Lecture No.14

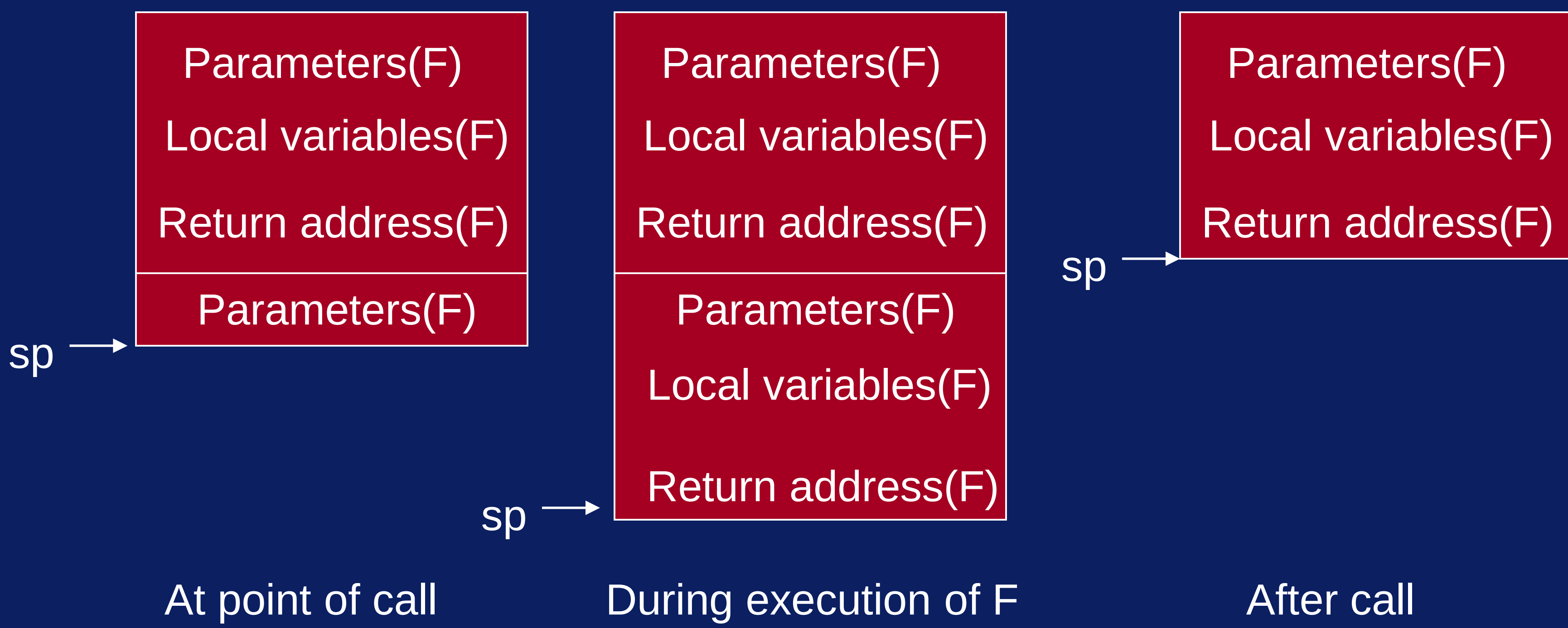
Data Structures

Recursive Call

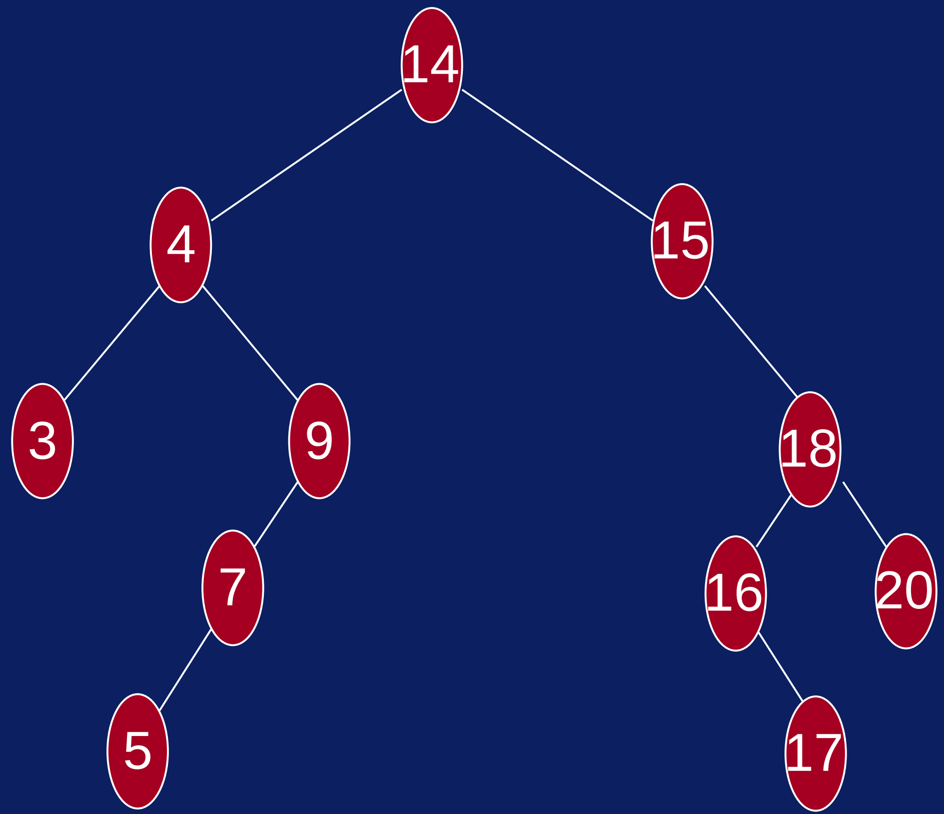
- Recall that a stack is used during function calls.
- The caller function places the arguments on the stack and passes control to the called function.
- Local variables are allocated storage on the call stack.
- Calling a function itself makes no difference as far as the call stack is concerned.

Stack Layout during a call

- Here is stack layout when function F calls function F (recursively):

