**Batch: B3 Roll No.: 121**

**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:**

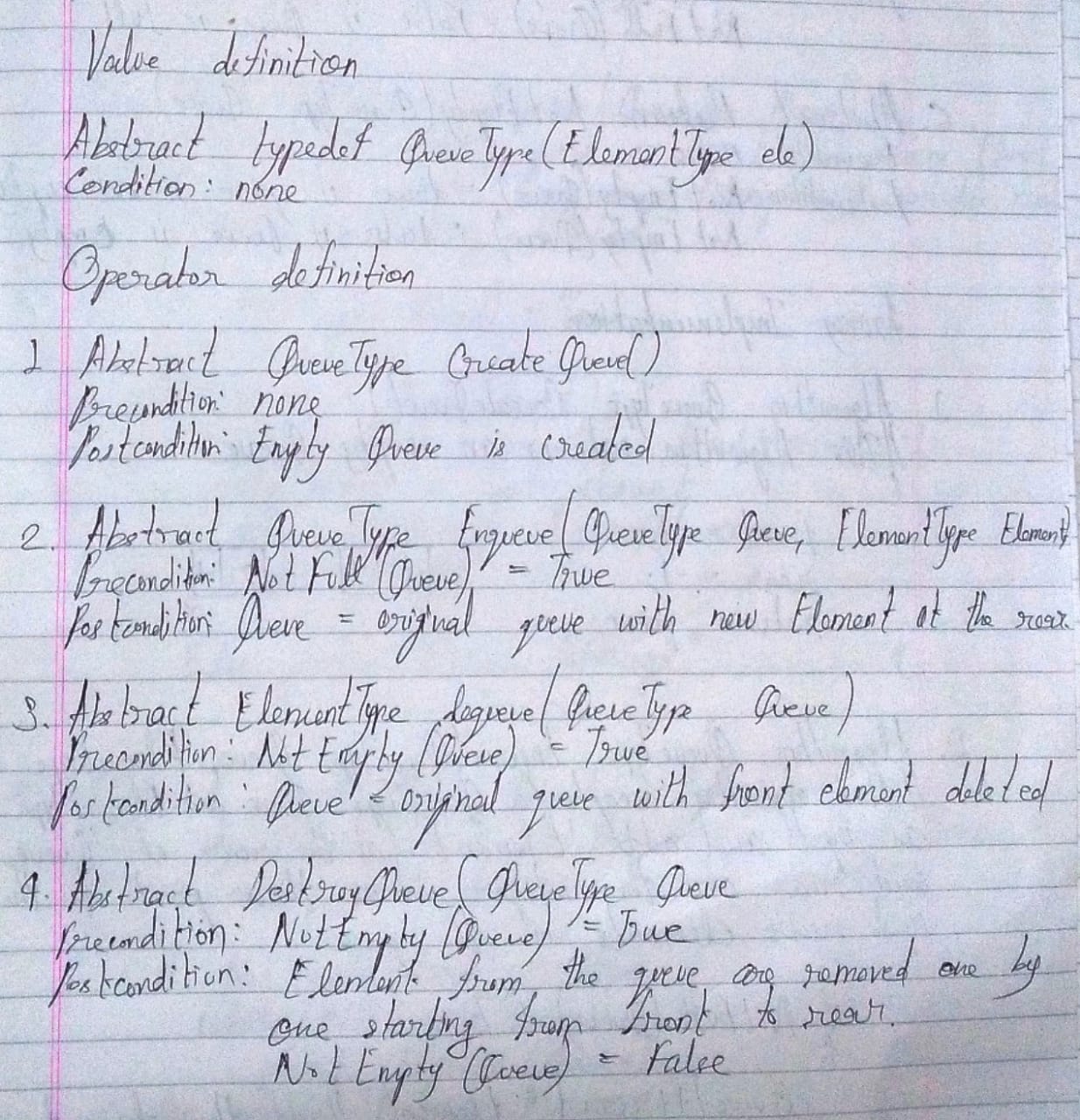
1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. *Data Structures Using C -* Aaron M. Tenenbaum, Yedidyah Langsam, and Moshe J. Augenstein

**Abstract**:

(Define Queue, enlist queue operations).

