CHARUTAR VIDYA MANDAL'S
SEMCOM
Vallabh Vidyanagar

Faculty Name: Ami D. Trivedi
Class: FYBCA
Subject: US02CBCA01
(Advanced C Programming and Introduction to Data Structures)

*UNIT – 4 (Some More Data structures - Queue and Linked lists)

*QUEUES

SIMPLE QUEUE

Another important type of lists allows deletion to be performed at one end of list and insertion at the other end.

The information in such list is processed in the same order as it was received. That is first-in first-out **(FIFO)** or a first-come first-serve **(FCFS)** basis. This type of a list is known as Queue.
Definition: Queue is a linear non primitive data structure in which insertion of an element occurs at one end and deletion occur at other end.

Representation of a queue

A queue can be represented by a vector as follow:

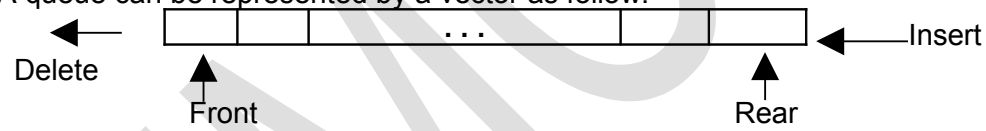


Fig: Representation of a queue by a vector

A vector representation of a queue requires two pointers **f and r. f** denotes the positions of its front (leftmost) elements and **r** represents rear (rightmost) elements.

EXAMPLES OF QUEUE

1. A checkout line at the supermarket cash register. Here the first person in line will check out first.
2. The line of cars waiting at traffic signal. The first car at signal will be deleted (move ahead) first. A car will be inserted (joining) at the end of the line of existing waiting cars.
3. A queue in a time-sharing computer machine where many users share the system simultaneously. Such a system has a single CPU and one main memory. These resources must be shared. It allows one user's program to execute for a short time. Then execution of another user's program will be done and so on. This will be done until it returns to the execution of the initial user's program. The user programs that are waiting in a waiting queue may be processed based on FIFO.

CHARACTERISTICS

A queue data structure has following characteristics:

- Its nature is FIFO (First In First Out).
- The insertion and deletion operation occur at opposite end.

OPERATIONS ON QUEUE

Following operations can be performed on Queue.

1. Insert	Used to insert an element in a queue.
2. Delete	Used to delete an element from a queue.

1. QINSERT(Q, F, R, N, Y)

This function inserts element Y at the rear of queue.

Q	a queue containing N elements
F	a pointer to front element of a queue
R	a pointer to rear element of a queue
Y	element to be inserted at the rear of queue

Before the first call of this function, F and R have been set to zero.

1.	[Overflow?] If R ≥ N then Write ('OVERFLOW') Return
2.	[Increment rear pointer] R ←- R + 1
3.	[Insert element] Q [R] ← Y
4.	[Is front pointer properly set?] If F = 0 then F ←1 Return

