**Lab Exercise -7: Queue Data Structure Implementation using Array and**

**Linked Lists and Applications of Queue ADT**

**Experiment 1: Linear Queue using Array**

**Objectives:**

**• Implement a simple linear queue.**

**• Perform enqueue, dequeue, peek, isEmpty, and isFull operations.**

**Tasks:**

**1. Define an array queue[MAX] and variables front and rear.**

**2. Implement enqueue operation with overflow check.**

**3. Implement dequeue operation with underflow check.**

**4. Implement peek/front operation.**

**5. Implement display function to show all elements.**

#include <stdio.h>

#define max 100

int queue[max];

int front = -1, rear = -1;

int isFull();

int isEmpty();

void Enqueue(int);

void Dequeue();

void Peek();

void Display();

int main()

{

int choice = 1, value;

while (choice != 0)

{

printf("\n\n\*\*\*\*\* MAIN MENU || QUEUE(array) || (C) \*\*\*\*\*");

printf("\n1. Insert an Element");

printf("\n2. Delete an Element");

printf("\n3. Peek in the Queue");

printf("\n4. Display the Queue");

printf("\n0. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter Value to Insert : ");

scanf("%d", &value);

Enqueue(value);

break;

case 2:

Dequeue();

break;

case 3:

Peek();

break;

case 4:

Display();

break;

case 0:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !!\nTry again !!");

break;

}

}

return 0;

}

int isFull()

{

if (rear == max - 1)

{

return 1;

}

return 0;

}

int isEmpty()

{

if (front == -1 || front > rear)

{

return 1;

}

return 0;

}

void Enqueue(int value)

{

if (isFull())

{

printf("\nQUEUE OVERFLOW");

}

else if (front == -1 && rear == -1)

{

front = rear = 0;

queue[rear] = value;

printf("\nFirst Element Inserted in the Queue");

}

else

{

rear++;

queue[rear] = value;

printf("\nElement Inserted in the Queue");

}

}

void Dequeue()

{

if (isEmpty())

{

printf("\nQUEUE UNDERFLOW");

}

else

{

printf("Element with Value %d deleted from the Queue", queue[front]);

front++;

if (front > rear)

{

front = rear = -1;

}

}

}

void Peek()

{

if (isEmpty())

{

printf("\nQUEUE UNDERFLOW");

}

else

{

printf("\nFirst Element of the Queue is : %d", queue[front]);

}

}

void Display()

{

printf("\nQUEUE :\n");

if (isEmpty())

{

printf("\nEMPTY QUEUE");

}

else

{

for (int i = front; i <= rear; i++)

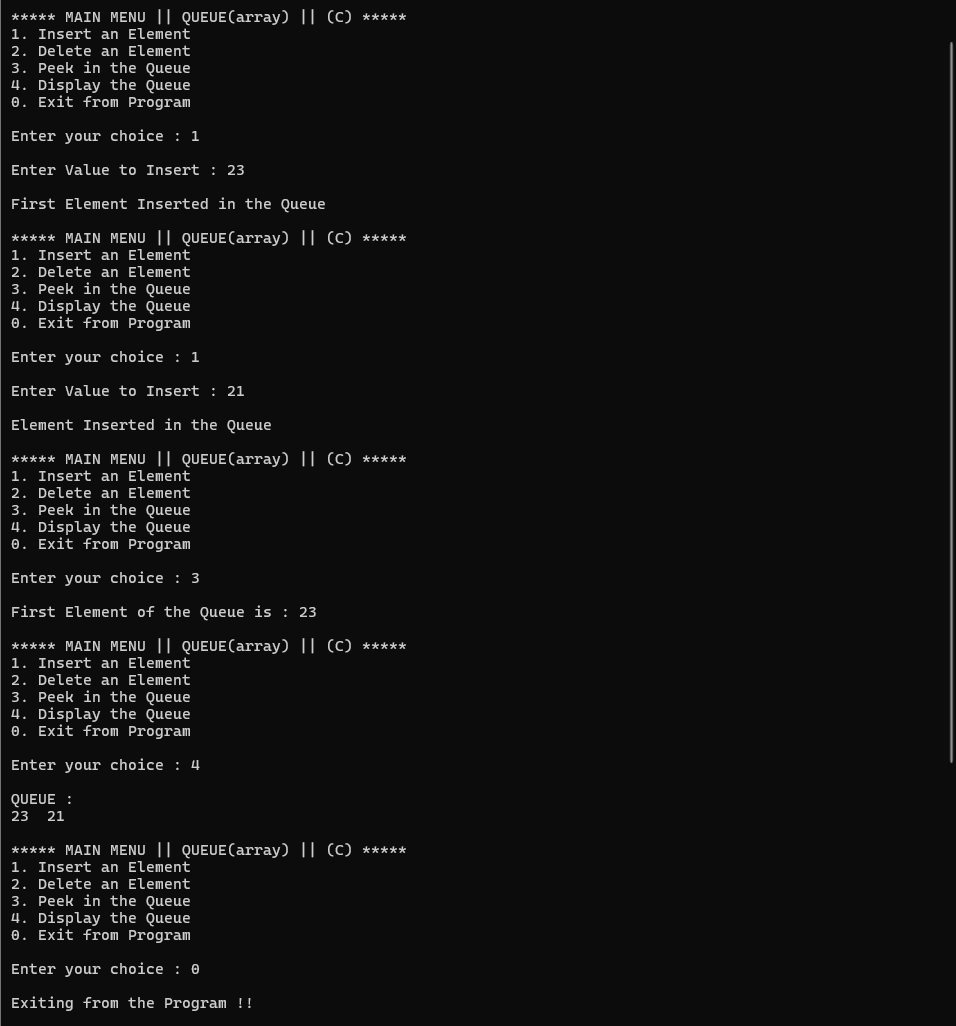
{

printf("%d ", queue[i]);

}

}

}

****

**Experiment 2: Queue using Linked List**

**Objectives:**

**• Implement a queue using singly linked list.**

**• Understand dynamic memory allocation and pointer management.**

**Tasks:**

**1. Create a node structure with data and next.**

**2. Implement enqueue at the rear.**

**3. Implement dequeue at the front.**

**4. Implement peek operation.**

**5. Display queue elements.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct queue

{

struct node \*front;

struct node \*rear;

