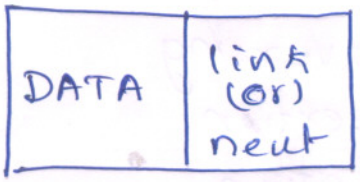
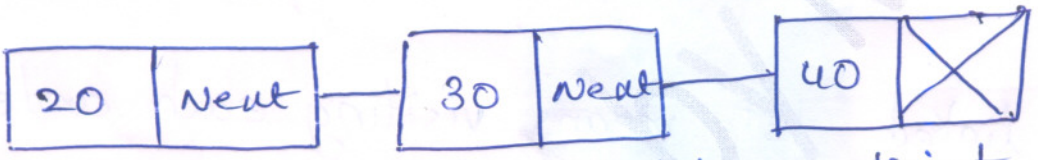
UNIT-3

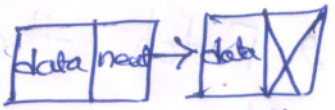
→ Introduction to linked list :-

* A linked list is a linear collection of data Elements. These data elements are called Node
* In a linked list there are different number of nodes each node consists of 2 fields. First field is data field and second field is link field



* DATA field holds the value or some information



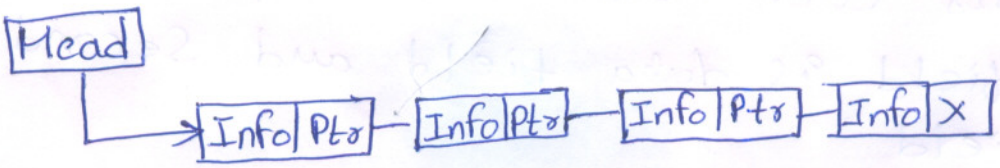
* link field or Next field contains the address to the next item in the list the last node in the list contains NULL that indicates End of the list
* NULL is represented as  Ex :- 
* linked list also contain a pointer variable called Head which represents the list is simply a pointer to the first node of the list

→ Types of linked list :-

1. Single linked list
2. Double linked list
3. circular linked list

Single Linked list :-

A single linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type



Operations on Singly linked list:

There are 3 operations on SLL
1. Traversing
2. Insertion
3. Deletion.

1. Traversing:-

Traversing linked list means visiting each and every node of the SLL.

Steps for traversing the SLL.

1. First move to the first node
2. Fetch the data from the node and perform the operation depending on data type.
3. After performing operation, advance pointer to next node and perform all above steps.

Algorithm for Traversing A linked list:-

1. Current ptr ← HEAD /* HEAD is a pointer to first of a linked list */
2. While current ptr ≠ NULL do
   begin /* any process that can be done with current node */

3. print 'The information field contains', info (current ptr)

4. current ptr <-- ptr (current ptr)
   End
5. Return.

Insertion:-
Elements are placed in list will be done.

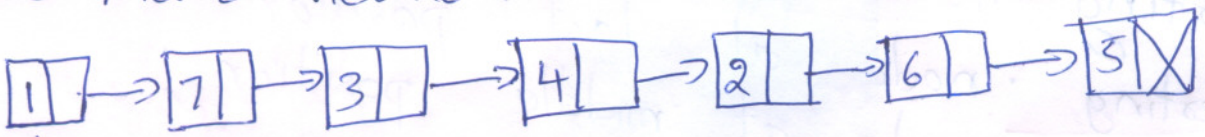
1. Inserting node at first position
2. Inserting node at last position
3. Inserting node at middle position

1. Inserting node at first Position:

~~Insert AVAIL~~
Insert First (item, HEAD)
1. If AVAIL = NULL then
   begin
   print "space not available in the free list"
   Return (HEAD)
   End
2. new <-- AVAIL /*Remove first node from free list*/
3. AVAIL <-- ptr (AVAIL)
4. Info (new) <-- item /* Initialize the new node with item */
5. ptr (new) <-- HEAD /*set HEAD to new node pointer*/
6. HEAD <-- new
   /* now new node is the first node */
7. Return (HEAD)

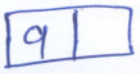
Steps for Inserting node at start in the SSL:

1- Create new node
2- Fill Data into "Data field"
3- make its "pointer" on "next Field" as NULL
4- Attach this newly created node to start
5- make newnode as starting node

