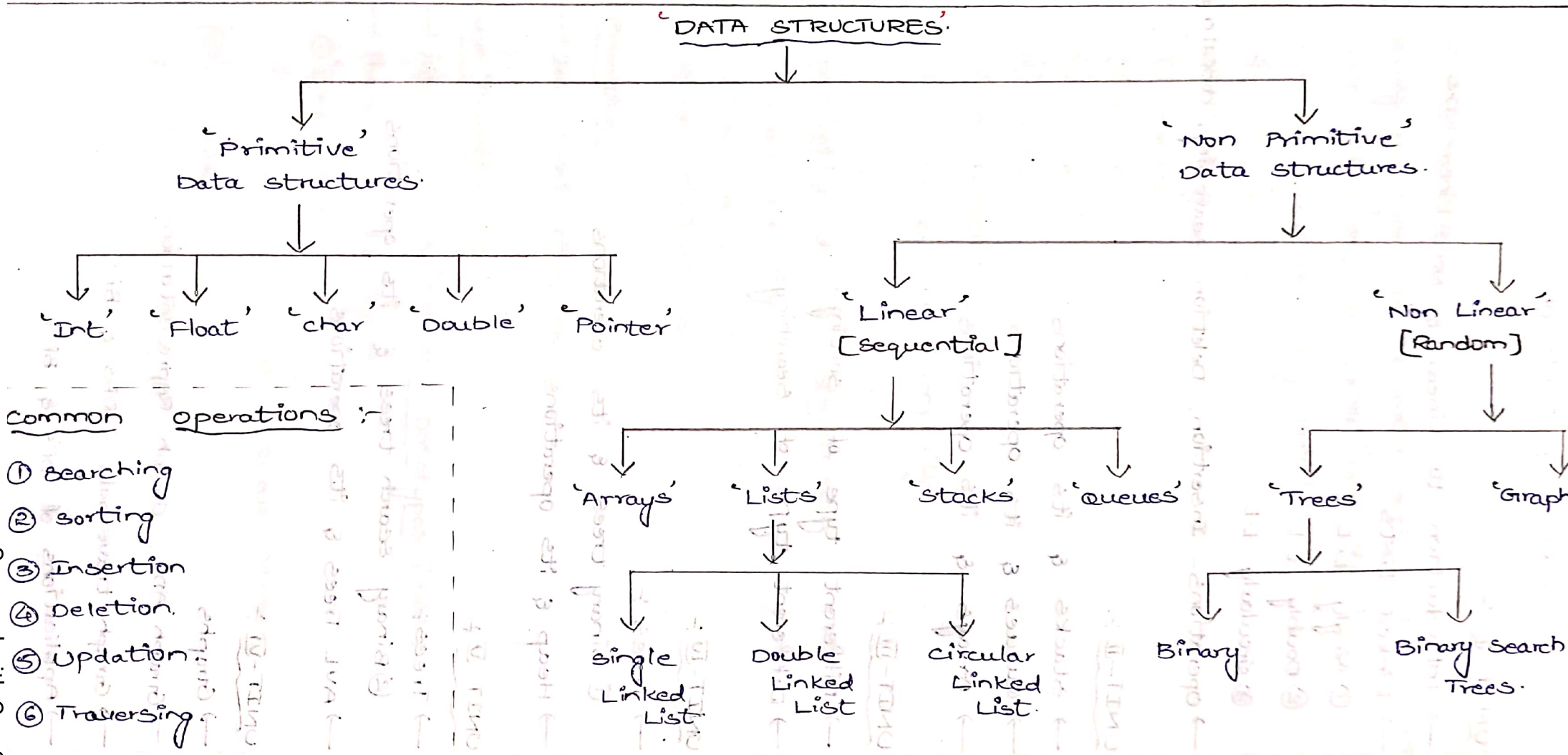
DATA STRUCTURES



Common operations :-

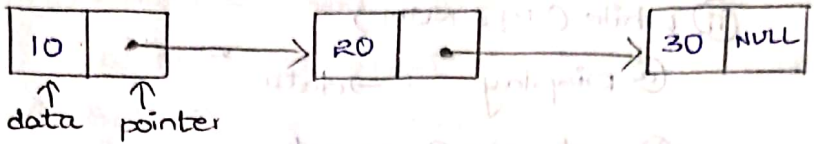
1. Searching
2. Sorting
3. Insertion
4. Deletion
5. Updation
6. Traversing

03-03-2023

UNIT - I

* Single Linked List :-

→ contains one Node with two fields [Data field and Node type of pointer field].