};

struct queue \*q;

void CreateQueue();

void Enqueue(int);

void Dequeue();

void Peek();

void Display();

int main()

{

int choice = 1, value;

CreateQueue();

while (choice != 0)

{

printf("\n\n\*\*\*\*\* MAIN MENU || QUEUE(LL) || (C) \*\*\*\*\*");

printf("\n1. Insert an Element");

printf("\n2. Delete an Element");

printf("\n3. Peek in the Queue");

printf("\n4. Display the Queue");

printf("\n0. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter Value to Insert : ");

scanf("%d", &value);

Enqueue(value);

break;

case 2:

Dequeue();

break;

case 3:

Peek();

break;

case 4:

Display();

break;

case 0:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !!\nTry again !!");

break;

}

}

return 0;

}

void CreateQueue()

{

q = (struct queue \*)malloc(sizeof(struct queue));

q->front = NULL;

q->rear = NULL;

}

void Enqueue(int value)

{

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

ptr->data = value;

if (q->front == NULL)

{

q->front = ptr;

q->rear = ptr;

q->front->next = q->rear->next = NULL;

}

else

{

q->rear->next = ptr;

q->rear = ptr;

q->rear->next = NULL;

}

printf("Element added to the Queue");

}

void Dequeue()

{

struct node \*ptr = q->front;

if (q->front == NULL)

{

printf("\nQUEUE UNDERFLOW");

}

else

{

q->front = q->front->next;

printf("Element removed from the Queue");

free(ptr);

}

}

void Peek()

{

if (q->front == NULL)

{

printf("\nQUEUE UNDERFLOW");

}

else

{

printf("\nElement at front of the Queue = %d", q->front->data);

}

}

void Display()

{

struct node \*ptr = q->front;

if (ptr == NULL)

{

printf("\nQUEUE UNDERFLOW");

}

else

{

printf("\nQUEUE : \n");

while (ptr != q->rear)

{

printf("%d <- ", ptr->data);

ptr = ptr->next;

}

printf("%d ", ptr->data);

}

}

****

**Experiment 3: Circular Queue using Array**

**Objectives:**

**• Implement a circular queue to overcome the wastage of linear queue.**

**Tasks:**

**1. Define array cq[MAX], front, and rear.**

**2. Implement enqueue with circular increment (rear + 1) % MAX.**

**3. Implement dequeue with circular increment (front + 1) % MAX.**

**4. Implement display function handling circular nature.**

#include <stdio.h>

#define max 5

int queue[max];

int front = -1, rear = -1;

void Enqueue(int);

void Dequeue();

void Peek();

void Display();

int main()

{

int choice = 1, value;

while (choice != 0)

{

printf("\n\n\*\*\*\*\* MAIN MENU || CIRCULAR QUEUE(array) || (C) \*\*\*\*\*");

printf("\n1. Insert an Element");

printf("\n2. Delete an Element");

printf("\n3. Peek in the Queue");

printf("\n4. Display the Queue");

printf("\n0. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter Value to Insert : ");

scanf("%d", &value);

Enqueue(value);

break;

case 2:

Dequeue();

break;

case 3:

Peek();

break;

case 4:

Display();

break;

case 0:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !! Try again !!");

break;

}

}

return 0;

}

void Enqueue(int value)

{

if ((rear + 1) % max == front)

{

printf("\nQUEUE OVERFLOW");

return;

}

if (front == -1 && rear == -1)

{

front = rear = 0;

}

else

{

rear = (rear + 1) % max;

}

queue[rear] = value;

printf("\nElement Added to C.Queue = %d", queue[rear]);

}

void Dequeue()

{

if (front == -1 && rear == -1)

{

printf("\nQUEUE UNDERFLOW");

return;

}

printf("\nElement Removed from the C.Queue = %d", queue[front]);

if (front == rear)

{

front = rear = -1;

}

else

{

front = (front + 1) % max;

}

}

void Peek()

{

if (front == -1 && rear == -1)

{

printf("\nQUEUE UNDERFLOW");

}

else

{

printf("\nElement at the front of the C.Queue = %d", queue[front]);

}

}

void Display()

{

int i;

printf("\nQUEUE : \n");

if (front == -1 && rear == -1)

{

printf("\nQUEUE UNDERFLOW");

return;

}

for (i = front; i != rear; i = (i + 1) % max)

