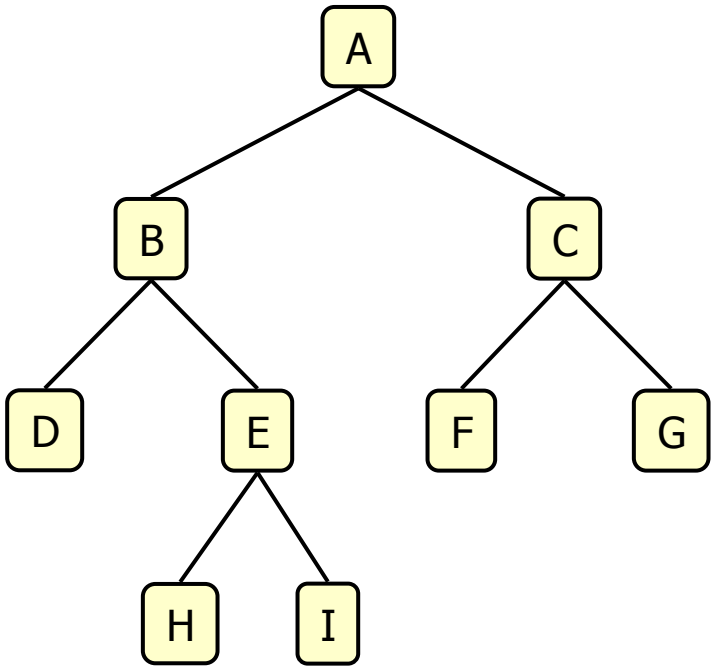
Binary trees



- Binary tree is a specific type of tree
- "In computer science, a binary tree is a tree data structure in which each node has at the most two children, which
- are referred to as the left child and the right child."
- Properties:
  - Each internal node has at most two children (degree of two)
    - Each node is called the parent of its children
  - The children of a node are an ordered pair
  - We call the children of an internal node left child and right child
  - a binary tree is either:
    - a tree consisting of a single node, OR
    -a tree whose root has an ordered pair of children, each of which is a binary tree
  - A node with no children is called a leaf.
  - Left child is always less that its parent, while right child is greater than its parent

- Applications:
  - arithmetic expressions
  - decision processes
  - searching

Tree Representing

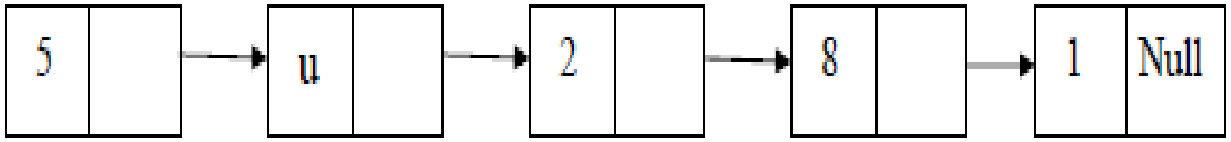
- Tree data structure combines advantages of an ordered array and a linked list.
- Searching as fast as in ordered array.
- Insertion and deletion as fast as in linked list.
- Similar to Linked List but with 2 Links
- Store the nodes at unrelated locations in memory and connect them using references in each node that point to its children.
- Can also be represented as an array, with nodes in specific positions stored in corresponding positions in the array.

Binary Search Tree (BST)

- The binary search tree is a binary tree in which the left child , if there is , of any node Contains a smaller value than does the parent node and the right child , if there is, contains a larger value than the parent node .