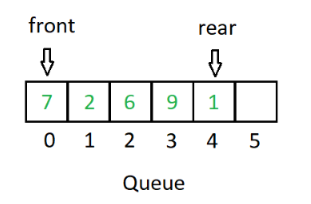
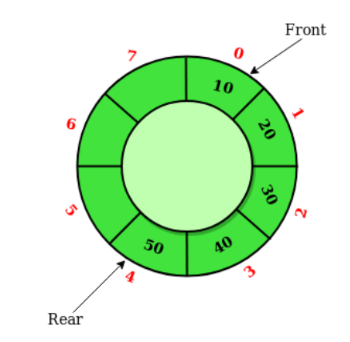
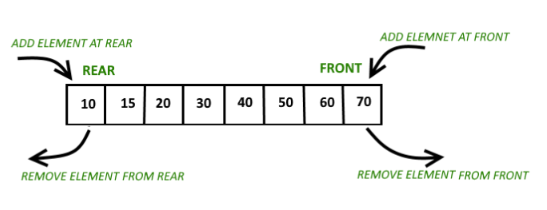
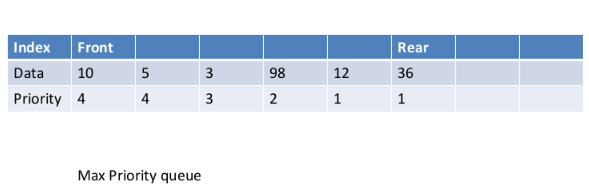


**List 5 Real Life applications of Queue:**

1. **People waiting for a bus**
2. **Cashier line in a store**
3. **A car wash line**
4. **One way exits**

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

**Ans.**

1. Queue:
   1. It is an ordered collection of homogenous elements which are added at the rear of queue and removed from the front of queue.
2. Simple Queue:
   1. In a Simple Queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First In First Out) Rule.
   2. Applications include CPU Scheduling and Disk Scheduling, handling of interrupts in real-time systems, etc.
   3. ****
3. Circular Queue:
   1. In a Circular Queue, the last element points to the first element making a circular link. The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full and the first position is empty, we can insert an element in the front position. This is not possible in a simple queue.
   2. Applications include CPU scheduling, Memory Management and Traffic Management.
   3. ****
4. Doubly Ended Queue (Dequeue)
   1. In a Doubly Ended Queue, insertion and removal of elements can be performed from both front as well as rear ends. Thus, it does not follow the FIFO Rule.
   2. Applications include undo operation on software, to store history in browsers, and for implementing both stacks and queues.
   3. ****
5. Priority Queue
   1. A Priority Queue is a special type of Queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in queue. Insertion occurs based on the arrival of values and removal based on priority.
   2. They are of two types – Min Priority Queue and Max Priority Queue. In Min Priority Queue, the element with the lowest integer priority is removed first. In Max Priority Queue, the element with the highest integer priority is removed first.
   3. ****
   4. Applications include implementation of stack, load balancing and interrupt handling in an operating system, data compression in Huffman code.

**Queue ADT:**

#include<stdio.h>

#include<stdlib.h>

struct qnode

{

int qdata;

struct qnode \*next;

};

struct qnode \*front, \*rear;

int max\_cap, cur\_cap = 0;

void create();

void enqueue();

void dequeue();

void display();

void main()

{

int ch;

printf("\nQUEUE\tIMPLEMENTATION\tIN\tC");

printf("\nEnter the maximum capacity of the queue: ");

scanf("%d", &max\_cap);

printf("\nDuring the start, enter option '1' to create the queue.\nHenceforth during the operation, don't press '1' again.");

do

{

printf("\nEnter\n'1' to create a queue.\n'2' to insert an element at the rear.\n'3' to delete an element from the front.\n'4' to display the queue in FIFO order.\n'5' to exit.\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

case 2:

enqueue();

break;

case 3:

dequeue();

break;

case 4:

display();

break;

case 5:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default:

printf("\nPlease enter '1', '2', '3', '4' or '5' only and try again!");

