Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Data Structures and Algorithms – C++ Implementation

Huỳnh Tấn Đạt

Email: htdat@cse.hcmut.edu.vn

Home Page: http://www.cse.hcmut.edu.vn/~htdat/

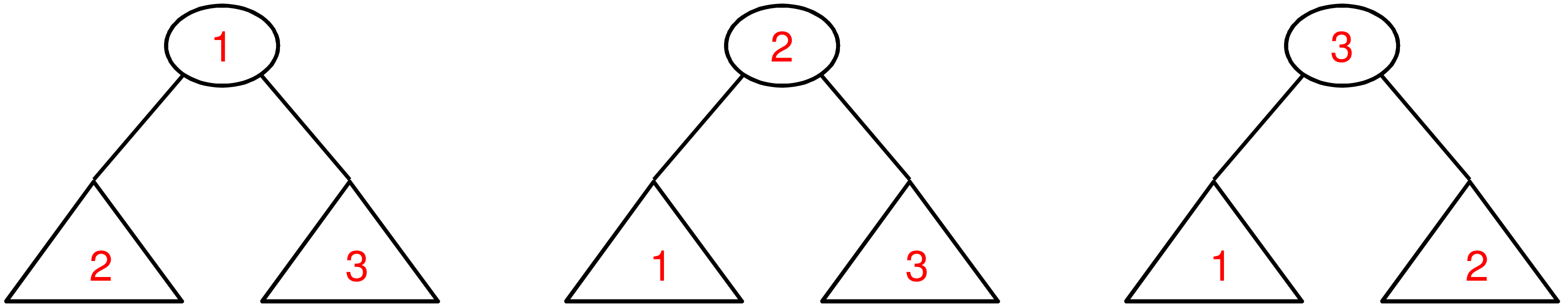
Binary Tree Structure

```
Node
    data <dataType>
    leftSubTree <nodePointer>
    rightSubTree <nodePointer>
End Node
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
class BinaryNode {
public:
    EntryDataType data;
    BinaryNode* left;
    BinaryNode* right;
    BinaryNode(EntryDataType &x){
        data = x;
        left = right = NULL;
    }
};
```

