Hello Everyone

Course Code : CSE201

Course Title : Data Structure

Course Teacher:

Md. Jamal Uddin

Assistant Professor

Department of CSE,BSMRSTU,

Gopalganj-8100.

Presented By:

Tushar Sarkar

Student ID: 18CSE035

Department of CSE,BSMRSTU,

Gopalganj-8100.

Queue

Outline

- What is Queue?
- Queue Operations
- Illustration/Example
- Storing a Queue in Static Data
- Storing a Queue in Dynamic data
- Pseudo Code
- Complexity
- Advantages & Disadvantages
- Conclusion

What is Queue?

- A Queue is a linear data structure. The concept is quite similar with stack.
- Additions are made at the end or tail of the queue while removals are made from the front or head of the queue.
- Access system a queue is referred to FIFO(First-In First-Out) and LILO(Last-in Last-Out) structure.
- Example:

Queue Operations

- Add : adds a new node

  Add(X,Q)-> add the value X to the tail of queue.

- Remove : remove a node

  Remove(Q)-> removes the head node and returns its value.

- IsEmpty : reports whether the queue is empty

  IsEmpty(Q)-> report whether the queue is empty().

- IsFull : reports whether the queue is full

  IsFull(Q)-> report whether the queue is full.

- Initialize : cleats/initializes the queue

  Initilize(Q)-> create a new empty queue named Q.

- Destroy : deletes the contents of the queue

  Destroy(Q)-> delete the contents of the queue Q.

Illustration/Example

Operation	Queue's contents	Return value
1. Initialiaze(S)	<empty>	-
2. Add(A,Q)	A	-
3. Add(B,Q)	A B	-
4. Add(C,Q)	A B C	-
5. Remove(Q)	B C	A
6. Add(D,Q)	B C D	-
7. Remove(Q)	C D	B
8. Remove(Q)	D	C
9. Remove(Q)	<empty>	D

Storing a Queue in Static Data

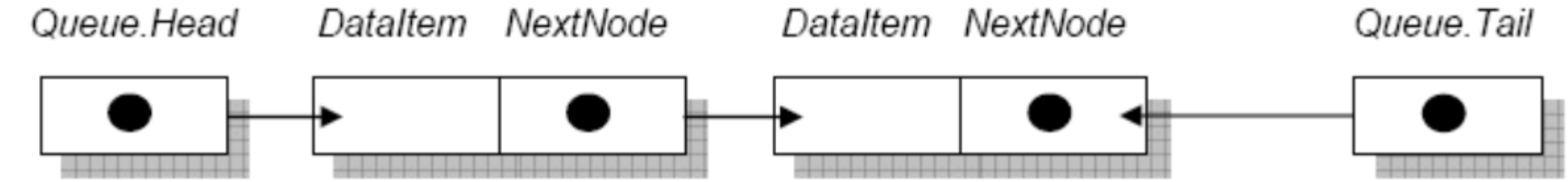
- This implementation stores the queue in an array
- The array indices at which the head and tail of the queue are currently stored must be maintained.
- The head of the queue is not necessarily at index 0.
- The array can be “circular array” – the queue “wraps round” if the last index of the array is reached.

Storing a queue in an array of length

Add(A,Q)		Head : 0 Tail : 0
Add(D,Q)		Head : 0 Tail : 1
Add(Z,Q)		Head : 0 Tail : 2
Remove(Q)		Head : 1 Tail : 2
Add(X,Q)		Head : 1 Tail : 3