}

}while(ch!=5);

}

void create()

{

front = NULL;

rear = NULL;

}

void enqueue()

{

struct qnode \*temp;

temp = (struct qnode\*)malloc(sizeof(struct qnode));

printf("\nEnter data for the new node: ");

scanf("%d", &temp -> qdata);

temp -> next = NULL;

if(cur\_cap == max\_cap)

{

printf("\nQueue Overflow");

}

else

{

if(front == NULL)

{

front = temp;

rear = temp;

front -> next = NULL;

rear -> next = NULL;

}

else

{

rear -> next = temp;

rear = temp;

rear -> next = NULL;

}

cur\_cap++;

}

}

void dequeue()

{

struct qnode \*ptr;

if(front == NULL)

{

printf("\nQueue Underflow!");

}

else

{

ptr = front;

front = front -> next;

printf("\nThe deleted element is %d.", ptr->qdata);

free(ptr);

cur\_cap--;

}

}

void display()

{

struct qnode \*ptr;

if(front == NULL)

{

printf("\nQueue Underflow!");

}

else

{

ptr = front;

while(ptr != NULL)

{

printf("\t%d", ptr->qdata);

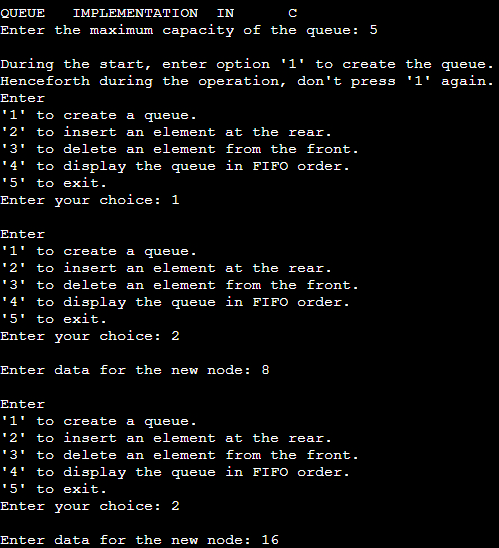
ptr = ptr -> next;

