**Algorithm :-INSFIRST(INFO, LINK, START, AVAIL, ITEM)**

**This algorithm inserts ITEM as the first node in the list**.

Step 1 : [OVERFLOW?]   
 If AVAIL = NULL, then :   
 Write : OVERFLOW, and Exit.

Step 2 : [Remove first node from AVAIL list.]  
 Set NEW := AVAIL and

AVAIL := LINK[AVAIL].

Step 3 :Set INFO[NEW] := ITEM.  
 [Copies new data into new node].

Step 4 : Set LINK[NEW] := START.  
 [New node now points to original first node.]

Step 5 : Set START := NEW.  
 [Changes START so it points to new node.]

Step 6 : Exit.

**Algorithm:-INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)  
This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.**

Step 1 : [OVERFLOW?]   
 If AVAIL = NULL, then :

Write : OVERFLOW, and Exit.

Step 2 : [Remove first node from AVAIL list.]   
 Set NEW := AVAIL and AVAIL :=   
 LINK[AVAIL].

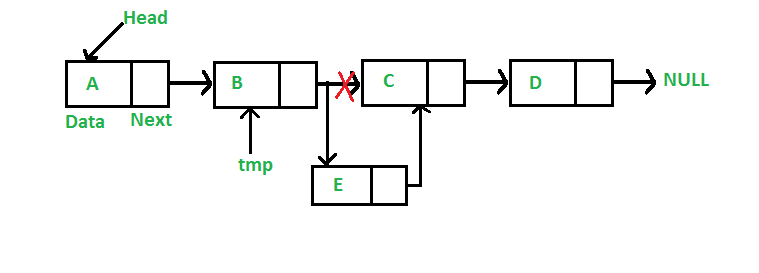
Step 3 : Set INFO[NEW] := ITEM.  
 [Copies new data into new node].

Step 4 : If LOC = NULL, then :

[Insert as first node.]  
 Set LINK[NEW] := START and

START := NEW.  
 Else : [Insert after node with location LOC.]  
 Set LINK[NEW] := LINK[LOC]   
 and LINK[LOC] := NEW.  
 [End of If structure.]

Step 5 : Exit.



**Algorithm:- INSERT(INFO,LINK,START,AVAIL,ITEM)**

**This algorithm inserts ITEM into a sorted linked list.**

1. [Use Procedure to find the location of the node preceding ITEM.]

Call FINDA(INFO,LINK,START,ITEM,LOC)

2. [Use Algorithm to insert ITEM after the node with location LOC].

Call INSLOC(INFO,LINK,START,AVAIL,LOC,ITEM).

3. Exit.

**Procedure:-FINDA(INFO,LINK,START,ITEM,LOC)**

**This procedure finds the location LOC of the last node in a sorted list such that INFO[LOC ]<ITEM or sets LOC=NULL.**

1.[List empty?] If START=NULL, then:

Set LOC:=NULL, and return.

2.[Special case ?] If ITEM< INFO[START], then: Set

LOC:=NULL, and return.

3.Set SAVE:=START and PTR:= LINK[START].

[Initializes pointers].

4.Repeat steps 5 and 6 while PTR!=NULL.

5. If ITEM< INFO[PTR]. then:

Set LOC:= SAVE, and return.

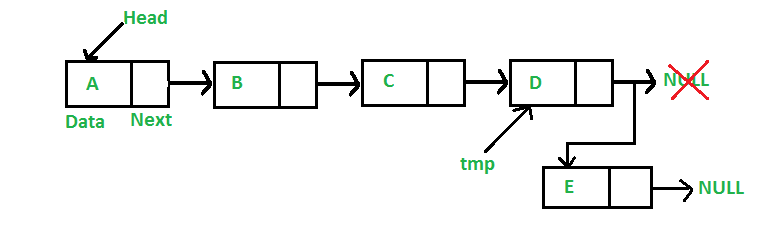
[End of if structure]

6.Set SAVE:=PTR and PTR:=LINK[PTR]

[End of Step 4 loop]

7.Set LOC:=SAVE.