* Defining a single Linked List :-

```
struct node
{
    int data;
    struct node * next; // self Referential structure member.
} *start=NULL, *temp, *p;
```

*start :- Always points to starting node.
*temp :- It is used to create/delete a node.
*p :- It is used for traversing.

* Algorithm for creation of single Linked List :-

Step-1 :- create a new node i.e, temp.

[temp = (struct node *) malloc (size of (struct node));

Step-2 :- Input the data to be inserted [using x].

step-3 :- set temp → data = x ; temp → next = NULL ;

Step-4 :- if (start == NULL) then

(i) set start = temp.

(ii) set P = temp

step-5 :- else

(i) P → next = temp.

(ii) set P = temp.

Step-6 :- If you want create one more node repeat steps 1-5 otherwise exist.

* Algorithm for Traversal of S.L.L :-

step-1 :- if (start == NULL) then

(i) Display "Linked List is Empty"

step-2 :- else

(i) set P = start

(ii) while (P != NULL)

(a) Display "P → data"

(b) set P = P → next.

step-3 :- Exit.

* Insertion :-

* Algorithm for Inserting a node at beginning of the single Linked List :-

step-1 :- Create a new Node i.e. temp.

[temp = (struct node *) malloc (size of (struct node));]

step-2 :- Input the data to be inserted [using x variable]

step-3 :- set temp → data = x., temp → next = NULL.

step-4 :- if (start == NULL) then

(i) set start = temp.

step-5 :- else

(i) set temp → next = start

(ii) set start = temp.

step-6 :- If we want insert another node at begining repeat steps 1 to 5 otherwise Exit.

* Algorithm for Inserting a node at end of the single Linked List :-

Step-1 :- create a new node i.e, temp.

step-2 :- Input the data to be inserted [using x variable];

step-3 :- set temp→data = x, temp→next = NULL

step-4 :- if (start == NULL) then

(i) set start = temp

step-5 :- else

(i) set P = start

(ii) while (P→next != NULL)

(a) set P = P→next.

(iii) Set P→next = temp

step-6 :- If you want insert another node at ending of single Linked List then Repeat from steps 1-5 otherwise EXIT.

* Algorithm for inserting a node at specified position of the single Linked List :-

step-1 :- create a new node i.e temp.

step-2 :- Input the data to be inserted [using x variable]

step-3 :- set temp→data = x, temp→next = NULL

Step-4 :- if (start == NULL) then

(i) set start = temp.

step-5 :- else

(i) set count = 0

(ii) P = start

(iii) while (P != NULL)

(a) set count = count + 1

(b) set P = P→next.

(iv) Input the position of inserting value [using position variable].

(v) if (position == 1)

(a) call insert_node_beg_sLLC()

(vi) if (position == count + 1)

(a) call insert_node_end_sLLC()

(vii) else if (Position > 1 and Position <= count)

(a) set P = start

(b) for (k = 1; k < Position - 1; k++)

(A) set P = P→next.

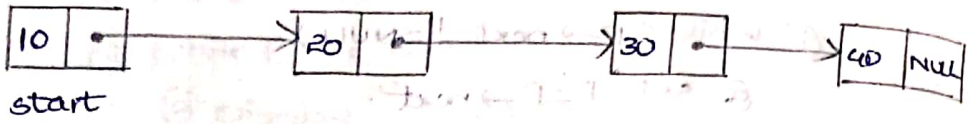
(c) set temp→next = p→next

(d) Set P→next = temp

(iii) else
  (a) Display "Invalid position"

step-6 :- EXIT.