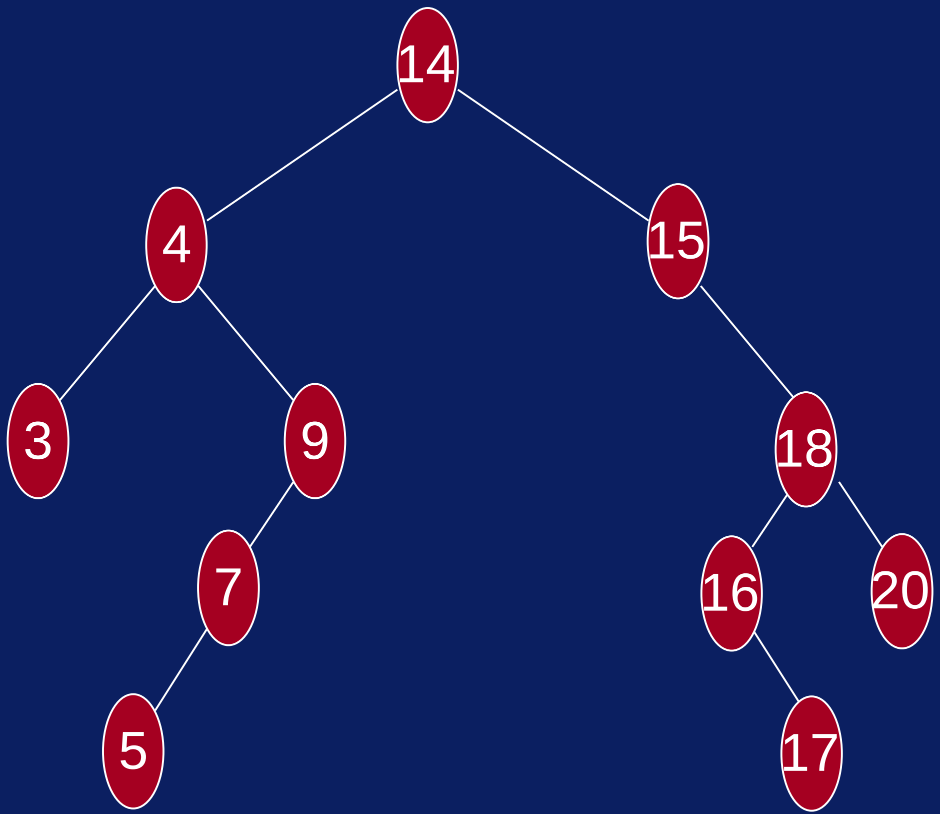


Recursion: preorder



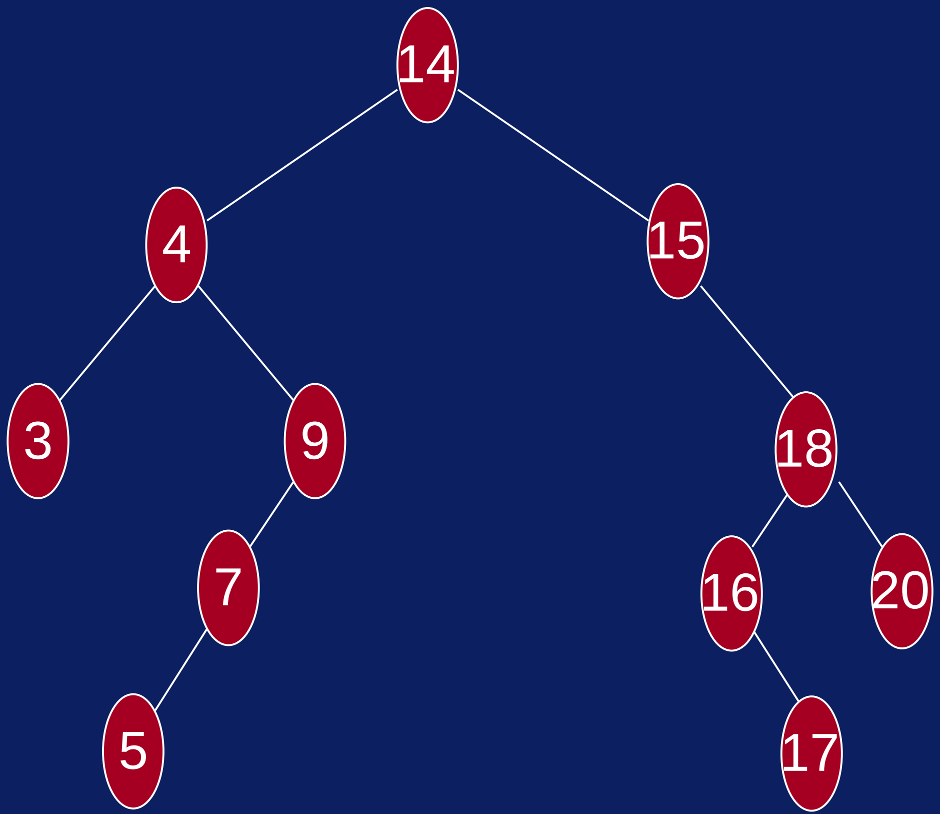

```
preorder(14)
14
..preorder(4)
4
....preorder(3)
3
......preorder(null)
......preorder(null)
....preorder(9)
9
......preorder(7)
7
........preorder(5)
5
..........preorder(null)
..........preorder(null)
........preorder(null)