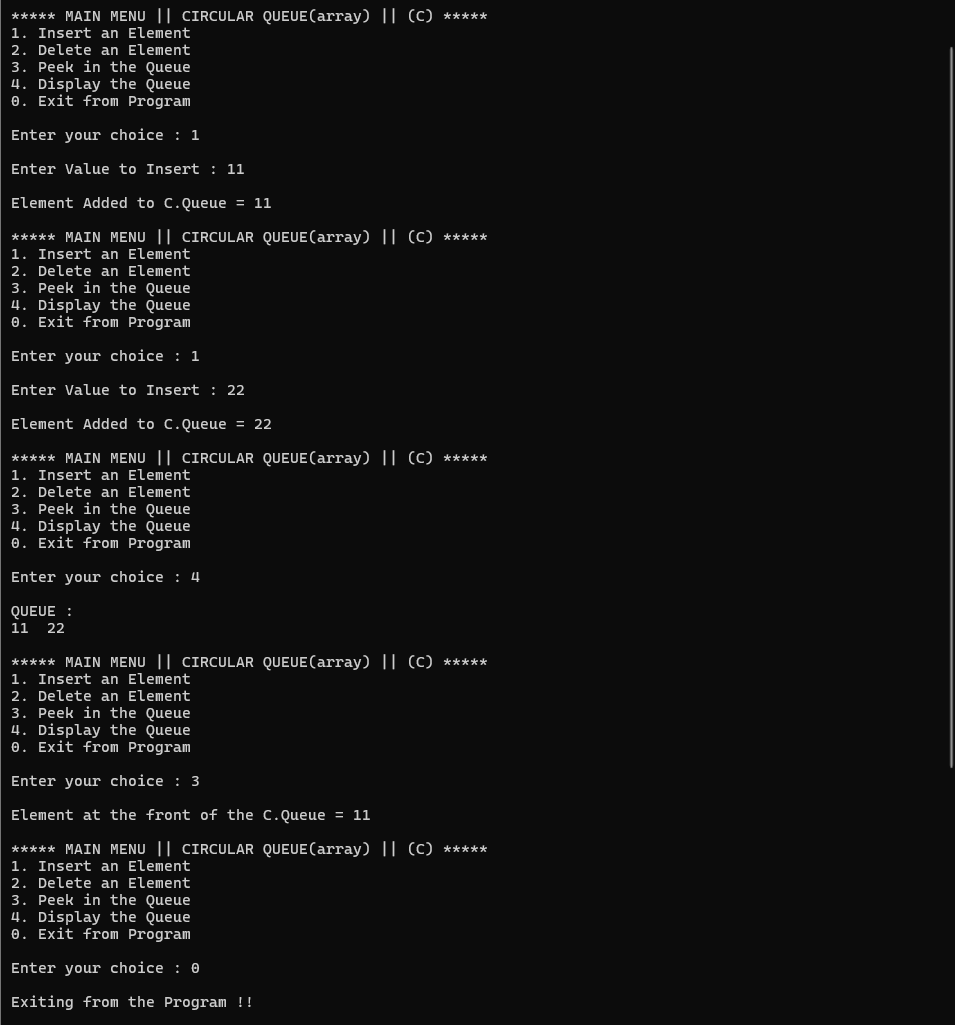
{

printf("%d ", queue[i]);

}

printf("%d ", queue[i]);

}

****

**Experiment 4: Double-Ended Queue (Deque)**

**Objectives:**

**• Implement deque using array.**

**• Insert and delete elements from both ends.**

**Tasks:**

**1. Define front and rear pointers.**

**2. Implement insertFront, insertRear, deleteFront, deleteRear.**

**3. Display queue elements after each operation.**

#include <stdio.h>

#define max 100

int deque[max];

int front = -1, rear = -1;

void EnqueueFront(int);

void DequeueFront();

void PeekFront();

void EnqueueRear(int);

void DequeueRear();

void PeekRear();

void Display();

int main()

{

int choice = 1, value;

while (choice != 0)

{

printf("\n\n\*\*\*\*\* MAIN MENU || DE-QUEUE(array) || (C) \*\*\*\*\*");

printf("\n1. Insert an Element at Front");

printf("\n2. Delete an Element from Front");

printf("\n3. Peek at Front in the Queue");

printf("\n4. Insert an Element at Rear");

printf("\n5. Delete an Element from Rear");

printf("\n6. Peek at Rear in the Queue");

printf("\n7. Display the Queue");

printf("\n0. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter Value to Insert : ");

scanf("%d", &value);

EnqueueFront(value);

break;

case 2:

DequeueFront();

break;

case 3:

PeekFront();

break;

case 4:

printf("\nEnter Value to Insert : ");

scanf("%d", &value);

EnqueueRear(value);

break;

case 5:

DequeueRear();

break;

case 6:

PeekRear();

break;

case 7:

Display();

break;

case 0:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !! Try again !!");

break;

}

}

return 0;

}

void EnqueueFront(int value)

{

if ((rear + 1) % max == front)

{

printf("\nDEQUE OVERFLOW");

return;

}

if (front == -1 && rear == -1)

{

front = rear = 0;

}

else

{

front = (front + max - 1) % max;

}

deque[front] = value;

printf("\nElement Added in DE.Queue = %d", deque[front]);

}

void DequeueFront()

{

if (front == -1 && rear == -1)

{

printf("\nDEQUE UNDERFLOW");

return;

}

printf("\nElement Removed from the DE.Queue = %d", deque[front]);

if (front == rear)

{

front = rear = -1;

}

else

{

front = (front + 1) % max;

}

}

void PeekFront()

{

if (front == -1 && rear == -1)

{

printf("\nDEQUE UNDERFLOW");

}

else

{

printf("\nElement at the front of the DE.Queue = %d", deque[front]);

}

}

void EnqueueRear(int value)

{

if ((rear + 1) % max == front)

{

printf("\nDEQUE OVERFLOW");

return;

}

if (front == -1 && rear == -1)

{

front = rear = 0;

}

else

{

rear = (rear + 1) % max;

}

deque[rear] = value;

printf("\nElement Added in DE.Queue = %d", deque[rear]);

}

void DequeueRear()

{

if (front == -1 && rear == -1)

{

printf("\nDEQUE UNDERFLOW");

return;

}

printf("\nElement Removed from the DE.Queue = %d", deque[rear]);

if (front == rear)

{

front = rear = -1;

}

else

{

rear = (rear + max - 1) % max;

}

}

void PeekRear()

{

if (front == -1 && rear == -1)

{

printf("\nDEQUE UNDERFLOW");

}

else

{

printf("\nElement at the Rear of the DE.Queue = %d", deque[rear]);

}

}

void Display()

{

int i;

printf("\nDEQUE : \n");

if (front == -1 && rear == -1)

{

printf("\nDEQUE UNDERFLOW");

return;

}

for (i = front; i != rear; i = (i + 1) % max)

