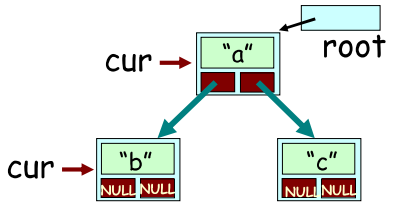
Lecture #12

- Binary Tree Traversals
- Evaluate Expressions Using Binary Trees
- Binary Search Trees
  - Searching for an item
  - Inserting a new item
  - Finding the minimum and maximum items
  - Printing out the items in order
  - Deleting the whole tree

The In-order Traversal

1. Process the nodes in the left sub-tree.
2. Process the current node.
3. Process the nodes in the right sub-tree.



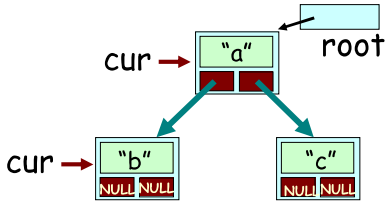

```
void InOrder(Node *cur)
{
   if (cur == NULL)           // if empty, return...
      return;
   InOrder(cur->left);     // Process nodes in left sub-tree.
   cout << cur->value;     // Process the current node.
   InOrder(cur->right);  // Process nodes in right sub-tree.
}
```

Output:

b

The Post-order Traversal

1. Process the nodes in the left sub-tree.
2. Process the nodes in the right sub-tree.
3. Process the current node.




```
void PostOrder(Node *cur)
{
   if (cur == NULL)           // if empty, return...
      return;
   PostOrder(cur->left);     // Process nodes in left sub-tree.
   PostOrder(cur-> right);  // Process nodes in right sub-tree.
   cout << cur->value;      // Process the current node.
}
```

Output:

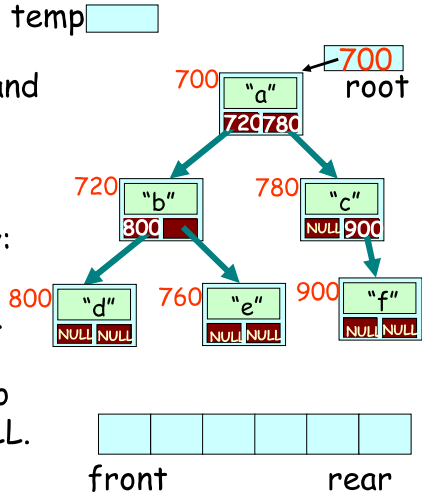
b

The Level Order Traversal

In a *level order traversal* we visit each level's nodes, from left to right, before visiting nodes in the next level.

Here's the algorithm:

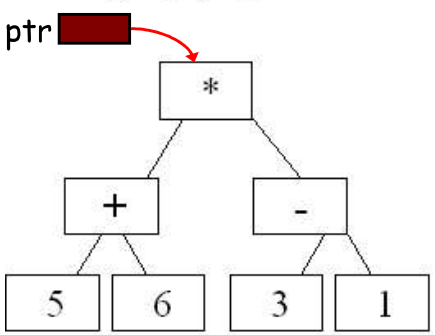
1. Use a temp pointer variable and a queue of node pointers.
2. Insert the root node pointer into the queue.
3. While the queue is not empty:
   A. Dequeue the top node pointer and put it in temp.
   B. Process the node.
   C. Add the node's children to queue if they are not NULL.



abcd Etc...

Expression Evaluation

We can represent arithmetic expressions using a binary tree.



For example, the tree on the left represents the expression: (5+6)*(3-1)

Once you have an expression in a tree, its easy to evaluate it and get the result.

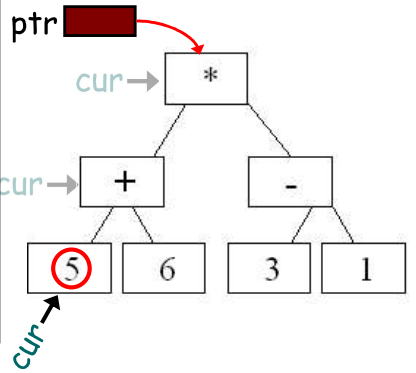
Let's see how!

Expression Evaluation

Here's our evaluation function. We start by passing in a pointer to the root of the tree.

1. If the current node is a number, return its value.
2. Recursively evaluate the left subtree and get the result.
3. Recursively evaluate the right subtree and get the result.
4. Apply the operator in the current node to the left and right results; return the result.

