14) Write algorithms for Singly / Doubly & Circular Linked List.

⇒ Singly Linked List.

① Insert at first position
② Insert at last position
③ Insert in Ordered Linked List
④ Delete element

① Insertion at first position

```
① [Underflow ?]
      If AVAIL = NULL
        then Write ("Underflow")
        return (Avail)

② [Initialize new node]
        new ← avail

③ [Remove free node]
        ~~new → In~~
        Avail ← Link (Avail)

④ [Initializes field of new node]
        new → Info ← x
        new → Link ← start

⑤ Start = New

⑥ [Finished]
        Return (new)
```

② Insertion at last

```
① If  Avail = Null
      then write ("Stack Underflow")
      return ( Avail )

② [Initialize a new node ]
      New ← Avail

③ [Remove free node from available Stack]
      Avail ← Avail → link

④ [Initializes field of a new node]
      Info (New) ← x
      Link (New) ← Null

⑤ [Is the list empty ?]
      If  first = NULL
      then Return( first)

⑥ [Initialize search for the last node]
      save ← first

⑦ [Search for and of list]
      Repeat while (Link (save) ≠ NULL)
          save ← link (save)

⑧ [Set link field of list node of new]
      Link (save) ← New

⑨ [Return first node]
      return (first)
```

③ Insertion at middle

```
① [Underflow ?]
      If avail = NULL
          than write ("Stack underflow")
          return (avail)
② [Initialize a new node]
      New ← avail
③ [Remove free node from available stack]
      Avail ← avail → link
④ [Is the list empty]
      If first = NULL
      then Link (New) ← NULL ~~FIRST~~
          Return (New)
⑤ [Does the new node precede all other node in the li
      If  INFO (NEW) ≤ INFO(FIRST)
      then  LINK (NEW) ← FIRST
          Return (NEW)
⑥ [Initialize temporary pointer]
      Save ← First
⑦ [Search for predecessor of new node]
      Repeato while  Link (SAVE) ≠ NULL &
                  INFO (NEW) ≥ INFO (LINK (SAVE))
              SAVE ← LINK (SAVE)
⑧ [Set link field of New node and its predecesso
      LINK (NEW) ← LINK (SAVE)
      LINK (SAVE) ← NEW
⑨ [Return first node pointer]
      Return (FIRST)
```

④ Delete the ~~Oper~~ Element

```
① [Is Empty list ?]
      If FIRST = NULL
      then write (Underflow)
      return
② [Initialize search for x]
      Save ← First
```

```
(3) [Find X]
      Repeat through (5) while
          SAVE ≠ X  and Link (Save) ≠ NULL

(4) [Update predecessor marker]
      PRED ← SAVE

(5) [MOVE to next node]
      SAVE ← Link (SAVE)
(6) [End of the list]
      If SAVE ≠ X
      then write ('Node not found')
      return

(7) [Delete X]
      If X = First
        then First ← Link (First)
        else Link (PRED) ← Link(X)

(8) [FREE Deleted Node]
      FREE(X)
```

⇒ Doubly Linked List.

1. Insertion at begin
2. Insertion at last
3. Insertion at middle
4. Deletion at begin
5. Deletion at last
6. Deletion at middle

(1) Insertion from begining.

```
(1)
      Avail ← NULL
(2)
      NEW ← AVAIL
```

```
(3)
    AVAIL ← Avail → Next
(4)
    New → Pre ← Null
(5)
    New → data ← Value(x)
(6)
    New → Next ← Start
(7)
    Start → Pre ← new
(8)
    Start ← New
(9)
    Exit
```

(2) Insertion at end.

```
(1)
    Avail ← NULL
(2)
    New ← Avail
(3)
    Avail ← Avail → Next
(4)
    New → Next ← Null
(5)
    New → data ← value
(6)
    ptr ← start
(7)
    While (ptr → next != NULL)
(8)
        ptr ← ptr → next
(9)
    ptr → next ← new
(10)
    New → pre ← ptr
```

(3) Insertion at Middle

```
(1) Avail ← NULL
(2) New ← Avail
(3) Avail ← Avail → Next
(4) New → data ← value
(5) ptr ← start
(6) Repeat Step 4 : ptr → data ≠ value
(7) ptr ← ptr → next
(8) New → Next ← ptr → next
(9) ptr → next → pre ← new
(10) New → prev ← ptr
(11) ptr → next ← new
```

(4) Deletion at begining

```
(1) ptr ← start
(2) start ← start → Next
(3) Start → prev ← null
(4) free (ptr)
```

(5) Deletion at last

```
(1) ptr ← start
(2) Repeat step 3 ptr → next ≠ NULL
(3) ptr ← ptr → next
(4) ptr → prev → next ← null
(5) free (ptr)
```

(6) Deletion at Middle

```
(1) ptr ← start
(2) Repeat step 3 : ptr → data ! = value
(3) ptr ← ptr → next
(4) temp ← ptr → next
(5) ptr → next ← temp → next
(6) temp → next → prev ← ptr
```

→ Circular Linked List

(1) Insertion at begining
(2) ~~Deletion~~ Insertion at last
(3) Insertion in Ordered Linked List
(4) Delete Element

(1) Insertion at begining

```
    (1) [Create new empty node]
            New ← Node
    (2) [Initialize fields of new node and its link to the list]
            info (New) ← x
```

```
If   First = NULL
 then LINK (NEW) ← NEW
      FIRST ← LAST ← NEW
 else LINK (NEW) ← FIRST
      LINK (LAST) ← NEW
      FIRST ← NEW

 Return
```

② Insertion at last.

```
① [Create New Empty node]
      New ← Node
② [Initialize fields of new node and its link to
      the list]
   If   FIRST = NULL
   then LINK (NEW) ← NEW
        FIRST ← LAST ← NEW
   else LINK (NEW) ← FIRST
        LINK (LAST) ← NEW
        LAST ← NEW
   Return
```

③ Insertion in Ordered Linked List

```
① [Create New Empty Node]
      New ← Node
② [Copy information content into new node]
      INFO (NEW) ← X
③ [Is linked list is empty ?]
      If   FIRST = NULL
      then LINK (NEW) ← NEW
           FIRST ← LAST ← NEW
           Return
④ [Does new node preceedes all other nodes in list
      If   INFO (NEW) ≤ INFO (FIRST)
      then LINK (NEW) ← FIRST
           LINK (LAST) ← NEW
           FIRST ← NEW
           Return
```

```
(5) [Initialize Temporary pointer]
        save ← First
(6) [Search for Predecessor of new node]
        Repeat while save ≠ LAST and INFO(NEW) ≥
                                        INFO(LINK(SAVE))

            Save ← Link(Save)
(7) [Set link field of New node and its Predecessor]
        LINK(New) ← Link(save)
        LINK(Save) ← NEW
        If      save = Last
        then    Last ← New
(8) [Finished]
        Return
```

(4) Delete Element

```
    (1) [Is Empty List ?]
            If   First = NULL
              then write ("Empty ");
                 return
    (2) [Initialize Search for x]
            Save ← First
    (3) [Find x]
            Repeat thrugh step 5 while Save ≠ x and ~~Sa~~
                                        Save ≠ Last
    (4) [Update Predecessor marker]
            Pred ← save
    (5) [Move to Next Node]
            Save ← Link(Save)
    (6) [End of linked list]
            If save ≠ x
            then write ("Node not found")
              return
    (7) [Delete Element]
            If     x = First
            then   First ← Link(first)
                   Link(Last) ← First
            else   Link (Pred) ← Link(x)
```

```
     If x = Last
      then LAST ← PRED
8  [FREE Deleted Node]
        free (X)
```