Assignment 2: Design

Wednesday, January 13, 2021 3:43 PM

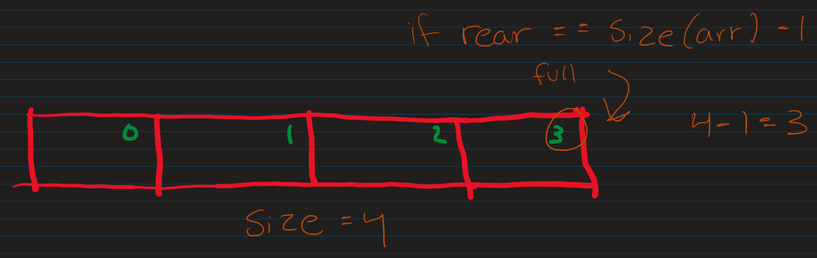
Queue Data Structure:

Assignment 2 (Design)

Assignment 2 (Design):

1. **Do not implement any code this week!**
2. Based on what we know about linked lists, stacks, and queues, design a queue data structure:
    1. What functions are we likely to need for a queue to function like the one discussed in class?
    2. What values will we need to know about the structure for our queue to function properly?

1.) functions we most likely need for a queue to function would be a function to initialize a queue, a function to add values to the queue (enqueue) values to remove stuff from the queue (dequeue), and functions that check if a queue is empty or full. We also need an initializer and main function.




```
is_empty () {
  if ( front == -1 && rear == -1)
        return true
  else
        return false
  }
```

```
is_full () {
  if (rear == size(arr) - 1
        return true
  else
        return false
  }
```

* must do -1 to account that we are starting at 0

```
enqueue ( element) {
  if ( is_full () )          * Checks if queue is full
      return
  else if (is_empty() )      * If not full, Check it it is empty and if so, set the rear and front equal to zero
      rear = front = 0
  else
      ++ rear                *If is is not empty or full, switch the rear one value higher, then set a value into the position of the rear
  arr[rear] = element
```

```
dequeue () {
  int i = 0;          * Set a temporary variable to use later
  if (is_empty() )    * Check to see if queue is empty and return if so.
      return;
  else if (front == rear)  * If there is one element in the list, set the
      i = arr [front ]       frontValue as i and then set the front and
      front = rear = -1      rear pointers to -1 to give invalid position (AKA empty queue)
  else
      i = arr [ front ]      * When there's multiple values in the queue, i gets assigned the value of the front pointer and the front pointer gets incremented by one.
      ++ front
  return i                   * Removes value
```

```
queue_init(curr_queue, curr_q_size)
    front = -1
    rear = -1
    for ( i=0, i < curr_q_size , ++i )
          arr[i] = 0
```

* initializes the queue by iterating through the size of the array, setting the pointers equal to -1, and setting the elements in the array equal to zero.

```
Main () {
  array_size = 10
  array_q [array_size ]
  Current_index = 0
  is_full()
  is_empty()
  queue_init ( Curr_q , array_size)
  enqueue ( 100)
  dequeue()
  return 0
```

2.) We would need to know the values and position of the front and rear in order to have everything incrimenting correctly and have the structure working correctly.

List data Structure

3. Based on what we know about linked lists, design a list data structure that allows us to add (insert) or remove (delete) values at a given location in the list (instead of the top of a stack or the front or back of a queue):
    1. What functions are we likely to need for a list to function like this?
    2. What values will we need to know about the structure for our list to function properly?

1.) In order for a list that lets us add or delete values at a given location, we must utilize functions that ititializes the list, a function that adds values to list, a function that deletes values to a list, and a main function that puts everything together.

```
Main () {
  array_size = 10
  array_list [array_size ]
  Current_index = 0                    *initializes list
  return_addy = initialize_list (array_list, array_size )
  Print ("return address: " + return_addy )
  New_Val = 710                        * adds to list
```

```
initialize_list ( Curr_list, Curr_list_size ) {
    for( i=0, i < Curr_list_size, ++i ) {
        Curr_list[i] = -1
    }
    return Curr_list
}
```

* takes in Curr_list and Curr_list size and uses a for loop to iterate through the size of Curr_list. All values are set to -1, (means empty list)

```
remove_Value ( Curr_list, arr_size , targ_index ) {
    if((targ_index -1) >= 0 && (targ_index -1) < arr_size)
        Curr_list [targ_index -1 ] = -1
        ++ targ_index
    else
        Print ( "Cannot remove from list, targ_index out of reach!" )
    return targ_index
}
```

* Checks to see if list is empty by checking the conditions that the position before the target index is not less than zero. Sets the position, (targ_index -1) in Curr_list to -1. Removes the value and sets it equal to -1. Increments the target index.

```
add_Value ( Curr_list, arr_size, targ_index, added_Val ) {
        if ( targ_index >= 0 && targ_index < arr_size )
            Curr_list [targ_index ] = added_Val
            ++ targ_index
        else
            Print ( "Cannot add to list, too full!")
        return targ_index
    }
```

* first checks to see if the target index is greater than zero and less than the size of the array. If so, the value at the target index in Curr_list gets set to added value. If target index is less than zero or greater than the size of the array, then an error message pops up. Target index is returned

2.) In order for our list to function properly, we must know the values of our array size, our target index, and the values we'd like to add