BinaryTreeDisplay

*Recursive Display*



1. *LDR - Left\_Root\_Right*

*void LDR ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*LDR ( root -> left );*

*print ( root -> value );*

*LDR ( root -> right );*

*return;*

*}*

1. *DLR - Root\_Left\_Right*

*void DLR ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*print( root -> value );*

*DLR ( root -> left );*

*DLR ( root -> right );*

*return;*

*}*

1. *LRD - Left\_Right\_Root*

*void LRD ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*LRD ( root -> left );*

*LRD ( root -> right );*

*print ( root -> value );*

*return;*

*}*

*Non - Recursive Display - Stack*



1. *LRD - Left\_Right\_Root*

*Sequence of Btree - 4, 5, 2, 6, 7, 3, 1*

*void LRD ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*stack < BinaryTreeNode \* > s;*

*s.push\_back ( root );*

*stack< int > array;*

*BinaryTreeNode \* node = s.top ( );*

*while ( ! s.empty ( ) )*

*{*

*node = s.top ( );*

*array.push\_back ( node -> value );*

*s.pop ( );*

*if ( node -> left )*

*s.push\_back ( node -> left );*

*if ( node -> right )*

*s.push\_back ( node -> right );*

*}*

*}*

1. *DLR - Root\_Left\_Right*

*Sequence of Btree - 1, 2, 4, 5, 3, 6, 7*

*void DLR ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*stack < BinaryTreeNode \* > s;*

*s.push\_back ( root );*

*vector < int > v;*

*BinaryTreeNode \* node = s.top ( );*

*while ( ! s.empty ( ) )*

*{*

*node = s.top ( );*

*array.push\_back ( node -> value );*

*s.pop ( );*

*if ( node -> right )*

*s.push\_back ( node -> right );*

*if ( node -> left )*

*s.push\_back ( node -> left );*

*}*

*}*

1. *LDR - Left\_Root\_Right*

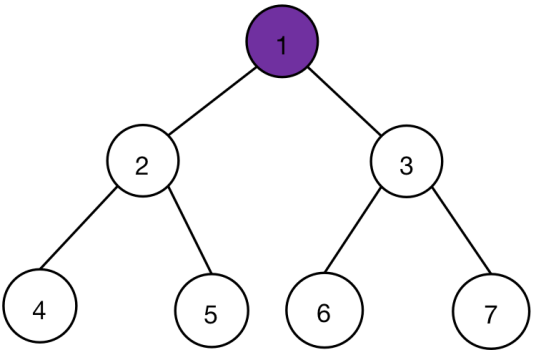
*Sequence of Btree - 4, 2, 5, 1, 6, 3, 7*

*Procedure:*



*Round 1 - Access Root 1 in the Tree, and push Root Node into stack.*

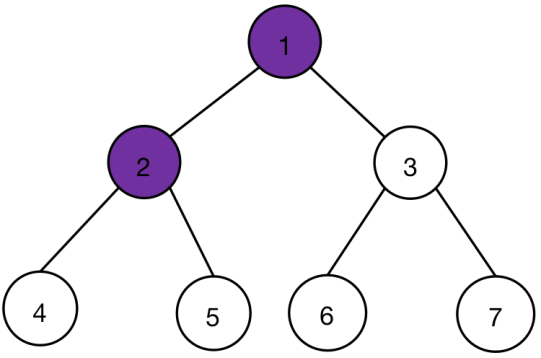
|  |
| --- |
|  |
|  |
|  |
|  |
|  |
| *1* |

**

*Output: Empty*

*Round 2 - Check whether Internal Node 2 of Root 1 equals to NULL, and push the Internal Node 2 into stack if not NULL.*

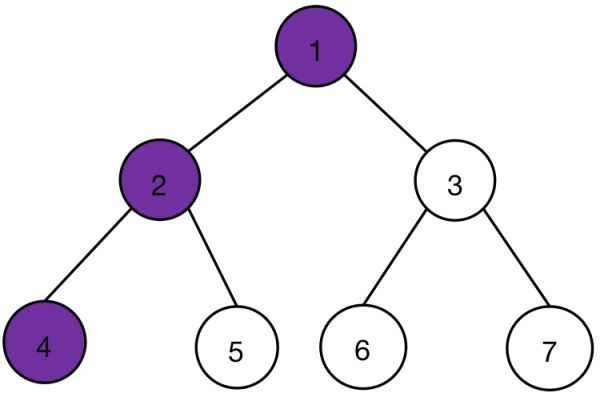
|  |
| --- |
|  |
|  |
|  |
|  |
| *2* |
| *1* |