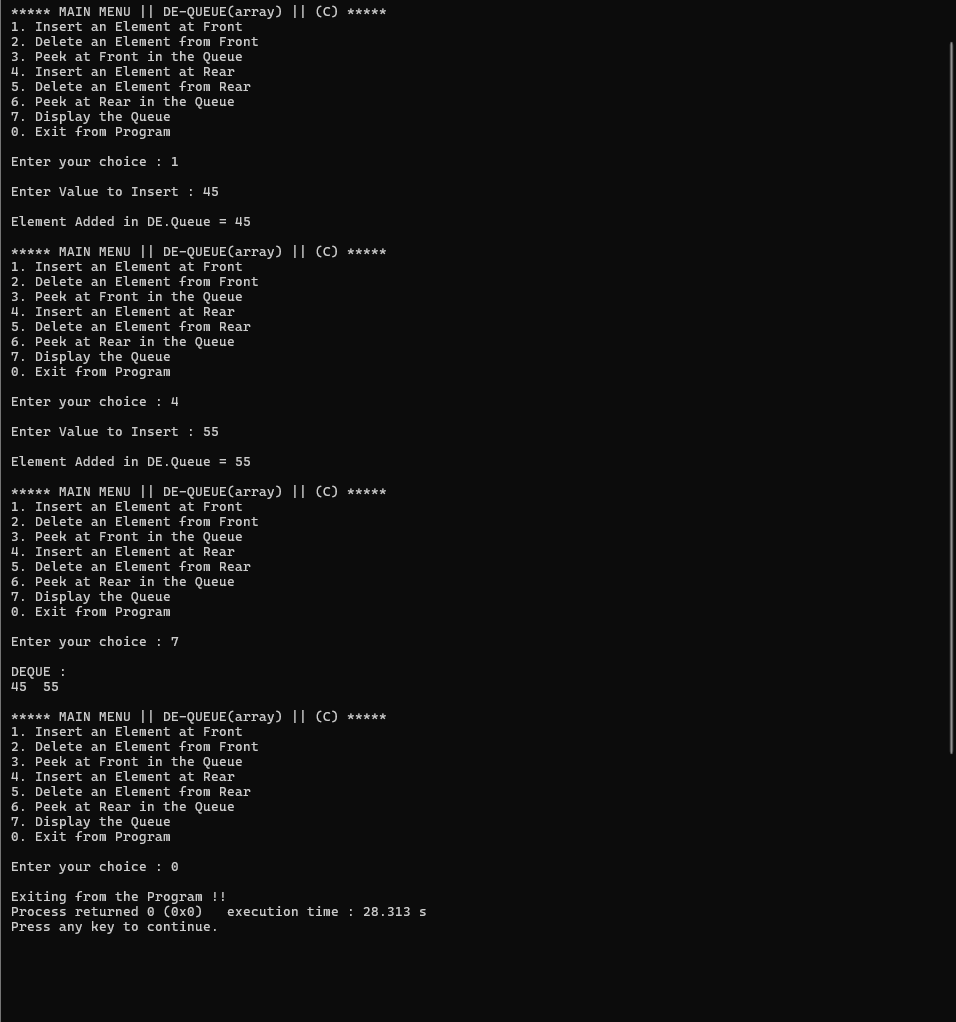
{

printf("%d ", deque[i]);

}

printf("%d ", deque[i]);

}

****

**Experiment 5: Priority Queue using Array**

**Objectives:**

**• Implement a priority queue using array.**

**• Understand priority-based element selection.**

**Tasks:**

**1. Define an array of Element {data, priority}.**

**2. Implement enqueue (unsorted array → O(1)).**

**3. Implement dequeue (find highest priority → O(n)).**

**4. Implement peek operation.**

**5. Display queue with priorities.**

#include <stdio.h>

#define max 100

struct element

{

int value;

int priority;

};

struct element pq[max];

int len = 0;

void Enqueue(int, int);

void Dequeue();

void Display();

int GetPriority();

int main()

{

int choice = 1, value, priority;

while (choice != 0)

{

printf("\n\n\*\*\*\*\* MAIN MENU || PRIORITY QUEUE(array) || (C) \*\*\*\*\*");

printf("\n1. Insert an Element");

printf("\n2. Delete an Element");

printf("\n3. Display the Queue");

printf("\n0. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("\nEnter Value to Insert : ");

scanf("%d", &value);

printf("Enter Priority for Element : ");

scanf("%d", &priority);

Enqueue(value, priority);

break;

case 2:

Dequeue();

break;

case 3:

Display();

break;

case 0:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !! Try again !!");

break;

}

}

return 0;

}

void Enqueue(int value, int priority)

{

if (len == max)

{

printf("\nPRIORITY QUEUE OVERFLOW");

return;

}

pq[len].value = value;

pq[len].priority = priority;

len++;

printf("Element Added to the Priority Queue");

}

int GetPriority()

{

int h = 0;

for (int i = 1; i < len; i++)

{

if (pq[i].priority > pq[h].priority)

{

h = i;

}

}

return h;

}

void Dequeue()

{

if (len == 0)

{

printf("\nPRIORITY QUEUE UNDERFLOW");

return;

}

int high = GetPriority();

printf("\nElement with Value: %d and Priority: %d REMOVED", pq[high].value, pq[high].priority);

for (int i = high; i < len; i++)

{

pq[i] = pq[i + 1];

}

len--;

}

void Display()

{

if (len == 0)

{

printf("\nPRIORITY QUEUE UNDERFLOW");

return;

}

printf("PRIORITY QUEUE :\n");

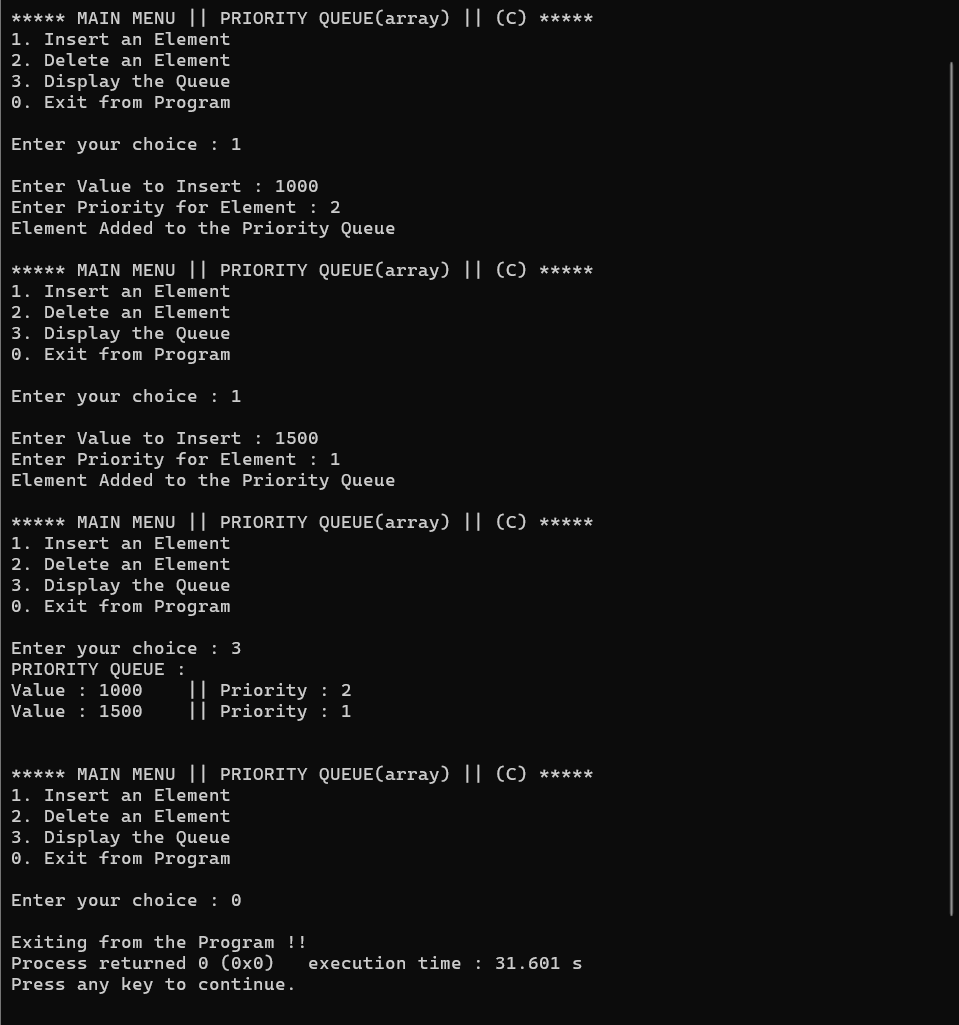
for (int i = 0; i < len; i++)