2. QDELETE (Q, F, R)

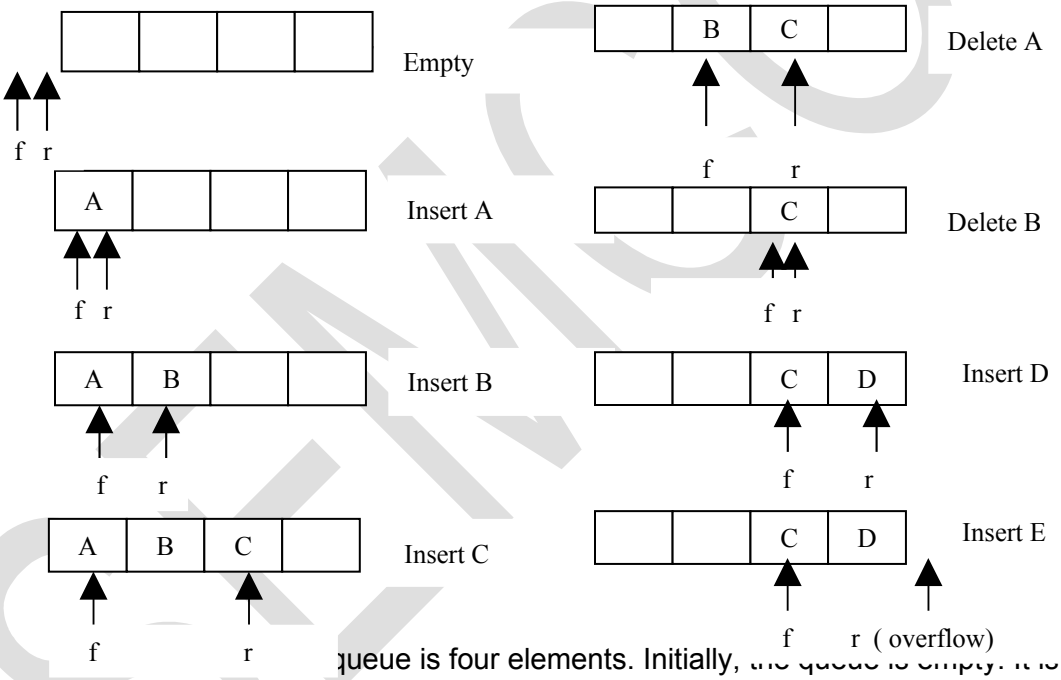
This function will delete an element from the front of queue. And returns this element.

Q	a queue containing N elements
F	a pointer to front element of a queue
R	a pointer to rear element of a queue

Y is a temporary variable.

1.	[Underflow?] If F = 0 then Write (' UNDERFLOW') Return (0) (0 denotes an empty queue)
2.	[Delete element] Y ← Q [F]
3.	[Queue Empty?] If F = R then F ←R ← 0 else F ←- F + 1 (increment front pointer)
4.	[Return element] Return(Y)

Trace of operations on a simple queue

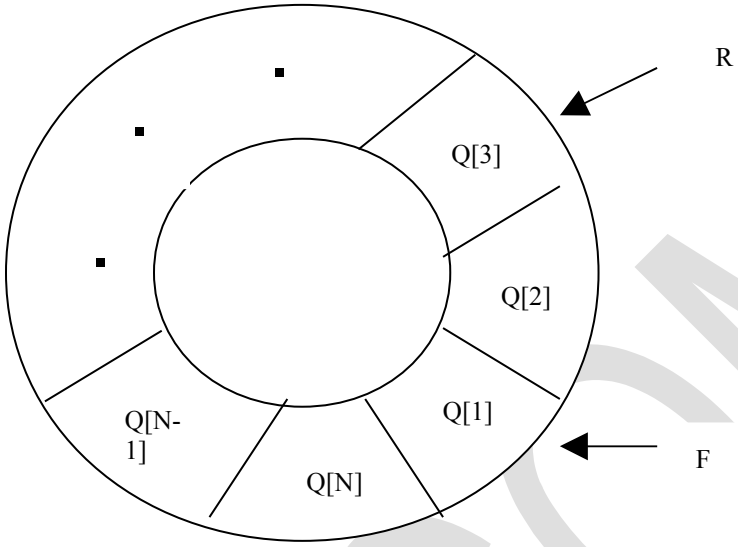


queue is four elements. Initially, the queue is empty. It is required to insert symbols 'A','B' and 'C', delete 'A' and 'B' and insert 'D' and 'E'. A trace of the contents of the queue is given in following figure. Note that an overflow occurs on trying to insert symbol 'E', even though the first two locations are not being used.

CIRCULAR QUEUE

Circular queue is more suitable method of representing a queue. It stops an unnecessary use of memory. It arranges the elements Q[1],Q[2],…,Q[n] in a circular fashion with Q[1] following Q[n].

