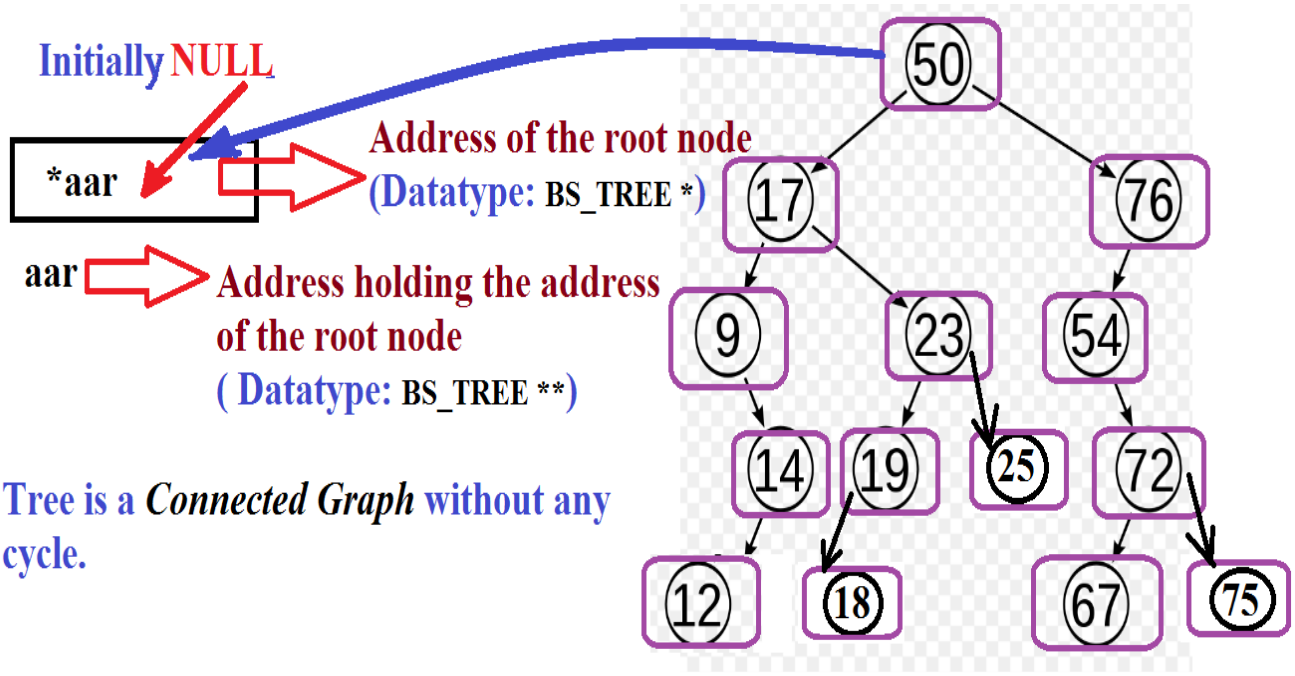
Binary Search Tree

Dr. Rahul Das Gupta

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

typedef  struct bs_tree
{
    int data;
    struct bs_tree *left,*right;
}BS_TREE;
```




```
/*Recursive insertion*/
/* ============================
```

aar = address holding the address of the root node (Data type: BS_TREE **)
***aar = address of the root node (Data type: BS_TREE *)**
==========*/

```
void initialise_tree (BS_TREE **aar)
{
      *aar = NULL;
}

void rec_insert (BS_TREE **aar, int n)
{
   if (*aar == NULL)
   {
     *aar=(BS_TREE *)malloc(sizeof(BS_TREE));
     (*aar)->data=n;
     (*aar)->left=(*aar)->right=NULL;
   }
   else if (n<(*aar)->data)
        rec_insert(&((*aar)->left), n);
   else
       rec_insert(&((*aar)->right), n);
}

/*
```

Iterative Insertion in Binary Search Tree
***/**

```
void insertion_iterative (BS_TREE **aar, int n)
{
   BS_TREE *t, *parent, *cur;
   t=(BS_TREE *)malloc(sizeof(BS_TREE));
   t->data=n;
   t->left=t->right=NULL;
   if(*aar == NULL)
         *aar = t;
   else
   {
         par=NULL;
         cur=*aar;
         while (cur !=NULL)
          {
              par = cur;
              if (n<cur->data)
                        cur = cur->left;
              else
                        cur = cur->right;
          }
         if (n< par->data)
```