8.Return.



**Algorithm:- DEL(INFO,LINK,START,AVAIL,LOC,LOCP)**

**Let LIST be a linked list in memory. Suppose we are given the location LOC of node N in LIST. And ,suppose we are given the location LOCP of the node preceding N or, when N is the first node ,we are given LOCP=NULL. The following algorithm deletes N from the list**.

1.If LOCP=NULL, then:

Set START:=LINK [START]. [ Deletes first node].

else:

Set LINK[LOCP]:= LINK[LOC]. [ Deletes N node.]

[End of if structure]

2.[Return deleted node to the AVAIL list.]

set LINK[LOC]:=AVAIL and AVAIL:=LOC.

3.Exit.

Prac 3 :

**Algorithm:**

**Assume that there are two polynomials A and B. The polynomial storing the addition result is stored in one large array C and i,j,k represent the pointers to the polynomials and arrays resp.**

1. Read n1=no of terms in A, n2=no. of terms in B.
2. while i<n1 and j<n2
3. If exp at ith position of A=exp at jth position of B, then

C at kth position =coeff of A at ith position +coeff of B at jth position.

Increment i,j,k to point to next position in A,B,C.

1. Else

If exp position i>exp position j in B then

Copy coeff at ith position from A to coeff at kth position in C.

Copy exp pointed by i into exponent field at position k in C

Increment i,k

Else

Copy coeff at jth position from B to coeff at kth position in C

Copy exp pointed by j into exponent field at position k in C

Increment j,k.

1. While i<n1
2. Copy coeff at ith position from A to coeff at kth position in C.
3. Copy exp pointed by i into exponent field at position k in C
4. Increment i,k
5. While j<n2
6. Copy coeff at jth position from B to coeff at kth position in C
7. Copy exp pointed by j into exponent field at position k in C
8. Increment j,k.
9. Display C[result of array addition]
10. Exit.

Prac 4 :

**Algorithm :- INSFIRST(INFO, LINK, START, AVAIL, ITEM)**

**This algorithm inserts ITEM as the first node in the list**.

Step 1 : [OVERFLOW?]   
 If AVAIL = NULL, then :   
 Write : OVERFLOW, and Exit.

Step 2 : [Remove first node from AVAIL list.]  
 Set NEW := AVAIL and

AVAIL := LINK[AVAIL].

Step 3 :Set INFO[NEW] := ITEM.  
 [Copies new data into new node].

Step 4 : Set LINK[NEW] := START.  
 [New node now points to original first node.]

Step 5 : Set START := NEW.  
 [Changes START so it points to new node.]

Step 6 : Exit.

**Algorithm:-INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)  
This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.**

Step 1 : [OVERFLOW?]   
 If AVAIL = NULL, then :

Write : OVERFLOW, and Exit.

Step 2 : [Remove first node from AVAIL list.]   
 Set NEW := AVAIL and AVAIL :=   
 LINK[AVAIL].

Step 3 : Set INFO[NEW] := ITEM.  
 [Copies new data into new node].

Step 4 : If LOC = NULL, then :

[Insert as first node.]  
 Set LINK[NEW] := START and

START := NEW.  
 Else : [Insert after node with location LOC.]  
 Set LINK[NEW] := LINK[LOC]   
 and LINK[LOC] := NEW.  
 [End of If structure.]

Step 5 : Exit.

**Algorithm:- DEL(INFO,LINK,START,AVAIL,LOC,LOCP)**

**Let LIST be a linked list in memory. Suppose we are given the location LOC of node N in LIST. And ,suppose we are given the location LOCP of the node preceding N or, when N is the first node ,we are given LOCP=NULL. The following algorithm deletes N from the list**.

1.If LOCP=NULL, then:

Set START:=LINK [START]. [ Deletes first node].

else:

Set LINK[LOCP]:= LINK[LOC]. [ Deletes N node.]

[End of if structure]

2.[Return deleted node to the AVAIL list.]

set LINK[LOC]:=AVAIL and AVAIL:=LOC.