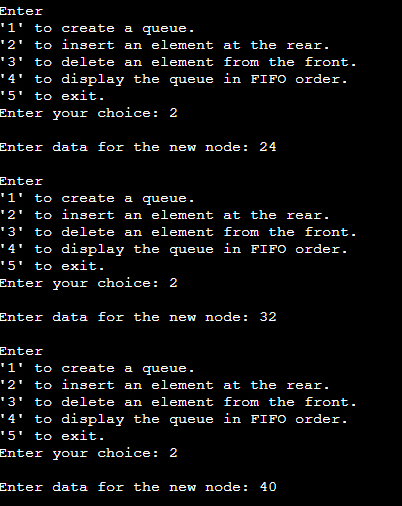
}

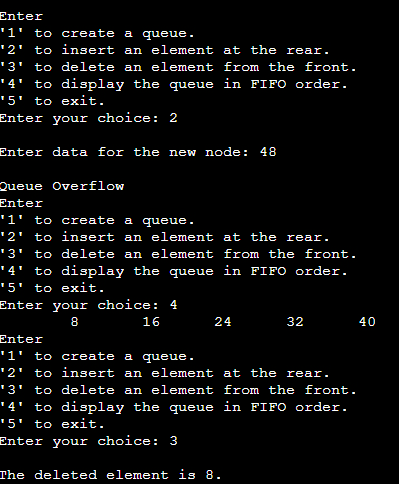
}

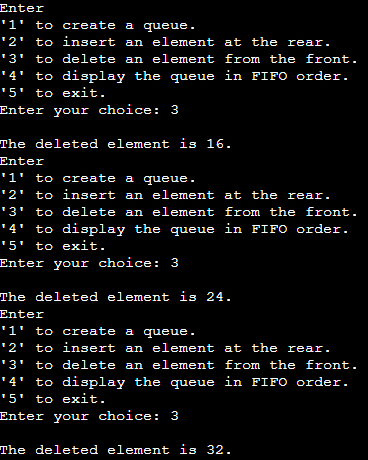
}

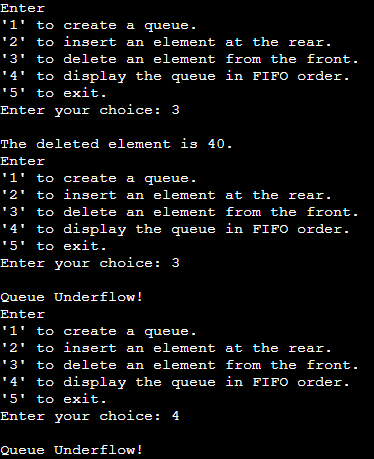
Output:

