Addition of Two Polynomials [Application of Linked List]

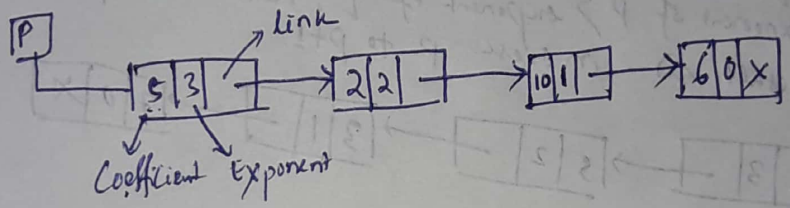
$P_1(x) = a_n x_n^e + a_{n-1} x_{n-1}^e + \cdots + a_1 x_1^e + a$

$P_2(x) = b_n x_n^e + b_{n-1} x_{n-1}^e + \cdots + b_1 x_1^e + b$

$P_1(x) + P_2(x)$: Linked List

Node :
- Coefficient field.
- Exponent field.
- Link field

$P(x) = 5x^3 + 2x^2 + 10x + 6$.




```
Struct Poly
 {
   int Coeff;
   int expo;
   struct poly * ptr;
 };
```

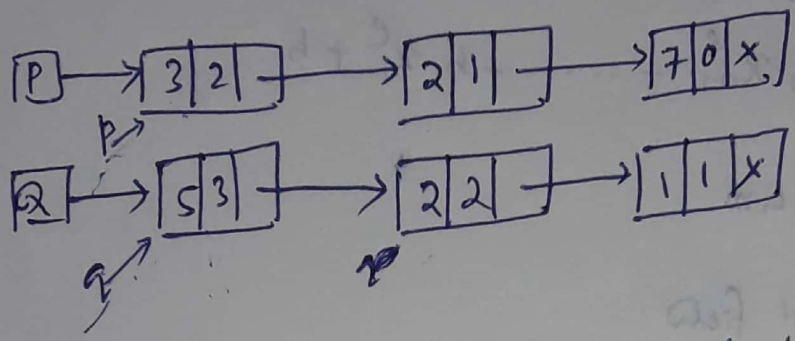
Algo :-

i) Read no. of terms in first polynomial 'p'.
ii) Read Coefficient and exponent of first polynomial 'p'.
iii) Read no. of terms in 2nd polynomial 'q'.
iv) Read Coefficient and exponent of 2nd polynomial 'q'
v) Set temp pointers 'p' & 'q' to traverse to polynomial respectively
vi) Compares the exponents of two polynomials starting from 1st node.

EX

$P = 3x^2 + 2x + 7$

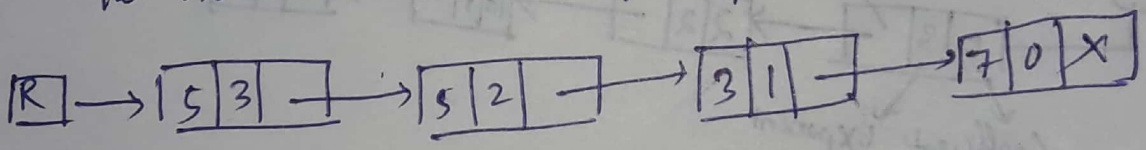
$Q = 5x^3 + 2x^2 + x$



vii) If both exponents are equal then add coefficients and store in result link list

viii) If exponent of P < exponent of q then add term of q in result and move q to point next node.

ix) If exponent of P > exponent of q, then add 'p' term in the result and increase p to p++