......preorder(null)
```

Recursion: preorder



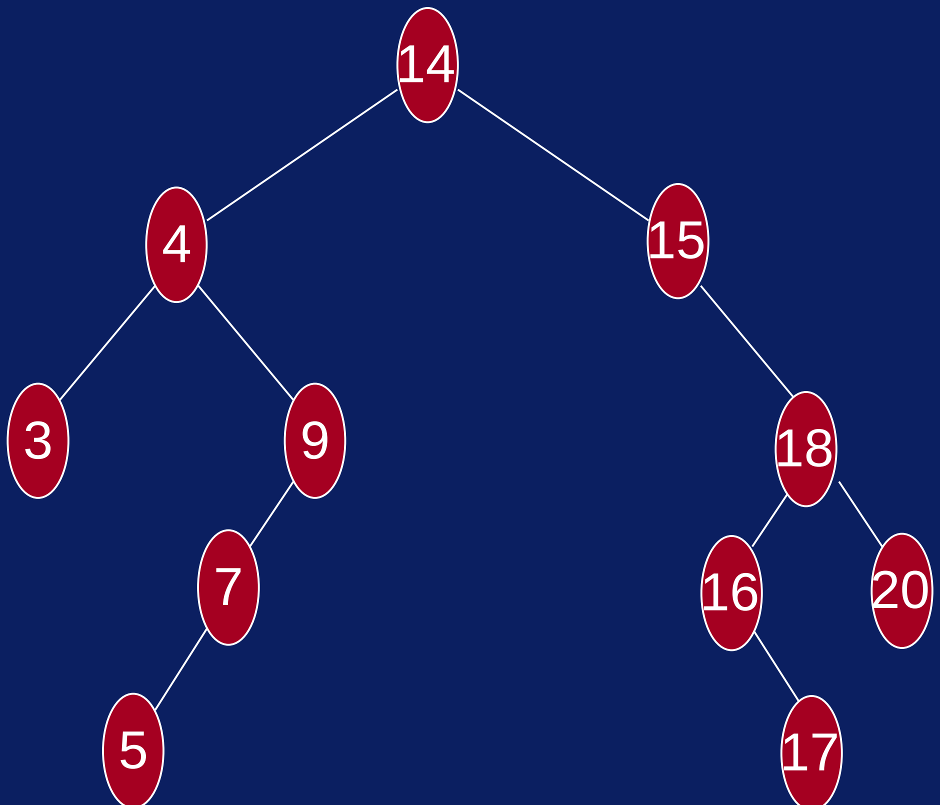

```
..preorder(15)
15
....preorder(null)
....preorder(18)
18
......preorder(16)
16
........preorder(null)
........preorder(17)
17
..........preorder(null)
..........preorder(null)
......preorder(20)
20
........preorder(null)
........preorder(null)
```