**

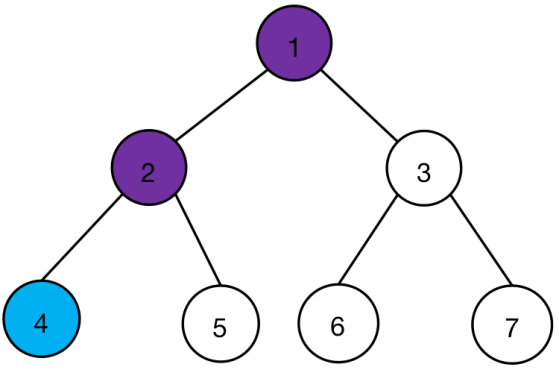
*Output: Empty*

*Round 3 - Check whether Leaf Node 4 of Internal Node 2 equals to NULL, and push the Internal Node 4 into stack of it does not equal to NULL.*

|  |
| --- |
|  |
|  |
|  |
|  |
| *4* |
| *2* |
| *1* |

**

*Output: Empty*

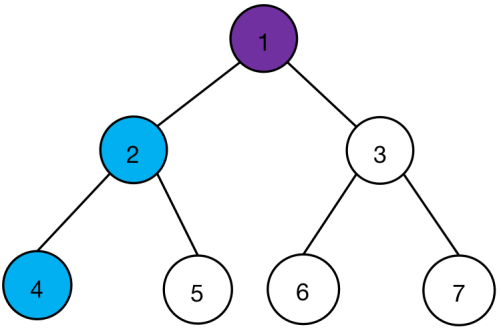
*Round 4 - Leaf Node 4 has no Left Node, but the stack still has element. Print the current Node Value in the top of stack, and pop out the Leaf Node 4. Assign NULL value to the Node.*

|  |
| --- |
|  |
|  |
|  |
|  |
| *2* |
| *1* |

*Output: 4.*

*Round 5 - Visit Top Node Element 2 of stack, and print the Element Value of Node 2. Pop out the Internal Node 2. The next step is to visit the Right Node 5 of the Internal Node 2.*

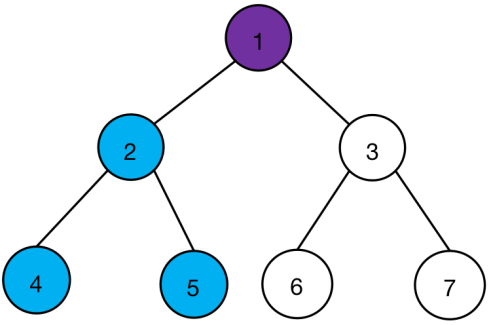
|  |
| --- |
|  |
|  |
|  |
|  |
| *1* |

**

*Output: 4 -> 2.*

*Round 6 - Visit the Right Node 5 of the Last Node 2. Push the Right Node 5 into the stack. Print out the value of 5. The next step is to visit the Left Node of Node 5.*

|  |
| --- |
|  |
|  |
|  |
| *5* |
| *1* |

**

*Output: 4 -> 2 -> 5.*

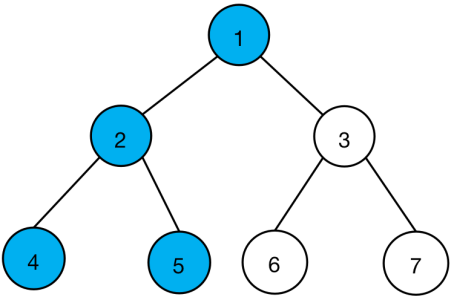
*Round 7 - The Node 5 is the Leaf Node, and Left Node and Right Node of Node 5 are all NULL. Skip visiting this Node. Go check Bottom Node of value 1.*

|  |
| --- |
|  |
|  |
|  |
|  |
| *1* |

*Output: 4 -> 2 -> 5.*

*Round 8 - Pop out Root Node with value 1, and print out value 1. The next step is to visit Right Node of Node 1.*

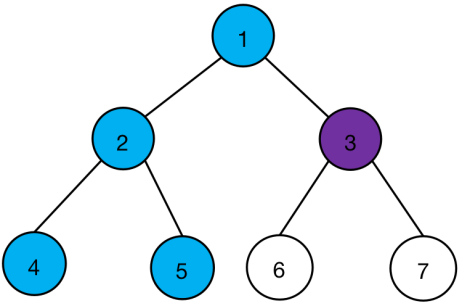
|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**

*Output: 4 -> 2 -> 5 -> 1.*

*Round 7 - Visit the Right Node 3 of Node 1, and push the Node 3 into the stack. The next step is to visit the Left Node of Node 3.*

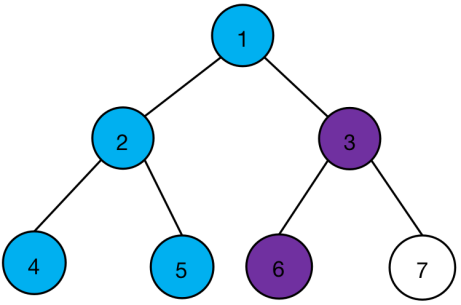
|  |
| --- |
|  |
|  |
|  |
|  |
| *3* |

