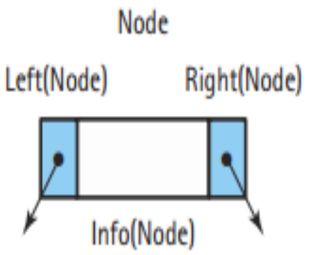
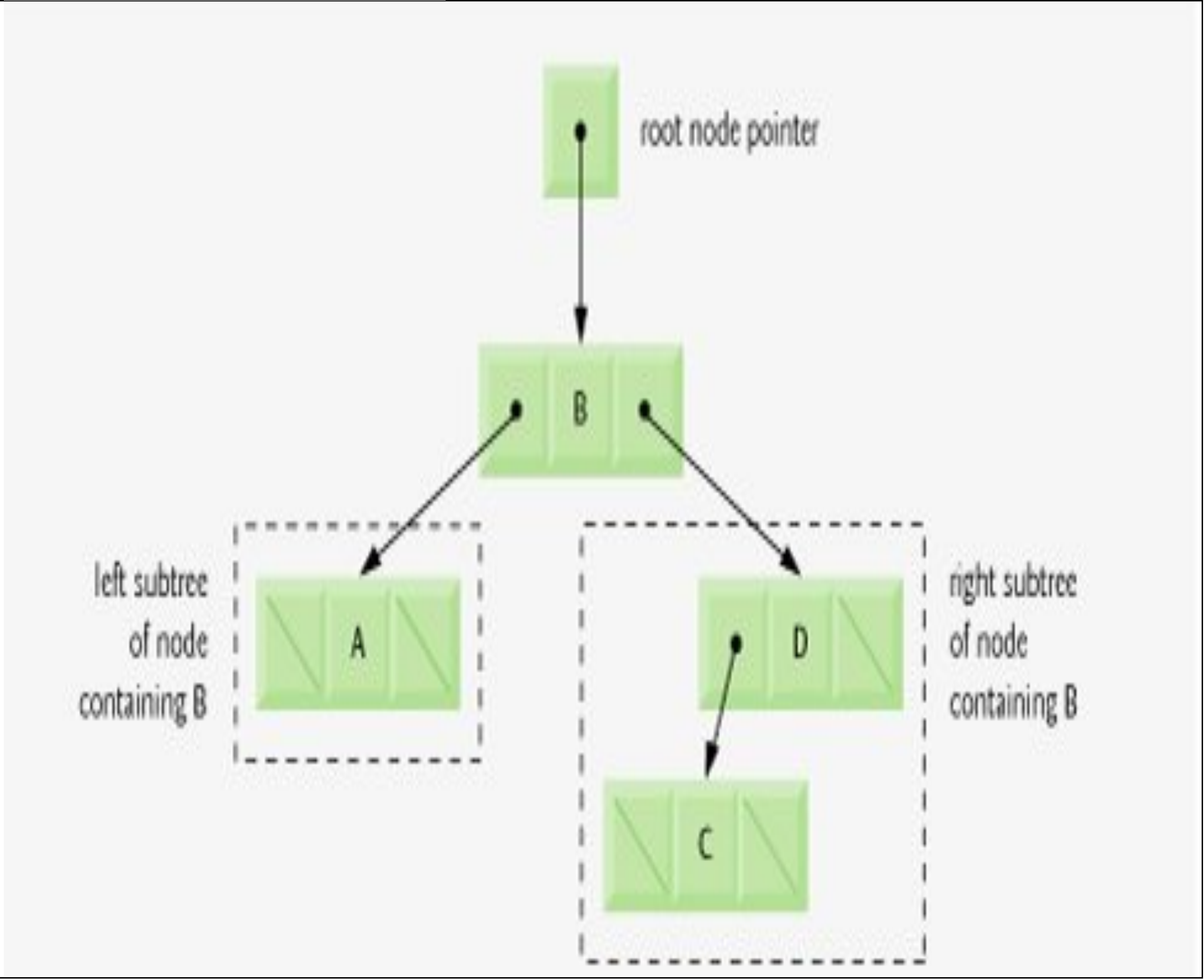
Binary tree