Recursion: inorder




```
inorder(14)
..inorder(4)
....inorder(3)
......inorder(null)
3
......inorder(null)
4
....inorder(9)
......inorder(7)
........inorder(5)
..........inorder(null)
5
..........inorder(null)
7
........inorder(null)
9
......inorder(null)
14
```

Recursion: inorder




```
..inorder(15)
....inorder(null)
15
....inorder(18)
......inorder(16)
........inorder(null)
16
........inorder(17)
..........inorder(null)
17
..........inorder(null)
18
......inorder(20)
........inorder(null)
20
........inorder(null)
```

Non Recursive Traversal

- We can implement non-recursive versions of the preorder, inorder and postorder traversal by using an explicit stack.
- The stack will be used to store the tree nodes in the appropriate order.
- Here, for example, is the routine for inorder traversal that uses a stack.

Non Recursive Traversal

```
void inorder(TreeNode<int>* root)
{
    Stack<TreeNode<int>* > stack;
    TreeNode<int>* p;
    p = root;
    do
    {
        while( p != NULL )
        {
            stack.push( p );
            p = p->getLeft();
        }
        // at this point, left tree is empty
```

Non Recursive Traversal

```
void inorder(TreeNode<int>* root)
{
    Stack<TreeNode<int>* > stack;
    TreeNode<int>* p;
    p = root;
    do
    {
        while( p != NULL )
        {
            stack.push( p );
            p = p->getLeft();
        }
        // at this point, left tree is empty
```

Non Recursive Traversal

```
void inorder(TreeNode<int>* root)
{
    Stack<TreeNode<int>* > stack;
    TreeNode<int>* p;
    p = root;
    do
    {
        while( p != NULL )
        {
            stack.push( p );
            p = p->getLeft();
        }
        // at this point, left tree is empty
```

Non Recursive Traversal

```
        if( !stack.empty() )
        {
            p = stack.pop();
            cout << *(p->getInfo()) << " ";
            // go back & traverse right subtree
            p = p->getRight();
        }
    } while ( !stack.empty() || p != NULL );
}
```

Non Recursive Traversal

```
      if( !stack.empty() )
      {
          p = stack.pop();
          cout << *(p->getInfo()) << " ";
          // go back & traverse right subtree
          p = p->getRight();
      }
   } while ( !stack.empty() || p != NULL );
}
```