* DELETION.



* Algorithm for Deleting a Node at Begining of S.L.L

step-1 :- if (start == NULL) then
  (i) Display "L.L is Empty"

step-2 :- else
  (i) Set temp = start.
  (ii) Set start = start → next.
  (or)
  Set start = temp → next.
  (iii) set temp → next = NULL.
  (iv) Delete temp [free (temp);]

step-3 :- EXIT.

* Algorithm for Deleting a Node at end of the S.L.L :-

step-1 :- if (start == NULL) then
  (i) Display "L.L is Empty"
  (ii) EXIT.

step-2 :- set p = start.

step-3 :- while (p → next → next != NULL)
  (i) set p = p → next

step-4 :- (ii) set p.temp = p → next.

step-5 :- set (p → next = NULL.

step-6 :- Delete temp.

step-7 :- EXIT

step :- else if (start → Next = NULL)
  (i) set temp = start
  (ii) set start = NULL
  (iii) Delete(temp)
  (iv) EXIT.

* Algorithm for Deleting a Node at specified position of S.L.L :-

step-1 :- if (start == NULL)

(i) Display "Linked List is Empty"

step-2 :- else if (start→next = NULL)

(i) Set temp = start
(ii) set start = start→Next
(iii) Delete temp.

} No need.

step-3 :- else

(i) set count = 0.
(ii) set P = start
(iii) while (P != NULL)
  (a) Set count = count + 1;
  (b) set P = P→next.
(iv) input the position of the deleting Node. using 'pos' variable
(v) if (pos == 1)
  (a) call delete_node_beg_SLL ( ).
(vi) else if (pos == count)
  (b) call del_node_end_S.L.L ( ).
(vii) else if (posint > 1 and posint < count)
  (a) set P = start.
  (b) for (k = 1; k < pos - 1; k++)
    (A) set P = P→next
  (c) set temp = P→next.
  (d) set P→next = temp→next.
  (e) Set temp→next = NULL.
  (f) Delete (temp)
(viii) else
  (a) Display "Entered position is Invalid".

step-4 :- EXIT.

SEARCHING

* Algorithm for searching :-

void searching (int x)
int flag=0;

step-1 :- if (start == NULL) then
  (i) Display "Linked List is Empty"

step-2 :- else
  (i) p = start
  (ii) while (p != NULL)
    (a) if (p → data == x)
      (A) set flag = 1.
      (B) break;
    (b) p = p → next.
  (iii) if (flag == 1)
    (a) Display "search is successful."
  (iv) else
    (a) Display "Element is not found."

step-3 :- EXIT.

SORTING

* Algorithm for sorting :-

*temp1, *temp2.

step-1 :- if (start == NULL)
  (i) Display "LL is Empty."

step-2 :- else if (start → next = NULL)
  (i) Display "LL having single Node."

step-3 :- else
  (i) for (temp1 = start; temp1 != NULL; temp1 = temp1 → next)
    (a) for (temp2 = start → next; temp2 != NULL; temp2 = temp2 → next)
      (A) if (temp1 → data < temp2 → data)
        (I) set x = temp1 → data.
        (II) set temp1 → data = temp2 → data
        (III) set temp2 → data = x.

step-4 :- EXIT.

REVERSING A SINGLE L.L

*P1, *P2, *P3.

* Algorithm for Reversing a single Linked List :-

step-1 :- if (start == NULL)

(i) Display "Linked List is empty".

step-2 :- else if (start → next = NULL).

(ii) Display "Linked List contains single Node".

step-3 :- else

(i) set P_1 = start.

(ii) set $P_2 = P_1 \rightarrow next$

(iii) set $P_3 = P_2 \rightarrow next$

(iv) set $P_1 \rightarrow next = NULL$.

