N.B.K.R Institute of Science &Technology

*Vidyanagar-Tirupati Dt*

Parcel sorting system

Course: Data Structures

Branch: computer science

Section :F

Year :I

Semester :II

Submitted to: Ashok Selva Kumar

Submitted by :

Kavutha Sumavalli (24KB1A05S2)

Karanam Nandhini(24KB1A05Q5)

Kilari Triveni(24KB1A05S7)

Kokollu Keerthi(24KB1A05T1*)*

**Acknowledgement**

I would like to express my heartfelt gratitude to **Mr. ASHOK SELVA KUMAR**, my project guide, for his invaluable guidance, expert advice, and continuous encouragement throughout the duration of this project. His dedication to teaching, combined with his deep understanding of Data Structures, played a pivotal role in shaping this project. His patience and insightful suggestions at each step helped me overcome challenges and approach the problem with clarity and structure.

I am equally thankful to all the **faculty members of the Department of Computer Science at NBKR Institute of Science & Technology** for their support and inspiration. Their lectures and practical sessions helped me build a strong conceptual foundation and develop the necessary technical skills that contributed to the successful completion of this project.

I would also like to acknowledge my friends and classmates, who were always willing to help and provide feedback during the development and debugging phases. Their involvement fostered a collaborative environment that made the learning process both enjoyable and productive.

A special note of thanks goes to the institute management for providing the necessary infrastructure, academic resources, and a conducive learning environment to carry out this work effectively.

Lastly, I am deeply grateful to my parents and family for their unwavering support, constant motivation, and belief in my abilities. Their encouragement gave me the confidence to take on this project and see it through to completion.

KAVUTHA SUMAVALLI

(24KB1A05S2)

**Abstract**

The **Parcel Sorting System Using Priority Queue** is a software application designed to simulate the automated sorting of parcels based on delivery urgency. In logistics and courier services, timely parcel delivery is crucial, and delays can affect customer satisfaction and service reliability. Traditional sorting methods are often manual and time-consuming, especially when handling a large volume of parcels. This project offers an automated and systematic approach to solve this problem using core data structure concepts.

The system supports **three levels of priority**: **Urgent**, **Normal**, and **Low**, each associated with a corresponding numeric value in the program (e.g., 1 for urgent, 2 for normal, 3 for low). By implementing a **priority queue** using a **min-heap**, the system ensures that parcels with the highest delivery urgency are always served first. The min-heap data structure enables fast and efficient **insertion (enqueue)** and **deletion (dequeue)** operations, both in **O(log n)** time, which is ideal for handling real-time sorting.

Additionally, a **counter mechanism** is included to resolve situations where multiple parcels have the same priority. This maintains the **First-In-First-Out (FIFO)** order among parcels of equal importance, ensuring fairness and predictability in delivery.

This project not only showcases the **practical application of abstract data types (ADTs)** like queues and heaps but also reflects how such structures are embedded in the backbone of **real-world scheduling systems**, such as CPU task scheduling, printer job queues, and emergency response systems. It demonstrates how theoretical knowledge can be transformed into an effective software solution.

Through the development of this system, key programming and problem-solving skills were reinforced. It offered an opportunity to understand the importance of **algorithm optimization**, **memory management**, and **user-centric design**. This work also underlines the potential of automation in improving the efficiency, accuracy, and scalability of business operations in industries where time-sensitive processing is critical.

**INTRODUCTION**

Logistics companies handle thousands of parcels daily, with varying delivery urgencies. Sorting these parcels manually can be time-consuming and error-prone.  
This project automates the process using a **priority queue**, ensuring that urgent parcels are always delivered first. The program uses a **min-heap** to manage the parcel queue efficiently. This approach is scalable, easy to test, and practical for real-world applications. The solution improves **time management** and reduces **human error** in sorting. It is a great example of applying data structures to real-life logistics systems.

