**Batch: B-1 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 5**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of Basic operations on queue for the assigned application using Array and Linked List- Create, Insert, Delete, Destroy |

**Objective:** To implement Basic Operations on Queue i.e. Create, Push, Pop, Destroy for the given application

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan

**Abstract**:

(Define Queue, enlist queue operations).

An Abstract Data Type (ADT) Queue is a linear data structure that follows the First-In-First-Out (FIFO) order. In a queue, elements are added (enqueued) at the rear end, and elements are removed (dequeued) from the front end. It models real-world queues, like people waiting in line at a ticket counter or cars waiting at a toll booth.

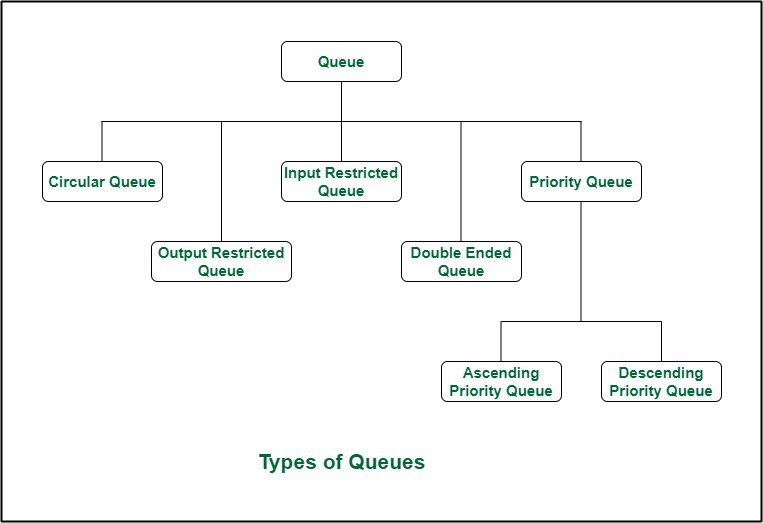
Here are the common operations associated with a Queue ADT:

1. **enqueue(item)**: Adds an element (item) to the rear or end of the queue.
2. **dequeue()**: Removes and returns the element from the front or head of the queue. It reduces the size of the queue.
3. **front()**: Returns the element at the front of the queue without removing it.
4. **isEmpty()**: Checks if the queue is empty. Returns true if the queue contains no elements; otherwise, returns false.
5. **size()**: Returns the number of elements currently in the queue.
6. **clear()**: Removes all elements from the queue, making it empty.

**List 5 Real Life applications of Queue:**

1. Print Job Management: Printers in offices or public places often have queues of print jobs waiting to be processed. The first job in is the first to be printed (FIFO). Users submit their print jobs, and the printer processes them in the order they were received.
2. Call Center Systems: In a call center, customer service representatives handle incoming calls. Calls are placed in a queue, and the next available representative takes the call from the front of the queue. This ensures fairness and efficiency in handling customer inquiries.
3. Breadth-First Search (BFS): In graph theory and algorithms, BFS uses a queue to explore nodes or vertices in a graph level by level. It's used in various applications, including social network analysis, web crawling, and shortest path finding.
4. Task Scheduling in Operating Systems: Operating systems often use queues to manage processes or tasks. The ready queue holds processes waiting to execute, and the CPU scheduler selects the next process from the front of the queue for execution.
5. Order Processing in E-commerce: When customers place orders online, those orders are often placed in a queue for processing. The orders are processed one by one in the order they were received. Queues help ensure that orders are handled fairly and efficiently.

**Define and explain various types of queue with suitable diagram and their application(s):**



1. **Linear Queue (Simple Queue):**
   * **Definition: A linear queue is a basic type of queue in which elements are added at the rear (enqueue) and removed from the front (dequeue).**
   * **Application: Linear queues are used in scenarios where tasks or processes are processed in a strict first-come, first-served (FIFO) order. For example, in print job management, CPU scheduling, and call center systems.**
2. **Circular Queue:**
   * **Definition: A circular queue is an extension of the linear queue where the front and rear wrap around to the beginning when they reach the end.**
   * **Application: Circular queues are suitable for scenarios where the queue needs to be managed efficiently in a circular or ring-like fashion. Examples include memory management in computer systems and task scheduling.**
3. **Priority Queue:**
   * **Definition: A priority queue assigns a priority to each element and dequeues elements based on their priority, not their arrival order.**
   * **Application: Priority queues are used in scenarios where tasks or items have different levels of urgency or importance. Examples include scheduling tasks in an operating system and managing patients in a hospital's emergency room.**
4. **Double-Ended Queue (Deque):**
   * **Definition: A double-ended queue is a queue where elements can be added or removed from both the front and the rear.**
   * **Application: Deques are versatile and can be used in applications that require insertion and deletion at both ends. For example, implementing a stack, managing a sliding window in data streaming, and parsing expressions.**
5. **Blocking Queue:**
   * **Definition: A blocking queue is a queue that blocks the enqueue or dequeue operation if the queue is full or empty, respectively, until the condition changes.**
   * **Application: Blocking queues are used in multithreading and concurrent programming to synchronize and coordinate tasks among threads. They ensure that threads wait when necessary and continue when conditions allow.**
6. **Priority Deque:**
   * **Definition: A priority deque combines the features of a double-ended queue and a priority queue, allowing elements to be inserted or removed from both ends while maintaining priority order.**
   * **Application: Priority deques are useful in scenarios that require a combination of FIFO and priority-based processing. Examples include real-time systems and event-driven programming.**
7. **Concurrent Queue:**
   * **Definition: A concurrent queue is designed to be thread-safe, allowing multiple threads to enqueue and dequeue elements concurrently without data corruption.**
   * **Application: Concurrent queues are essential in multithreaded and parallel processing applications, such as parallel computing, where data sharing between threads must be synchronized to prevent race conditions.**