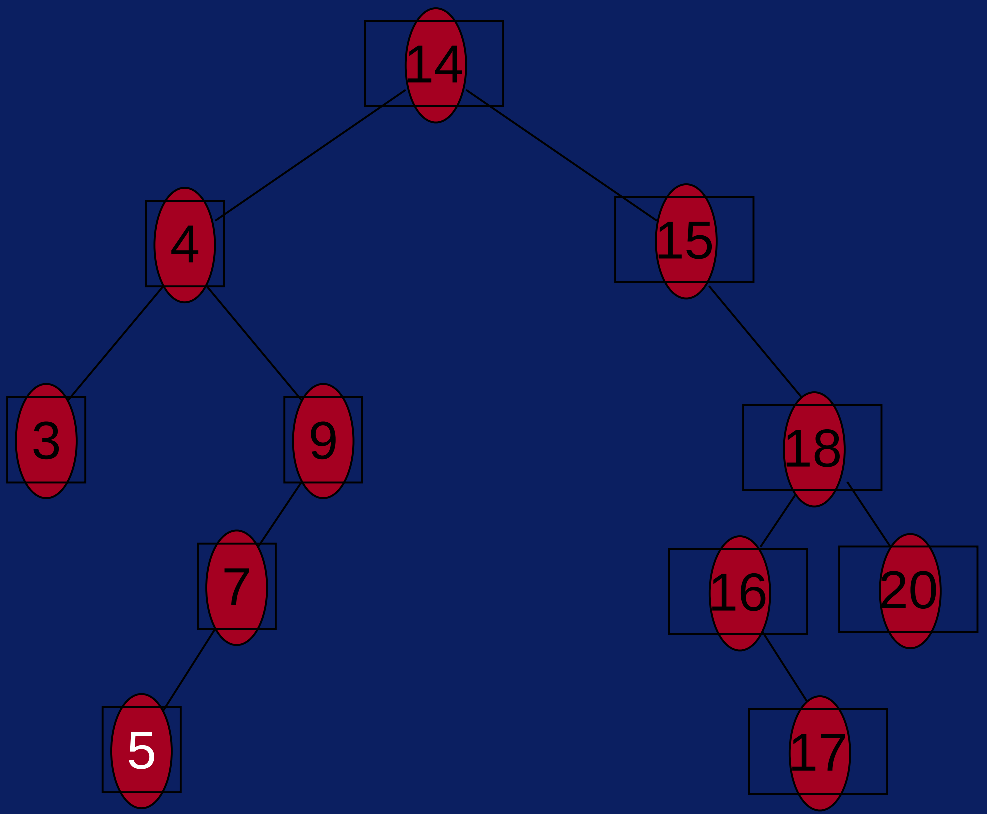
Non Recursive Traversal

```
    if( !stack.empty() )
    {
        p = stack.pop();
        cout << *(p->getInfo()) << " ";
        // go back & traverse right subtree
        p = p->getRight();
    }
  } while ( !stack.empty() || p != NULL );
}
```

Non Recursive Traversal

```
      if( !stack.empty() )
      {
          p = stack.pop();
          cout << *(p->getInfo()) << " ";
          // go back & traverse right subtree
          p = p->getRight();
      }
  } while ( !stack.empty() || p != NULL );
}
```

Nonrecursive Inorder




```
push(14)
..push(4)
....push(3)
3
4
..push(9)
....push(7)
......push(5)
5
7
9
14
push(15)
15
push(18)
..push(16)
16
..push(17)
17
18
push(20)
20
```

Traversal Trace

recursive inorder	*nonrecursive inorder*
inorder(14)	**push(14)**
..**inorder(4)**	..**push(4)**
....**inorder(3)**	....**push(3)**
3	3
4	4
..**inorder(9)**	..**push(9)**
....**inorder(7)**	....**push(7)**
......**inorder(5)**	......**push(5)**
5	5
7	7
9	9
14	14
inorder(15)	**push(15)**
15	15
inorder(18)	**push(18)**
..**inorder(16)**	..**push(16)**
16	16
..**inorder(17)**	..**push(17)**
17	17
18	18
inorder(20)	**push(20)**
20	20

Traversal Trace

recursive inorder

```
inorder(14)
..inorder(4)
....inorder(3)
3
4
..inorder(9)
....inorder(7)
......inorder(5)
5
7
9
14
inorder(15)
15
inorder(18)
..inorder(16)
16
..inorder(17)
17
18
inorder(20)
20
```

nonrecursive inorder

```
push(14)
..push(4)
....push(3)
3
4
..push(9)
....push(7)
......push(5)
5
7
9
14
push(15)
15
push(18)
..push(16)
16
..push(17)
17
18
push(20)
20
```

Traversal Trace

recursive inorder	*nonrecursive inorder*
`inorder(14)`	`push(14)`
`..inorder(4)`	`..push(4)`
`....inorder(3)`	`....push(3)`
`3`	`3`
`4`	`4`
`..inorder(9)`	`..push(9)`
`....inorder(7)`	`....push(7)`
`......inorder(5)`	`......push(5)`
`5`	`5`
`7`	`7`
`9`	`9`
`14`	`14`
`inorder(15)`	`push(15)`
`15`	`15`
`inorder(18)`	`push(18)`
`..inorder(16)`	`..push(16)`
`16`	`16`
`..inorder(17)`	`..push(17)`
`17`	`17`
`18`	`18`
`inorder(20)`	`push(20)`
`20`	`20`

Traversal Trace

recursive inorder

```
inorder(14)
..inorder(4)
....inorder(3)
3
4
..inorder(9)
....inorder(7)
......inorder(5)
5
7
9
14
inorder(15)
15
inorder(18)
..inorder(16)
16
..inorder(17)
17
18
inorder(20)
20
```