Representation of a circular queue



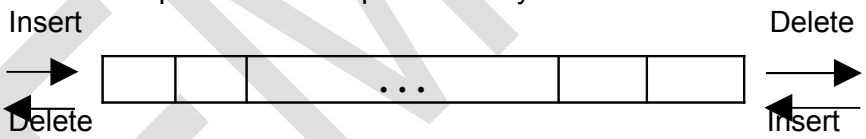
Definition: A queue in which the elements are arranged in the circular fashion is known as Circular queue.

DOUBLE-ENDED QUEUE / DEQUE

It is a linear list in which insertions and deletions are done at both end of the structure. It is also known as Deque.

Representation of a doubly ended queue

A double ended queue can be represented by a vector as follow:



There are two types of a deque:

1. Input-restricted deque: It allows the insertion at only one end and deletion at both ends.
2. Output-restricted deque: It allows the insertion at both ends and deletion at only one end.

APPLICATION OF DQUEUE

- Simulation

***LINKED LIST**

INTRODUCTION

Linked Allocation

The storage of data where some kinds of links or pointers are used to access the elements is known as linked allocation. This makes the elements adjacent to each other, not physically, but logically.

Linked list

Linked list is a data structure in which each node contains data field as well as pointer field pointing to other nodes in the list.

Node

A typical element or node consists of two fields –

- Information fields called **INFO** and
- Pointer field called **LINK**

The name of element is **NODE**.

Node structure is as follows:

NODE

INFO	LINK

ADVANTAGES OF LINKED LIST

1. Insertion and deletion of an element in array requires lot of processing of moving element. This movement requires time and it is tedious. The same operation with linked list requires just changing of 2 or 3 pointer values.

2. An array size is fixed. So there will be wastage of memory for an application where memory requirements are unknown. But it uses memory as per requirements. Means nodes are created only when they are required.

3. It is a Dynamic structure. i.e. Memory allocated at run-time.

4. We can have information fields of more than one datatype.

DISADVATAGES OF LINKED LISTS

1. In linked list, if we want to access any node it is difficult and time consuming.

2. It is occupying more memory. Linked list uses pointers which are overhead.

TYPES OF LINKED LIST

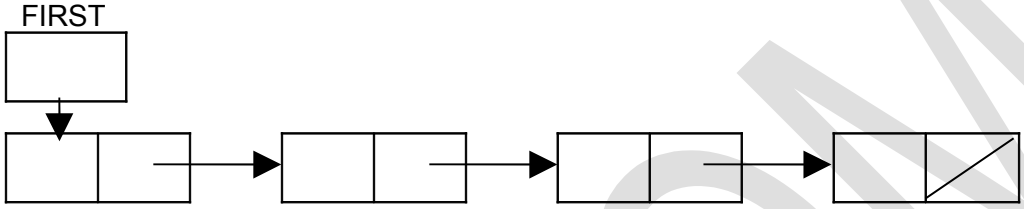
1. Singly linked list
2. Circular linked list
3. Doubly linked list

1. SINGLY LINKED LIST

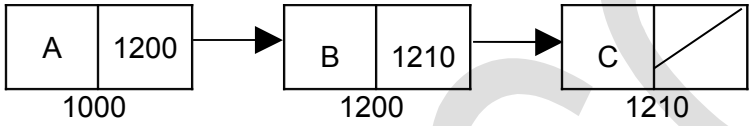
It is a linked list in which each node contains two fields. One contains data and other contains pointer to the next node in the list.

Definition: A linked list in which pointer fields of each node is pointing to the next node is known as singly linked list. An information fields contains the actual data about node.

Thus a list can be viewed as follows:



Example: FIRST = 1000



FIRST is a variable, which will always point to the first node of the list.