{

printf("Value : %d\t|| Priority : %d\n", pq[i].value, pq[i].priority);

}

}

****

**Experiment 6: Queue using Two Stacks**

**Objectives:**

**• Implement a queue using two stacks.**

**• Compare enqueue-costly vs dequeue-costly methods.**

**Tasks:**

**1. Define two stacks s1 and s2.**

**2. Implement enqueue-costly method: push O(n), pop O(1).**

**3. Implement dequeue-costly method: push O(1), pop O(n).**

**4. Test with multiple enqueue and dequeue operations.**

#include <stdio.h>

#define max 20

int queue[max], front1 = -1, rear1 = -1, front2 = max, rear2 = max;

void InsertA(int);

int DeleteA();

void DisplayA();

void InsertB(int);

int DeleteB();

void DisplayB();

int main()

{

int choice = 0, value, op1 = 0, op2 = 0;

while (choice != 3) // Main loop

{

printf("\n\n\*\*\*\*\*\* MAIN MENU || QUEUE || (C) \*\*\*\*\*\*");

printf("\n1. operations for 1st STACK-QUEUE");

printf("\n2. operations for 2nd STACK-QUEUE");

printf("\n3. exit from the program !");

printf("\nEnter the Choice :");

scanf("%d", &choice);

switch (choice)

{

case 1:

op1 = 0; // Reset menu choice for queue 1

while (op1 != 4)

{

printf("\n\n\*\*\*\* MENU || QUEUE 1 \*\*\*\*");

printf("\n1. Insert");

printf("\n2. Delete");

printf("\n3. Display");

printf("\n4. Exit to main menu");

printf("\nEnter the operation :");

scanf("%d", &op1);

switch (op1)

{

case 1:

printf("\nEnter Value :");

scanf("%d", &value);

InsertA(value);

break;

case 2:

DeleteA();

break;

case 3:

DisplayA();

break;

case 4:

printf("\nExiting to main menu...");

break;

default:

printf("\nTRY AGAIN");

break;

}

}

break;

case 2:

op2 = 0; // Reset menu choice for queue 2

while (op2 != 4)

{

printf("\n\n\*\*\*\* MENU || QUEUE 2 \*\*\*\*");

printf("\n1. Insert");

printf("\n2. Delete");

printf("\n3. Display");

printf("\n4. Exit to main menu");

printf("\nEnter the operation :");

scanf("%d", &op2);

switch (op2)

{

case 1:

printf("\nEnter Value :");

scanf("%d", &value);

InsertB(value);

break;

case 2:

DeleteB();

break;

case 3:

DisplayB();

break;

case 4:

printf("\nExiting to main menu...");

break;

default:

printf("\nTRY AGAIN");

break;

}

}

break;

case 3:

printf("\nExiting from the Program !!\n");

break;

default:

printf("\nInvalid choice, Enter again !!");

}

}

return 0;

}

void InsertA(int value)

{

if (rear1 == rear2 - 1)

{

printf("\nOVERFLOW");

return;

}

if (rear1 == -1 && front1 == -1)

{

rear1 = front1 = 0;

queue[rear1] = value;

}

else

{

queue[++rear1] = value;

}

printf("\nValue %d added to Queue 1", queue[rear1]);

}

int DeleteA()

{

if (front1 == -1)

{

printf("\nUNDERFLOW");

return -1;

}

printf("\nValue %d removed from Queue 1", queue[front1]);

front1++;

if (front1 > rear1)

{

front1 = rear1 = -1;

}

return 0;

}

void DisplayA()

{

int i;

if (front1 == -1)

{

printf("\nEMPTY QUEUE 1");

return;

}

printf("\nQueue 1: ");

for (i = front1; i <= rear1; i++)

{

printf("%d ", queue[i]);

}

}

void InsertB(int value)

{

if (rear2 == rear1 + 1)

{

printf("\nOVERFLOW");

return;

}

if (rear2 == max && front2 == max)

{

rear2 = front2 = max - 1;

queue[rear2] = value;

}

else

{

queue[--rear2] = value;

}

printf("\nValue %d added to Queue 2", queue[rear2]);

}

int DeleteB()

{

if (front2 == max)

{

printf("\nUNDERFLOW");

return -1;

}

printf("\nValue %d removed from Queue 2", queue[front2]);

front2--;

if (front2 < rear2)

{

front2 = rear2 = max;

}

return 0;

}

void DisplayB()

{

int i;

if (front2 == max)

{

printf("\nEMPTY QUEUE 2");

return;

}

printf("\nQueue 2: ");