```
                par->left=t;
        else
                par->right=t;
    }
}
```

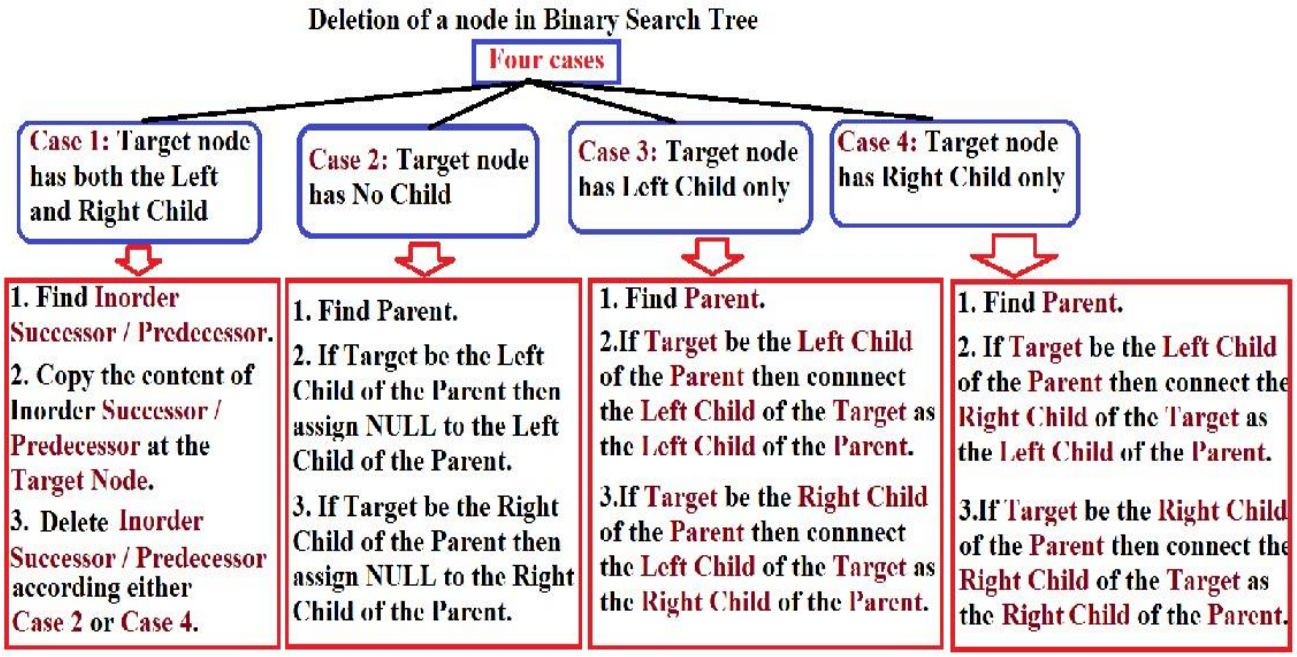
```
/*Pre-order Traversal*/
void preorder(BS_TREE *ar)
{
    if(ar)
   {
      printf("%d", ar->data);
      preorder(ar->left);
      preorder(ar->right);
    }
}
```

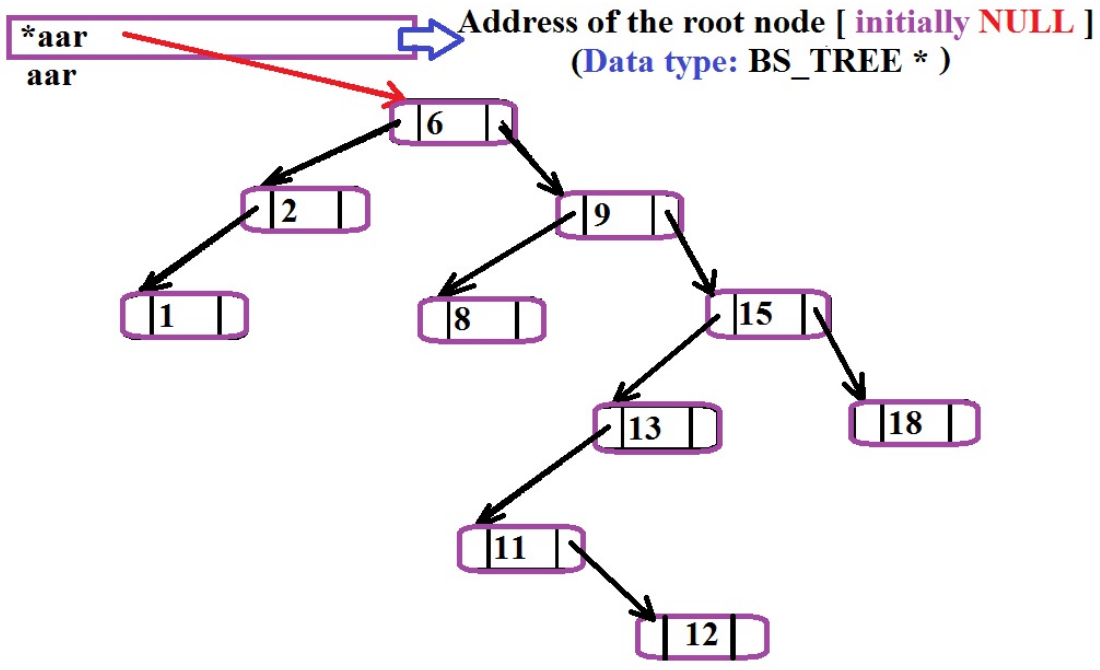
```
/*In-order Traversal*/
void inorder(BS_TREE *ar)
{
    if(ar)
   {
      inorder(ar->left);
      printf("%d", ar->data);
      inorder(ar->right);
    }
}

