7 2018 August
Tuesday

July

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Binary search tree

1) insertion

```
struct node * insert(struct node * tree, int x)
{
    if (tree == NULL)
        return (newNode(x));
    if (x > tree->data)
        tree->right = insert(tree->right, x);
    else
        tree->left = insert(tree->left, x);
    return tree;
}
struct node * newNode(int x)
{
    struct node * temp = (struct node*) malloc (sizeof(struct node
    temp->data = x;
    temp->right = temp->left = NULL;
    return temp;
}
```

8 Wednesday

⇒ in BST fn make all functions return struct node not int or void.
else we have to do call by address which will be hard.

2) minimum

```
Struct node * Min(struct node * tree)
{
    Struct node * temp = tree;
    while (temp->left != NULL)
        temp = temp->left
    return (temp);
}
```

3) Sum of leaf nodes

```
void leafsum (struct node* root, int *sum) {
if (!root)
    return;
if (!root->left && !root->right)
    *sum+= root->data;

leafSum (root->left, &(*sum));
leafSum (root->right, &(*sum));
}
```

4) Kth largest root (traverse from opposite end) decreasing order

```
void Kl ( Node* root, int &h, int k)
{
if (!root || h>=k)
    return;
Kl (root->right, h, k);
h++;
if (h==k)
    printf ("%d\n", root->data);
Kl (root->left, h, k);
}
void kth largest (Node* root, int k)
{
int h=0;
Kl (root, h, k);
}
```

Friday 10

5] Predecessor

```
struct node * predecessor (struct node *tree, int i)
{
  struct node * temp = tree;
  struct node * new = NULL;
  struct node* a = search (tree, i);
  if (a == NULL)
  {  printf ("NULL\n");
     return new;
  }
  if ((Min (tree)) -> data == i) {
     printf ("-1\n");
     return new;
  }
  while (temp -> data != i)
  {
     if (temp -> data != i)
     {
        new = temp;
        temp = temp -> left;
     }
     else
        temp = temp -> right;
  }
  if (temp -> left != NULL)
     printf ("%d\n", (Max (temp -> left)) -> data);
  else
     printf ("%d\n", new -> data);
}
```

for successor

if ((Max (tree)) -> data ==

temp = temp -> right

temp = temp -> left

-> Min (temp -> right

12 Sunday

6) Binary tree to parenthesis using preorder traversal

```
void paren (struct node *tree)
{
  if (tree! =NULL)
{
    printf ("(");
    printf ("%d", tree->data);
    paren ( tree->left);
    paren(tree->right);
    printf (")");
}
  else
    printf ("()");
}
```

Tuesday

7) Search

```
struct node* search (struct node * tree , int k)
{
  if (tree == NULL || tree->data == k)
      return tree;
  if (k > tree->data)
      return search ( tree->right, k);
  else
      return search (tree->left, k);
}
```

8) Delete

```
struct node * delete (struct node * tree, int i)
{
    if (tree == NULL)
    {
        printf ("NULL \n");
        return tree;
    }
    if (i < tree -> data)
        tree -> left = delete (tree -> left, i);
    else if (i > tree -> data)
        tree -> right = delete (tree -> right, i);
    else
        {
            if (tree -> left == NULL)        // Single child
            {
                struct node * temp = tree -> right;
                free (tree);
                return temp;
            }
    else if (tree -> right == NULL) {
        struct node * temp = tree -> left;    free (tree);
        return temp;
    }
                // 2 child
    struct node * temp = Min (tree -> right);
    tree -> data = temp -> data
    tree -> right = delete (tree -> right, temp -> data
}
Return tree;
}
```

16 Thursday

9) to take parethesis as input

calling → tree = insr();

```
struct node* insr()
{
    char ch;
    int n;
    scanf("%c", &ch);
          space
    if (scanf("%d", &n))
    {
        struct node *temp = (struct node*) malloc(sizeof(struct node));
        temp → data = n;
        temp → left = insr();
        temp → right = insr();
        scanf("%c", &ch);
        return temp;
    }
    scanf("%c", ch);
    return NULL;
}
```

Saturday 18

10) get level

```
int getLevel (struct node* tree, int data)
{
return getLevelUtil (tree, data, 1);
}
int getLevelUtil (struct node* tree, int data, int level)
{
if (tree == NULL);
    return NULL
if (tree -> data == data)
    return level;
int downlevel = getLevelUtil (tree->left, data, level+1);
if (downlevel != 0)
    return downlevel;
downlevel = getLevelUtil (tree->right, data, level+1);
    return downlevel;
}
```

20 Monday

11) height

```
int height (struct node* tree) {
    int r=0, l=0;
    struct node* temp1 = tree;
    while (temp1->left != NULL) {
            l++; temp1 = temp1->left;
    } struct node* temp2 = tree;
    while (temp2->right != NULL) {
            r++; temp2 = temp2->right;
    }
    if (r>l)
        return (r+1);
    else
        return (l+1);
```