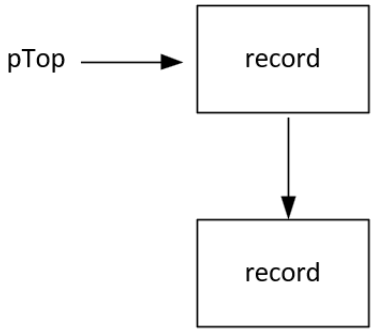
PLAYBOOK NOTES: STACKS

INTRODUCTION

What is a stack:

A stack is a list of data that you can visualize going up and down. New data is always added to the top (PUSH), and when taking data off the stack (POP) it always taken off the top. So, the last data onto the stack is the first one out. This is called a **LIFO** (Last In, First Out).



An example of a real-life stack:

Think of a stack of plates at buffet. You always add plates to the top of the stack and when you want a plate you always take a plate off the top.

CREATING A STACK

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* pTop;

} stackType;
```

First let's write a few functions:

```
//-------------------------------------------------------------------------
// Function Name: InitializeStack
// Description:
//   Set the top pointer to equal NULL
//
//-------------------------------------------------------------------------
void InitializeStack(stackType* pStack)
{
    // Set pointer to NULL
    pStack->pTop = NULL;
}


//-------------------------------------------------------------------------
// Function Name: StackIsEmpty
// Description:
//   Returns true or false based on if the pointer to top is NULL
//
//-------------------------------------------------------------------------
int StackIsEmpty(stackType stack)
{
    // check to see if the stack is empty
    return stack.pTop == NULL;
}
```

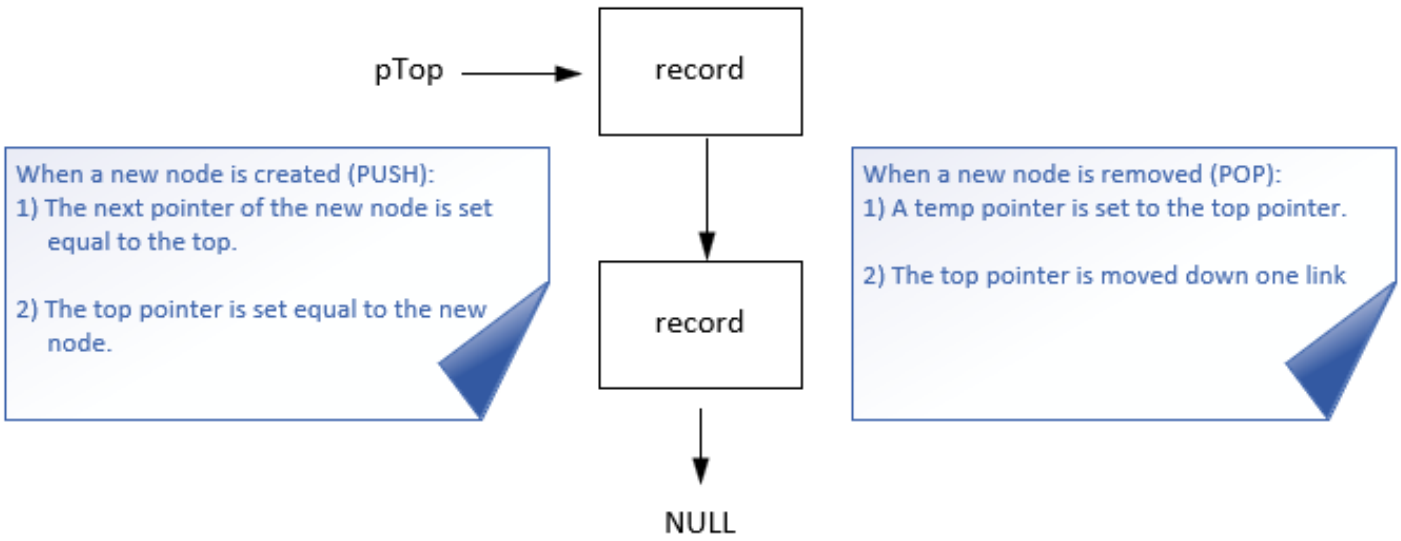
NOTE: If your stack has a limit of how many nodes can be in the stack, you will want to implement a function called IsFull. A variable that holds such a number would be created in the stackType definition.

Creating our stack:

```
// Declare our stackType
stackType stack;

