Queue

Definition
→ An ordered collection of homogeneous data items. where element are added at rear & removed from the front end

Queue ADT
(QT)

1) Abstract QueueType CreateQueue()
→ Precondition: none
→ Post condition: Empty Queue is created

2) Abstract QT Enqueue (QT Queue, Element Type Element)
→ Pre: Queue not full = True
→ Post: Queue = Queue' + Element at the rear

3) Abtract Element Type dequeue (QT Queue)
→ Pre: Queue not Empty = True
→ Post: Dequeue = Element at front, Queue = Queue' - Element at front

4) Abstract DestroyQueue (QT Queue)
→ Pre: Queue not Empty = True
→ Post: Element removed one by one from front to rear till not Empty (Queue) = False

5)

5) Abstract Boolean NotFull(QT Queue)
→ Pre: none
→ Post: NotFull (Queue) = True → Not Full
NotFull (Queue) = False → Full

6) Abstact Boolean NotEmpty (QT Queue)
→ Pre: none
→ Post: Not Empty (Queue) = True → not Empty
Not Empty (Queue) = False → Empty

Types of Queue
1) Simple Queue
→ Insert at rear
→ Deletes at front
→ Array Implementation

1) Algo QT Create Queue ()
{
front = rear = -1;
return queue;
}

2) Algo QT Enqueue (QT Queue, ElementType Element) (ET)
{
if NotFull(Queue) = True
{
Queue [++rear] = ~~value~~ Element;
if (front == -1) then front = 0
}
else "Error"

3) Algo ET Dequeue (QT Queue)
{
if Not Empty (Queue) = True
{ temp = Queue [front];
if (front == rear) then front = rear = -1;
else front ++ ;
return(temp);
}
else "Error" }

4) Abstract DestroyQueue (QT Queue)
{ if Not Empty (Queue) = True
while (Not Empty(Queue))
Print Dequeue (Queue);
else "Error" }

5) Abstract Boolean NotFull (QT Queue)
{
if(rear != MaxSize - 1)
return True;
else return False;
}

6) Abstract Boolean Not Empty (QT Queue)
{
if (front != -1)
return True;
else return False;
}

→ Linked List

```
Struct NodeType
{
ET Element;
NodeType * Next;
}
```

Let's call it SNT

1) Algo QT Create Queue ()

```
{
create Node(front);
create Node(rear);
front = rear = NULL;
}
```

2) QT Enqueue(QT Queue, NodeType NewNode

```
{
if (front == rear == NULL)
{
NewNode -> next = NULL;
front = rear = New Node;
}
else rear -> Next = NewNode;
     rear = NewNode
}
```

3) Algo ET DeQueue (QT Queue)

```
{
if (front == rear == Null)
Print "underflow";
exit;
```

else { Creat Node (Temp);
Temp = front;
front = front -> next;
if (front = NULL)
rear = NULL;
return (Temp -> data);
}

4) Abstract Destroy Queue (QT Queue)
{ if front == NULL
print "Underflow";
exit;
else
create Node (Temp);
While (Not Empty (Queue))
return DeQueue (Queue);
}

5) Abstract Display Queue (QT Queue)
{
if front == NULL
print "Error"
else {
create Node (Temp);
Temp = front;
While (Temp != NULL)
print (Temp -> Data
Temp = Temp -> Next;
}
}

2) Circular Queue
→ Last node is connected to first Node
→ Deletion at front
→ Insertion at rear.
→ Array Implementation

1) Algo QT Create CQueue()
{
front = -1;
rear = -1;
return queue;
}

2) Algo QT CEnqueue (QT CQueue, ET Element)
{
if NotFull(CQueue) = True
{
if(rear == size - 1 && front != 0)
rear=0;
else rear = rear+1;
CQueue[rear]=Element;
if (front == -1) then front = 0 }
Else "Error";
}

3) Algo ET Dequeue (QT CQueue)
{ if NotEmpty(CQueu) = True
{ temp = CQueue [front];
if (front == rear) then front = rear = -1;
else if (front = size -1) the front = 0;
Else front ++;

return(temp);
}
Else "Error"
}

4) Abstract DestroyQueue (QT (Queue)
{
if NotEmpty ((Queue) = True
while (NotEmpty(CQueue))
print Dequeue ((Queue)
else "Error
}

5) Abstract Boolean NotFull (QT (Queue)
{
if (rear = size - 1 && front == 0 || rear
== front == -1)
return False;
else return True;
}

6) Abstract Boolean NotEmpty (QT (Queue)
{
if(front != -1)
return True;
else return False;
}