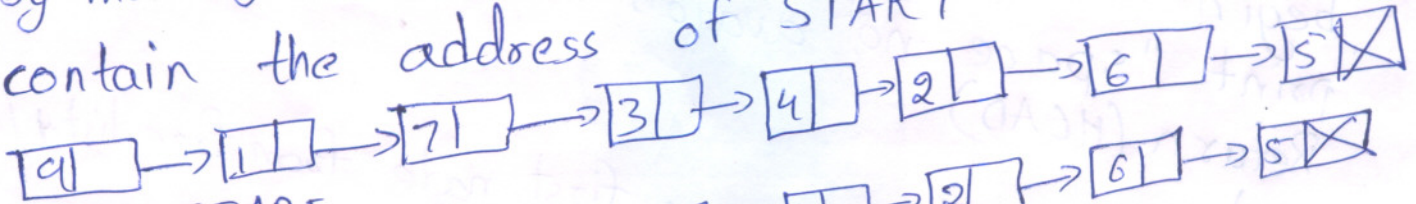


start

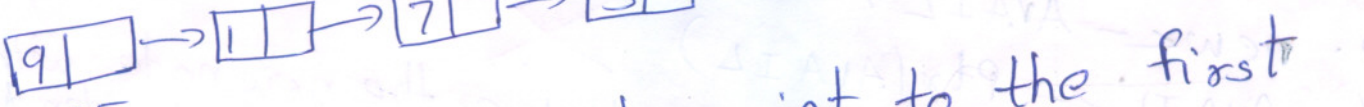
Allocate memory for the new node and initialize its DATA part to 9



Add the new node as the first node of the list by making the NEXT part of the new node contain the address of START



START



START

Now make START to point to the first node of the list.

2. Inserting node at last position:

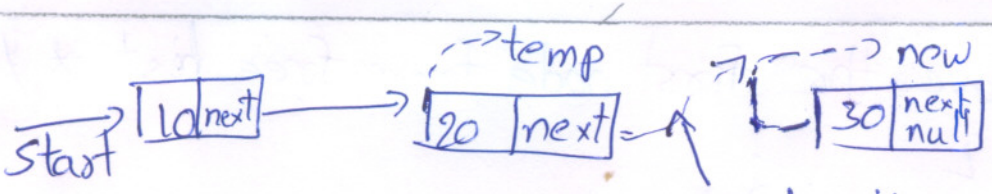
1. If AVAIL = NULL then
   begin
   print 'space not available in the free list'
   Return (HEAD)

End /* remove the first node from free list */

2. new <- AVAIL
3. AVAIL <- ptr(AVAIL)
4. Info(new) <- item /* initialize new node with item */
5. ptr(new) <- NULL /* set the new node of NULL pointer to make it last node */
6. If HEAD = NULL then /* If the list is empty then new will be */
   Return(new) /* the first node and so return new */
7. Temp ptr <- HEAD
8. While ptr(Temp ptr) ≠ NULL do
   Temp ptr <- ptr(Temp ptr)
9. ptr(Temp ptr) <- new /* set earlier last node pointer to new */
10. Return(HEAD)

Steps for Inserting node at last :-

1. Create new node
2. Fill Data into "Data field"
3. make its "pointer" or "next field" as NULL
4. Node is to be inserted at least position so we need to traverse SLL upto last node
5. Make link between last node and new node



Node at middle position :-

Algorithm :- (P is a pointer that points to a given node)

1. If AVAIL = NULL then
   begin
   print 'space not available in the free list'
   Return (HEAD)
   End
2. new ← AVAIL /* remove first node from freelist
3. AVAIL ← ptr(AVAIL)
4. Info(new) ← item /* initialize new node with item
5. If P = NULL then /* if the new node is to be inserted at first position
   begin
6. ptr(new) ← HEAD /* make new as first node
7. HEAD ← new
   End
   Else
   begin
8. ptr(new) ← ptr(P) /* insert after the node which pointed by P
9. ptr (P) ← new.
   End
10. Return (HEAD)

Deltion :- remove an element from the list

Algorithm :-

1. If HEAD = NULL then /* test whether the list is empty
   begin
   print 'underflow'
   Return
   End
2. Temp ptr ← HEAD
3. while Temp ptr ≠ NULL and ptr(Temp Ptr) ≠ P do
   begin
4. prev ← Temp ptr /* Save the current pointer in prev and.
5. Temp ptr ← ptr (Temp ptr) /* move to next node
   End
6. If Temp Pt ≠ P then /* test whether required node found or not */
   begin
   Print 'Not found the node to be deleted'
   Return
   end
7. If P = HEAD then /* if the element to be deleted is the first node */
   HEAD ← ptr (HEAD) /* set next node as the first node
   else
   ptr (prev) ← ptr (p) /* element to its previous element */

8. ptr (P) <— AVAIL /* Assign deleted node to first block of */

9. AVAIL <— P /* Available list */

10. Return