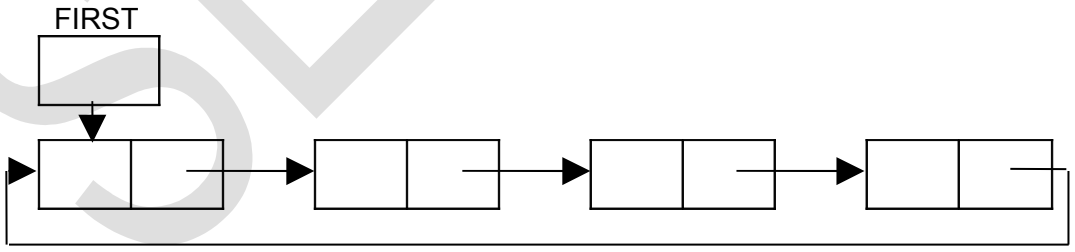
The last node of the list always contains NULL in link field. NULL is a special value that is used as delimiter.

Disadvantage

Sometimes we want to go back to the previous element in a linked list. For that we have to keep track of previous element by using PRED (predecessor) pointer. With PRED pointer we can keep traverse again from FIRST to come to position of PRED.

2. CIRCULAR LINKED LIST

Definition: A Singly linked list in which pointer fields of last node is pointing to the first node is known as circular linked list.



Advantages

1. It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes.
2. If we are at a node, then we can go to any node. But in linear linked list it is not possible to go to previous node.

Disadvanges

1. It is not easy to reverse the linked list.
2. If proper care is not taken, then the problem of infinite loop can occur.
3. If we are at a node and want to go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through in between nodes and then we will reach the required node.

3. DOUBLY LINKED LIST

In some applications, it is necessary that a list should be traversed (pass through) in either a forward or reverse manner. To do this we need two link fields in each node of linked list.

These linked fields are used to denote predecessor (ancestor) and successor of the node.

The link denoting the predecessor of the node is called the left link. The link denoting the successor is called right link. A list containing this type of node is called doubly linked linear list or a two-way chain.

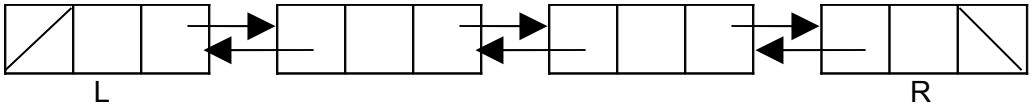
Definition: A linked list which has an information field and two pointers fields is known as doubly linked list. Each of the pointer field contains address of left or right node. An information field contains the actual data about node.

A node structure of doubly linked list is as follows:

LPTR – pointer to the left node
RPTR – pointer to the right node
INFO – actual value

In doubly linked list,

- An address of the first node (leftmost node) is represented by L.
- An address of the last node (rightmost node) is represented by R.
- A left pointer field of first node and right pointer field of last node are set to NULL.



Advantage

Doubly linked list is using the link to the preceding as well as succeeding node. So we can traverse in both directions in a linked list. This provides faster searching and deletion.

Disadvantage

List uses extra memory pointer to store each element. This increases the overhead associated with it. Means to store a single element of data like 'A', two extra pointers are required. But this technique is fruitful when each element requires more space.

***ALGORITHMS**

1. An algorithm for inserting a node at the first position in the linked list

Function INSERT(X, FIRST)

X	new element
FIRST	a pointer to the first element of a linked list
AVAIL	pointer to the top element of the availability stack
NEW	temporary pointer variable

This function inserts X. It is required that X precedes the node whose address given by FIRST.

1.	[Underflow ?] If AVAIL = NULL then Write ('AVAILABILITY STACK UNDERFLOW') Return (FIRST)
2.	[Obtain address of next free node] NEW ⬅ AVAIL
3.	[Remove free node from availability stack] AVAIL ⬅ LINK (AVAIL)
4.	[Initialize fields of new node] INFO (NEW) ⬅ X LINK (NEW) ⬅ FIRST
5.	[Return address of new node] Return (NEW)

2. An algorithm for inserting a node at the first position in the linked list

Function INSEND (X, FIRST)

X	new element
FIRST	a pointer to the first element of a linked list
AVAIL	pointer to the top element of the availability stack
NEW	temporary pointer variable
SAVE	temporary pointer variable

This function inserts X. It is required that X is to be inserted at the end of the list.