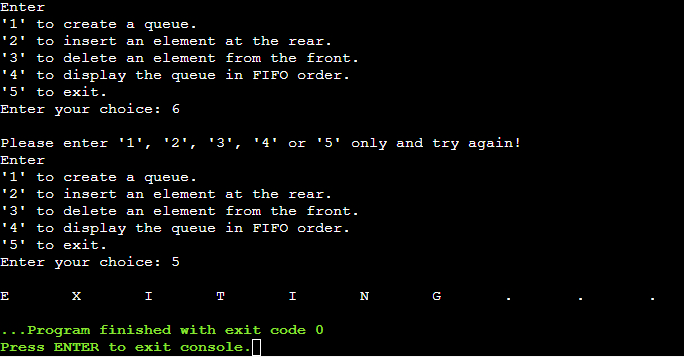




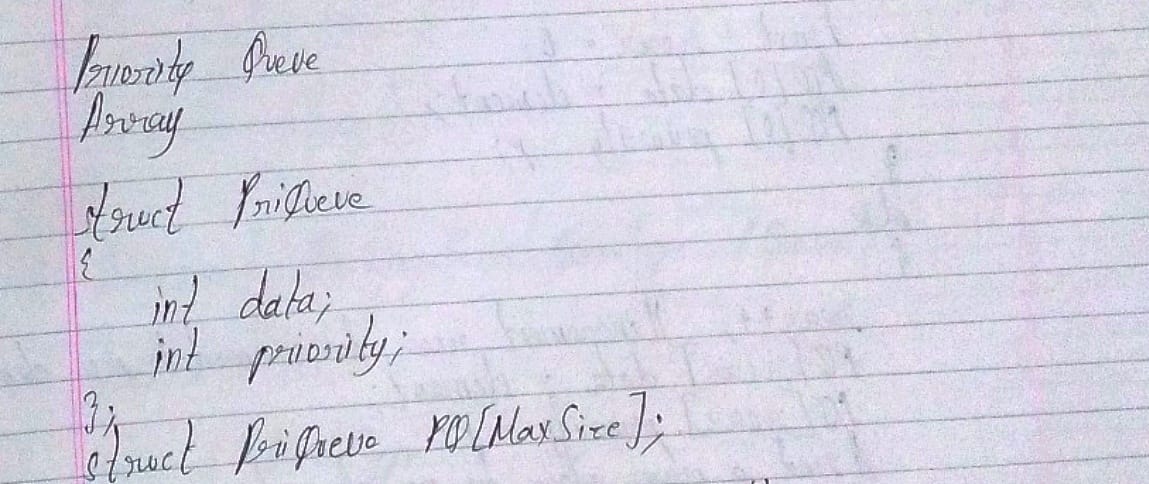