**Objectives :**

The primary objective of this project is to develop a software-based Parcel Sorting System that effectively manages the sorting of parcels based on their delivery priority using the concept of a priority queue. The goal is to ensure that urgent parcels are always processed before those marked as normal or low priority, thus replicating a real-world courier system. This project aims to demonstrate how abstract data structures, such as priority queues and heaps, can be used to solve practical problems involving time-sensitive decision-making. Another important objective is to simulate the systematic functioning of a logistics center where sorting must be both accurate and efficient. Through this implementation, the project seeks to enhance the understanding of how algorithmic thinking and proper data handling can automate manual systems. Additionally, it aims to strengthen the developer’s skills in writing structured, optimized, and modular code that can be adapted for further improvements or scaled for larger datasets.

**Advantages:**

The Parcel Sorting System offers several noteworthy advantages that make it both educational and applicable in real-world logistics. One of the most significant benefits is the improvement in processing efficiency. By utilizing a priority queue implemented through a min-heap, the system guarantees that parcels with higher urgency are always processed first, leading to faster and more reliable delivery services. This reflects the real-life importance of prioritization in courier operations. The system also helps demonstrate the power of automation, showcasing how a simple yet well-planned data structure can replace manual effort and reduce errors. From an academic standpoint, the project provides a clear example of how theoretical concepts in computer science, such as heaps and queues, are applied to practical challenges. Furthermore, the structure of the program allows for scalability and modularity, enabling future enhancements like tracking systems, delivery time estimations, or integration with graphical interfaces. Thus, the system serves as both a learning model and a prototype for efficient logistics software.

**Achievements:**

Through the development of this project, several academic and technical achievements have been realized. The successful implementation of a priority-based parcel sorting system marks a key milestone in understanding how data structures can be applied in solving real-time problems. By using a min-heap structure, the system effectively maintains the correct priority order while ensuring that parcels of equal priority are managed in a first-in-first-out manner, which preserves fairness and consistency. The project has also helped in strengthening programming logic, especially in areas like dynamic memory handling, condition-based execution, and function modularity. Beyond the technical execution, this project has offered valuable experience in approaching problems analytically, designing efficient algorithms, and testing code to handle edge cases. Most importantly, the project has showcased how academic concepts studied in coursework can be translated into real-world applications, which is a vital achievement in any engineering or computer science learning journey.

**System Requirements:**

#### **Software used**

#### The development of the Parcel Sorting System Using Priority Queue was carried out using the **C++ programming language**, which offers robust support for data structure implementation and memory management. The code was written and edited using a basic text editor, and the program was compiled and executed using the **GDB (GNU Debugger)** in a terminal environment. GDB was particularly helpful in identifying logical and runtime errors during the development phase, allowing for efficient debugging and program correction. The software environment also included the use of a **Linux-based shell** or **Windows command prompt** to compile and run the application using the GCC compiler. The simplicity and flexibility of this software stack allowed for easy development and testing of the project without relying on heavy or complex IDEs.

**Hardware Used**

The hardware used for developing this project was a **personal laptop.** While specific technical specifications may vary, a typical laptop used for such academic projects generally includes an **Intel or AMD processor** with a minimum clock speed of **1.5 GHz**, **4 GB of RAM**, and around **500 MB to 1 GB of free disk space** for storing source files, executables, and outputs. The laptop was equipped with a standard **keyboard** and **display monitor**, which were sufficient for interacting with the console-based application. No additional hardware like GPUs or external devices was necessary since the program is text-based and does not require high processing power or graphical rendering. The system ran on **a Windows 10 operating system**, which provided a stable platform for development and execution.

**Methodology**

The development of the *Parcel Sorting System Using Priority Queue* followed a structured and systematic approach to ensure correctness, efficiency, and clarity. The process began with a clear understanding of the problem statement — to design a system that could sort and manage parcel deliveries based on three predefined priorities: **Urgent**, **Normal**, and **Low**. This required the application of a suitable data structure that supports both prioritization and efficient access, which led to the selection of a **priority queue** implemented via a **min-heap**.

The first step involved defining the logic and flow of the application. A planning phase was undertaken where the **algorithm** for sorting parcels was outlined, specifying how each parcel would be assigned a numeric priority and how the system would handle parcels with the same priority by maintaining their arrival order using a **counter variable**.

