Implementation of a Queue as a Circular Linked List with Dummy Node

A *queue* is a priority queue where priority is given to the item which has been in the queue the longest. Informally, this rule is known as FIFO. The items of a queue belong to some type which could be anything, but in our examples we assume that type is char. The standard operators of the ADT queue are

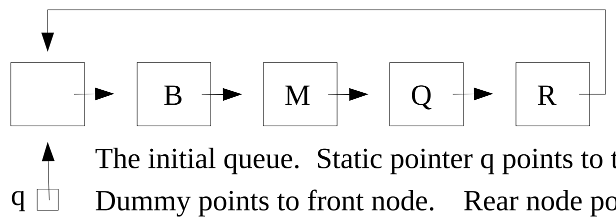
1. `Initialize(queue q)` initializes a queue q with no items.
2. `Empty(queue q)` of Boolean type, returns **true** if q has no items, false otherwise.
3. `Enqueue(queue q,char newitem)` Inserts newitem into q. The length of the queue increments.
4. `Dequeue(queue q)` returns the item which has been in q the longest, and deletes it from q.

A standard implementations of a queue is as a *list*, where items are inserted at the *rear* of the list and deleted from the *front*. This list is typically implemented as an array or a linked list. The front and rear of the list are stored as indices of the array or as pointers to nodes of the linked list.

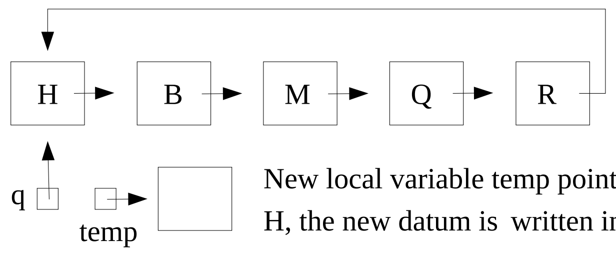
Memory Leak. If the list is simplemented as a linked list, deletion of an item could cause the memory of the computer to become cluttered with variables that are no longer accessible to the program. There are ways to guard against memory leak.

False Overflow. In the list implementation of queue, there could be no room at the rear of the list to insert an item despite there being room at the beginning of the list. The *false overflow* problem can be solved in several ways.

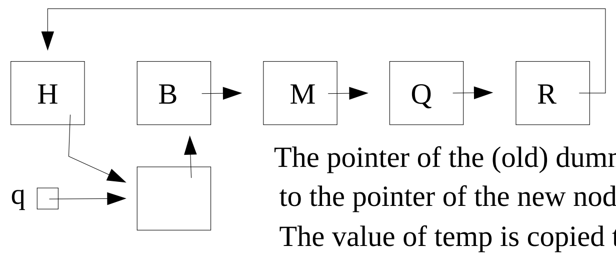
1. *Wrap.* Make the list circular.
2. *Slide.* Move all items closer to the front.
3. *Bigger.* If your list is an array, simply make the array at least as long as the total number of inserts during the life of the program.



The initial queue. Static pointer q points to the dummy node.
Dummy points to front node. Rear node points to dummy.
All nodes are private; q is the only publically visible part of the queue.



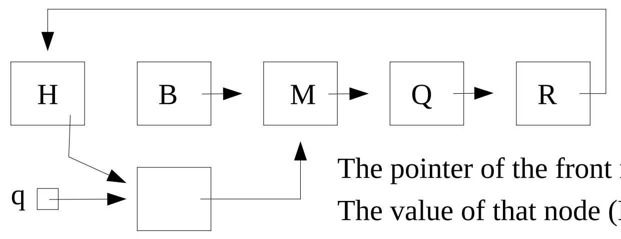
New local variable temp points to a new node.
H, the new datum is written into the dummy node.



The pointer of the (old) dummy node is copied
to the pointer of the new node.
The value of temp is copied to the pointer q

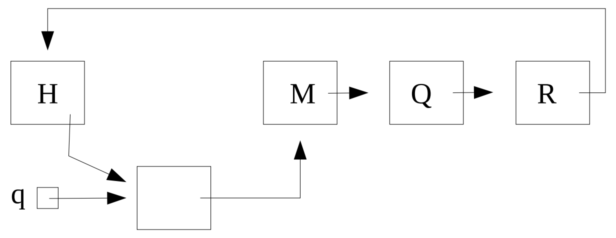
The new node becomes the dummy node, and the old dummy is the rear node.
temp is deallocated. Static q is still the only public part of the structure..

Now execute dequeue.



The pointer of the front node is copied to q.
The value of that node (B in this example)
is returned.

If memory space is a problem, deallocate the old front node.



Each of these solutions has disadvantages. Slide can lead to thrashing. Bigger only works if you know, in advance, how many inserts there will be. Wrap and Slide with seaparate pointers for front and rear have boundary condition issues; separate code might be needed for boundary cases.

The figure above illustrates the circular linked list with dummy node implementation of a queue q. If q contains n items, it is implemented as a circular list of $n+1$ nodes, one of which is the dummy node. The first picture shows q with 4 items: B, M, Q, R, in that order. The dummy node is shown blank. The dummy points to the front node, and the rear node points to the dummy. There is a pointer variable q which points to the front node. That pointer is the only externally visible part of the structure.

We execute enqueue(q,H), in the following steps.

- Write the new item, H, into the item field of the dummy node.
- Declare a local pointer variable temp which points to a new node, which will become the dummy node.
- The pointer field of the new dummy points to the front node of the queue, which has item B in the figure.
- The old dummy's pointer is changed to point to the new dummy node. The external variable q is now changed to point to the new dummy.

The structure has been updated. The node that contains H is the new rear node.

We now execute dequeue(q), in just one step. The pointer of the dummy node is changed to point to the second from the front node, shown as M in the figure. The previous front node, shown with B, is now inaccessible, causing a memory leak.