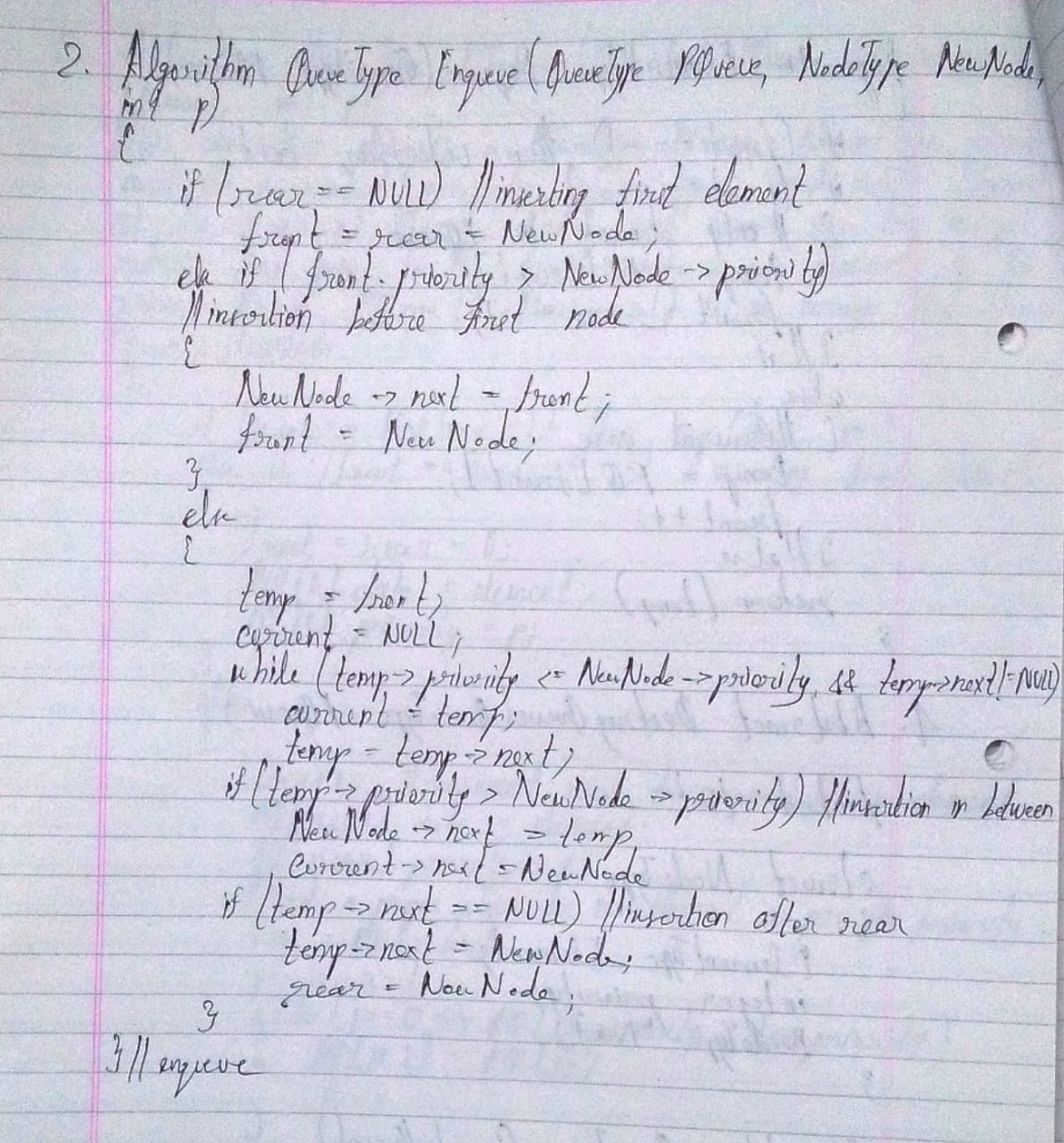
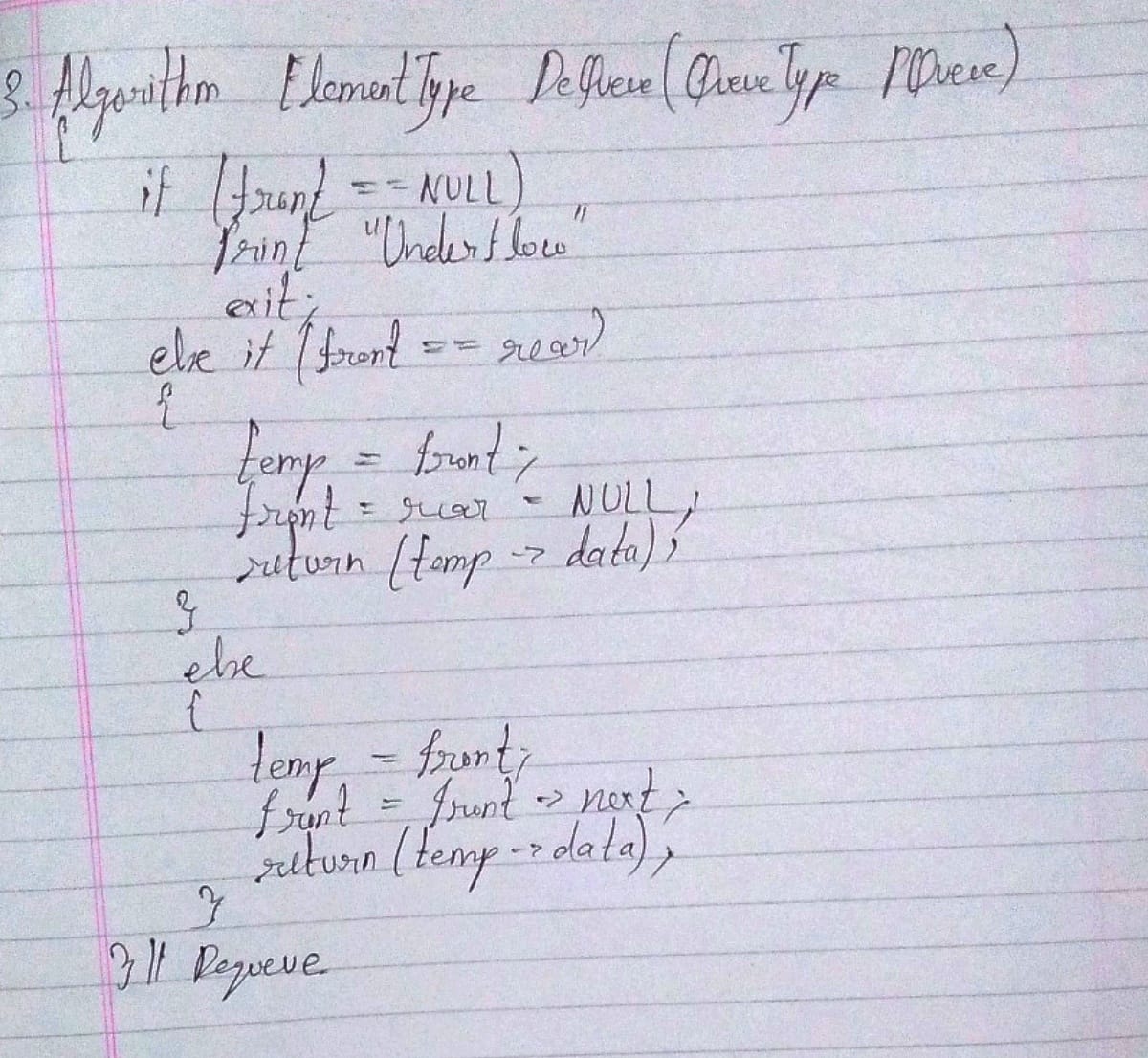