→ Linked List

SNT

1) Algo QT createQueue()
```
{
Create Node(front);
create Node(rear);
front = rear = NULL;
}
```
2) QT Enqueue(QT (Queue, NodeType NewNode)
```
{
if (front == rear == NULL)
  front = rear = NewNode;
  rear -> next = NewNode;
else
  temp = front;
  while(temp! = rear)
  {
  temp = temp -> next;
  temp -> next = NewNode;
  NewNode -> next = rear -> next;
  rear = NewNode;
  }
}
```
OR
```
if (front == rear == NULL)
front = rear = NewNode;
rear -> next = NewNode;
```

```
else
 rear->next = NewNode;
 rear = NewNode;
 NewNode -> next = front;
 }
```

3) Algo ET DeQueue (QT(Queue)

```
{
if(front == rear == NULL)
Print "Underflow"
exit;
else if(front == rear)
 { temp = front;
 front = rear = NULL;
 return(temp->data);
 }
else {
   temp = front;
   front = front -> next;
   rear -> next = front;
   return (temp -> data);
   }
 }
```

4) Abstract Destroy Queue (QT(Queue)

```
{
if front == NULL
Print "Underflow"
exit;
```

Else { CreateNode (temp);
while (NotEmpty (cQueue))
return (DeQueue (cQueue)
}
}

5) Abstract Display Queue (QT (Queue)
{
if front == NULL
Print "Error"
else
{ Create Node (Temp);
Temp = front;
While (Temp != NULL)
Print (Temp → data);
Temp = Temp → next;
}
}

3) Doubly ended Queue
→ Deletions & insertion's can be done at both the ends.
→ Has two pairs od fronts & rears.
→ Array Implementation
1) Algo QT CreateDQueue()
{
~~front1 = -1;~~
front1 = rear 1 = -1;
front2 = rear 2 = -1;
Return DQueue;
}
2) Algo QT DEnqueue (QT DQueue, ET Element, int end)
{
id (end == 2 & rear 2 == 0) then Left End = Full;
exit;
id (end == 1 & rear 1 == Max-1) ~~then Right End~~
then Right End = Full;
exit;
id (rear 1 == -1)
{
front 1 = front 2 = rear 1 = rear 2 = Max/2;
DQueue[rear 1] = element;
}
else id (end == 1)
DQueue[++rear 1] = element;
front 2 = rear 2;

else if (end == 2)
DQueue [--rear2] = element;
front1 = rear2;
}

3) Algo ET Dequeue (QT DQueue, int end)
{
if (front 1 == -1)
then underflow;
exit;
if (front1 == rear1 == front2 == rear2) {
temp = DQueue [front1]
front1 = front 2 = rear 1 = rear 2 = -1;
}
else if (end == 1) {
temp = DQueue [front 1]
front++, rear++;
}
else if (end == 2) {
temp = DQueue [front 2];
front2 --, rear 2 --;
}
return (temp)
}

4) Abstract DestroyQueue (QT DQueue)
{
if NotEmpty (DQueue) = true
while (NotEmpty (DQueue))
Print Dequeue (DQueue)
else "Error"
}

5) Abstract Boolean NotFull (QT DQueue)
{
if (front1 != 0 || rear2 != size - 1)
return True;
else return False;
}

6) Abstract Boolean NotEmpty (QT DQueue)
{
if (front1 != -1 || front2 != -1)
return True;
else return False;
}

→ Linked List

SNT

1) Algo QT CreateQueue()
{
CreateNode (front1);
CreateNode (rear1);
CreateNode (front2);
CreateNode (rear2);
front1 = rear1 = front2 = rear2 = NULL;
}

2) QT Enqueue(QT DQueue, NodeType NewNode, int end)
{
if (rear 1 = = NULL)
front 1 = rear1 = front2 = rear 2 = NewNode;
else if (end = = 1
{
rear -> next = NewNode;
front 2 = NewNode;
rear1 = NewNode;
}
else if (end = = 2)
{
NewNode -> next = rear2;
rear2 = NewNode;
front 1 = NewNode;
}
}

3) Algo ET DeQueue(QT Dqueue , int end)

```
{
if ( front1 == NULL)
print "Underflow"
exit;
else if (front1 == rear1)
{
temp = front1
front1 = rear1 = front2 = rear2 = NULL;
return(temp -> data);
Else if ( end == 1)
{
temp = front1;
front1 = front1 -> next;
rear2 = front1 or ( rear2 = rear2 -> next)
return (temp -> data);
}
Else if (end == 2)
{
temp = front2
temp2 = front1
while (temp2 -> next != front2)
temp2 = temp2 -> next;  // LOOP
rear1 = temp2;
front2 = temp2;
rear1 -> next = NULL
return (temp -> data);
}
}
```

4) Abstract Destroy Queue (QT DQueue)
{
if (front1 = = NULL)
Print "underflow"
exit;
Else
{
createNode (temp);
while (Not Empty (DQueue)) {
return (Dequeue (DQueue, 1)); }
}
}