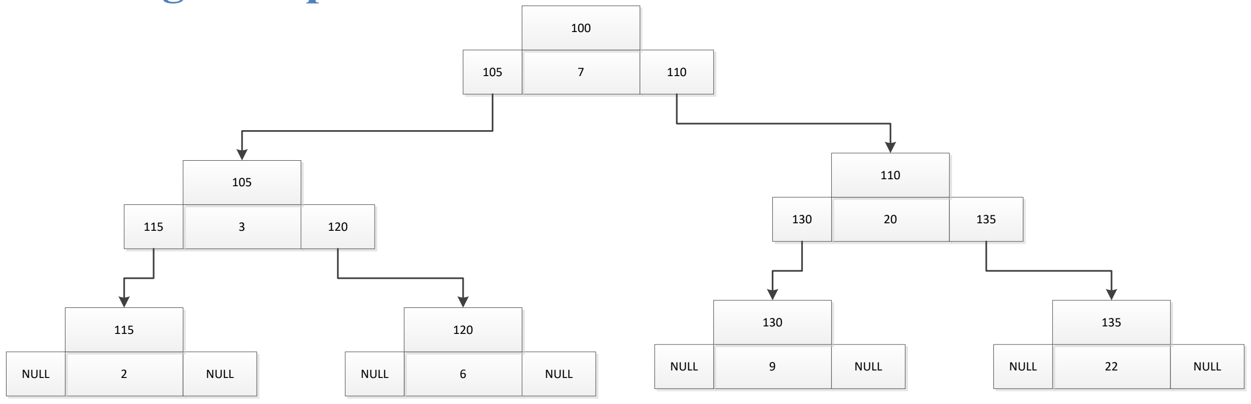
```
struct TreeNode
{
  ItemType info;
  TreeNode* left;
  TreeNode* right;
};
```



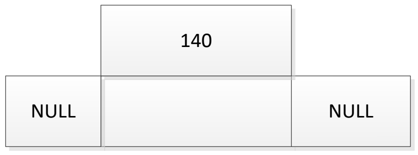

root node pointer
B
A
D
C
left subtree of node containing B
right subtree of node containing B

Binary Search Tree

Creating a simple tree



Creating a new node:



Example 1a

```
            7
     3              20
2    |  6         9  |   22
```

```
struct BinaryTree {
    int key;
    BinaryTree *left;
    BinaryTree *right;
};

BinaryTree * newNode(int key){
    BinaryTree *node=new BinaryTree;
    node-> key = key;
    node->left=NULL;
    node->right=NULL;
    return (node);
}

int main( ){
    BinaryTree *root;

    root=newNode(7);
    root->left=newNode(3);
    root->right=newNode(20);
    root->left->left=newNode(2);
    root->left->right=newNode(6);
    root->right->left=newNode(9);
    root->right->right=newNode(22);
}
```

Inserting a node

Create node first in (ins), then decide where to put it, either at the root or at the leaf position (insertNoRecursion). When placing in the leaf position, return back the current leaf and add to its either left or right(insert)
Ins->insert->insertNoRecursion

Example 1D

```
struct BinaryTree {
    int key;
    node *left;
    node *right;
};
```

```
          7
    3           20
2   |  6      9  |  22
```

```
void insertNoRecursion(BinaryTree *&root, int key){
    bool done=false;
    BinaryTree *leaf=root;
    if (!root)
      add(root, key)
    else
      while (!done){
        if (key== leaf->key)
            done=true;
        else if (key < leaf->key)
            if (leaf ->left !=NULL)
                leaf = leaf->left;
            else{
                add(leaf->left, key);
                done=true;
            }
        else{
            if (leaf ->right!=NULL)
                leaf = leaf->right;
            else {
                add(leaf->right, key);
                done=true;
            }
        }
  }

void add(BinaryTree *&leaf, int key){
    leaf=new node;
    leaf->key=key;
    leaf->left=NULL;
    leaf->right=NULL;
}
int main( ){
    struct BinaryTree *root;
    insert (root, 7);
    insert (root, 3);
    ...
}
```

```
        7
   3          20
2  |  6     9  |  22
```

Example 1E (Tail Recursive)

```
struct BinaryTree {
    int key;
    node *left;
    node *right;
};


//The key here is the ampersand, Tail recursive algorithm
//Connects because of the ampersand
void Insert(struct BinaryTree *& leaf, int item){
      if (!leaf){// Insertion place found.
            leaf = new node;
            leaf->right = NULL;
            leaf->left = NULL;
            leaf->key = item;
      }
      else if (item < leaf->key)
          Insert(leaf->left, item) ;    //Insert in left subtree.
      else if (item > leaf->key)
          Insert(leaf->right, item) ;   //Insert in right subtree.
}

int main( ){
    struct BinaryTree *root;
    insert (root, 7);
    insert (root, 3);
    ...
}
```

```
        7
   3          20
2  |  6     9  |  22
```