**Algorithm for Queue operations using array/Linked list : (Write only the algorithm for assigned type)**

****

****

**Implementation Details:**

1. **Mention the application assigned to you and explain how you implemented the solution using the assigned type of Queue.**

**Ans.** The application assigned is OS FIFO Queue for processes considering some scenario.

In this attempt, the first step was to get the code for a general Priority Queue. Once that was done, that code was modified by changing the integer data parameter of the node to a character array parameter, so that the data contained inside the nodes of the Priority Queue would not be integers, but would be character arrays which describes the OS process (say, “BOOT UP”, “MOVE”, etc.). Then the user enters many such process names while execution and the system assigns priority to those processes based on certain rules (like, “BOOT UP” instruction would have greater priority than “DEVICE DETECT” instruction – the user would be provided a library of instructions out of which he can choose any one and enter into the console).

**Program source code:**

#include<stdio.h>

#include<malloc.h>

#include<string.h>

#include<stdlib.h>

struct node

{

int priority;

char process\_name[50];

struct node \*next;

};

struct node \*front = NULL;

void insert();

void Delete();

void display();

void main()

{

int choice;

printf("\nLibrary of Process States for OS:\nState Name\tPriority\nReady\t\t1\nWaiting\t\t2\nNew\t\t3\nRunning\t\t4\nTerminated\t5\n");

do

{

printf("\nEnter: '1' to insert a process.\n'2' to delete a process.\n'3' to display the sequence of processes.\n'4' to exit from the program.\nEnter your choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

Delete();

break;

case 3:

display();

break;

case 4:

printf("\nE\tX\tI\tT\tI\tN\tG\t.\t.\t.");

break;

default :

printf("\nWrong choice");

}

}while(choice!=4);

}

void insert()

{

struct node \*temp,\*ptr;

char process\_state[50];

temp = (struct node \*)malloc(sizeof(struct node));

printf("Enter the name of the process to be added in the queue : ");

scanf("%s", temp->process\_name);

printf("Enter its state : ");

scanf("%s", process\_state);

fflush(stdin);

if(strcmp(process\_state, "Ready") == 0){

temp -> priority = 1;

fflush(stdin);

}

else if(strcmp(process\_state, "Waiting") == 0){

temp -> priority = 2;

fflush(stdin);

}

else if(strcmp(process\_state, "New") == 0){

temp -> priority = 3;

fflush(stdin);

}

else if(strcmp(process\_state, "Running") == 0){

temp -> priority = 4;

fflush(stdin);

}

else if(strcmp(process\_state, "Terminated") == 0){

temp -> priority = 5;

fflush(stdin);

}

else

{

printf("\nAn unidentified process state has been entered. Please try again.");

exit(0);

}

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

{

temp->next = front;

front = temp;

}

else

{

ptr = front;

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

ptr=ptr->next;

temp->next = ptr->next;

ptr->next = temp;

}

}

void Delete()

{

struct node \*temp;

if(front == NULL)

printf("Queue Underflow\n");

else

{

temp = front;

printf("Deleted process is %s\n",temp->process\_name);

front = front->next;

free(temp);

}

}

void display()

{

struct node \*ptr;

ptr = front;

if(front == NULL)

printf("Queue is empty\n");

else

{

printf("Queue is :\n");

printf("Priority\tItem\n");

while(ptr != NULL)

{

printf("%d\t\t%s\n",ptr->priority,ptr->process\_name);

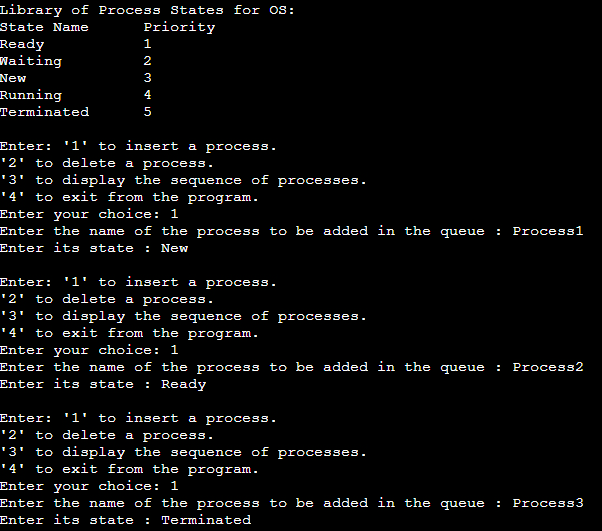
ptr = ptr->next;