(5+6)*(3-1)

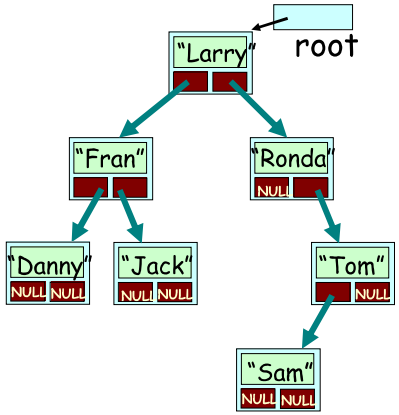


Binary Search Trees

Binary Search Trees are a type of binary tree with specific properties that make them very efficient to search for a value in the tree.

Like regular Binary Trees, we store and search for values in Binary Search Trees…

Here's an example BST…



Operations on a Binary Search Tree

Here's what we can do to a BST:

- Determine if the binary search tree is empty
- Search the binary search tree for a value
- Insert an item in the binary search tree
- Delete an item from the binary search tree
- Find the height of the binary search tree
- Find the number of nodes and leaves in the binary search tree
- Traverse the binary search tree
- Free the memory used by the binary search tree

Searching a BST

Here are two different BST search algorithms in C++, one recursive and one iterative:

```
bool Search(int V, Node *ptr)
{
  if (ptr == NULL)
    return(false);  // nope
  else if (V == ptr->value)
    return(true);  // found!!!
  else if (V < ptr->value)
    return(Search(V,ptr->left));
  else
    return(Search(V,ptr->right));
}
```

```
bool Search(int V,Node *ptr)
{
  while (ptr != NULL)
  {
    if (V == ptr->value)
      return(true);
    else if (V < ptr->value)
      ptr = ptr->left;
    else
      ptr = ptr->right;
  }
  return(false); // nope
}
```

Let's trace through the recursive version…

Inserting A New Value Into A BST

Input: A value V to insert

If the tree is empty
- Allocate a new node and put V into it
- Point the root pointer to our new node. DONE!

Start at the root of the tree

While we're not done…
- If V is equal to current node's value, DONE! (nothing to do…)
- If V is less than current node's value
  - If there is a left child, then go left
  - ELSE allocate a new node and put V into it, and set current node's left pointer to new node. DONE!
- If V is greater than current node's value
  - If there is a right child, then go right
  - ELSE allocate a new node and put V into it, set current node's right pointer to new node. DONE!

Finding Min & Max of a BST

How do we find the minimum and maximum values in a BST?

```
int GetMin(node *pRoot)
{
  if (pRoot == NULL)
    return(-1);  // empty

  while (pRoot->left != NULL)
     pRoot = pRoot->left;

  return(pRoot->value);
}
```

```
int GetMax(node *pRoot)
{
  if (pRoot == NULL)
    return(-1);  // empty

  while (pRoot->right != NULL)
     pRoot = pRoot->right;

  return(pRoot->value);
}
```

Question: What's the big-oh to find the minimum or maximum element?

Finding Min & Max of a BST

And here are recursive versions for you…

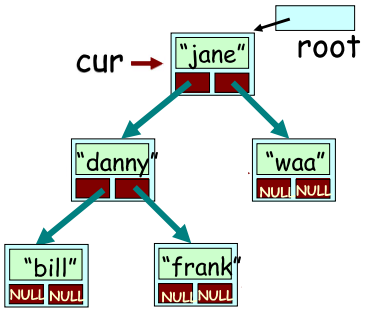
```
int GetMin(node *pRoot)
{
  if (pRoot == NULL)
    return(-1);  // empty

  if (pRoot->left == NULL)
    return(pRoot->value);

  return(GetMin(pRoot->left));
}
```

```
int GetMax(node *pRoot)
{
  if (pRoot == NULL)
    return(-1);  // empty

  if (pRoot->right == NULL)
    return(pRoot->value);

  return(GetMax(pRoot->right));
}
```

Hopefully you're getting the idea that most tree functions can be done recursively…

Printing a BST In Alphabetical Order

Can anyone guess what algorithm we use to print out a BST in alphabetical order?



Output:

bill
danny
frank
jane
waa

Big-oh Alert!

So what's the big-Oh of printing all the items in the tree?

Right! O(n) since we have to visit and print all n items.

Freeing The Whole Tree

When we are done with our BST, we have to free every node in the tree, one at a time.

Question: Can anyone think of an algorithm for this?

```
void FreeTree(Node *cur)
{
   if (cur == NULL)          // if empty, return…
      return;

   FreeTree(cur->left);     // Delete nodes in left sub-tree.

   FreeTree (cur->right);  // Delete nodes in right sub-tree.

  delete cur;               // Free the current node
}
```