# PLAYBOOK NOTES: QUEUES

## INTRODUCTION

**What is a queue:**

A queue is a list of data where all data is removed from the front of the queue and all new data is added at the end of the queue. So, the first data into the queue is it the first data out of the queue. This is called a **FIFO** (First in, First Out)

**An example of a real-life queue:**

Getting in line to…. do anything. (I prefer to think of lines to ride a roller coaster, but lines to check out at a store is just as good!)

**An example of queues used in computer:**

(Data Structures, Algorithms & Software Principles In C, Thomas A. Standish, ©1995)

- Regulate flow of tasks within a system (Example: Operating Systems)
- Synchronize the exchange of data between processes running at different speeds (Example: printer buffer)
- Writing simulations

## CREATING A QUEUE

**First let's create our structures:**

```
typedef struct
{
    //
    // All the data that will be stored in one record
    // will be declared here.
    //

    int sampleData;

} recordType;

typedef struct Node
{
    // Declare a record
    recordType data;

    struct Node* pNext;

} nodeType;

typedef struct
{
    nodeType* pFront;
    nodeType* pRear;

} queueType;
```

**Let's start by writing some functions:**

```
//--------------------------------------------------------------------------------
// Function Name: InitializeQueue
// Description:
//    Assign both pointers of the queue to NULL
//
//--------------------------------------------------------------------------------

void InitializeQueue(queueType* queue)
{
    // Assign both pointers to NULL
    queue->pFront = NULL;
    queue->pRear = NULL;
}


//--------------------------------------------------------------------------------
// Function Name: QueueIsEmpty
// Description:
//   Check to see if the front pointer is NULL
//
//--------------------------------------------------------------------------------

int QueueIsEmpty(queueType queue)
{
    // Check to see if pointer is NULL
    return queue.pFront == NULL;
}
```

**Creating our queue:**

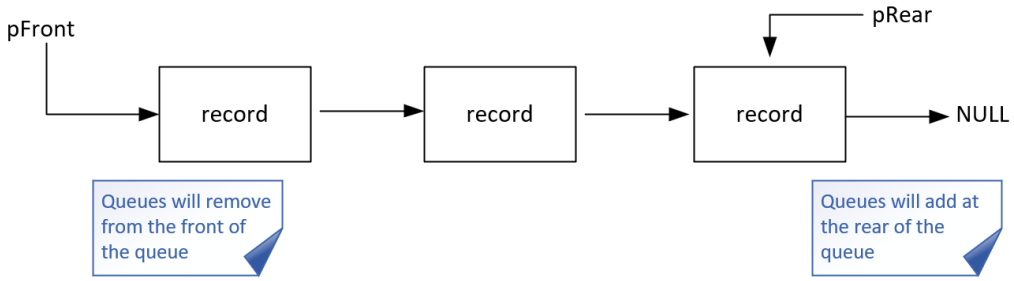
After creating the queue DON'T FORGET to initialize your pointers to NULL

```
int main(void)
{
    // Declare the queue
    queueType queue;

    // DON'T FORGET TO INITIALIZE YOUR POINTERS TO NULL;
    InitializeQueue(&queue);
```

**Thinking big picture:**

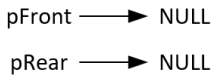
As we put our records into our queue, we will develop a link list that will start to resemble the following picture:



# INSERTING NODES INTO OUR CODE (ENQUEU)

**First case (both pointers equal NULL):**

The first time we insert a node, both bothers are NULL.



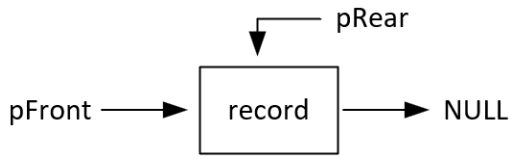

So, we assign both pointers to the node:

```
void InsertNode(queueType* queue, nodeType* pTemp)
{
    if (pTemp == NULL)
    {
        // Print error message
        printf("ERROR, temp is NULL\n");
    }
    else
    {
        // Check to see if the queue is empty
        if (queue->pFront == NULL)
        {
            // Assign both pointers to our temp node
            queue->pFront = pTemp;
            queue->pRear = pTemp;
        }
```

Notice the defensive coding, checking our temp pointer for NULL.