1.	[Underflow ?] If AVAIL = NULL then Write ('AVAILABILITY STACK UNDERFLOW') Return (FIRST)
2.	[Obtain address of next free node] NEW ⬅ AVAIL
3.	[Remove free node from availability stack] AVAIL ⬅ LINK (AVAIL)
4.	[Initialize fields of new node] INFO (NEW) ⬅ X LINK (NEW) ⬅ NULL
5.	[Is the list empty?] If FIRST = NULL then Return (NEW)
6.	[Search for the last node] SAVE ⬅ FIRST
7.	[Search for end of list node] Repeat while LINK (SAVE) != NULL SAVE⬅ LINK (SAVE)
8.	[Set LINK field of last node] LINK (SAVE)⬅ NEW
9.	[Return first node pointer] Return (FIRST)

3. <u>An algorithm for inserting a node in the increasing order of the data stored in the INFO field of liner linked list</u>

Function INSORD (X, FIRST)

X	new element
FIRST	a pointer to the first element of a linked list
AVAIL	pointer to the top element of the availability stack
NEW	temporary pointer variable
SAVE	temporary pointer variable

This function inserts X. It is required that X be inserted so that it preserves the ordering of the term in increasing order of their INFO fields.

1.	[Underflow ?] If AVAIL = NULL then Write ('AVAILABILITY STACK UNDERFLOW') Return (FIRST)
2.	[Obtain address of next free node] NEW ⬅ AVAIL
3.	[Remove free node from availability stack] AVAIL ⬅ LINK (AVAIL)
4.	[Initialize field of new node] INFO (NEW) ⬅ X
5.	[Is the list empty?] If FIRST = NULL then LINK (NEW) ⬅ NULL Return (NEW)
6.	[New node precedes all other nodes?] If INFO (NEW) <= INFO (FIRST) then LINK (NEW) ⬅ FIRST Return (NEW)
7.	[Initialize temporary pointer] SAVE ⬅ FIRST
8.	[Search for predecessor of new node] Repeat while LINK (SAVE) != NULL and INFO (LINK (SAVE)) <= INFO (NEW) SAVE ⬅ LINK (SAVE)
9.	[Set LINK field of new node and its predecessor] LINK (NEW) ⬅ LINK (SAVE) LINK (SAVE) ⬅ NEW
10.	[Return first node pointer] Return (FIRST)

4. <u>An algorithm for deleting a node whose address is X in the linked list</u>

Function DELETE (X, FIRST)

X	Address of a node in linked list
FIRST	a pointer to the first element of a linked list
AVAIL	pointer to the top element of the availability stack
TEMP	temporary pointer variable
PRED	temporary pointer variable which keeps track of the predecessor of TEMP

Note: This function will delete by address.

1.	[Empty list?] If FIRST = NULL then Write ('UNDERFLOW') Return
2.	[Search for X] TEMP ⬅ FIRST
3.	[Find X] Repeat while TEMP != X and LINK (TEMP) != NULL PRED ⬅ TEMP TEMP ⬅ LINK (TEMP)
4.	[End of the list ?] If TEMP != X then Write ("NODE NOT FOUND") Return

5.	[Delete X]
	If X = FIRST (is X first node ?) then
	FIRST ⬅ LINK (FIRST)
	Else
	LINK (PRED) ⬅ LINK (X)

6.	[Return node to availability area] LINK (X) ⬅ AVAIL AVAIL ⬅ X Return

APPLICATION OF LINKED LIST

- Polynomial manipulation
- Linked dictionary
- Multiple precision arithmetic
- Stack, Queue and Tree

Disclaimer

The study material is compiled by Ami D. Trivedi. The basic objective of this material is to supplement teaching and discussion in the classroom in the subject. Students are required to go for extra reading in the subject through library work.