3.Exit.

**Algorithm:-INSERT(INFO,LINK,START,AVAIL,ITEM)**

**This algorithm inserts ITEM into a sorted linked list.**

1. [Use Procedure to find the location of the node preceding ITEM.]

Call FINDA(INFO,LINK,START,ITEM,LOC)

2. [Use Algorithm to insert ITEM after the node with location LOC].

Call INSLOC(INFO,LINK,START,AVAIL,LOC,ITEM).

3. Exit.

**Procedure:-FINDA(INFO,LINK,START,ITEM,LOC)**

**This procedure finds the location LOC of the last node in a sorted list such that INFO[LOC ]<ITEM or sets LOC=NULL.**

1.[List empty?] If START=NULL, then:

Set LOC:=NULL, and return.

2.[Special case ?] If ITEM< INFO[START], then: Set

LOC:=NULL, and return.

3.Set SAVE:=START and PTR:= LINK[START].

[Initializes pointers].

4.Repeat steps 5 and 6 while PTR!=NULL.

5. If ITEM< INFO[PTR]. then:

Set LOC:= SAVE, and return.

[End of if structure]

6.Set SAVE:=PTR and PTR:=LINK[PTR]

[End of Step 4 loop]

7.Set LOC:=SAVE.

8.Return.

Prac 5 :

**Algorithms**

Algorithm for PUSH operation

1. Check if the stack is **full** or not.
2. If the stack is full, then print error of overflow and exit the program.
3. If the stack is not full, then increment the top and add the element.

Algorithm for POP operation

1. Check if the stack is empty or not.
2. If the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.

**Algorithm : POLISH(Q,P)**

**Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.**

1. Push“(” onto STACK, and add “)” to the end of Q.

2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the STACK is empty:

3. If an operand is encountered , add it to P.

4. If a left parenthesis is encountered , push it onto STACK.

5.If an operator is encountered, then:

(a)repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than operator.

(b) Add operator to STACK.

[End of if structure.]

6. If a right parenthesis is encountered, then:

(a)Repeatedly pop from stack and add to P each operator (on the top of STACK) until a left parenthesis is encountered.

(b) Remove the left parenthesis.

[do not add the parenthesis to P.]

[End of If structure.]

[End of step 2 loop.]

7. Exit.

**Algorithm for Evaluating Postfix Expressions**

**Algorithm : EVALUATE(P)**

**This algorithm scan Postfix expression and evaluate it**

1. Add a right parenthesis “)” at the end of P.[this act as a sentinel]

2. Scan P from left to right and repeat steps 3 & 4 for each element of P until sentinel “)” is encountered

3. If an operand is encountered , put it on STACK.

4. If an operator is encountered, then:

a) Remove the two top elements of STACK, where A is the top element & B is next to top element.

b) Evaluate B operator A.

c) Place the result of (b) back on stack

[End of If structure.]

[End of Step 2 loop.]

5. Set VALUE equal to the top element on STACK .

6. Exit

Prac 7 :

**Algorithm :- INSFIRST(INFO, LINK, START, AVAIL, ITEM)**

**This algorithm inserts ITEM as the first node in the list**.

Step 1 : [OVERFLOW?]   
 If AVAIL = NULL, then :   
 Write : OVERFLOW, and Exit.

Step 2 : [Remove first node from AVAIL list.]  
 Set NEW := AVAIL and

AVAIL := LINK[AVAIL].

Step 3 :Set INFO[NEW] := ITEM.  
 [Copies new data into new node].

Step 4 : Set LINK[NEW] := START.  
 [New node now points to original first node.]

Step 5 : Set START := NEW.  
 [Changes START so it points to new node.]

Step 6 : Exit.

**Algorithm:-INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)  
This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.**

Step 1 : [OVERFLOW?]   
 If AVAIL = NULL, then :

Write : OVERFLOW, and Exit.

Step 2 : [Remove first node from AVAIL list.]   
 Set NEW := AVAIL and AVAIL :=   
 LINK[AVAIL].

