```
printf ("Top element : %d \n", peek(&stack));

// Popping elements from the stack
printf("\n Popped element : %d \n", pop(&stack));
printf ("\n Popped element : %d \n", pop (&stack));

// check if the stack is empty
printf ("Is the stack empty? %s \n",
        isEmpty(&stack) ? "Yess" : "No");

return 0;
}
```

2023/06/21

## Tutorpals (8)

(1) What is a circular queue?

A circular queue also known as "Ring Buffer" is a data structure that follows first-in-First-out (FIFO) principle.

It is implemented as an array / queue with a fixed size, where the last position is connected to the first position / first index forming a circular behaviour.

(2) What are the characteristics of circular queue?

- front and rear as pointers: In circular queues two pointers ~~front~~ maintain the order of the circular queue.

- The circular behaviour: When the data structure has reached its capacity the next

element will be inserted in to a vacant position in the front of the data structure.

- Fixed size : The circular queues are initialized with a fixed size which prevents them from growing dynamically.

(3) Give applications of circular queue?

- Cache management.
- Printer spooling
- Buffer management
- Event handaling.
- Traffic management
- Memory management
- OS / cpu scheduling.

(4) What is the algorithm of circular queue?

algo t8 q4

File Edit View

ALGO tute 8 question 4 answer;
The algorithm for a circular queue typically involves the following operations:
1. Initialize the circular queue:
Create an array of a fixed size to hold the elements of the queue.
Initialize two pointers, "front" and "rear," both set to -1 (indicating an empty queue).
Check if the queue is empty:

If front == -1 and rear == -1, the queue is empty.
Check if the queue is full:
If (rear + 1) % size == front, the queue is full, where "size" represents the maximum capacity of the queue.
Enqueue (insert) an element into the circular queue:

2. Check if the queue is full.
If it is, return an error or perform appropriate error handling.
If the queue is empty (front == -1 and rear == -1), set both front and rear pointers to 0.
Otherwise, increment the rear pointer by 1: rear = (rear + 1) % size.
Insert the new element at the rear position in the array: queue[rear] = element.
Dequeue (remove) an element from the circular queue:

3. Check if the queue is empty.
If it is, return an error or perform appropriate error handling.
Store the value of the front element to return it later: element = queue[front].
If front == rear (indicating the queue has only one element), set both front and rear pointers back to -1.
Otherwise, increment the front pointer by 1: front = (front + 1) % size.
Get the front element of the circular queue without removing it:

4. Check if the queue is empty.
If it is, return an error or perform appropriate error handling.
Return the value of the front element: element = queue[front].
Get the size of the circular queue:

If front == -1 and rear == -1, the queue is empty, so return 0.
If rear ≥ front, the size of the queue is (rear - front + 1).
If rear < front, the size of the queue is (size - front + rear + 1).

Ln 2, Col 43 80%