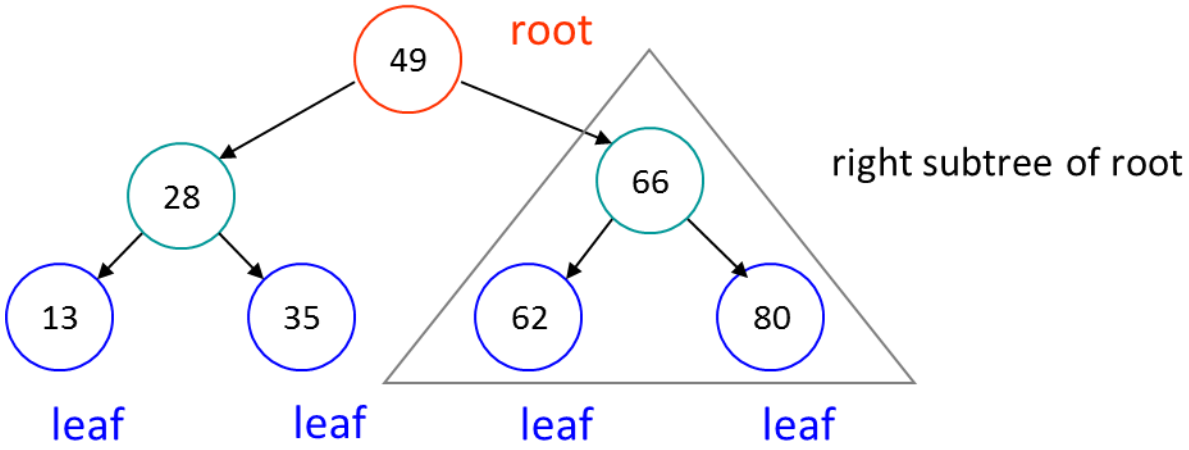
- Advantages of using Binary Search Trees:

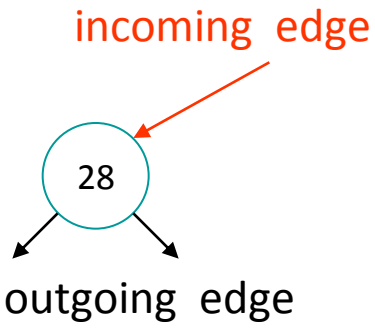
One of the drawbacks of using linked lists to store ordered information , is the time it takes to search along the list to find any value .
For example , if we have the following linked list :

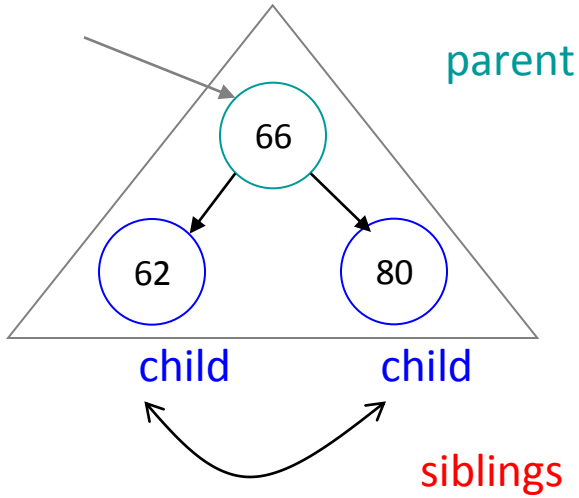


- To find the value L , we must make a sequential traversal of all the nodes in the list .
- The binary Search tree provides us with a structure that retains the flexibility of the linked list, while allowing quicker access to any node in the list .

Binary tree example:





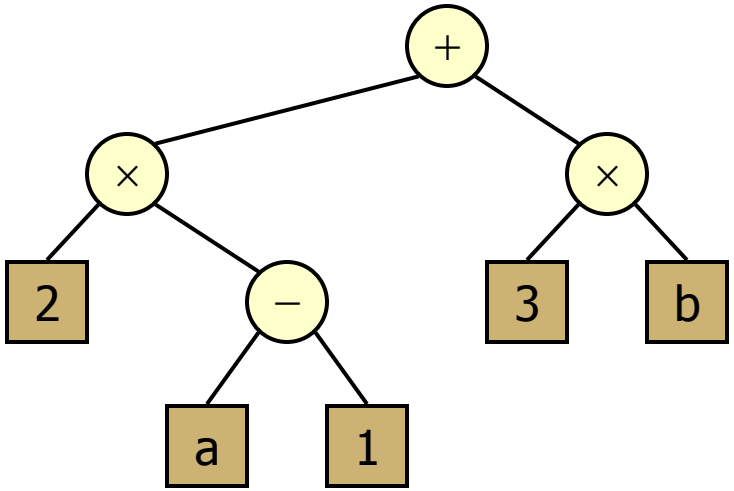


Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands

- Example:

arithmetic expression tree for the expression **(2 × (a – 1) + (3 × b))**



Implementation of Binary Search tree

There are two methods to implement binary tree Structure :