(v) set $P_2 \rightarrow next = P_1$

(vi) while (P_3 != NULL)

(a) set $P_1 = P_2$

(b) set $P_2 = P_3$

(c) Set $P_3 = P_2 \rightarrow next$ (or) $P_3 \rightarrow next$.

(d) set $P_2 \rightarrow next = P_1$

(vii) set start = P_2

step-4 :- EXIT.

CONCATENATION

* Algorithm for concatenation of two S.L.L :-

struct node * creation (struct node * start).

step-1 :- set start1 = creation (start1). *start1 = NULL;

step-2 :- Set start2 = creation (start2). *start2 = NULL.

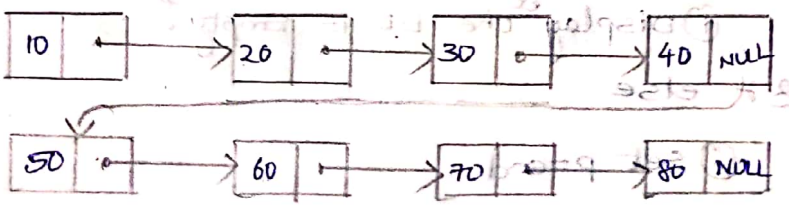
step-3 :- set p = start1

step-4 :- while (P → next != NULL)

(a) P = P → next.

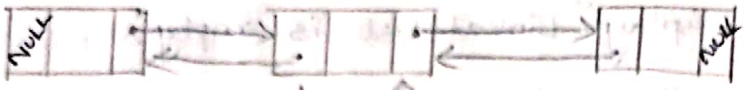
step-5 :- Set P → next = start2.

step-6 :- EXIT.



** DOUBLE LINKED LIST

→ It contains Nodes. one Node having three fie
[one data field and two pointer fields].

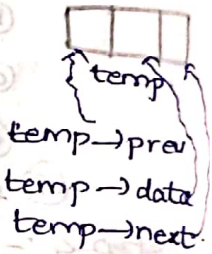


* Defining a D.L.L:-

```
struct node
{
  struct node *prev;
  int data;
  struct node *next;
} *start=NULL, *end, *P, *temp;
```

* Algorithm for creating of D.L.L:-

step-1:- create a new node i.e, temp.



step-2:- Input the data to be inserted [using x].

step-3:- set temp→data=x,
set temp→prev=NULL,
set temp→next=NULL.

step-4:- if (start==NULL) then
(i) set start=temp
(ii) set end=temp

step-5:- else
(i) set end→next=temp.
(ii) set temp→previous=end.
(iii) set end=temp.

step-6:- EXIT.

* Algorithm for Traversal of D.L.L in Backward directi

step-1:- if (start==NULL)
(i) Display "the L.L is Empty."

step-2:- else
(i) set p=end

(ii) while (P != NULL)

(a) Display "P→data"

(b) Set P = P→prev.

Step-3 :- EXIT.

* Algorithm for Traversal of D.L.L in Forward Direction :-

step-1 :- if (start == NULL)

Display The L.L is Empty.

step-2 :- else

(i) set p = start

(ii) while (P! = NULL)

(a) Display P→data

(b) set P = P→next

step-3 :- EXIT.

INSERTION

* Algorithm for Inserting a Node at begining in D.L.L :-

step-1 :- create a new node i-e temp

step-2 :- Input the data to be inserted [using x]

step-3 :- set temp→data = x,

set temp→prev = NULL,

set temp→next = NULL.

step-4 :- if (start == NULL) then

set start = temp

set end = temp.

step-5 :- else

(i) set temp→next = start

(ii) set start→prev = temp.

(iii) set start = temp.

step-6 :- EXIT.

* Algorithm for inserting a Node at end of D.L.L:

step-1 to step-4 :- same as creation.

step-5 :- else

(i) set end → next = temp

(ii) set temp → prev = end

(iii) set end = temp

step-6 :- EXIT.