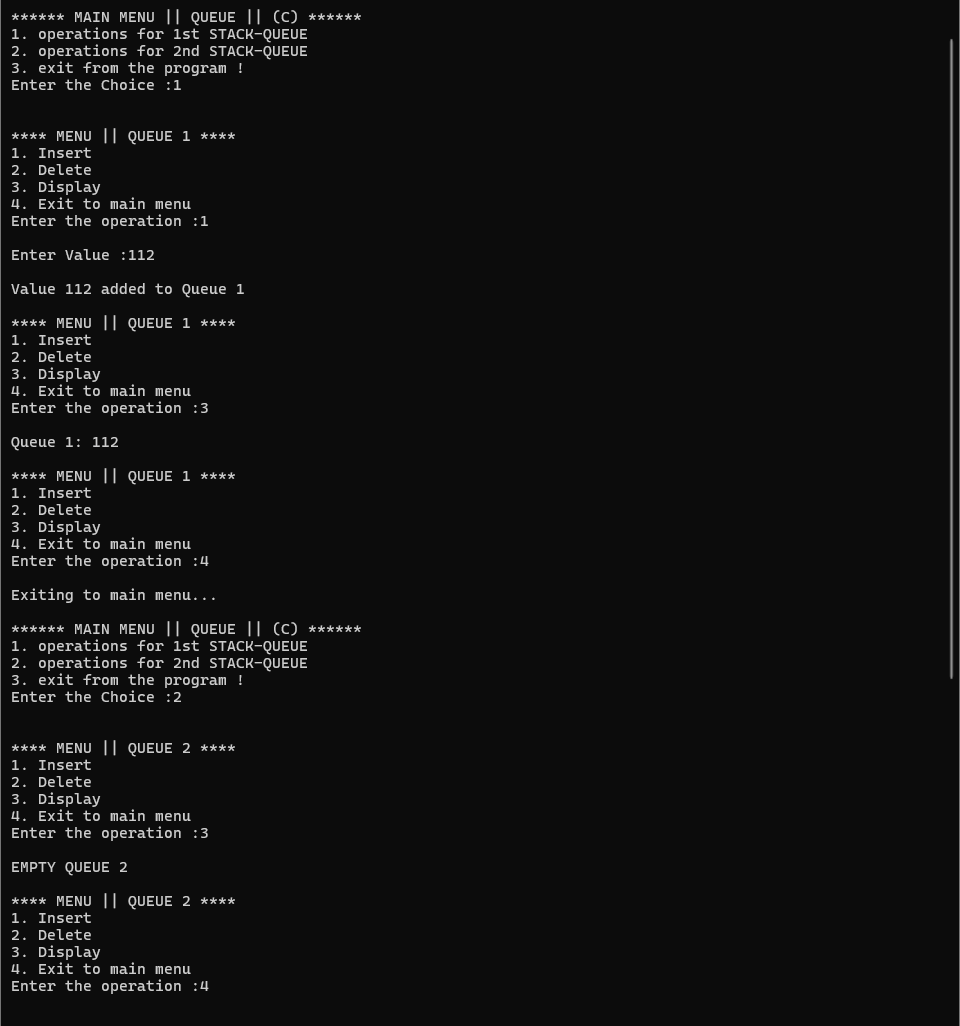
for (i = front2; i >= rear2; i--)

{

printf("%d ", queue[i]);

}

}

****

**Experiment 7: Stack using Two Queues**

**Objectives:**

**• Implement a stack using two queues.**

**• Compare push-costly vs pop-costly implementations.**

**Tasks:**

**1. Define two queues q1 and q2.**

**2. Implement push-costly method: push O(n), pop O(1).**

**3. Implement pop-costly method: push O(1), pop O(n).**

**4. Test LIFO behavior using different sequences.**

#include <stdio.h>

#include <stdlib.h>

#define max 100

int q1[max], q2[max];

int front1 = -1, rear1 = -1;

int front2 = -1, rear2 = -1;

void Enqueue1(int);

int Dequeue1();

void Enqueue2(int);

int Dequeue2();

void PushCostly(int);

void PopCostly();

void PushCheap(int);

void PopCheap();

void Display();

void ResetStack();

int main()

{

int choice = 1, value, op;

while (choice != 0)

{

printf("\n\n\*\*\*\*\* MAIN MENU || STACK using 2 QUEUES || (C) \*\*\*\*\*");

printf("\n1. Use Push-Costly Method");

printf("\n2. Use Pop-Costly Method");

printf("\n0. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

ResetStack();

switch (choice)

{

case 1:

op = 1;

while (op != 0)

{

printf("\n\n--- Push-Costly Menu ---");

printf("\n1. Push Element");

printf("\n2. Pop Element");

printf("\n3. Display Stack");

printf("\n0. Back to Main Menu");

printf("\n\nEnter your operation : ");

scanf("%d", &op);

switch (op)

{

case 1:

printf("\nEnter Value to Push : ");

scanf("%d", &value);

PushCostly(value);

break;

case 2:

PopCostly();

break;

case 3:

Display();

break;

case 0:

printf("\nReturning to main menu...");

break;

default:

printf("\nInvalid Operation !!");

break;

}

}

break;

case 2:

op = 1;

while (op != 0)

{

printf("\n\n--- Pop-Costly Menu ---");

printf("\n1. Push Element");

printf("\n2. Pop Element");

printf("\n3. Display Stack");

printf("\n0. Back to Main Menu");

printf("\n\nEnter your operation : ");

scanf("%d", &op);

switch (op)

{

case 1:

printf("\nEnter Value to Push : ");

scanf("%d", &value);

PushCheap(value);

break;

case 2:

PopCheap();

break;

case 3:

Display();

break;

case 0:

printf("\nReturning to main menu...");

break;

default:

printf("\nInvalid Operation !!");

break;

}

}

break;

case 0:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !!\nTry again !!");

break;

}

}

return 0;

}

void Enqueue1(int value)

{

if ((rear1 + 1) % max == front1)

{

printf("\nQUEUE 1 OVERFLOW");

return;

}

if (front1 == -1)

{

front1 = rear1 = 0;

}

else

{

rear1 = (rear1 + 1) % max;

}

q1[rear1] = value;

}

int Dequeue1()

{

if (front1 == -1)

{

return -1;

}

int item = q1[front1];

if (front1 == rear1)

{

front1 = rear1 = -1;

}

else

{

front1 = (front1 + 1) % max;

}

return item;