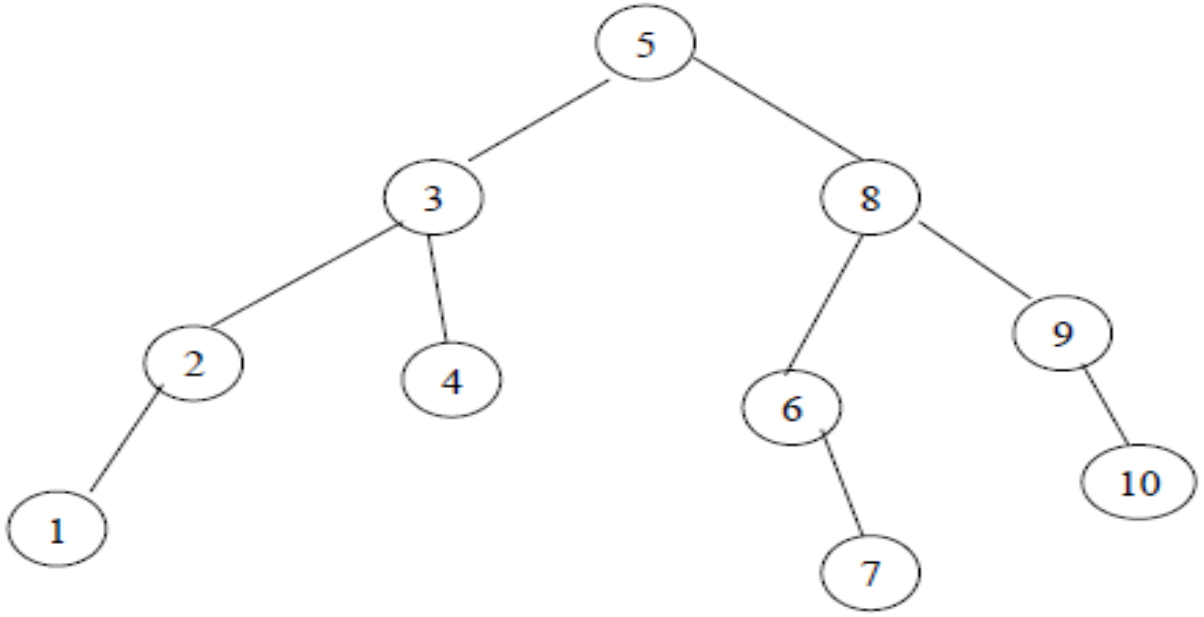
1- Linear representation : which does not require the use of the pointers.

2- Linked representation : uses pointers.

Linear Representation

- The maximum number of nodes in a binary tree of height k is $2^{k+1}-1$, k>=0.

- The linear representation method of a binary Search tree uses a one – dimensional array of size $2^{k+1}-1$, where k is the high of the tree.
- In the following tree , k =3 and this tree require an array of size = 15 to be represented



Implementation of Binary Search tree (cont.)

- **Once the size of the array has been determine , the following method is used is represent the tree :**

❖ Store the root in the 1st location of the array .

❖ If anode is in the1st location of the array , Store its left child at location (2n) and its right child at location (2n+1) .

- **The main advantages of this method are :**

❖ Its simplicity and the fact that , given a child node, its parent node can be determined immediately .

❖ If a child node is at location N in the array, then its parent node can be determined immediately . If a child is at location N in the array, then its parent node is at location N / 2, unless N = 1 .

❖ Node 1 is the root and has no parent.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	3	8	2	4	6	9	1	-	-	-	-	7	-	10

- **The disadvantages of this method:**

❖ Insertion and deletion of a node causes Considerable data monument up and down the array , using an excessive amount of processing time , because

❖ insertions and deletions are the major data processing activities .