* Algorithm for Inserting a Node at specified position in D.L.L :-

step-1 :- create a New node temp.

step-2 :- Input the data to be inserted using x.

step-3 :- set temp → data = x,
set temp → prev = NULL,
set temp → next = NULL.

step-4 :- if (start == NULL).

(i) set start = temp

(ii) set end = temp

step-5 :- else

(i) set count = 0

(ii) set P = start

(iii) While (P != NULL)

(a) set (count = count++)

(b) P = P → next

(iv) Input the position of the value to be inserted (using pos)

(v) if (pos == 1)

(a) call insert_node_begining_dll ( ).

(vi) else if (pos == count+1)

(a) call insert_node_end_dll ( ).

(vii) else if (pos > 1 and pos <= count).

(a) set P = start

→

(b) for (k=1; k<pos-1; k++)

(A) set p=p→next;

(c) set temp→next = p→next;

(d) set p→next→prev = temp

(e) set p→next = temp

(f) set temp→prev = p

(viii) else

Display "Invalid position."

step-6: EXIT.

DELETION

* DELETION at Begining :-

* Algorithm for Deletion of a Node at begining of D.L.L:-

step-1 :- if (start == NULL)

(i) Display "L.L is Empty"

step-2 :- else.

(i) Set temp = start

(ii) Set start = start→next (or) temp→next

(iii) set start→prev = NULL

step-3 (iv) free (temp)

step-4 :- EXIT.

* Deletion at ending of a D.L.L :-

Step-1 :- If (start==NULL)

(i) Display "L.L is Empty".

step-2 :- else if (start→next ==NULL)

(i) set temp = start

(ii) set start = NULL

(iii) free (temp)

step-3: ELSE.

(i) set temp = start

(ii) while (temp→next != NULL)

(A) temp = temp→next

(iii) temp→prev→next = NULL

(iv) free (temp).

step-4: EXIT.

* Algorithm for Deletion of a Node at specified position in Doubly Linked List :-

Step-1 :- if (start == NULL)

(i) Display "Linked List is empty."

Step-2 :- else.

(i) set count = 0

(ii) set P = start

(iii) while (P != NULL)

(a) set (count = count++)

(b) P = P → next

(iv) Input the position to be deleted (using pos variable).

(v) if (pos == 1)

(a) call del_node_begin( );

(vi) else if (pos == count+1)

(a) call del_node_end_DLLC( );

(vii) else if (Pos > 1 and Pos <= count)

(a) set P = start

(b) for (k=1 ; k <= Pos-1 ; k++)

(A) set P = P → next.

(c) set temp = p → next

(d) set P → next = temp → next;

(e) set temp → next → prev = P.

(f) Delete temp.

(viii) else

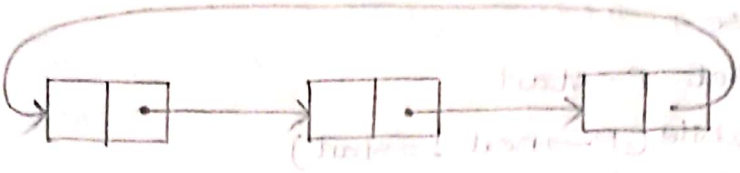
(a) Display "Invalid position."

Step-3 :- EXIT.

Circular Linked List (CLL).

* Single circular Linked List [S.C.L.L]

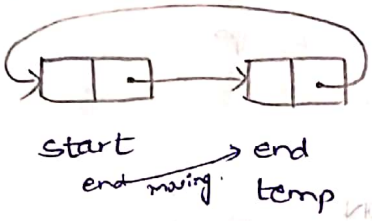
→ Node consists of two fields [Data, Pointer fields].



For single Node :-

start→next = start
(or)
end→next = start.

For Two nodes :-



start→next = temp
temp→next = start.

* Algorithm for creating single circular List :-

~~Step-1~~ :-

structure :-

```
Struct node
{
   int data;
   Struct node *next;
} *start=NULL, *temp, *p/*end;
```

Step-1 :- create a New node i.e temp.