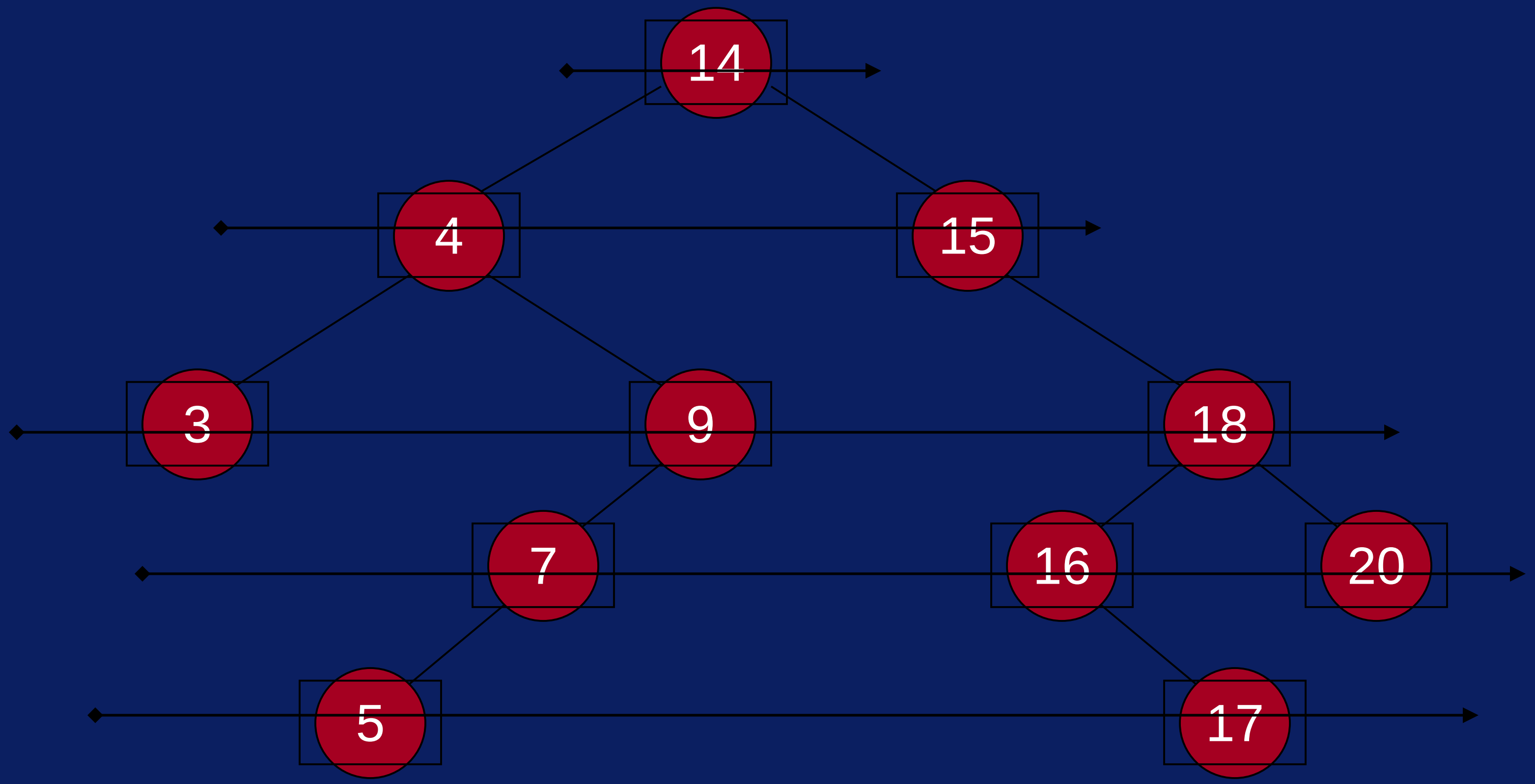
nonrecursive inorder

```
push(14)
..push(4)
....push(3)
3
4
..push(9)
....push(7)
......push(5)
5
7
9
14
push(15)
15
push(18)
..push(16)
16
..push(17)
17
18
push(20)
20
```

Level-order Traversal

- There is yet another way of traversing a binary tree that is not related to recursive traversal procedures discussed previously.
- In level-order traversal, we visit the nodes at each level before proceeding to the next level.
- At each level, we visit the nodes in a left-to-right order.

Level-order Traversal



Level-order: 14 4 15 3 9 18 7 16 20 5 17