**Queue ADT:**

class Queue:

def \_\_init\_\_(self):

self.items = []

def enqueue(self, item):

self.items.append(item)

def dequeue(self):

if not self.isEmpty():

return self.items.pop(0)

def front(self):

if not self.isEmpty():

return self.items[0]

def isEmpty(self):

return len(self.items) == 0

def size(self):

return len(self.items)

**Algorithm for Queue operations using array/Linked list : (Write only the algorithm for assigned type)**

1. Structure Definition:
   * Define a structure named Patient containing fields for patient name, priority, arrival time, and a pointer to the next patient.
2. Create Patient Function (createPatient):
   * Input: Name, priority, and arrival time of a patient.
   * Output: A pointer to a newly created Patient structure.
   * Allocate memory for a new Patient structure.
   * Set the name, priority, arrival time, and next pointer for the new patient.
   * Return the pointer to the new patient.
3. Insert Patient Function (insertPatient):
   * Input: A pointer to the head of the patient queue, name, priority, and arrival time of a patient.
   * Output: Modifies the patient queue by inserting the new patient.
   * Create a new patient using the createPatient function.
   * If the queue is empty or the new patient has higher priority or arrives earlier than the current head of the queue:
     + Make the new patient the new head of the queue.
   * Otherwise, traverse the queue until the correct position for the new patient is found based on priority and arrival time:
     + Insert the new patient into the queue before the patient with lower priority or later arrival time.
4. Print Treatment Order Function (printTreatmentOrder):
   * Input: A pointer to the head of the patient queue.
   * Output: Prints the treatment order of patients.
   * Traverse the patient queue:
     + Print the name, priority, and arrival time of each patient.
5. Main Function (main):
   * Initialize a pointer to the head of the patient queue as NULL and an integer n to store the number of patients.
   * Prompt the user to input the number of patients (n).
   * For each patient (loop i from 0 to n-1):
     + Prompt the user to input the patient's name, priority, and arrival time.
     + Insert the patient into the queue using the insertPatient function.
   * Print the treatment order of patients using the printTreatmentOrder function.
   * Return 0 to indicate successful execution.

**Implementation Details:**

1. **Mention the application assigned to you and explain how you implemented the solution using the assigned type of Queue.**
2. Queue Representation: The patient queue is represented as a linked list. Each patient is a node in the linked list, and the order in which they are treated is determined by their priority and arrival time.
3. Enqueue Operation (Insertion): The insertPatient function is responsible for enqueuing (inserting) patients into the queue. Patients are inserted into the queue based on their priority and arrival time.
4. Dequeue Operation (Treatment): While the code doesn't explicitly include a dequeue operation, the treatment order is effectively the same as dequeueing from the front of the queue. Patients at the front of the queue (those with the highest priority and earliest arrival time) are treated first. The printTreatmentOrder function prints the treatment order, effectively displaying the patients in the order they are dequeued for treatment.
5. Queue Initialization: The main function initializes the queue by creating a pointer to the head of the queue and setting it to NULL. This represents an empty queue at the beginning.
6. Insertion: Patients are inserted into the queue based on their priority and arrival time. The insertPatient function inserts each patient into the queue in the correct order, ensuring that patients with higher priority and earlier arrival times are placed at the front of the queue.
7. Printing: The printTreatmentOrder function traverses the queue and prints the patients in the order they are treated. This order effectively represents the dequeue operation, where patients at the front of the queue are treated first.