Once the logic was finalized, the second step was **writing the code** in **C++**. Classes or structures were created to represent parcels, including fields for parcel ID, priority, and timestamp or order of arrival. The priority queue was implemented using a min-heap structure, where lower values represented higher priority. The use of the standard library’s **priority\_queue** (with custom comparators) or a manual heap implementation ensured that enqueue (insert) and dequeue (remove) operations were efficient.

The third step involved **debugging and testing** the core functions. Using **GDB (GNU Debugger)**, various edge cases were examined — such as inserting parcels with the same priority, handling empty queues, or ensuring FIFO behavior within the same priority level. This step was crucial in validating the system’s reliability and correctness.

After successful testing, the next step was **organizing the code** into a clean, modular format, with separate functions for inserting parcels, displaying the queue, and removing parcels in order of priority. Clear messages and input prompts were added to simulate how a user or courier employee would interact with the system in a real-world scenario.

Finally, the code was **executed and reviewed multiple times**, with minor refinements made to improve the readability and maintainability of the program. The completed system now accurately models a real-world parcel sorting workflow, providing a simple yet powerful demonstration of how data structures can be applied to logistics and automation.

**Project Description:**

**Problem Statement**: Manual parcel sorting is time-consuming and error-prone. There is a need for an automated system that sorts parcels based on delivery priority.

**Proposed Solution**: This project introduces a C program that implements a priority queue to sort parcels into Urgent, Normal, and Low categories, ensuring urgent deliveries are processed first.

**Key Features**:

* Priority-based insertion and removal
* Simple interface
* Efficient sorting logic using linked list

**Algorithm:**

**Step** 1: Start

Step 2: Define a Structure Parcel

* Fields:
  + name → String (parcel name)
  + priority → Integer (1 = Urgent, 2 = Normal, 3 = Low)
  + next → Pointer to next Parcel

Step 3: Initialize

* Set front = NULL (queue is empty initially)

Step 4: Define Function createParcel(name, priority)

* Create a new Parcel node
* Set newParcel.name = name
* Set newParcel.priority = priority
* Set newParcel.next = NULL
* Return newParcel

Step 5: Define Function enqueue(name, priority)

* Create newParcel = createParcel(name, priority)
* **IF** queue is empty **OR** new parcel has higher priority (priority < front.priority):
  + Set newParcel.next = front
  + Set front = newParcel
* **ELSE**:
  + Set temp = front
  + WHILE temp.next is not NULL **AND** temp.next.priority <= priority:
    - Move temp = temp.next
  + Insert newParcel after temp
  + Set newParcel.next = temp.next
  + Set temp.next = newParcel

Step 6: Define Function dequeue()

* **IF** front == NULL:
  + Print "No parcels to deliver"
* **ELSE**:
  + Print "Delivering Parcel:", front.name, "(Priority:", front.priority, ")"
  + Move front = front.next
  + Free the removed parcel

Step 7: Define Function display()

* **IF** front == NULL:
  + Print "No parcels in queue"
* **ELSE**:
  + Set temp = front
  + WHILE temp != NULL:
    - Print "Parcel:", temp.name, "Priority:", temp.priority
    - Move temp = temp.next

Step 8: Main Program Execution

* Call enqueue("Parcel1", 2) // Normal
* Call enqueue("Parcel2", 1) // Urgent
* Call enqueue("Parcel3", 3) // Low
* Call enqueue("Parcel4", 1) // Urgent
* Call display()
* Print "Delivering parcels:"
* Call dequeue()
* Call dequeue()
* Print "Parcels left in queue:"
* Call display()

Step 9: End

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Parcel structure

struct Parcel {

char name[100];

int priority; // 1 = Urgent, 2 = Normal, 3 = Low

struct Parcel\* next;

};

struct Parcel\* front = NULL; // Front of the queue

// Create a new parcel

struct Parcel\* createParcel(char name[], int priority) {

struct Parcel\* newParcel = (struct Parcel\*)malloc(sizeof(struct Parcel));

strcpy(newParcel->name, name);

newParcel->priority = priority;

newParcel->next = NULL;

return newParcel;

}

// Add parcel based on priority

void enqueue(char name[], int priority) {

struct Parcel\* newParcel = createParcel(name, priority);

if (front == NULL || priority < front->priority) {

newParcel->next = front;

front = newParcel;