/*Post-order Traversal*/
void postorder(BS_TREE *ar)
{
    if(ar)
   {
      postorder(ar->left);
      postorder(ar->right);
      printf("%d", ar->data);
    }
}
```

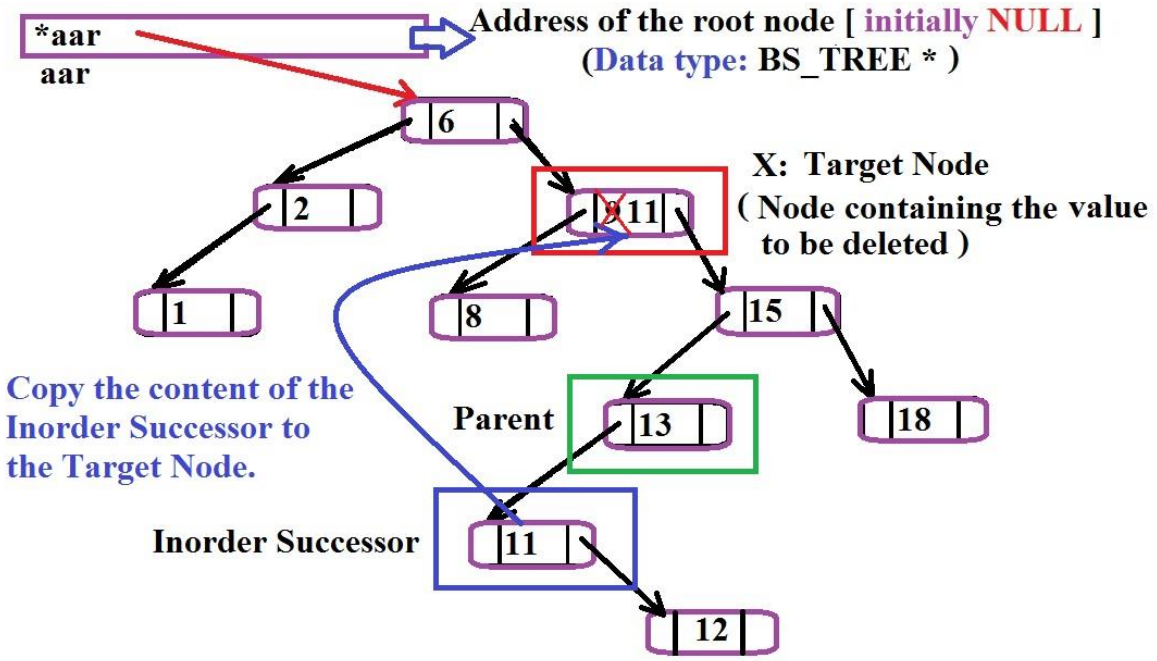
```
void nonrecursive_inorder (BS_TREE *ar)
{
   int top=-1;
   BS_TREE **stk;
   stk = (BS_TREE **) malloc (sizeof (BSTREE *)*MAX);
   while(1)
   {
      while(ar)
      {
         if(top==MAX-1)
         {
            printf("\n Overflow…");
            exit(1);
         }
          stk[++(top)]=ar;
          ar= ar->left;
      }
      if(top!=-1)
      {
         ar = stk[top--];
```