$R = 5x^3 + 5x^2 + 3x + 7$

Program

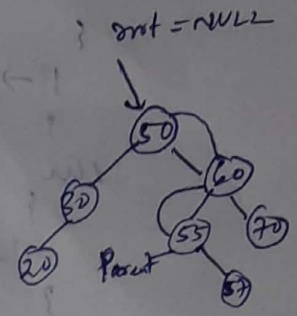
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

struct node
  {
   int num
   int coeff;
   struct node *next;
  };
```

Struct node

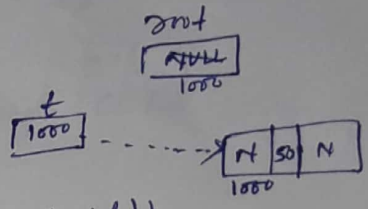
Insertion of Element into BST :

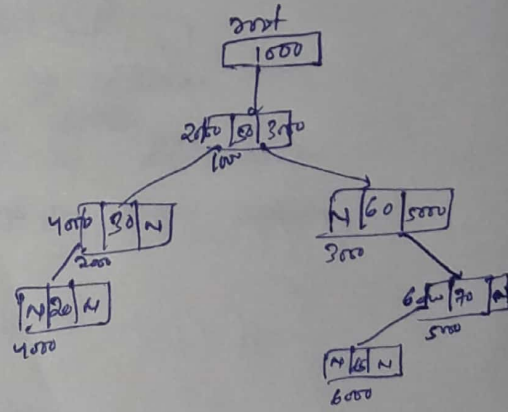
→ 1st we need to create the node



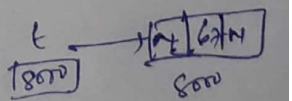

```
Struct Node
{
   int data
   Struct Node * left;
   Struct Node * Right;
};
Struct Node * root = NULL
```

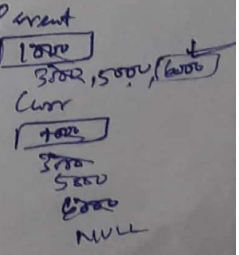
```
Void insert (int d)
{
   Struct Node * t, *p
   t = (Struct Node*) malloc( sizeof (struct Node));
   t->data = d;
   t->left = NULL;
   t->Right = NULL;
   p = root;
   if (root== NULL)
    {
      root = t;
    }
```



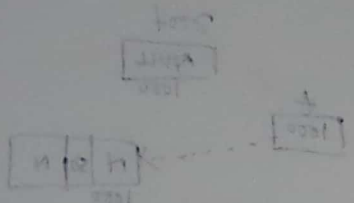


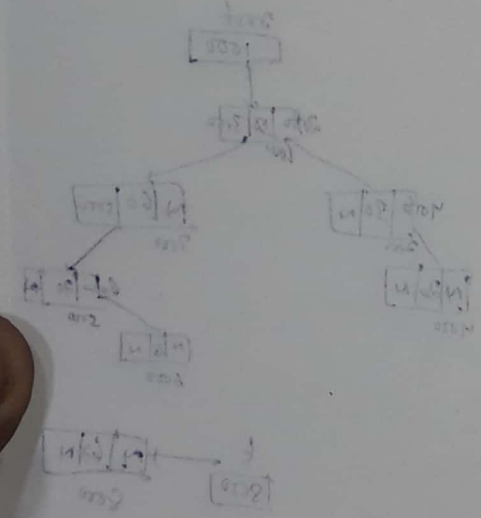

```
else
{
   Struct Node * Curr;
   Curr = root;
   while (Curr)
   {       p = Curr;
      if (t->data > Curr->data)
            {
               Curr = Curr->right;
            }
      else
            {
            Curr = Curr->left;
            }
   }
```



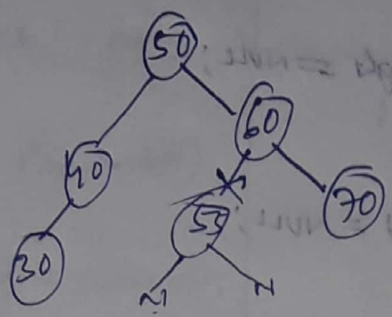


