Tutorials 6

1. What is circular queue?

A circular queue is a linear data structure that follows FIFO principle. the last node is connected back to the first node to make a circle.

2. What are the characteristics of circular queue.

- Last element is connected to the first element to make the circular structure.

- The circular queue solves the major limitati of the normal queue.

- In a normal queue, after a bit of insertion and deletion, there will be non-usable empty space.

3. Give applications of circular queue.

* Memory management : circular queue is used in memory management.

* Process Scheduling : A CPU uses a queue to shedule process.

* Traffic system : Queues are also used in traffic systems.

No: Date: 07 06 2023

(4) What is algorithm of circular queue.

A circular queue is similar to a linear queue as it is also based on the FIFO principle except that last position is connected to the 1st position in a circular queue that forms a circle.

- This algorithm has the following operations Initializing, Enqueue, Dequeue, IsEmpty, IsFull, Front and Rear. The circular nature is achieved by the modulo operator (%) with the size of the array to calculate the next position.

(5) Write a simple (c) program of a circular queue

```
# include <stdio.h>
# define MAX_SIZE 5
int queue [MAX_SIZE>;
int front = -1 , rear = -1;
Void enqueue (int item)
 {
    if ((rear+1) % MAX_SIZE == front)
     {
        Printf ("Queue is full. Unable to enqueue. \n");
     }
     else if (front == -1 && rear == -1)
     {
        front = rear = 0;
        queue[rear] = item;
     }
     else
     {
        rear = (rear+1) % MAX_SIZE;
        queue [rear] = item;
     }
 }
```

```
int  dequeue ()
{
    int item ;
    if (front == -1 && rear == -1)
    {
        printf ("Queue is empty. Unable to dequeue \n");
        return -1;
    }
    else if (front == rear)
    {
        item = queue [front];
        front = rear = -1;
        return item;
    }
    else
    {
        item = queue [front];
        front = (front +1) % MAX_SIZE;
        return item;
    }
}
int is_empty ()
{
    return front == -1 && rear == -1;
}
int get_front ()
{
    if (front == -1 && rear == -1)
    {
        Printf ("Queue is empty. \n");
        return -1;
    }
    else { return queue [front]; }
}
```

```
int    get_rear ()
{
    if (front == -1 && rear == -1)
  {
     printf (" Queue is empty. \n");
     return -1 ;
  }
    else
   {
     return  queue [rear];
   }
}

 int main()
{
      enqueue (1);
      enqueue (2);
      enqueue (3);
      enqueue (4);
      enqueue (5);
      printf( "Front : %d \n", get-front ());  // output : 1
      printf (" Rear : %d \n", get_rear ());  // output : 5

      dequeue ();
      enqueue (6);

      printf ("Front : %d \n", get-front ()); //output: 2
      printf( "Rear : %d \n", get-rear());  //output : 6

   return 0;
}
```

⑥ Compare and contrast linear queue and circular queue.

	Linear queue	Circular queue
Similarities	• FIFO principle	• FIFO principle
	• supports the operations enqueue and dequeue.	• Supports the operations enqueue and dequeue
	• Size limit - Have a fixed size limit determined by the amount of allocated memory or the maximum capacity specified during initialization.	• Size limit – Have a fixed size limit determined by the amount of allocated memory or the maximum capacity specified during initialization
	• Have ways to check whether the queue is empty/full.	• Have ways to check whether the queue is empty /full.
	• Pointers :- front and rear	• Pointer := front and rear.

(to keep track of the first, last elements respectively)

• Memory allocation :- requires a fixed amount of memory allocations to store the elements.

• In both queues elements are stored in a sequential manner.

Linear queus vs Circular queue

comparison

• Follows a linear structure	• Follows a circular structure.
• Easier to implement	• slightly more complex to implement.
• Linear queues are suitable in situations where elements are inserted and removed strictly from one end.	• Circular queues are useful in scenarios where the process of insertion and removal wraps around.

Tutorials ⑦ 14/06/2023

Recap -

① Why are stacks useful?

to call functions and execution, to evaluate expressions, to undo/redo operations, to manage memory, to browse history, to backtrack algorithms etc.

② Reverse a string using stack?

```
#include <stdio.h>
#include <string.h>

#define MAX_LENGTH 100

//Function to reverse a String using a stack
Void reverseString (char *inputstring, char *reveredstring)
{
        int length = strlen (inputstring);
        char stack [MAX_LENGTH];
        int top = -1;
```