```
            printf ("\t %d", ar->data);
            ar=ar->right;
        }

      else
          break;
  }
}

void nonrecursive_preorder (BS_TREE
*ar)
{
   int top=-1;
   BS_TREE *stk[MAX];
   while(1)
   {
      while(ar)
      {
            printf ("\t %d",ar->data);
             if(top==MAX-1)
             {
```

```
                printf("\n Overflow…");
                exit(1);
            }
            stk[++(top)]=ar;
            ar= ar->left;
        }
        if(top!=-1) /*when stack is non-empty */
        {
            ar=stk[top--];
            ar=ar->right;
        }
        else   /* when stack is empty */
            break;
    }
}
```


Deletion of a node in Binary Search Tree
Four cases
Case 1: Target node has both the Left and Right Child
Case 2: Target node has No Child
Case 3: Target node has Left Child only
Case 4: Target node has Right Child only
1. Find Inorder Successor / Predecessor.
2. Copy the content of Inorder Successor / Predecessor at the Target Node.
3. Delete Inorder Successor / Predecessor according either Case 2 or Case 4.
1. Find Parent.
2. If Target be the Left Child of the Parent then assign NULL to the Left Child of the Parent.
3. If Target be the Right Child of the Parent then assign NULL to the Right Child of the Parent.
1. Find Parent.
2.If Target be the Left Child of the Parent then connnect the Left Child of the Target as the Left Child of the Parent.
3.If Target be the Right Child of the Parent then connnect the Left Child of the Target as the Right Child of the Parent.
1. Find Parent.
2. If Target be the Left Child of the Parent then connect the Right Child of the Target as the Left Child of the Parent.
3.If Target be the Right Child of the Parent then connect the Right Child of the Target as the Right Child of the Parent.


*aar
aar
Address of the root node [initially NULL]
(Data type: BS_TREE *)
6
2
9
1
8
15
13
18
11
12



/**********

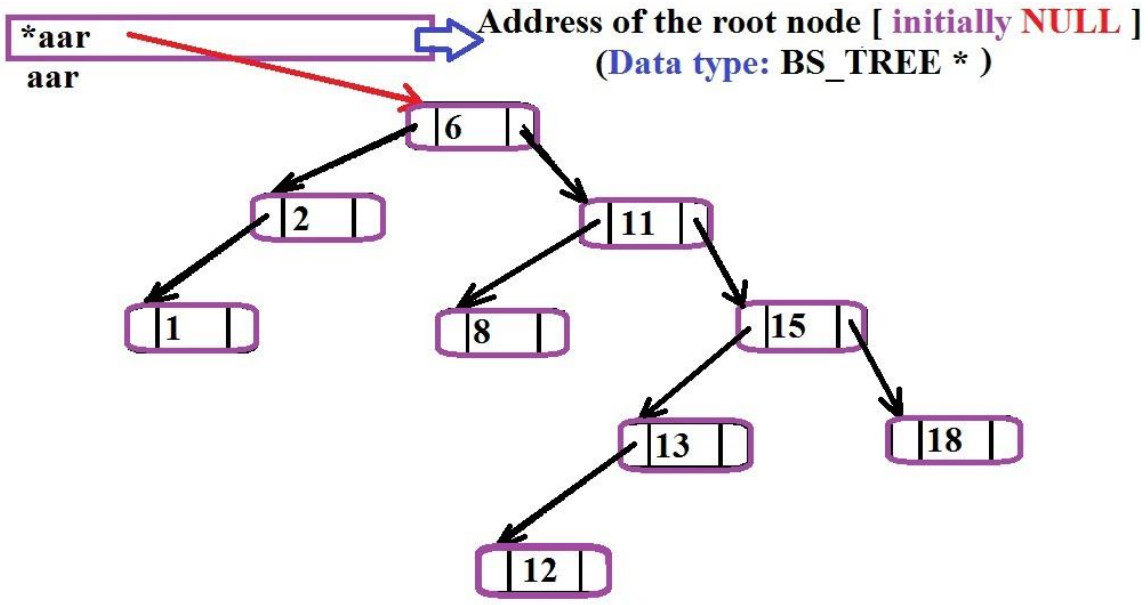
How to find In-order Successor??

Answer: Go one step right immediately below the target node and then go to extreme left.

How to find In-order Predecessor??

Answer: Go one step left immediately below the target node and then go to extreme right.

**********/