}

void Enqueue2(int value)

{

if ((rear2 + 1) % max == front2)

{

printf("\nQUEUE 2 OVERFLOW");

return;

}

if (front2 == -1)

{

front2 = rear2 = 0;

}

else

{

rear2 = (rear2 + 1) % max;

}

q2[rear2] = value;

}

int Dequeue2()

{

if (front2 == -1)

{

return -1;

}

int item = q2[front2];

if (front2 == rear2)

{

front2 = rear2 = -1;

}

else

{

front2 = (front2 + 1) % max;

}

return item;

}

void PushCostly(int value)

{

Enqueue2(value);

while (front1 != -1)

{

Enqueue2(Dequeue1());

}

while (front2 != -1)

{

Enqueue1(Dequeue2());

}

printf("\nElement %d Pushed to Stack", value);

}

void PopCostly()

{

if (front1 == -1)

{

printf("\nSTACK UNDERFLOW");

return;

}

printf("\nElement %d Popped from Stack", Dequeue1());

}

void PushCheap(int value)

{

Enqueue1(value);

printf("\nElement %d Pushed to Stack", value);

}

void PopCheap()

{

if (front1 == -1)

{

printf("\nSTACK UNDERFLOW");

return;

}

while (front1 != rear1)

{

Enqueue2(Dequeue1());

}

int item = Dequeue1();

while (front2 != -1)

{

Enqueue1(Dequeue2());

}

printf("\nElement %d Popped from Stack", item);

}

void Display()

{

if (front1 == -1)

{

printf("\nSTACK IS EMPTY");

return;

}

printf("\nSTACK (Top to Bottom):\n");

for (int i = front1; i != rear1; i = (i + 1) % max)

{

printf("%d\n", q1[i]);

}

printf("%d\n", q1[rear1]);

}

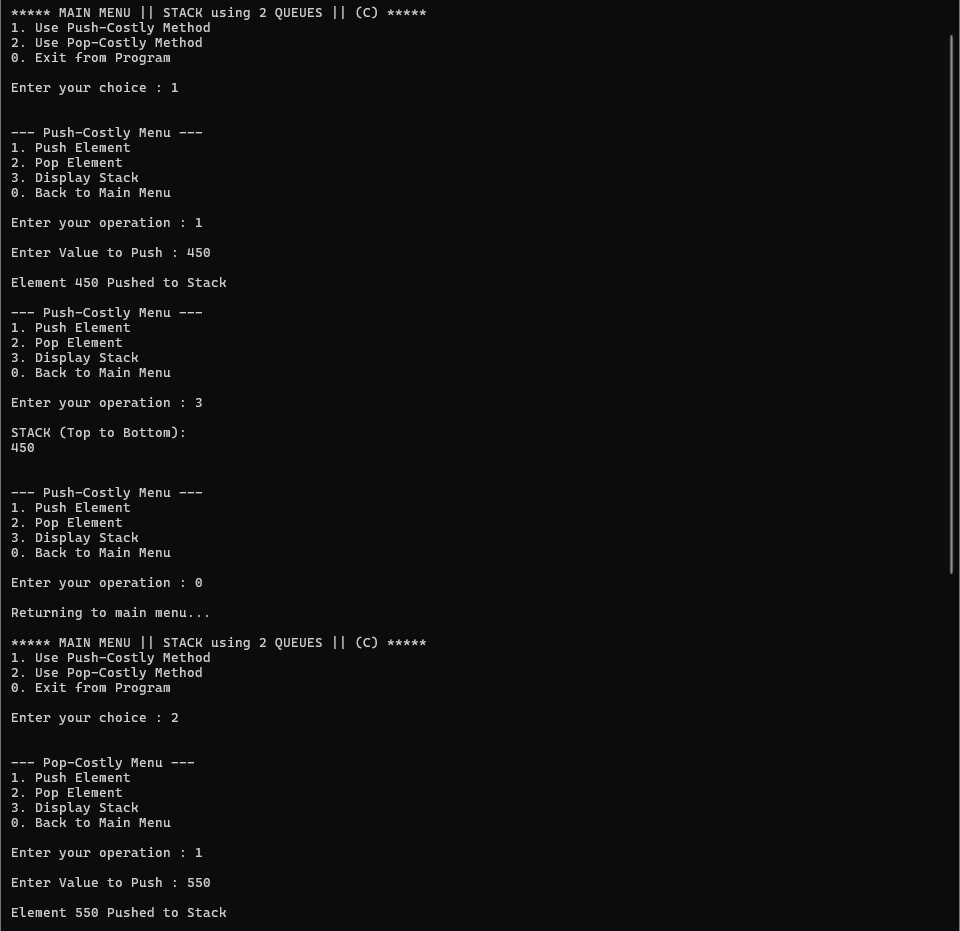
void ResetStack()

{

front1 = rear1 = -1;

front2 = rear2 = -1;

}

****

**Experiment 8: Multiple Queues in a Single Array**

**Objectives:**

**• Implement two queues in one array.**

**• Efficient memory usage by sharing array space.**

**Tasks:**

**1. Define arr[MAX], front1, rear1, front2, rear2.**

**2. Queue1 grows left → right, Queue2 grows right → left.**

**3. Implement enqueue and dequeue for both queues.**

**4. Display elements of both queues.**

#include <stdio.h>

#define max 100

int queue[max];

int front1 = -1, rear1 = -1;

int front2 = max, rear2 = max;

void Enqueue1(int);

void Dequeue1();

void Display1();

void Enqueue2(int);

void Dequeue2();

void Display2();

int main()

{

int choice = 0, value, op;

while (choice != 3)

{

printf("\n\n\*\*\*\*\* MAIN MENU || MULTIPLE QUEUES in Array || (C) \*\*\*\*\*");

printf("\n1. Operations for Queue 1");

printf("\n2. Operations for Queue 2");

printf("\n3. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

op = 0;

while (op != 4)

{

printf("\n\n--- Menu for Queue 1 ---");

printf("\n1. Enqueue Element");

printf("\n2. Dequeue Element");

printf("\n3. Display Queue 1");

printf("\n4. Back to Main Menu");

printf("\n\nEnter your operation : ");

scanf("%d", &op);

switch (op)