Step-2 :- Input the data to be inserted (using x).

Step-3 :- Set temp→data = x, temp→next = NULL.

Step-4 :- if (start == NULL) then

(i) Set start = temp.

(ii) Set start→next = start.

(iii) Set end = temp/start.

Step-5 :- else

(i) Set end→next = temp.

(ii) Set temp→next = start.

(iii) Set end = temp.

Step-6 :- EXIT.

* Algorithm for Traversal of S.C.L.L :-

Step-1 :- if (start == NULL) then

(i) Display "L.L is Empty."

Step-2 :- else

(i) Set P = start.

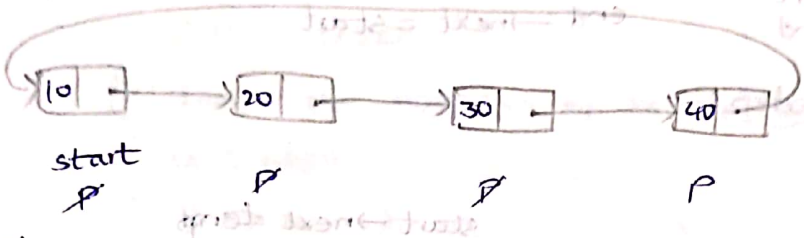
(ii) while (P → next != start)

(a) Display P → data

(b) set P = P → next.

step-3 :- ~~Exit~~ Display p → data

step-4 :- EXIT.



O/P :- 10 20 30 40.

INSERTION

* Algorithm to insert a Node at begining of S.C.L.L

Step-1 to 4 :- Same as creation.

Step-5 :- else.

(i) set temp → next = start.

(ii) set end → next = temp.

(iii) set (~~temp →~~) start = temp

(or)

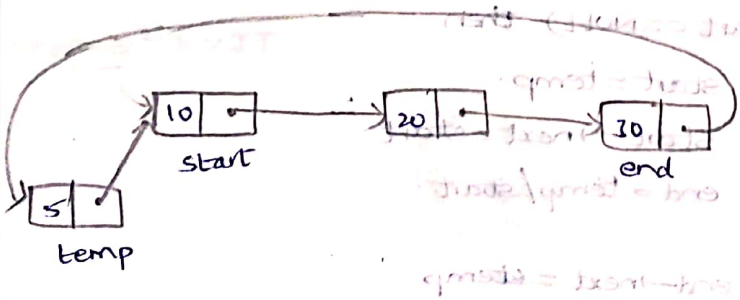
Step-5 :- else

(i) Set temp → next = start

(ii) set start = temp

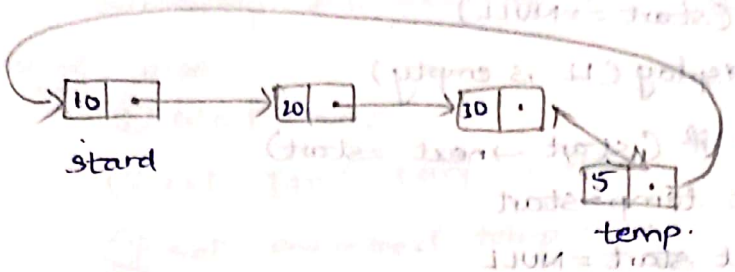
(iii) set end → next = start

Step-6 :- EXIT



* Algorithm to insert a node at end of S.C.L.L:

step-1 to 3 :- same as creation of S.C.L.L.