❖ Wasted memory location.

Implementation of Binary Search tree (cont.)

- **Linked Representation of a Binary Search Tree**

❖ Because each node in a binary tree may have 2 children , a node in a linked representation has 2pointer fields , one for each child , and one or more data fields containing specific information about the node itself .

❖ when anode has no children the corresponding pointer fields are NULL.

❖ As in a linear linked list, the first node in the tree is pointed to by an external pointer.

- **Operations on binary search trees**

❖ Insert: add a new node to a binary search tree.

❖ Search: search for a node at the binary search tree.

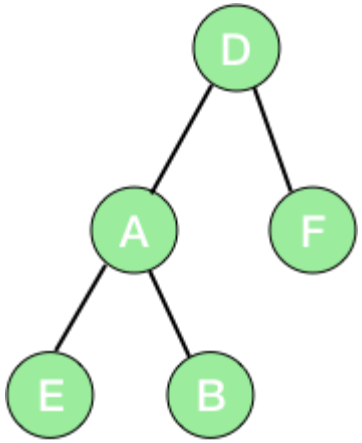
❖ Delete: delete a node from the binary search tree.

❖ Print: prints all elements in the binary search tree.

Linked Representation of a Binary Search Tree

Example: let's make a node using a structure in C++:

```
struct node
{
   char data;                    //node will store character
   struct node *right_child;     // right child
   struct node *left_child;      // left child
};
```



char data – The data which we are going to store in this node is of character type.

struct node *right_child – 'right_child' is a pointer to the node (our defined structure). Thus, we will use this to link the current node with its right child.

struct node *left_child – Similarly, we will use this to link the current node with its left child.

Now, we have a structure of our node and we need a function to create new nodes to make a tree. So, let's write it.

Linked Representation of a Binary Search Tree

```
struct node* new_node(char data)
{
   struct  node *p;                    // node
   p = new node;                       // allocating space to p
   p->data = data;                     // assinging data to p
   p->left_child = NULL;               // making children NULL
   p->right_child = NULL;

   return(p);                          // returning the newly made node
}
```

struct node *p – '*p' is a node and 'p' is a pointer to the node. It means that p will store the address of the node. Please note that we have just declared a node here and no memory has been yet assigned to store data. Thus, the next task is to allocate the memory to the node and we will do it using the new function.

p = new node: We are allocating a space in the memory to the node and storing the address of the memory in the variable 'p'.
p->data = data : Now, we have our node and we also have a space in memory allocated to it. Here, we are just storing data in it.

p->left_child = NULL : We are making the left child of the node null.

p->right_child = NULL : Similarly, we also made the right child null.

return(p) : We are just returning the address of the node we created.

Traversals in a Binary Tree

- Now, we have made our node and a function to add new nodes. Thus, the next task is to make the tree described in the above picture and implement inorder, postorder and preorder traversals to it.

- So, let's first make the tree in the main function.

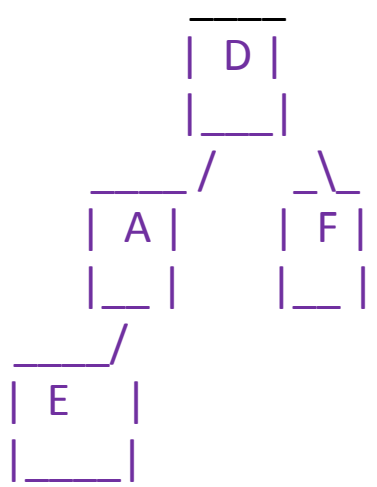
```
main()
{
   struct node *root;                          //new structure
   root = new_node('D');                       // making a new node

     ___
   | D  |
   |____|
   root->left_child = new_node('A');     //left child of root

        ____
       | D |
       |___|
     ____/
   | A |
   |__ |
root->right_child = new_node('F');        //right child of root

        ____
       | D |
       |___|
     ____/     \___
   | A  |        F   |
   |____|      |____|

 root->left_child->left_child = new_node('E');     // new node
```



root->left_child->right_child = new_node('B'); // new node