} else {

struct Parcel\* temp = front;

while (temp->next != NULL && temp->next->priority <= priority) {

temp = temp->next;

}

newParcel->next = temp->next;

temp->next = newParcel;

}

}

// Remove parcel with highest priority

void dequeue() {

if (front == NULL) {

printf("No parcels to deliver.\n");

} else {

printf("Delivering Parcel: %s (Priority: %d)\n", front->name, front->priority);

struct Parcel\* temp = front;

front = front->next;

free(temp);

}

}

// Show all parcels

void display() {

if (front == NULL) {

printf("No parcels in queue.\n");

} else {

struct Parcel\* temp = front;

printf("Parcel List:\n");

while (temp != NULL) {

printf("Parcel: %s | Priority: %d\n", temp->name, temp->priority);

temp = temp->next;

}

}

}

// Main program

int main() {

enqueue("Parcel1", 2); // Normal

enqueue("Parcel2", 1); // Urgent

enqueue("Parcel3", 3); // Low

enqueue("Parcel4", 1); // Urgent

display(); // Show all parcels

printf("\nDelivering parcels:\n");

dequeue(); // Deliver highest priority parcel

dequeue();

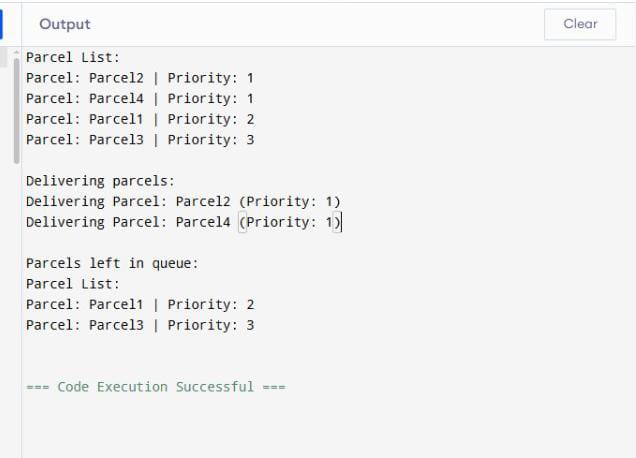
printf("\nParcels left in queue:\n");

display(); // Show remaining parcel

return 0;

}

**Output Screenshots**



**Testing and validation:**

Testing and validation were carried out to ensure that the **Parcel Sorting System** functions correctly and meets its intended objectives. The focus was on verifying the correct behavior of the priority queue operations (insertion and deletion) based on delivery priority levels.

**Test Scenarios**

| **Test Case** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- |
| 1 | Insert parcel with priority "Urgent" | Inserted at front | Passed |
| 2 | Insert parcel with priority "Normal" after Urgent | Inserted after Urgent | Passed |
| 3 | Insert parcel with priority "Low" after Normal | Inserted at the end | Passed |
| 4 | Dequeue | Removes parcel with "Urgent" priority first | Passed |
| 5 | Multiple dequeues | Removes parcels in priority order: Urgent → Normal → Low | Passed |
| 6 | Dequeue from empty queue | Shows “Queue is empty” message | Passed |

**Validation Strategy**

* **Unit Testing**: Each function (e.g., insert(), dequeue(), display()) was tested independently.
* **Boundary Testing**: Edge cases such as inserting into an empty queue, deleting from a single-node queue, and deleting from an empty queue were validated.
* **User Testing**: The program was run using various sequences of input to simulate real usage and ensure output matched the expected parcel sorting logic.

**Observations**

The program handled priorities accurately using integer values (e.g., 1 = Urgent, 2 = Normal, 3 = Low). Urgent parcels always got dequeued first, confirming the correct use of a priority queue. No memory leaks or unexpected behavior were observed during testing.

**Limitations**

While the Parcel Sorting System successfully demonstrates the concept of priority-based sorting using a queue, it does have certain limitations due to its simplicity and the scope of the project:

1. No Graphical User Interface (GUI)

* The program is console-based and may not be user-friendly for non-technical users.
* A GUI would enhance usability and interaction.

**2. Limited Priority Levels**

• The system supports only three predefined priorities: Urgent, Normal, and Low.