Step 3 : Set INFO[NEW] := ITEM.  
 [Copies new data into new node].

Step 4 : If LOC = NULL, then :

[Insert as first node.]  
 Set LINK[NEW] := START and

START := NEW.  
 Else : [Insert after node with location LOC.]  
 Set LINK[NEW] := LINK[LOC]   
 and LINK[LOC] := NEW.  
 [End of If structure.]

Step 5 : Exit.

**Algorithm:- DEL(INFO,LINK,START,AVAIL,LOC,LOCP)**

**Let LIST be a linked list in memory. Suppose we are given the location LOC of node N in LIST. And ,suppose we are given the location LOCP of the node preceding N or, when N is the first node ,we are given LOCP=NULL. The following algorithm deletes N from the list**.

1.If LOCP=NULL, then:

Set START:=LINK [START]. [ Deletes first node].

else:

Set LINK[LOCP]:= LINK[LOC]. [ Deletes N node.]

[End of if structure]

2.[Return deleted node to the AVAIL list.]

set LINK[LOC]:=AVAIL and AVAIL:=LOC.

3.Exit.

**Algorithm:-INSERT(INFO,LINK,START,AVAIL,ITEM)**

**This algorithm inserts ITEM into a sorted linked list.**

1. [Use Procedure to find the location of the node preceding ITEM.]

Call FINDA(INFO,LINK,START,ITEM,LOC)

2. [Use Algorithm to insert ITEM after the node with location LOC].

Call INSLOC(INFO,LINK,START,AVAIL,LOC,ITEM).

1. Exit

Prac 8 :

**Algorithm: PostOrder Traversal**

**POSTORD(INFO,LEFT,RIGHT,ROOT)**

A binary tree T is in memory. This algorithm does a postorder traversal of T , applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

1. [Push NULL onto STACK and initialize PTR.]

Set TOP:=1,STACK[1]:=NULL and PTR:=ROOT.

2. [Push left-most path onto STACK.]

Repeat steps 3 to 5 while PTR!=NULL:

3.Set TOP:=TOP+1 and STACK [TOP]:=PTR.[Pushes PTR on STACK].

4.If RIGHT[PTR]!=NULL , then :[Push on STACK] Set TOP:=TOP+1 and STACK[TOP]:=-RIGHT[PTR][End of If structure.]

5.Set PTR:=LEFT[PTR].[Updates pointer PTR.] [End of step 2 loop.]

6.Set PTR:=STACK[TOP] and TOP:=TOP-1. [Pops node from STACK.]

7.Repeat while PTR>0:

(a)Apply PROCESS to INFO[PTR].

(b)Set PTR:=STACK[TOP] and TOP:=TOP-1.

[pops node from STACK.]

[End of loop.]

8.If PTR<0,then:

(a)Set PTR:=-PTR.

(b)Go to step 2.

[End of If structure.]

9.Exit.

Prac 9 :

**Step by Step Process**

The Heap sort algorithm to arrange a list of elements in ascending order is performed using following steps...

* **Step 1 -**Construct a **Binary Tree** with given list of Elements.
* **Step 2 -**Transform the Binary Tree into **Max Heap.**
* **Step 3 -**Delete the root element from Max Heap using **Heapify** method.
* **Step 4 -**Put the deleted element into the Sorted list.
* **Step 5 -**Repeat the same until Max Heap becomes empty.
* **Step 6 -**Display the sorted list.

Prac 10 :

**Algorithm:-**

1. Repeat for I, J=1,2,……….,M:

W[I,J]=0 then Set Q[I,J]:=INFINITY.

Else: Set Q[I,J]:=w[I,J].

[End of loop].

2. Repeat steps 3 and 4 for K=1,2,…………..,M:

3. Repeat Step 4 for 1, 2,…..,M:

4. Repeat for J=1, 2,…..,M:

Set Q[I,J]:=MIN(Q[I,J],Q[I,K]+Q[K,J]).

[End of loop].

[End of step 3 loop]

[End of Step 2 loop].

5. Exit.