Question

1. What is circular queue.

2. What are the characteristics of circular queue.

3. Give applications of circular queue.

4. What is algorithm of circular queue.

5. Write a simple program of circular.

6. Compare and contrast linear queue and circular queue.

Answers

1. A circular queue is a data structure that follows the FIFO (First In First Out) principle, but the last element is connected to the first element, forming a circle. This means that the queue can be treated as a ring buffer, where the elements are stored in contiguous memory locations.
2. The characteristics of circular queue are:

* It is a linear data structure that follows the FIFO principle.
* The last element is connected to the first element, forming a circle.
* The queue can be treated as a ring buffer, where the elements are stored in contiguous memory locations.
* It can be implemented using an array or a linked list.



* CPU Scheduling: Circular queues are used in operating systems for CPU scheduling algorithms, where processes are scheduled in a circular manner.
* Disk Space Management: Circular queues can be used to manage disk space allocation, where the allocation of free space is done in a circular order.
* Printer Spooling: In printer spooling systems, circular queues are employed to handle multiple print requests efficiently.
* Traffic Management: Circular queues can be utilized in traffic management systems to manage incoming and outgoing traffic in a circular manner.



* Initialization: Create an array of a fixed size and initialize the front and rear pointers to -1.
* Enqueue: Check if the queue is full. If not, increment the rear pointer circularly and add the new element at the rear position.
* Dequeue: Check if the queue is empty. If not, increment the front pointer circularly and remove the element at the front position.
* Overflow and Underflow Conditions: Maintain a count variable to keep track of the number of elements in the queue and check for overflow and underflow conditions accordingly.

1. Simple program circular in c#

using System;

public class CircularQueue

{

private int[] queue;

private int front;

private int rear;

private int maxSize;

private int currentSize;

public CircularQueue(int size)

{

maxSize = size;

queue = new int[maxSize];

front = -1;

rear = -1;

currentSize = 0;

}

public void Enqueue(int item)

{

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

{

Console.WriteLine("Queue is full. Enqueue operation cannot be performed.");

return;

}

if (front == -1)

front = 0;

rear = (rear + 1) % maxSize;

queue[rear] = item;

currentSize++;

Console.WriteLine($"Enqueued: {item}");

}

public void Dequeue()

{

if (front == -1)

{

Console.WriteLine("Queue is empty. Dequeue operation cannot be performed.");

return;

}

int item = queue[front];

if (front == rear)

{

front = -1;

rear = -1;

}

else

{

front = (front + 1) % maxSize;

}

currentSize--;

Console.WriteLine($"Dequeued: {item}");

}

public void Display()

{

if (front == -1)

{

Console.WriteLine("Queue is empty.");

return;

}

Console.Write("Queue: ");

int i = front;

while (i != rear)

{

Console.Write($"{queue[i]} ");

i = (i + 1) % maxSize;

}

Console.WriteLine(queue[rear]);

}

}

public class Program

{

public static void Main(string[] args)

{

CircularQueue queue = new CircularQueue(5);

queue.Enqueue(10);

queue.Enqueue(20);

queue.Enqueue(30);

queue.Display(); // Output: Queue: 10 20 30

queue.Dequeue();

queue.Display(); // Output: Queue: 20 30

queue.Enqueue(40);

queue.Enqueue(50);

queue.Display(); // Output: Queue: 20 30 40 50

queue.Enqueue(60); // Output: Queue is full. Enqueue operation cannot be performed.

queue.Dequeue();

queue.Dequeue();

queue.Dequeue();

queue.Dequeue();

queue.Display(); // Output: Queue is empty.

// To keep the console window open until a key is pressed

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

}

}



Linear Queue:

* In a linear queue, the front and rear pointers move linearly from the start to the end of the queue.
* When the rear reaches the end, no more elements can be inserted unless some elements are dequeued.
* After dequeueing, the space occupied by the dequeued element is not reused until the front pointer reaches the end.
* Linear queues are simpler to implement compared to circular queues.

Circular Queue:

* In a circular queue, the front and rear pointers form a loop, allowing elements to be inserted even if the rear reaches the end.
* When the rear reaches the end, it wraps around to the beginning of the queue, utilizing the empty spaces created by dequeueing.
* Circular queues utilize the available space more efficiently as the empty spaces are reused.
* Circular queues are particularly useful in scenarios where there is a need to repeatedly insert and remove elements in a cyclical manner.