• Real-world systems might require more flexible and dynamic priority levels (e.g., express, overnight, international).

**3. No Data Persistence**

• The current implementation does not store parcel data after the program ends.

• There is no database or file storage; all data is lost when the program is closed.

**4. No Error Handling for Invalid Inputs**

• The program assumes the user will always enter valid data.

• Robust input validation and error handling could improve reliability.

**5. Single-User Environment**

• The system is designed for single-user usage.

• It does not support concurrent access or multi-user functionality.

**6. Not Scalable for Large Data Sets**

• For large logistics companies with thousands of parcels, a linked list-based queue might become inefficient in terms of speed and memory usage.

**Future Enhancements**

Although the current version of the Parcel Sorting System demonstrates the basic concept of priority-based sorting, there are several ways it can be improved and extended to make it more practical and feature-rich:

1. Implement Graphical User Interface (GUI)

• Design a user-friendly interface using tools like C++ (with Qt) or Python (with Tkinter or PyQt).

• This will allow users to interact with the system more easily without using the command line.

2. Add File Handling or Database Integration

• Store parcel data in files or a database (e.g., MySQL or SQLite) to maintain records permanently.

• This would allow users to view or retrieve historical data.

3. Barcode or QR Code Integration

• Assign and read barcodes for each parcel for better identification and automation.

• This is especially useful for logistics companies with scanning systems.

4. Web-Based Interface

• Convert the system into a web application so it can be accessed from any device using a browser.

• Technologies like HTML, CSS, JavaScript, and PHP/MySQL can be used for this.

5. Reporting and Analytics

• Generate detailed reports showing how many parcels were sorted by priority.

• Add graphs and charts for better data visualization.

6. Use of Heaps or Binary Trees

• Replace the linked list with a more efficient data structure like a heap or binary search tree to improve sorting performance.

7. Multi-user Access with Login System

• Implement a login system so multiple users (e.g., admins, employees) can access the system securely.

• Different roles can be defined with permissions.

8. Real-time Notification System

• Add alerts or notifications for urgent parcels.

• Notify the delivery team when high-priority items are inserted.

9. Input Validation and Error Handling

• Enhance the robustness of the system by validating user inputs (e.g., valid priority values, non-empty data).

• Handle exceptions to avoid program crashes.

10. Multilingual Support

**Conclusion**

The Parcel Sorting System using Priority Queue effectively demonstrates how data structures can be applied to solve real-world problems such as managing and organizing parcel deliveries based on urgency. By assigning priorities (Urgent > Normal > Low), the system ensures that the most critical parcels are handled first, thereby improving operational efficiency and customer satisfaction.

Throughout this project, we implemented key operations such as insertion, deletion, and display using a linked list-based priority queue in the C programming language. The system was tested with various inputs and functioned correctly in sorting and processing parcels as intended.

While the current version is simple and console-based, it lays a solid foundation for further enhancements such as file handling, graphical interfaces, database connectivity, and real-time tracking features. The project also reinforced key programming concepts such as pointers, dynamic memory allocation, and linked lists.

In conclusion, the project served as a valuable learning experience, helping to strengthen both theoretical and practical understanding of data structures and their applications in real-world systems.

**References**

Yashavant Kanetkar, Data Structures Through C, BPB Publications

– This book was used to understand the implementation of queues, linked lists, and dynamic memory management in C.

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

– Provided in-depth knowledge of C programming constructs, functions, and pointers necessary for the development of the project.

GeeksforGeeks – https://www.geeksforgeeks.org/

– An online educational platform that offered detailed explanations and code samples related to priority queues, linked lists, and sorting logic.

TutorialsPoint – https://www.tutorialspoint.com/

– Used for reference on data structure concepts, particularly linked list traversal and queue operations.

Stack Overflow – https://stackoverflow.com

– A helpful forum to resolve programming issues and review community-tested solutions to common C programming errors.

GitHub – https://github.com/

– Referred to sample C projects and code snippets related to queues and data structure implementations for real-world applications.

Cprogramming.com – https://www.cprogramming.com/

– Used for learning best practices in C programming and understanding debugging techniques using GDB.