```
void deletion (BS_TREE **aar, int n)
{
   int found=0;
   BS_TREE *x,*par,*in_order_suc;
    if(*aar==NULL)
   {
      printf("\n Empty Tree…");
      return;
   }
   x=NULL;
   par=NULL;
   search(aar, &x, &par, &found, n);
```

```
    if (found==0)
    {
       printf("\n Record not found…");
       return;
    }
/* Case-1: When the node to be deleted has both left and right child.*/
if(x->left!=NULL && x->right!=NULL)
{
   par=x;
   in_order_suc=x->right;
   while (in_order_suc->left !=NULL)
    {
         par= in_order_suc;
         in_order_suc= in_order_suc->left;
    }
/* Copy the data of the inorder successor.*/
    x->data= in_order_suc->data;
    x= in_order_suc;
/* Not to return now. Now x to be deleted according to the method mentioned in the module Case-2 or Case -4.
```

```
Go to Case-2 when x has no child.
Go to Case-4 when x has right child.*/
}

/* Case-2: When the node to be deleted has neither the left child nor right child.*/
if(x->left==NULL && x->right==NULL)
{
  if (x==*aar) /* when the node to be deleted is the root node.*/
            *aar=NULL;
  else if (par->left==x)
            par->left=NULL;
  else
            par->right=NULL;
   free(x);
   return;
}
/* Case-3: When the node to be deleted has the left child only.*/
if(x->left !=NULL && x->right==NULL)
{
```

```
   if (x==*aar) /* when the node to be
deleted is the root node.*/
            *aar=(*aar)->left;
   else if (par->left==x)
            par->left=x->left;
   else
            par->right=x->left;
    free(x);
    return;
}

/* Case-4: When the node to be deleted has
the right child only.*/
if(x->left ==NULL && x->right !=NULL)
{
   if (x==*aar) /* when the node to be
deleted is the root node.*/
            *aar=(*aar)->right;
   else if (par->left==x)
            par->left=x-> right;
   else
            par->right=x-> right;
```

```
    free(x);
    return;
}
/* The end…*/
}

void search (BS_TREE **aar, BS_TREE
**a_x, BS_TREE **a_par, int *a_found, int
n)
{
    BS_TREE *q=*aar;
    *a_par=NULL;
    *a_found=0;
     while (q!=NULL)
      {
        if (n==q->data)
        {
          *a_x=q;
          *a_found=1;
          return;
        }
       else if (n<q->data)
```

```
        {
          *a_par=q;
           q = q->left;
         }
        else
        {
           *a_par=q;
            q = q->right;
         }
      }/* End of while (q!=NULL)*/
}

int recursive_search (BS_TREE  *ar, int n)
{
   if (ar!=NULL)
   {
      if (ar->data==n)
      {
         printf ("\n Data found…");
         return (1);
      }
      else if (n>ar->data)
```

```
                recursive_search (ar->right, n);
        else
                recursive_search (ar->left, n);
    }
    else
     {
        printf ("\n Data not found…");
        return (0);
     }
}
```