Linked List

- It is a linear data structure.
- Dynamic data structure.
- To access the n^{th} element of a linked list, you need to walk through each element before it one by one. Thus, the time required to do this is a *linear* function of *n* (the upper limit of which is the list size)
- A linked list can be considered linear if each node is pointing at another node in contrast to trees and other data structures where there may be multiple pointers within a node.

INSEND(X,FIRST)

(First = pointer to indicate first node of list
X = info of the node which is to be inserted)

1. [underflow]
If AVAIL = NULL
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

INSEND(X,FIRST)(CONT...)

5. [is the list empty?]
   if FIRST = NULL
   then return(NEW)
6. [initiate search for last node]
   SAVE = FIRST
7. [search for end of the list]
   Repeat while LINK(SAVE) ≠ NULL
   SAVE = LINK(SAVE)
8. [set LINK field of last node to NEW]
   LINK(SAVE) = NEW
9. Return(FIRST)

INSORD(X,FIRST)

1. [underflow]
   If AVAIL = NULL
      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. [copy info contents into new nod]
      INFO(NEW) = X

INSORD(X,FIRST)(CONT...)

5. [is the list empty?]
if FIRST = NULL
then LINK(NEW) = NULL
Return(NEW)
6. [Does the new node precede all others in the list?]
if INFO(NEW) ≤ INFO(FIRST)
then LINK(NEW) = FIRST
Return(NEW)
7. [initiate temporary pointer]
SAVE = FIRST

INSORD(X,FIRST)(CONT...)

8. [is the list empty?]
   repeat while LINK(SAVE) ≠ NULL and
   INFO(LINK(SAVE)) ≤ INFO(NEW)
   SAVE = LINK(SAVE)
9. [set link fields of new node and its predecessor]
   LINK(NEW) = LINK(SAVE)
   LINK(SAVE) = NEW
10. [return first node pointer]
    return(FIRST)

DELETE(X,FIRST)

1. [Empty list?]
   If FIRST = NULL
   then write(“underflow”)
   Return
2. [Initialize search for X]
   TEMP = FIRST
3. [Find X]
   Repeat thru step 5 while TEMP ≠ X and
   LINK(TEMP) ≠ NULL

DELETE(X,FIRST)(CONT...)

4. [Update predecessor marker]
PRED = TEMP
5. [Move to next node]
TEMP = LINK(TEMP)
6. [End of list?]
If TEMP ≠ X
Then write(“node not found”);
Return

DELETE(X,FIRST)(CONT...)

```
7. [delete X]
        If X = FIRST
                Then FIRST = LINK(FIRST)
        else
                LINK(PRED) = LINK(X)
 8. [Return node to availability area]
          LINK(X) = AVAIL
          AVAIL = X
          return
```

COPY(FIRST)

1. [Empty list?]
   If FIRST = NULL
   Return(NULL)
2. [Copy first node]
   If AVAIL = NULL
      Write (“availability stack underflow”)
      return
   else NEW = AVAIL
      AVAIL = LINK(AVAIL)
      FIELD(NEW) = INFO(FIRST)
      BEGIN = NEW

COPY(FIRST) (CONT...)

3. [Initial traversal]
SAVE = FIRST

4. [Move to next node if not at end of list]
Repeat thru step 6 while LINK(SAVE) ≠ NULL

5. [Update predecessor and save pointer]
PRED = NEW
SAVE = LINK(SAVE)

COPY(FIRST) (CONT...)

6. [copy node]
    if AVAIL = NULL
        Write(“availability stack underflow”);
        Return
    else
        NEW = AVAIL
        AVAIL = LINK(AVAIL)
        FIELD(NEW) = INFO(SAVE)
        PTR(PRED) = NEW
7. [set link of last node and return]
        PTR(NEW) = NULL
        Return(BEGIN)

DOUBINS(L,R,M,X)

(L = Left most Node, R = Right Most node,
M= node before which insertion takes place
X= info of the node to be inserted)

1. [Obtain new node from availability stack]
NEW = NODE

2. [Copy information field]
INFO(NEW) = X

3. [Insertion into an empty list?]
If R = NULL
LPTR(NEW) = RPTR(NEW) = NULL
L = R = NEW
Return

DOUBINS(L,R,M,X) (CONT...)

4. [left-most insertion]

```
If M= L
        LPTR(NEW) = NULL
        RPTR(NEW) = M
        LPTR(M) = NEW
        L = NEW
        Return
```

5. [Insert anywhere before M]

```
LPTR(NEW) = LPTR(M)
RPTR(NEW) = M
LPTR(M) = NEW
RPTR(LPTR(NEW)) = NEW
Return
```

DOUBINS(L,R,M,X) (CONT...)

6. [Right Most Insertion/ insert at the end]

RPTR(NEW) = NULL
LPTR(NEW) = R
RPTR(R) = NEW
R = RPTR(R)

DOUBDEL(L,R,OLD)

(L = Left most node , R = Right most Node
OLD = node which we want to delete)

```
1. [Underflow?]
    If R = NULL                          // Empty list
        Then write("underflow")
2. [delete node]
    If L = R                             // Single node in the list
        L = R = NULL
    Else if OLD = L                      // Left most deletion
                L = RPTR(L)
                LPTR(L) = NULL
     else if OLD = R                     // Right most deletion
                R = LPTR(R)
                RPTR(R) = NULL
     else
                RPTR(LPTR(OLD)) = RPTR(OLD)
                LPTR(RPTR(OLD)) = LPTR(OLD)
```

DOUBDEL(L,R,OLD) (CONT...)

3. [Return deleted node]
Restore(OLD)
Return

Circular LL(Insert)

1) [underflow]

If AVAIL = NULL

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)

Circular LL(Insert) (continue..)

4) [insertion in empty linked list]

```
if (head = NULL)
    head = temp
   temp(info)=x
   temp(link)=head;
```

5) [insert at front]

```
   temp(info)=x
   temp(link)=head
   head=temp
```

Circular LL(Insert) (continue..)

6) [insert at end]

```
save =head
repeat while save(link) != head
{
        save =save(link)
}
save(link)=temp
temp(info)=x
temp(link)=head
```

Circular LL(Insert) (continue..)

7) [insert in sorted list]

```
temp(info)=x
save =head
repeat while (temp(info)>=save(info)) and
        save(link)!= head
{
    pred=save
    save = save(link)
 }
pred(link)=temp
temp(link)=save
```