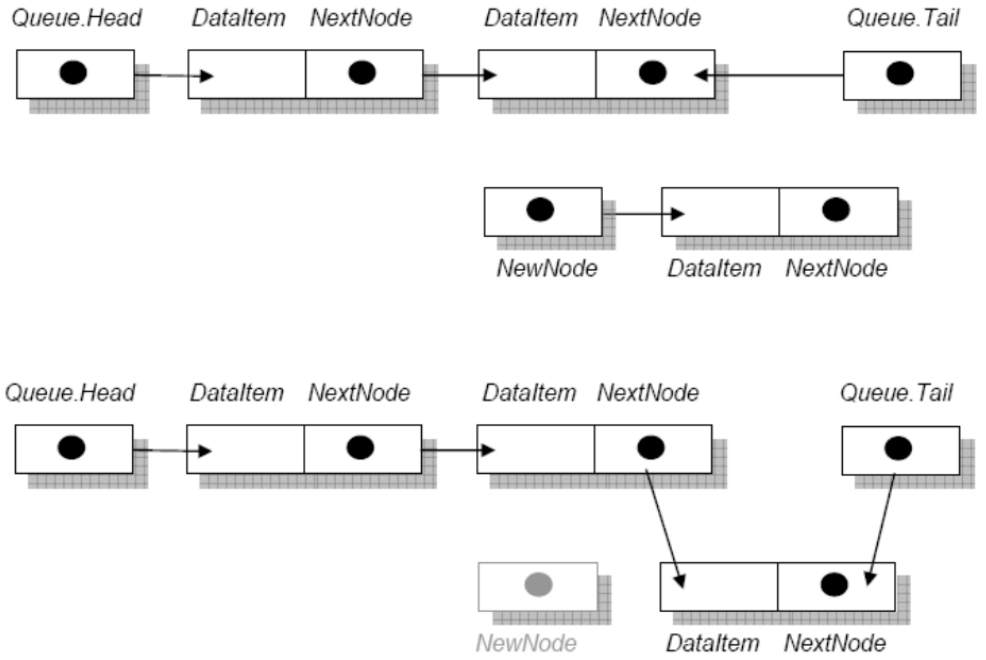
Storing a Queue in Dynamic Data

- Each node in a dynamic data structure contains data AND a reference to the next node.
- A queue also needs a reference to the head AND a reference to the tail node.
- The Following diagram describes the storage of a queue called Queue. Each node consists of data(DataItem) nd a reference (NextNode)