```
if (t->data > p->data)
{
    p->right = t;
}
else
{
    p->left = t;
}
```

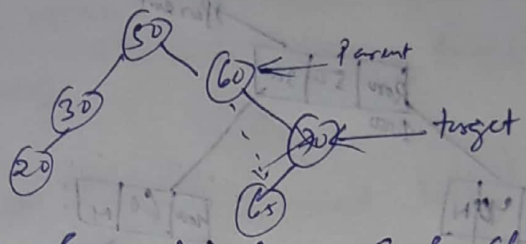




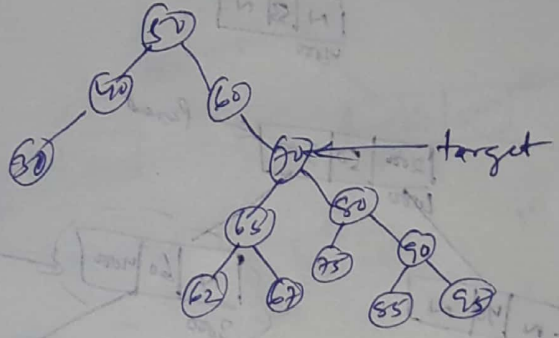
Deletion operation on BST:



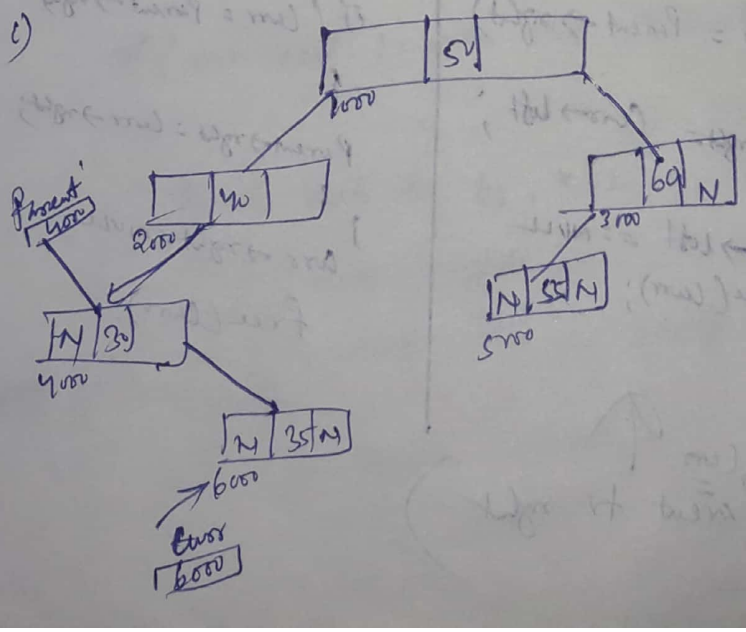
I) Delete a leaf node having no children



II) Deletion of node having single child



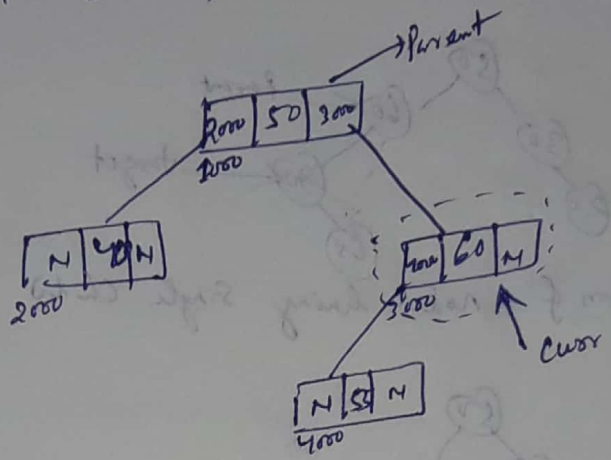
III) Deleting a node having 2 children
Deleted node replaced either either
→ Highest value of its left subtree
→ Least value of its Right subtree.

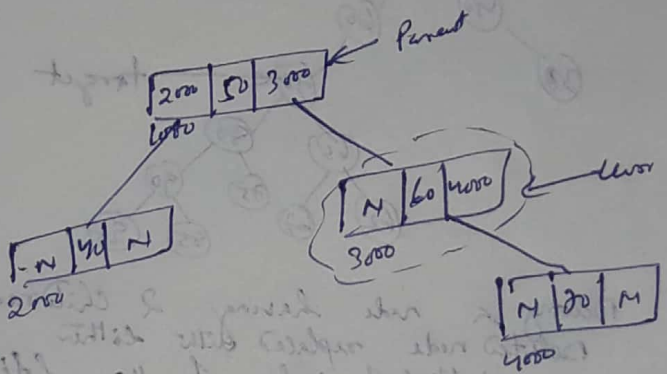


D