{

case 1:

printf("\nEnter Value to Enqueue : ");

scanf("%d", &value);

Enqueue1(value);

break;

case 2:

Dequeue1();

break;

case 3:

Display1();

break;

case 4:

printf("\nReturning to main menu...");

break;

default:

printf("\nInvalid Operation !!");

break;

}

}

break;

case 2:

op = 0;

while (op != 4)

{

printf("\n\n--- Menu for Queue 2 ---");

printf("\n1. Enqueue Element");

printf("\n2. Dequeue Element");

printf("\n3. Display Queue 2");

printf("\n4. Back to Main Menu");

printf("\n\nEnter your operation : ");

scanf("%d", &op);

switch (op)

{

case 1:

printf("\nEnter Value to Enqueue : ");

scanf("%d", &value);

Enqueue2(value);

break;

case 2:

Dequeue2();

break;

case 3:

Display2();

break;

case 4:

printf("\nReturning to main menu...");

break;

default:

printf("\nInvalid Operation !!");

break;

}

}

break;

case 3:

printf("\nExiting from the Program !!");

break;

default:

printf("\nInvalid Choice !!\nTry again !!");

break;

}

}

return 0;

}

void Enqueue1(int value)

{

if (rear1 == rear2 - 1)

{

printf("\nQUEUE OVERFLOW");

return;

}

if (front1 == -1)

{

front1 = rear1 = 0;

}

else

{

rear1++;

}

queue[rear1] = value;

printf("\nElement %d added to Queue 1", value);

}

void Dequeue1()

{

if (front1 == -1)

{

printf("\nQUEUE 1 UNDERFLOW");

return;

}

printf("\nElement %d removed from Queue 1", queue[front1]);

if (front1 == rear1)

{

front1 = rear1 = -1;

}

else

{

front1++;

}

}

void Display1()

{

if (front1 == -1)

{

printf("\nQUEUE 1 IS EMPTY");

return;

}

printf("\nQueue 1 (Front to Rear): ");

for (int i = front1; i <= rear1; i++)

{

printf("%d ", queue[i]);

}

}

void Enqueue2(int value)

{

if (rear2 == rear1 + 1)

{

printf("\nQUEUE OVERFLOW");

return;

}

if (front2 == max)

{

front2 = rear2 = max - 1;

}

else

{

rear2--;

}

queue[rear2] = value;

printf("\nElement %d added to Queue 2", value);

}

void Dequeue2()

{

if (front2 == max)

{

printf("\nQUEUE 2 UNDERFLOW");

return;

}

printf("\nElement %d removed from Queue 2", queue[front2]);

if (front2 == rear2)

{

front2 = rear2 = max;

}

else

{

front2--;

}

}

void Display2()

{

if (front2 == max)

{

printf("\nQUEUE 2 IS EMPTY");

return;

}

printf("\nQueue 2 (Front to Rear): ");

for (int i = front2; i >= rear2; i--)

{

printf("%d ", queue[i]);

}

}

****

**Experiment 9: Applications of Queue**

**Objectives:**

**• Implement real-world scenarios using queue.**

#include <stdio.h>

#define max 100

int service\_queue[max];

int vip\_front = -1, vip\_rear = -1;

int normal\_front = max, normal\_rear = max;

void AddVipCustomer(int);

void AddNormalCustomer(int);

void ServeNextCustomer();

void DisplayQueues();

int main()

{

int choice = 0, ticket\_no = 1;

while (choice != 4)

{

printf("\n\n\*\*\*\*\* CUSTOMER SERVICE QUEUE || (C) \*\*\*\*\*");

printf("\n1. Add VIP Customer");

printf("\n2. Add Normal Customer");

printf("\n3. Serve Next Customer");

printf("\n4. Exit from Program");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

AddVipCustomer(ticket\_no++);

break;

case 2:

AddNormalCustomer(ticket\_no++);

break;

case 3:

ServeNextCustomer();

break;

case 4:

printf("\nClosing service for the day !!");

break;

default:

printf("\nInvalid Choice !!\nTry again !!");

break;

}

DisplayQueues();

}

return 0;

}

void AddVipCustomer(int ticket)

{

if (vip\_rear == normal\_rear - 1)

{

printf("\nQUEUE OVERFLOW: Cannot add more customers.");

return;

}

if (vip\_front == -1)

{

vip\_front = vip\_rear = 0;

}

else

{

vip\_rear++;

}

service\_queue[vip\_rear] = ticket;

printf("\nVIP Customer with Ticket #%d has been added.", ticket);

}

void AddNormalCustomer(int ticket)

{

if (normal\_rear == vip\_rear + 1)

{

printf("\nQUEUE OVERFLOW: Cannot add more customers.");

return;

}

if (normal\_front == max)

{

normal\_front = normal\_rear = max - 1;

}

else

{

normal\_rear--;

}

service\_queue[normal\_rear] = ticket;

printf("\nNormal Customer with Ticket #%d has been added.", ticket);

}

void ServeNextCustomer()

{

if (vip\_front != -1)

{

printf("\nServing VIP Customer with Ticket #%d.", service\_queue[vip\_front]);

if (vip\_front == vip\_rear)

{

vip\_front = vip\_rear = -1;

}

else

{

vip\_front++;

}

}

else if (normal\_front != max)

{

printf("\nServing Normal Customer with Ticket #%d.", service\_queue[normal\_front]);

if (normal\_front == normal\_rear)

{

normal\_front = normal\_rear = max;

}

else

{

normal\_front--;

}

}

else

{

printf("\nUNDERFLOW: No customers are waiting to be served.");

}

}

void DisplayQueues()

{

int i;

printf("\n\n--- Current Waiting Customers ---");

if (vip\_front == -1)

{

printf("\nVIP Queue: [EMPTY]");

}

else

{

printf("\nVIP Queue (Front to Rear): ");

for (i = vip\_front; i <= vip\_rear; i++)

{

printf("%d ", service\_queue[i]);

}

}

if (normal\_front == max)

{

printf("\nNormal Queue: [EMPTY]");

}

else

{

printf("\nNormal Queue (Front to Rear): ");

for (i = normal\_front; i >= normal\_rear; i--)

{

printf("%d ", service\_queue[i]);

}

}

printf("\n---------------------------------");

}