Insert 8, R=Right, L=Left

INS(&7,8) INS (20,8)R	INS (&20,8) INS (9,8)L	INS (&9,8) INS(NULL,8)L	INS(&NULL,8) Create node

Example 1F (Tail Recursive WRONG)

```
struct BinaryTree {
    int key;
    node *left;
    node *right;
};

//This is problematic because the new node is not connecting to
//the left or right of the tree. It just creates it. It goes to
//the right spot, it goes to the left or right until it becomes null.
//So now pointing to nothing we create a node but we need to attach
//leaf->left or leaf-right to this new node.
//One way to correct the problem is to use an Ampersand so that
//it is being passed by reference. We are passing leaf->right or left which is
//containing a null and now we are going to point to a node.
void insert_wrong (BinaryTree *leaf, int key){

    if (!leaf){// Insertion place found.
             leaf = new node;
             leaf->right = NULL;
             leaf->left = NULL;
             leaf->key = item;
    }else if (key>leaf->key)
            insert_wrong(leaf->left, key);
     else if (key < leaf->key)
          insert_wrong (leaf->right, item) ;   //Insert in right subtree.

}
```

```
         7
   3           20
2  |  6      9  |  22
```

Insert 8

INS(7,8)	INS (20,8)	INS (9,8)	INS(NULL,8)
INS (20,8)R	INS (9,8)L	INS(NULL,8)L	Create node

Example 1G (Non-tail recursive)

```
struct BinaryTree {
    int key;
    node *left;
    node *right;
};

//It travels down until it gets to the bottom of the tree, creates node and
//then goes back up to connect it to the tree
node* insert_correct(BinaryTree * leaf, int data) {
  if (!leaf) ){// Insertion place found.
             leaf = new node;
             leaf->right = NULL;
             leaf->left = NULL;
             leaf->key = item;
  } else if (data < node->key)
        leaf->left  = insert_correct(leaf->left, data);
    else if (data > node->key)
        leaf->right = insert_correct(leaf->right, data);
  return(leaf);
}
```

```
         7
   3           20
2  |  6      9  |  22
```

Insert 8

Going	Coming back

INS(7,8)
leaf->right = INS (20,8)R
return (leaf)
	leaf->right=20 //reconnects old stuff
	return 7

INS (20,8) leaf->left = INS (9,8)L return (leaf)	leaf->left=9 //reconnects old stuff return 20
INS (9,8) leaf->left = INS(NULL,8)L return (leaf)	leaf->left=8 //connect 8 to 9 return 9
INS(NULL,8) Create node Return 8	

Searching for a value

```
        7
   3          20
2  |  6     9  |  22
```

Example 1B

```
struct BinaryTree {
    int key;
    node *left;
    node *right;
};
```

```
bool searchNoRecursion(BinaryTree *leaf, int key){
    bool found=false;

    //use a while loop and not recursion. Leaf is being changed
    //each and every time to point to the left or right, going down
    //the tree.
    while (leaf && !found){
        if (key==leaf->key){
            found=true
        }else if (key < leaf->key)
            leaf=leaf->left; //replaces recursion
        else
            leaf=leaf->right;
    }
    return found;
}
```

Example 1C (Tail Recursive)

```
struct BinaryTree {
    int key;
    node *left;
    node *right;
};

/*
    First make sure that the node has something in it. If it doesn't, then
    nothing is found; otherwise, we have one of 3 conditions:
      It is found (All done).
      It is less, which means move to our left
      It is greater, which means move to the right.

    This is a tail-recursive solution.
*/

bool searchRecursion(BinaryTree *leaf, int key){
    bool found=false;
    if (leaf){
        if (key==leaf->key)
           found=true ;
        else if (key < leaf->key) //go to the left if small
            found=searchRecursion (leaf->left, key);
        else //go to right if bigger
            found=searchRecursion (leaf->right, key);
    }
    return found;
}
```

```
        7
   3          20
2  |  6     9  |  22
```

Look for 6, SR stands for SearchRecursion

SR(7,6)	SR(3,6)	SR(6,6)
SR(3,6)	SR(6,6)	found