5) Abstract Display Queue (QT DQueue)
{
Create Node (Current1) [Front to Rear)
Current1 = front1
if (Current1 = = NULL)
"Error"
else {
while (Current1 != NULL)
print current1 -> data;
~~current1 = current1 next~~; Current1 = Current1 -> next
}
}
{
Create Node (Current2); [Rear to front]
Current2 = front2;

```
id (Current2 = = NULL)
 "Error"
else {
 while (Current 2 = = NULL)
  print current 2 -> data;
 current 2 = current 2 -> next;
 }
}
```

4) Priority Queue

→ Every element has predefined Priority
- Max Priority → Max Priority element removed
- Min Priority → Min " "
→ Definition
A collection of heterogeneous elements, accessed in FIFO manner where in each element has an additional priority associated with it.
→ Types
- Min Priority Queue - smaller no°, higher Priority
- Max " " - Larger no°, " "
→ $\boxed{\text{Array implementation}}$

```
Struct PriQueue {
int data;
int priority;
}
Struct PriQueue PQ[MaxSize];
```

1) Algo QT CreatePQueue()
```
{
front = rear = -1;
}
```

2) Algo QT PEnqueue(QT PQueue, ET Element, int P)
```
{ Struct PriQueue key;
if (rear == MaxSize-1) then Overflow; exit;
else if (front = rear = -1)
{ front = rear = 0;
PQ[rear].data = Element;
PQ[rear].priority = P;
}
```

```
}else
{ rear++;
PQ[rear].data = Element;
PQ[rear].Priority = P;
Key = PQ[rear];
j = rear - 1;
while(j>=0 && PQ[j].Priority < Key.Priority)
{
PQ[j+1] = PQ[j];
j--;
}
PQ[j+1] = Key;
}
```

3) Algo ET Dequeue(QT PQueue)

```
{ struct PriQueue Temp;
if (front == -1) then underflow; exit;
if (front == rear)
{
Temp = PQ[front];
front = rear = -1;
}
else
{
Temp = PQ[front];
front++;
}
return (Temp);
}
```

4) Abstract Destroy Queue (QT PQueue)
{
if NotEmpty (PQueue) = True
while (NotEmpty (PQueue))
Print Dequeue (PQueue)
else "Error"
}

5) Abstract Boolean Not Full (QT PQueue)
{
if (~~front != -1 &&~~ rear != MaxSize - 1)
return True;
else return False;
}

6) Abstract Boolean NotEmpty (QT PQueue)
{
if (front != -1)
{ return True;
~~ret~~ else return False;
}

→ Linked List

struct NodeType{
ET Element;
integer priority;
Nodetype next;
}

1) Algo QT CreateQueue()
{
createNode (front);
CreateNode (rear);
front = rear = NULL;
}

2) ~~Algo~~ QT Enqueue (QT PQueue, NodeType NewNode, int P)
{
if (rear == NULL) // first insertion
front = rear = NewNode;
else if (front→Priority > NewNode→Priority)
{ NewNode →next = front; // Before first Node
front = NewNode;
}
else
{
temp = front; Current = NULL;
While (temp→ prioriety < NewNode→ Priority
&& temp→next != NULL)
Current = temp; temp = temp→next;

if (temp -> prioripty > NewNode -> prioripty
NewNode -> next = temp; // In Between
Current -> next = NewNode;
if (temp -> next == NULL)
temp -> next = NewNode;
rear = NewNode;
}

}

3) Algo: ET Dequeue (QT PQueue)
{
if (front == NULL)
Print "underflow"
exit;
Else if (front == rear)
{
temp = front;
front = rear = NULL;
return (temp -> data);
}
else
{
temp = front;
front = front -> next;
return (temp -> data);
}
}

4) Abstract ~~QT~~ Destroy Queue(QT PQueue)
{
if(front == NULL)
print "Underflow"
exit;
else
{
CreateNode(temp);
while(NotEmpty(PQueue))
return(Dequeue(PQueue));
}
}

5) Abstract DisplayQueue(QT PQueue)
{
if front==NULL
Print "Error"
exit;
else
{
CreateNode(Temp);
Temp=front;
While(Temp!=NULL)
Print(Temp->data)
Temp = Temp->next
}
}

Catagory	Stack	Queue
Definition	It's a linear ds that ~~flows~~ follows LIFO (last in first out) where the last element added is first removed	It's a linear ds that follows FIFO (First in First out) where the first element added is first removed
Primary Operations	Push (add to top) Pop (Remove)	Enqueue (add to rear) Dequeue (remove front)
Used	Backtracking Reversing Call stack	BFS algorithms Task schduling Network requests