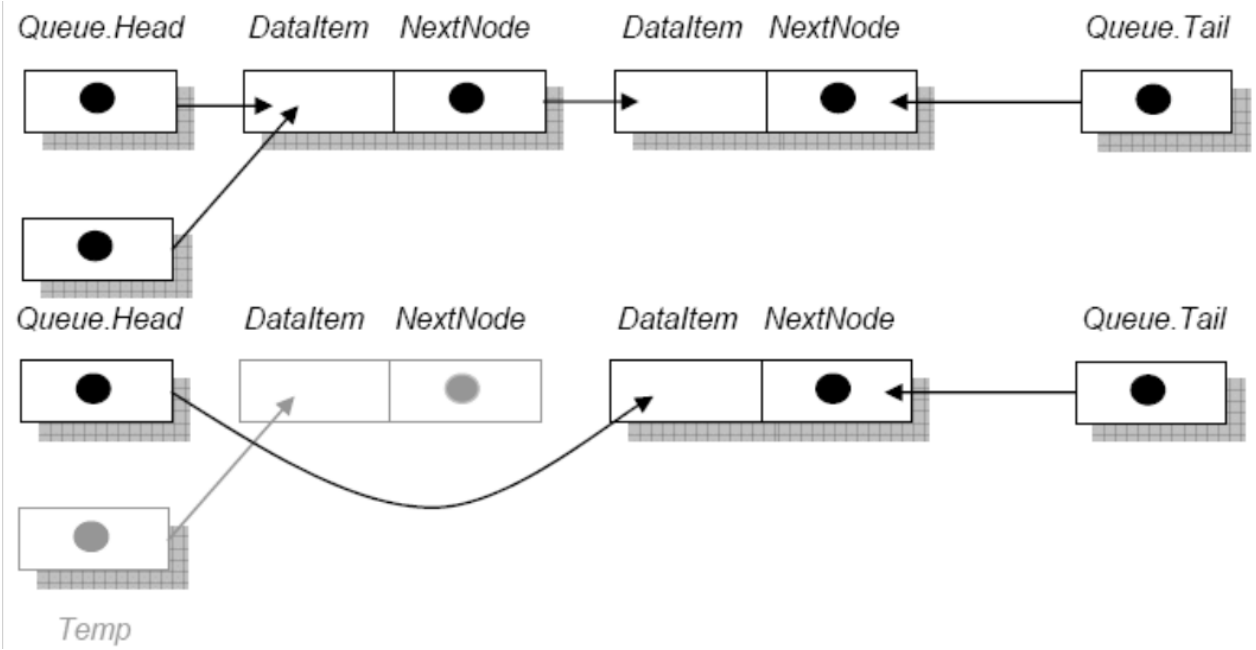
Storing a Queue in Dynamic Data

- The new node is to be added at the tail of the queue.
- The reference ***Queue.Tail*** should point to the new node, and the **NextNod**e reference of the node previously at the tail of the queue should point to the **DataItem** of the new node.



Storing a Queue in Dynamic Data

- The value of ***Queue.Head.DataItem*** is returned. A temporary reference Temp is declared and set to point to head node in the queue (**Temp=Queue.Head**)
- ***Queue.Head*** is then set to point to the second node instead of the top node.



Pseudo Code

D_Shortest_Path_Queri | A_Reconnaissance.cpp | Linear_Search.cpp | Quick_Sort.cpp | Selection_Sort.cpp | Queue.cpp | B_Borze.cpp | C_Fair_Candy_Distribution.cpp | 11631_Dark_roads.cpp

```cpp
#define     PI          3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534 2

void Show_Queue(queue<int>Q){
    while(!Q.empty()){
        cout << Q.front() << " ";
        Q.pop();
    }
    cout << endl;
}

int main(){
    int N,i,t;
    cout << "Enter Queue Size: ";
    cin >> N;

    queue<int>Q;
    cout << "Enter the elements of queue : ";
    for(i=0; i<N; i++){
        cin >> t;
        Q.push(t);
    }

    cout << "The queue is : ";
    Show_Queue(Q);

    cout << "Queue Front Value is : " << Q.front() << endl;
    cout << "Queue Back Value is : " << Q.back() << endl;

    Q.pop();
    cout << "After One element remove then queue is : ";
    Show_Queue(Q);


    biday;
```

```
C:\WINDOWS\system32\cmd.exe - pause
Enter Queue Size: 5
Enter the elements of queue : 6 7 3 4 8
The queue is : 6 7 3 4 8
Queue Front Value is : 6
Queue Back Value is : 8
After One element remove then queue is : 7 3 4 8
Press any key to continue . . .
```

Complexity

- The complexity of enqueuer and dequeuer operations in a queue using array is O(1).
- If you use pop(N) in code, then the complexity might be O(n) depending on the position of the item to be popped

Advantages of Queue

- ✓The main advantages is that adding or removing elements can be done quickly and efficiently because you would just add elements to the end of the queue or remove them from the front of the queue.
- ✓Queues have the advantages of being able to handle multiple data types and they are both flexible and flexibility and fast.
- ✓Moreover, queues can be of potentially infinite length compared with the use of fixed-length arrays.

Disadvantages of Queue

- ✓The main disadvantages is that a queue is not readily searchable (sure there are hacks around it but it is not supposed to be searchable) and adding or removing elements from the middle of the queue is very complex (again it is not MEANT to allow elements to be removed from the middle).
- ✓The **queue** is not readily searchable.
- ✓You have to start from the end and might have to maintain another **queue**.
- ✓So if you have some data, which later on you would want to be searchable, then don't even think about using a **queue**.
- ✓ Adding or deleting elements from the middle of the **queue** is complex as well.

Conclusion

- The **queue** data structure is a linear type of data structure that is used to store the elements. ...
- A **queue** data structure used an array or linked list during its implementation.
- Insertion in **queue** occurs at the REAR end, and deletion from **queue** occurs at the FRONT end.

Thank You