**

*Output: 4 -> 2 -> 5 -> 1.*

*Round 8 - Push the Left Node 6 of Node 3 into the stack.*

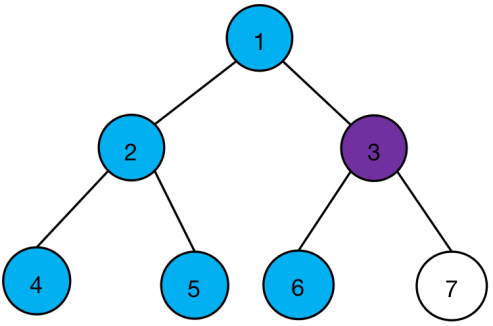
|  |
| --- |
|  |
|  |
|  |
| *6* |
| *3* |

**

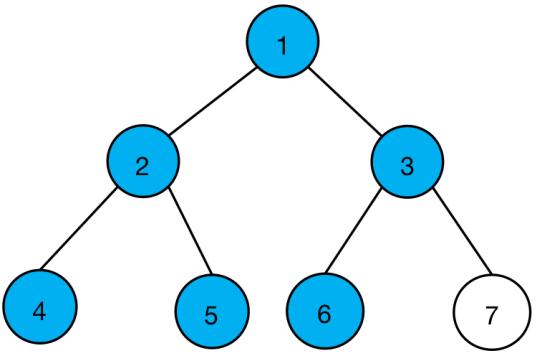
*Output: 4 -> 2 -> 5 -> 1.*

*Round 9 - Pop out Left Node 6, and print out the value 6. The next step is to visit the Right Node of Node 6. Attention that Right Node 6 has no Left Node.*

|  |
| --- |
|  |
|  |
|  |
|  |
| *3* |

**

*Output: 4 -> 2 -> 5 -> 1 -> 6.*

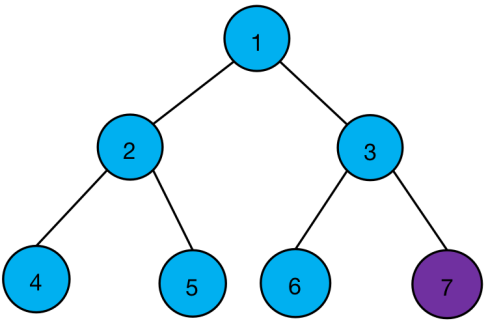
*Round 9 - Pop out Top Node 3, and print out value 3. The next step is to visit the Right Node 7 of Node 3.*

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

*Output: 4 -> 2 -> 5 -> 1 -> 6 -> 3.*

*Round 10 - Visit the Right Node 7 of Node 3. Push Node 7 into stack. The next step is to visit the Left Node of Node 7. Attention that there has no Left Node of Node 7.*

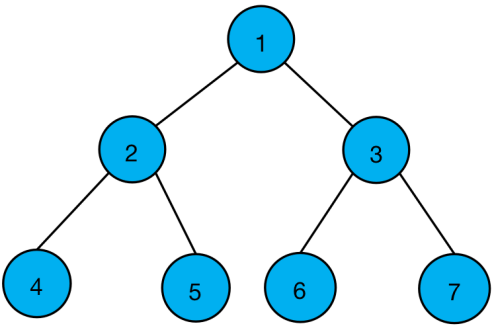
|  |
| --- |
|  |
|  |
|  |
|  |
| *7* |

**

*Output: 4 -> 2 -> 5 -> 1 -> 6 -> 3.*

*Round 11 - The Right Node 7 has no Left Node and Right Node, so visit the Node 7, print the Node Value 7. Pop the Right Node 7 out of stack. After that, the stack turns to Empty.*

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**

*Output: 4 -> 2 -> 5 -> 1 -> 6 -> 3 -> 7.*

*void LRD ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*stack < BinaryTreeNode \* > s;*

*BinaryTreeNode \* node = root;*

*while ( node | | ! s.empty ( ) )*

*{*

*if ( node )*

*{*

*s.push\_back ( node );*

*node = node -> left;*

*}*

*else*

*{*

*node = s.top( );*

*print ( node -> value );*

*s.pop( );*

*node = node -> right;*

*}*

*}*

*}*

1. *Level Display*

*void BtreeLevelDisplay ( BinaryTreeNode \* root )*

*{*

*if ( root == NULL )*

*return;*

*stack < BinaryTreeNode \* > s;*

*s.push\_back ( root );*

*while ( s.top( ) )*

*{*

*BinaryTreeNode \* node = s.top( );*

*print( s.top( ) -> value( ) );*

*if ( s.top() -> left )*

*s.push\_back( s.top() ->left );*

*if ( s.top() -> right )*

*s.push\_back( s.top() -> right );*

*s.pop( );*

*}*

*}*