```
if (curr == Parent -> right)
{
  Parent -> Right = NULL;
}
Else
{
  Parent -> Left = NULL;
}
free(curr);
```

II)






```
if (Curr -> Left != NULL)
{
  if (Curr = Parent -> right)
  {
    Parent -> right = Curr -> Left;
  }
  Curr -> Left == NULL
  free(Curr);
}
```

```
if (Curr -> Right != NULL)
{
  if (Curr = Parent -> right)
  {
    Parent -> right = Curr -> right;
  }
}
Curr -> right = NULL
free(Curr);
```

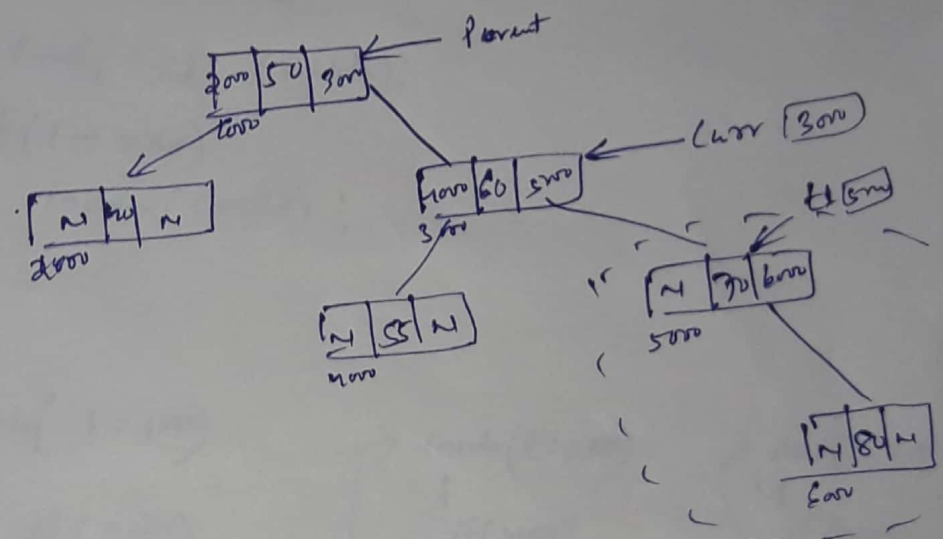
(Curr = Parent to right)

```
if (curr->right != NULL)
{
  if (curr = Parent->left)
    {
      Parent->left = curr->right;
    }
}
curr->right = NULL;
free(curr);
```

```
if (curr->left != NULL
{
  if (curr = Parent->left)
  {
    Parent->left = curr->left;
  }
}
curr->left = NULL;
free(curr);
```

if (Curr = Parent left)

iii)

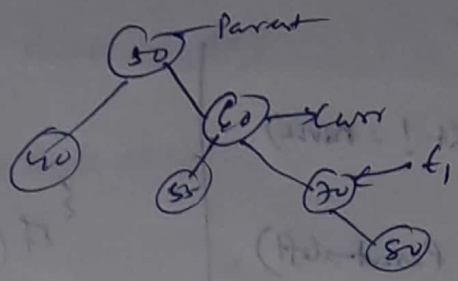


→ least element in the right subtree

