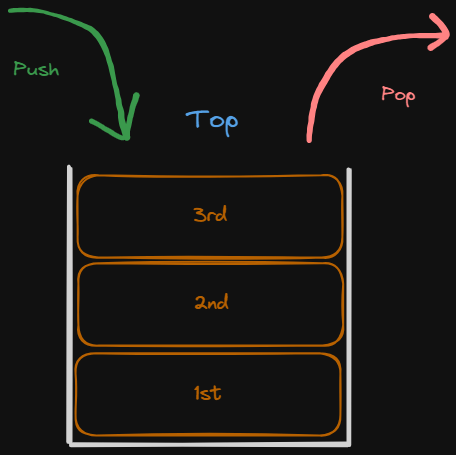
Code Documentation

**Stack** - a stack is an abstract data type that serves as a collection of elements with two main operations:

Push - Adds an element to the stack

Pop - Removes an element from the stack

The stack follows the LIFO (Last In, First out) order

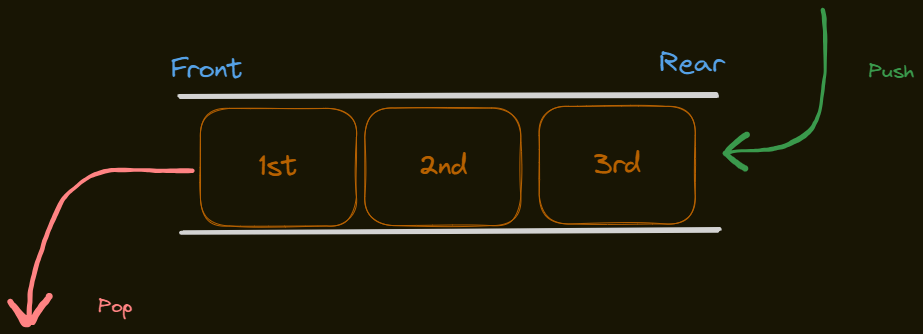


**Queue** - A Queue Data Structure is a fundamental concept in computer science used for storing and managing data in a specific order. It follows the principle of FIFO (First In, First Out), where the first element added to the queue is the first one to be removed.

Queue Operations:

Enqueue - Adds an element to the queue

Dequeue - Removes an element from the queue



Both the Stack and Queue Data Structures are very similar since they can be implemented identically (most commonly through Arrays or Linked Lists). What actually differs is just the order in which the elements are inserted/deleted (FIFO/LIFO).

What is pivotal are these four states that will always be present in each Data Structure Implementation:

1. If the Data Structure is empty
2. If the Data Structure is Full
3. If the Data Structure only has 1 single element
4. If the Data Structure has 2 or more elements inside

These 4 states will be the core logic behind the Data Structure Operations of Stacks and Queues. Now to cover all these operations clearly, we will arrange them either by Insertion/Deletion in order for us to also compare the differences in Logic between a Stack and a Queue Implementation.

INSERTION:

* Stack (Push):

void push(int num, int stkSize, int \*stack){

if(top == (stkSize-1)){ //if stack is full

printf("\nSTACK OVERFLOW: Cannot push a FULL stack\n");

printf("Select (2)Pop from the Operations Menu to delete an entry from the Stack\n\n");

}else if(top < 0 && bottom < 0){ //if stack is empty

top = 0;

bottom = 0;

stack[top] = num;

}else{ //if stack has content but is not full

top++;

stack[top] = num;

}

}

* Queue (Enqueue):

void enqueue(int num, int qSize, int \*queue){

if(rear == (qSize-1)){ //if queue is full

clearScreen();

printf("\nQUEUE OVERFLOW: Cannot enqueue to a FULL queue\n");

printf("Select (2)Dequeue from the Operations Menu to delete an entry from the Queue\n\n");

}else if(rear < 0 && front < 0){ //if queue is empty

front = 0;

rear = 0;

queue[rear] = num;

}else { //if queue has content but is not full

rear++;

queue[rear] = num;

}

}

DELETION:

* Stack (Pop):

void pop(int stkSize, int \*stack){

if(top < 0 && bottom < 0){ // if stack is empty

printf("\n\nSTACK UNDERFLOW: Cannot pop from an empty Stack\n");

printf("Select (1)Push from the Operations Menu to insert an entry into the Stack\n\n");

}else if(top == bottom){ //if the stack has only 1 entry

stack[top] = stack[0];

top = -1; // no more existing entries in the stack. STACK IS NOW EMPTY

bottom = -1;

printf("\nPopped Last Entry. STACK IS NOW EMPTY\n\n");

}else{ //if stack has more than 1 entry

stack[top] = stack[top-1];

top--;

printf("\nAn Entry Has been Removed\n");

printf("Current Entries in the Stack: %d/%d \n\n", top+1, stkSize);

}

}

* Queue (Dequeue):

void dequeue(int \*queue, int qSize){

if(front < 0 && rear < 0){ //if queue is empty

printf("\n\nQUEUE UNDERFLOW: Cannot dequeue from an EMPTY queue\n");

printf("Select (1)Enqueue from the Operations Menu to insert an entry into the Queue\n\n");

}else if(front == rear){ //if queue only has 1 entry

queue[front] = queue[front+1];

front = -1; //declare that there are no existing entries in the queue. QUEUE IS NOW EMPTY

rear = -1;

printf("\nDeleted Last Entry. Queue is now EMPTY\n\n");

}else{ //if queue has more than 1 entry

queue[front] = queue[front+1];

front++;

printf("\nAn Entry Has been Removed.\n");

printf("Current Entries in the Queue :%d/%d \n\n", qSize-front,qSize);

}

}

Since our Implementation uses a dynamically allocated array, then the logic is index-based. So if we were to delete or insert an element, we would just either increment/decrement the index of the array relative to what operation is being used.