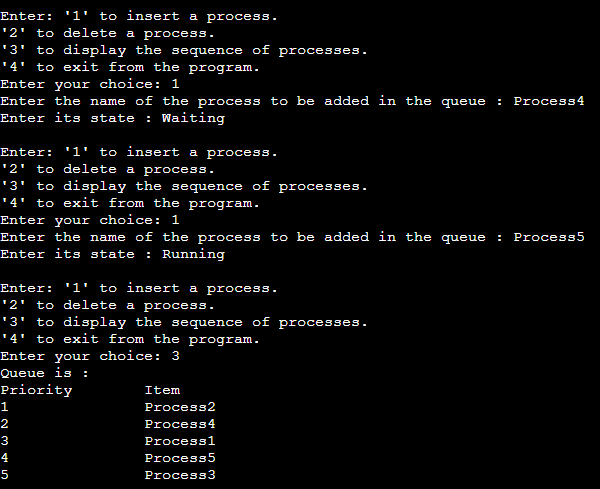
}

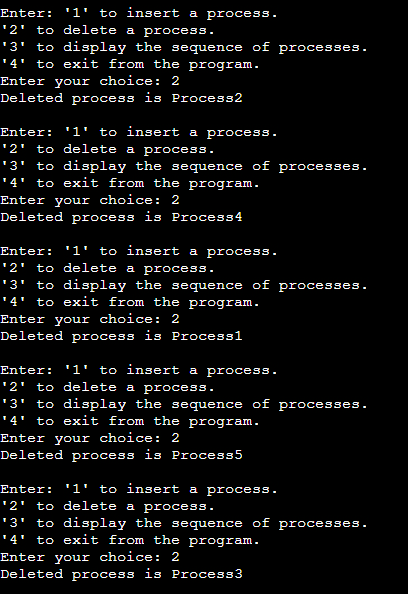
}

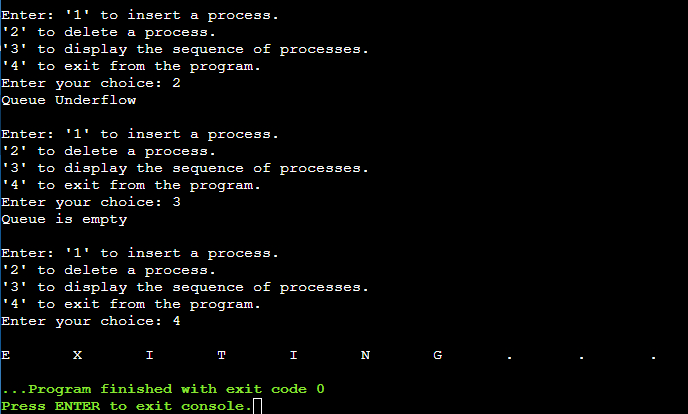
}

**Output Screenshots:**

****

****

****

****

**Applications of Queue in computer science:**

1. Managing requests on a single shared resource such as CPU scheduling and disk scheduling.
2. Handling hardware or real-time system interrupts.
3. Handling website traffic
4. Routers and switches in networking
5. Maintaining the playlist in media players.

**Conclusion:-**

Thus, in this experiment, the concept of Queue and different types of Queue has been learnt. The basic operations of Queue have been implemented in C language. Further, one assigned application of Queue was also implemented in C language. Queue is one of the most important data structures and has a wide range of applications encompassing almost all fields of computer software.