```
if (curr->left != NULL && Curr->Right != NULL)
  {
    struct Node * t1, * t2
    t1 = curr->right;
```

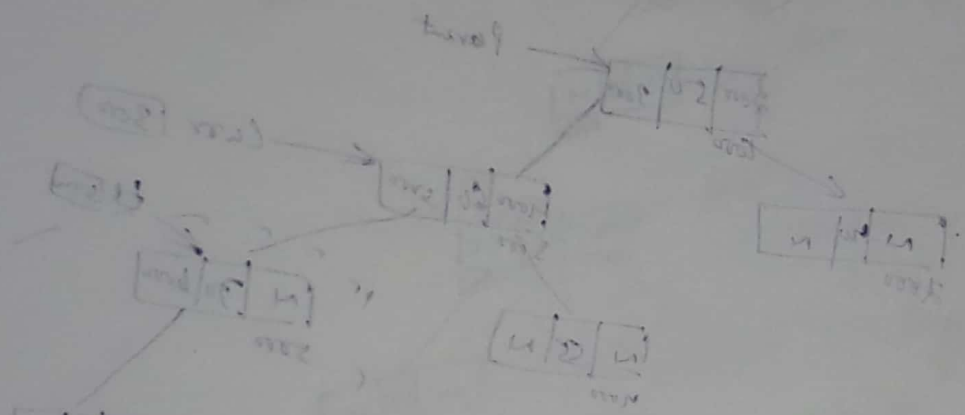
approach-1 → `if (t1->left == NULL && t1->right == NULL)`

```
    {
      Curr->data = t1->data;
      Curr->right = NULL
      free(t1);
    }
  }
```

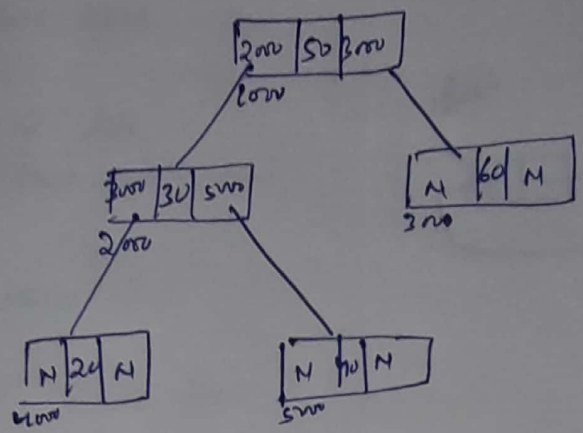


Approach ②

```
if ( t1 -> right != NULL && t1 -> left = NULL)
    {
    Curr -> data = t1 -> data;
    Curr -> right = t1 -> right;
        t1 -> right = NULL;
        free (t1);
    }
```



Inorder Traversal in BST.




```
inorder( struct Node *t)
{
  if (t->left)
     inorder (t->left);
  Printf("%d", t->data);
  if(t->right)
     inorder(t->right);
}
```

Execution:-

```
inorder( t = 1000)
{
  if( 2000)
  inorder(2000);
  Printf(50);
  if (3000)
  inorder 3000);
  Printf(60); }
```

```
inorder(t=2000)
{
if(4000)
  inorder(4000);
Printf(30);
if(5000)
  inorder(5000)
}
```

```
inorder(t=4000)
{
 if (NULL)
   X
 Printf(20);
 if (NULL)
```

```
inorder(t=5000)
{
 if(NULL)
   X
 Printf(40);
 if( NULL)
   X
```

Circular Linked List creation ..

```
Struct node
{
   int data;
   Struct node *next;
};
int main()
{
   int  i, n, item;
   Struct node *new_node, *p;
   Printf (" enter no. of nodes");
   Scanf (" %d", &n);
   Printf (" enter 0th node");
   Scanf (" %d", & item);
   new_node = (struct node *) malloc (size of (new_node));
   new_node → data = item;
   new_node → next = NULL;

   head = new_node;
   p = new_node;
   for (i=1; i<n; i++)
     { Printf (" enter next node");
       Scanf (" %d", & item);
     new_node = (Struct node *) malloc (sizeof (new_node));
     new_node → data = item;
     new_node → next = NULL;
        p → next = new_node;
         p = p → next;
     }
   p → next = head;
}
```