**Program source code:**

#include <stdio.h>

#include <stdlib.h>

struct Patient {

char name[50];

int priority;

int arrival;

struct Patient\* next;

};

struct Patient\* createPatient(char name[], int priority, int arrival) {

struct Patient\* newPatient = (struct Patient\*)malloc(sizeof(struct Patient));

strcpy(newPatient->name, name);

newPatient->priority = priority;

newPatient->arrival = arrival;

newPatient->next = NULL;

return newPatient;

}

void insertPatient(struct Patient\*\* head, char name[], int priority, int arrival) {

struct Patient\* newPatient = createPatient(name, priority, arrival);

if (\*head == NULL || priority < (\*head)->priority || (priority == (\*head)->priority && arrival < (\*head)->arrival)) {

newPatient->next = \*head;

\*head = newPatient;

} else {

struct Patient\* curr = \*head;

while (curr->next != NULL &&

(priority > curr->next->priority || (priority == curr->next->priority && arrival >= curr->next->arrival))) {

curr = curr->next;

}

newPatient->next = curr->next;

curr->next = newPatient;

}

}

void printTreatmentOrder(struct Patient\* head) {

printf("Treatment Order:\n");

while (head != NULL) {

printf("Name: %s, Priority: %d, Arrival: %d\n", head->name, head->priority, head->arrival);

head = head->next;

}

}

int main() {

struct Patient\* head = NULL;

int n;

printf("Enter the number of patients: ");

scanf("%d", &n);

for (int i = 0; i < n; i++) {

char name[50];

int priority, arrival;

printf("Enter patient name: ");

scanf("%s", name);

printf("Enter patient priority (1 for highest, 2 for second highest, and so on): ");

scanf("%d", &priority);

printf("Enter arrival time (in 24-hour clock format): ");

scanf("%d", &arrival);

insertPatient(&head, name, priority, arrival);

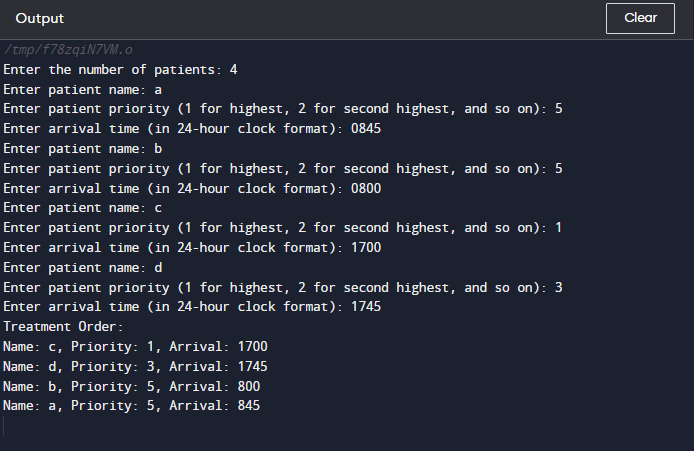
}

printTreatmentOrder(head);

return 0;

}

**Output Screenshots:**



**Applications of Queue in computer science:**

1. Job Scheduling: In operating systems, queues are used to manage processes in a ready queue, ensuring that the CPU executes processes based on their priorities and arrival times.
2. Print Spooling: Queues are used to manage print jobs in a print spooler. Print jobs are placed in a queue and printed in the order they are received.
3. Breadth-First Search (BFS): BFS traversal of graphs and trees is implemented using queues. Nodes are processed in the order they are dequeued from the queue.
4. Task Management: In multitasking operating systems, queues are used to manage tasks and threads, determining the order in which tasks are executed.
5. Buffering: Queues are used in buffering data between two processes or components. For example, a data buffer can be implemented as a queue to manage data flow between a producer and a consumer.
6. Call Center Systems: Queues are used in call centers to manage incoming calls. Calls are placed in a queue and directed to available agents in the order they are received.
7. Web Server Management: Web servers use queues to manage incoming requests. Requests are processed in the order they are received, ensuring fair access to server resources.
8. Print Queue Management: In networked printing environments, print queues are used to manage print jobs from multiple users and devices.
9. Task Synchronization: Queues are used for synchronization between threads or processes. They can be used to implement producer-consumer scenarios or other synchronization patterns.
10. Data Structures: Queues serve as building blocks for more complex data structures like priority queues and double-ended queues (dequeues).
11. Simulation: Queues are used in simulations to model real-world scenarios, such as waiting lines in banks, traffic systems, and manufacturing processes.
12. Task Queuing in Distributed Systems: Distributed systems often use queues to distribute tasks or messages among nodes for load balancing or fault tolerance.
13. Event Handling: Event-driven programs use queues to manage events and event handlers. Events are placed in a queue and processed sequentially.
14. Resource Management: Queues are used to manage access to shared resources, ensuring that resources are allocated fairly and efficiently.
15. Data Streaming: Queues can be used in streaming applications to buffer and process data as it arrives, ensuring smooth data flow.

**Conclusion:-**

We implemented priority queue using linked list through given application.