN], int maxCount)**  
Copies all the names from the queue into a 2D array for display purposes.
- **showMenu()**  
Displays the interface options (add, serve, view, exit) and returns the user's choice.

These functions work together within a loop in the main() function to continuously interact with the user until they choose to exit.

---

### Future Enhancements

This prototype can be extended into a more advanced system with added functionalities:

- **File I/O**  
Save and retrieve waitlists from disk so data isn't lost between sessions.
- **Time Logging**  
Store timestamps for each customer's entry to track wait durations.

- **Priority Queues**  
Add priority handling for VIP customers or reservations.
- **Graphical User Interface (GUI)**  
Build a GUI using libraries like GTK or migrate to mobile/web platforms using technologies such as React or Flutter.
- **Language Porting**  
Translate the program into languages like Python or JavaScript for wider platform compatibility and development speed.
- **Real-Time Notifications**  
Integrate SMS, email, or in-app notifications to inform customers when their table is ready.
- **Data Analytics Dashboard**  
Provide restaurant managers with insights such as average wait time, peak hours, and customer trends through visual charts.
- **Multi-Branch Support**  
Allow the system to handle multiple restaurant locations and share data between branches.
- **Database Integration**  
Use a database system (e.g., SQLite, MySQL) to store customer data securely and support advanced querying.
- **Mobile App Integration**  
Enable customers to join the waitlist remotely through a mobile app or web interface.
- **QR Code System**  
Generate QR codes for customers to check their status in the queue or update their reservation remotely.

These additions would not only enhance functionality but also improve user experience, scalability, and readiness for commercial deployment.

---

## Results and Outputs

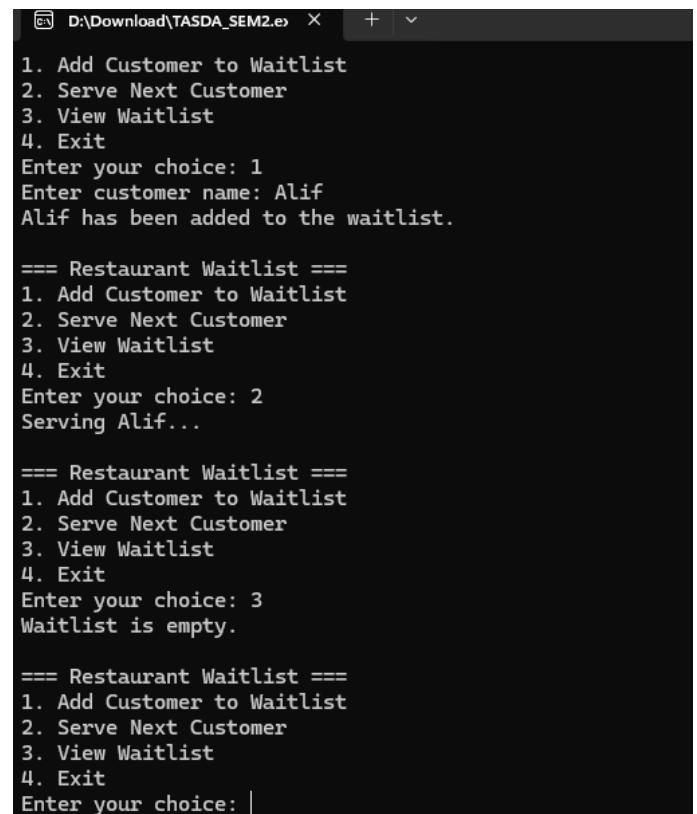
When tested, the program produced the expected outcomes for all operations:

Adding a customer displays:

Serving a customer displays:

Viewing the waitlist displays:

If the queue is empty:



```
D:\Download\TASDA_SEM2.e>
1. Add Customer to Waitlist
2. Serve Next Customer
3. View Waitlist
4. Exit
Enter your choice: 1
Enter customer name: Alif
Alif has been added to the waitlist.

=== Restaurant Waitlist ===
1. Add Customer to Waitlist
2. Serve Next Customer
3. View Waitlist
4. Exit
Enter your choice: 2
Serving Alif...

=== Restaurant Waitlist ===
1. Add Customer to Waitlist
2. Serve Next Customer
3. View Waitlist
4. Exit
Enter your choice: 3
Waitlist is empty.

=== Restaurant Waitlist ===
1. Add Customer to Waitlist
2. Serve Next Customer
3. View Waitlist
4. Exit
Enter your choice: |
```

These results confirm that the system maintains accurate order and memory integrity throughout each operation.

---

## Challenges and Lessons Learned

Key challenges included:

- **Memory Management:**  
Properly allocating and freeing memory for each customer was critical. Errors here could lead to memory leaks or crashes.

- **Pointer Manipulation:**  
When adding or removing customers, ensuring that the front and rear pointers were updated correctly was essential to maintain the queue structure.
  - **String Handling:**  
Accepting and cleaning input safely (removing trailing newlines from fgets) helped avoid string-related bugs.
- 

#### Lessons Learned:

- The importance of **defensive programming**, especially in C where manual memory management is required.
- How to simulate real-world systems using basic data structures.
- Breaking down complex problems into manageable, reusable functions.

This project successfully demonstrates how basic data structures—particularly queues—can be implemented in C to solve practical problems like restaurant waitlist management.

The current system performs well in handling core operations, but it can be further developed in several ways:

---

#### Suggested Enhancements:

- **Data Persistence:** Save and load waitlist from a file.
- **Timestamps:** Add the time each customer entered the queue.
- **Prioritization:** Serve VIP or reservation customers differently.
- **GUI or Web Interface:** Improve usability by creating a graphical version using a toolkit or web framework.

- **Language Porting:** Move the system to Python, JavaScript, or Java to improve accessibility and deployment.
- 

#### Reflection and Conclusion

This project provided a valuable opportunity to apply core concepts of **data structures, memory management, and user interaction in C programming**.

Though the project may appear basic at first glance, it solves a common real-world problem effectively and lays the groundwork for more complex systems. It also reinforced best practices in modular programming and taught how important it is to **plan for edge cases**, especially in languages like C that offer little protection by default.

The final result is a functioning program that meets its goals and is ready to be improved upon. It serves as a strong example of how **simple tools can create meaningful solutions**.

---

#### Flowchart Description: Customer Queue Management System

This flowchart represents a **Customer Queue Management System** that allows users to interact with a simple menu-driven program to manage a waitlist. The system operates as follows:

1. **Start:** The program begins.
2. **Display Menu:** The user is presented with four options:
  - 1. Add Customer
  - 2. Serve Next Customer
  - 3. View Waitlist
  - 4. Exit

3. **Input Choice:** The user inputs their menu selection.

4. Based on the input:

○ **Choice = 1 (Add Customer):**

- The system prompts the user for the customer's name.
- It calls the `addCustomer()` function.
- If successful, it continues; if not, it shows an error message.

○ **Choice = 2 (Serve Next Customer):**

- The system checks if the customer count is greater than 0.
- If yes, it calls `serveCustomer()` to serve the next customer.
- If no, it displays a list of names (possibly an empty list).
- Then it shows the message "exit is ehpre" (possibly a placeholder or typo).

○ **Choice = 3:**

- It displays the list of customers currently in the waitlist.
- Then shows the "exit is ehpre" message again.

○ **Choice = 4 (Exit):**

- The program shows an exit message and ends.

○ **Invalid Choice:**

- If the input does not match any valid options, it shows "Invalid choice" and then displays the exit message.

5. **End:** The program terminates.