InitializeStack(&stack);
```

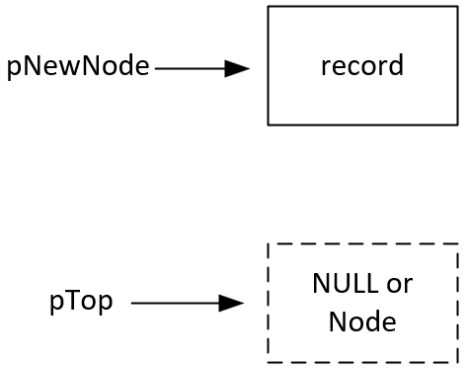
Thinking big picture:



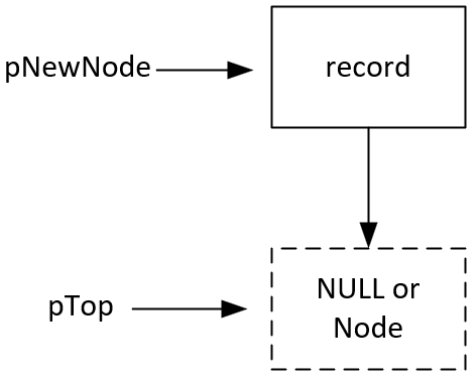
INSERTING NODES ON THE STACK (PUSH)

It does not matter if the stack is empty or if there are nodes in the stack.

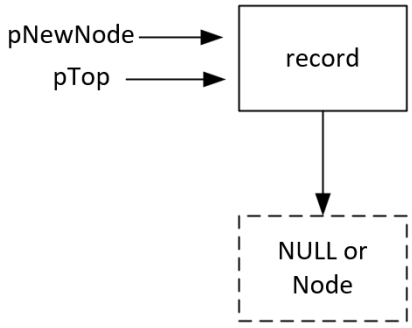
So, to start, our pointers look like this:



First, we assign the next pointer of our new node to equal to the top.



Second, we move the top pointer to the new node.



That's it! The new node has been inserted into the stack!

Here is the code:

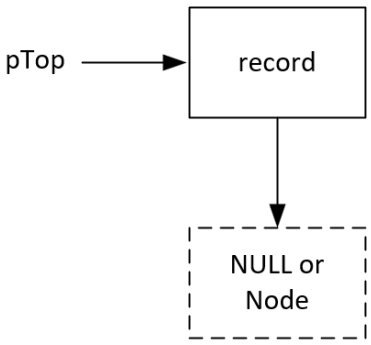
```
//-------------------------------------------------------------------------
// Function Name: Push
// Description:
//   Add one node to the stack.
//   Thus function assumes that the new node has already been
//  declared and initialized.
//
//-------------------------------------------------------------------------
void Push(stackType* pStack, nodeType* pNewNode)
{
    //Defensive coding
    if (pNewNode == NULL)
    {
        printf("ERROR - There is no new data\n");
    }
    else
    {
        // Set the next pointer of the new node to equal top
        pNewNode->pNext = pStack->pTop;

        // Move the top pointer
        pStack->pTop = pNewNode;
    }
}
```

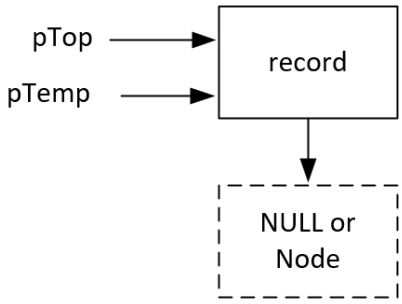
REMOVING NODES FROM THE STACK (POP)

To start, you will want to make sure the queue is not empty.

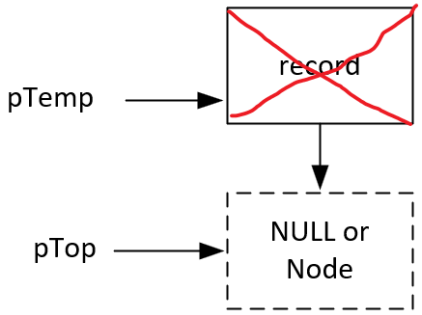
Assuming there is at least one node on the stack:



First, we assign a temp pointer to the top.



Second, we move the top pointer to the next node on the stack (or NULL) and free the



Here is the code:

```
//-------------------------------------------------------------------------
// Function Name: Pop
// Description:
//   Remove one node from the stack
//
//   Depending on the needs of your program, you may want to pass in a
//  nodeType pointer so you can return the data from the node.
//
//-------------------------------------------------------------------------

void Pop(stackType* pStack)
{
    nodeType* pTemp;

    // See if the stack is empty
    if (!StackIsEmpty(*pStack))
    {
        // Assign pTemp to the top of the stack
        pTemp = pStack->pTop;

        // Move the top pointer down one node
        pStack->pTop = pStack->pTop->pNext;

        // Free the memory pointed to by temp
        free(pTemp);

    }

}
```