* Algorithm to insert a Node at specific position of SCLL:

step-1 to 4 :- same as creation

step-5 :- else

(i) set count = 1

(ii) set P = start

(iii) while (P->next != start)

(a) count = count + 1

(b) set P = P->next;

(iv) Input the position of the Node to be inserted. [using pos]

(v) if (pos == 1)

(a) call insert_begin_S.C.L.L();

(vi) else if (pos <= count+1)

(a) call insert_end_S.C.L.L();

(vii) else if (Pos > 1 and Pos <= count)

(a) set P = start

(b) for(k=1; k<pos-1; k++)

(A) set P = P->next

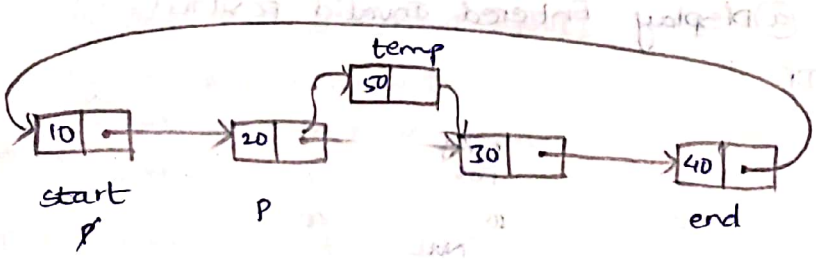
(c) set temp->next = P->next

(d) set P->next = temp.

(viii) else

(a) Display Entered Invalid position.

step-6 :- EXIT.



DELETION

* Algorithm for Deleting a Node at specified position in a

step-1 :- if (start == NULL)

(i) Display CLL is empty)

step-2 :- else if (start → next = start)

(i) set temp = start

(ii) set start = NULL

(iii) Set temp → next = NULL

(iv) Delete temp.

step-3 :- else.

(i) set count = 1

(ii) set p = start

(iii) while (p → next != start)

(a) count = count + 1

(b) set p = p → next

(iv) Input the position of Node to be deleted [using p

(v) if (pos == 1)

(a) call del_beg_sCLLC);

(vi) else if (pos == count)

(b) call del_end_sCLLC);

(vii) else if (pos > 1 and pos < count)

(a) Set p = start

(b) for (k = 1; k < pos - 1; k++)

(A) set p = p → next.

(c) set temp = p → next,

(d) set p → next = temp → next

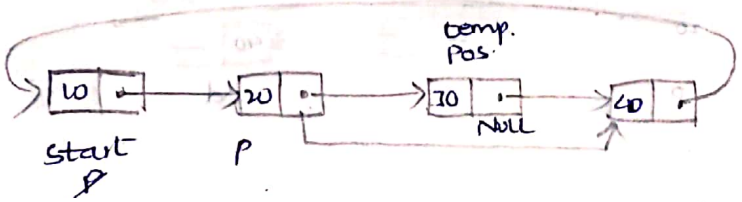
(e) set temp → next = NULL

(f) Delete temp.

(viii) else

(a) Display Entered Invalid position.

step-4 :- EXIT.



* Algorithm for Deleting a Node at begining:-

step-1 :- create a Ne: if (start == NULL)

(a) Display "Linked List is empty."

step-2 :- else

(i) start = temp

(ii) set start = temp->next

(iii) set end->next = temp->next.

(iv) Delete temp

step-4 :- EXIT.

(or)

step-1 :- if (start == NULL).

(i) Display 'Linked List is Empty'

step-2 :- else if (start->Next == start)

(i) start = NULL

(ii) free (start)

step-3 :- else

(i) set p = start

(ii) while (P->next != start)

(a) p = p->next

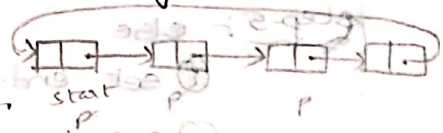
(iii) set P->next = start->next

(iv) Delete (start)

(v) Set start = P->next

step-4 :- EXIT.

* Algorithm for Deletion of a Node at ending in sCLL:-



step-1 :- if (start == NULL)

(i) Display Linked List is empty.

step-2 :- else if (start->next == start)

(i) start = NULL

(ii) Delete (start).

step-3 :- else

(i) set P = start

(ii) while (P->next->next != start).

(a) P = P->next

(iii) set P->next->next = NULL.

(iv) set P->next = start.

(v) Set temp = P->next->next

Step-4 :- Delete temp.

Step-5 :- EXIT.