Also notice that we are using pointers because the address will be changing.

Now our pointers look like this:

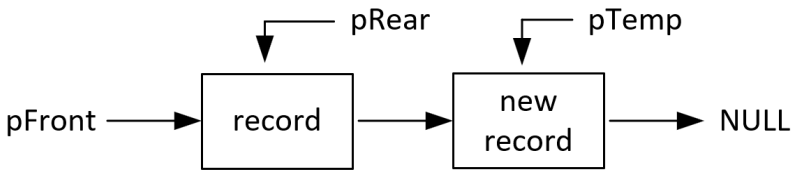



**Second case (there is at least one node in the list):**

Since we will insert the at the rear of the queue, we will not worry about pFront.

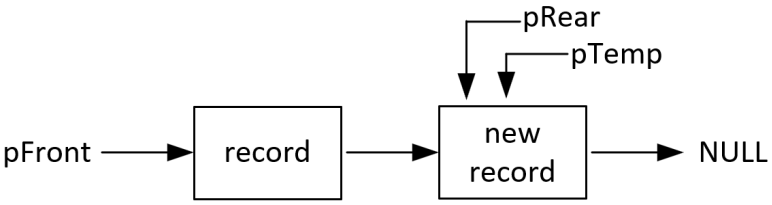
First, we will assign pRear->pNext to our temp node.

The pointers will look like this:




Second, we re-assign pRear to be equal to pTemp.

Now the pointers look like this:




Notice that pTemp is still there, but it is not relevant.

The code for the second case:

```
    else
    {
        // Add new node to the queue
        queue->pRear->pNext = pTemp;
        queue->pRear = pTemp;
    }
}
```

All other nodes to be inserted into the queue will follow the second case.

## REMOVING A NODE FROM THE QUEUE (DEQUEUE)

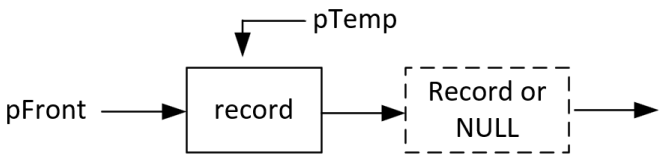
To remove a node from the queue, we first do defensive coding to make sure the queue isn't already empty.

```
void RemoveNode(queueType* queue)
{
    nodeType* pTemp;

    if (queue->pFront == NULL)
    {
        // Print error message to the screen
        printf("ERROR, queue is empty\n");
    }
```

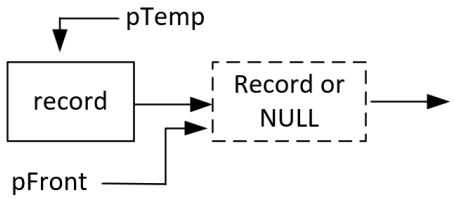
**First case (and only case): Remove the node at the front.**

First, set our temp variable to equal the front.

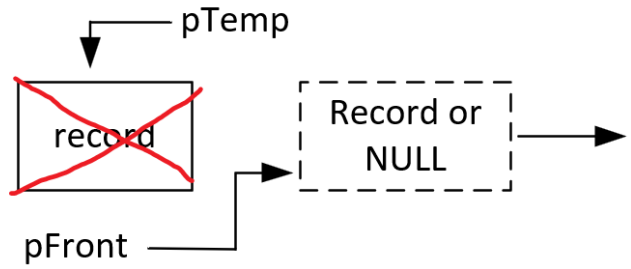



Note: There may be at least one other node in the queue or there may only be one node in the queue.

Second, move the front pointer forward one node. (This may be another node, or it may be NULL)




Third, free the temp node.




Here is the remaining code:

```
else
{
    // Assign temp to the front of the queue
    pTemp = queue->pFront;

    // Move the front pointer forward one
    queue->pFront = queue->pFront->pNext;

    // Check to see if the list is now empty
    if (queue->pFront == NULL)
    {
        // Set the rear pointer to NULL
        queue->pRear = NULL;
    }

    // Delete the temp pointer
    free(pTemp);
}
```

Notice that there is a check to see if the queue is now empty.

If the list is empty the rear pointer is set to NULL.

NOTE: Depending on the needs of your program, you may want to return a copy of the structure that you removed.