Depth-First Traversal

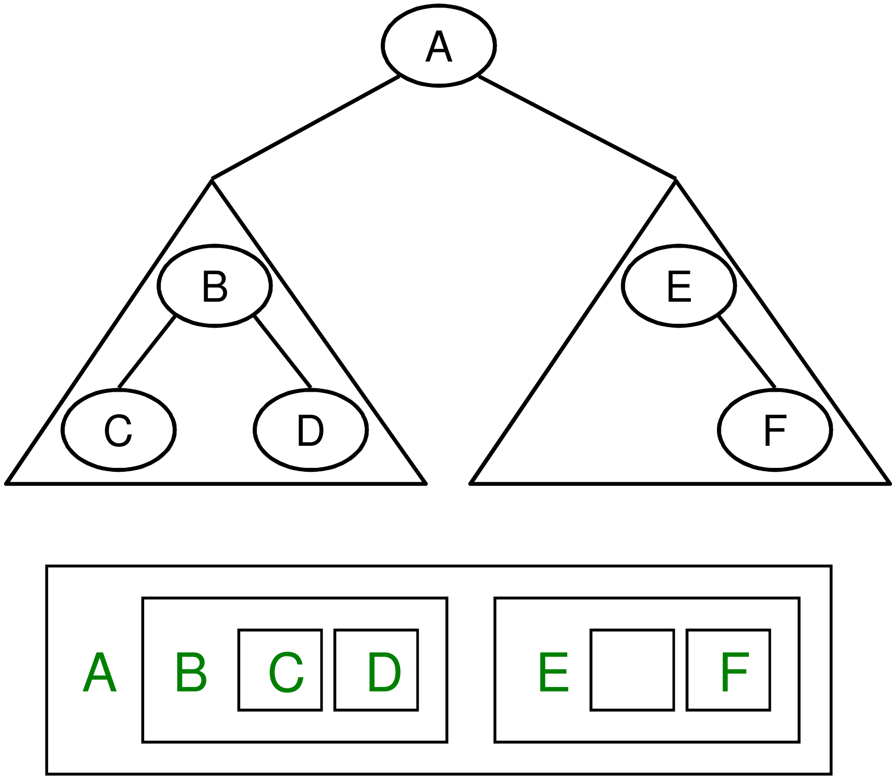


PreOrder
NLR

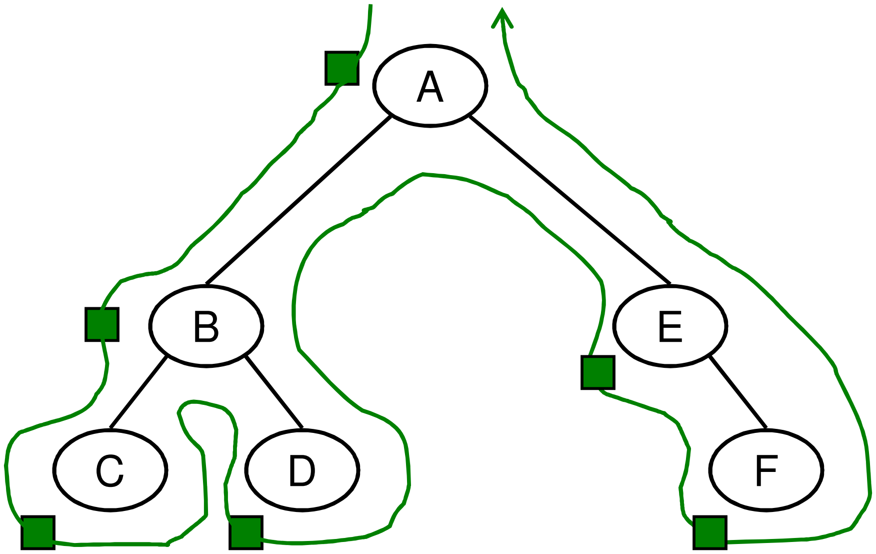
InOrder
LNR

PostOrder
LRN

PreOrder Traversal



Processing order

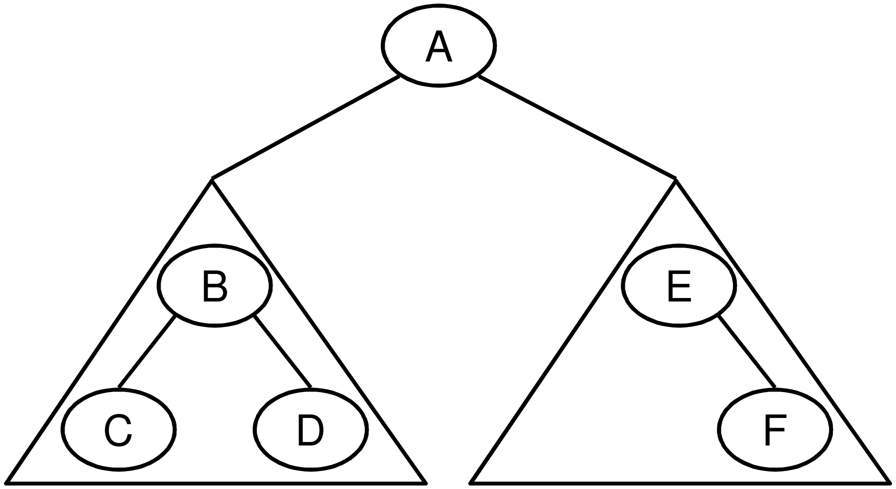


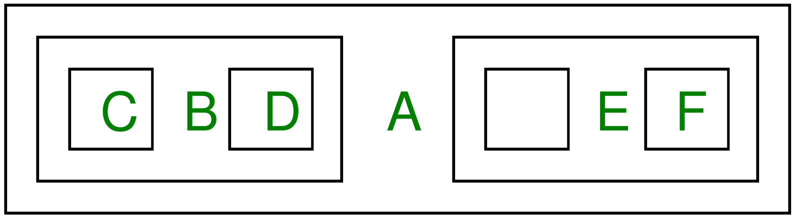
Walking order

PreOrder Traversal

```
Algorithm   preOrder (val root <nodePointer>)
Traverses a binary tree in node-left-right sequence
   Pre   root is the entry node of a tree/subtree
   Post        each node has been processed in order
   1  if (root is not null)
      1     process (root)
      2     preOrder (root –> leftSubTree)
      3     preOrder (root –> rightSubTree)
   4  return
End  preOrder
```

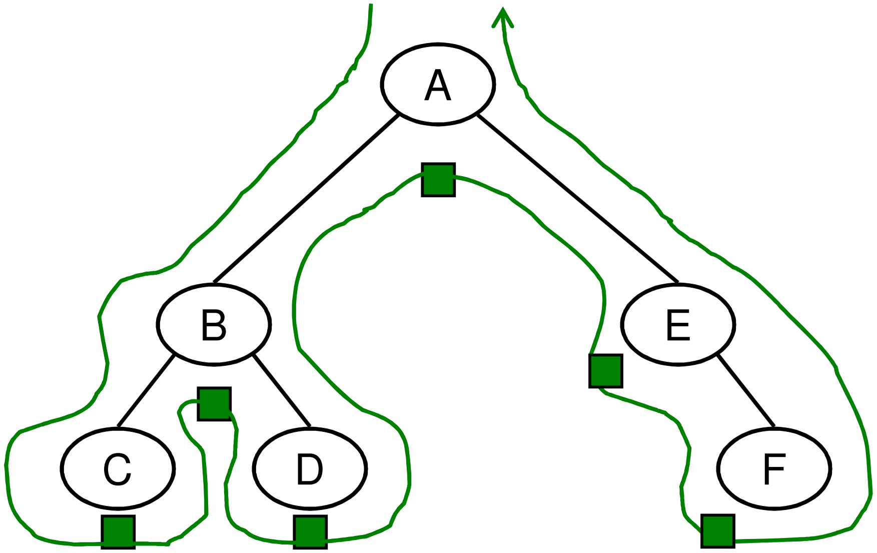
InOrder Traversal





C B D A E F

Processing order

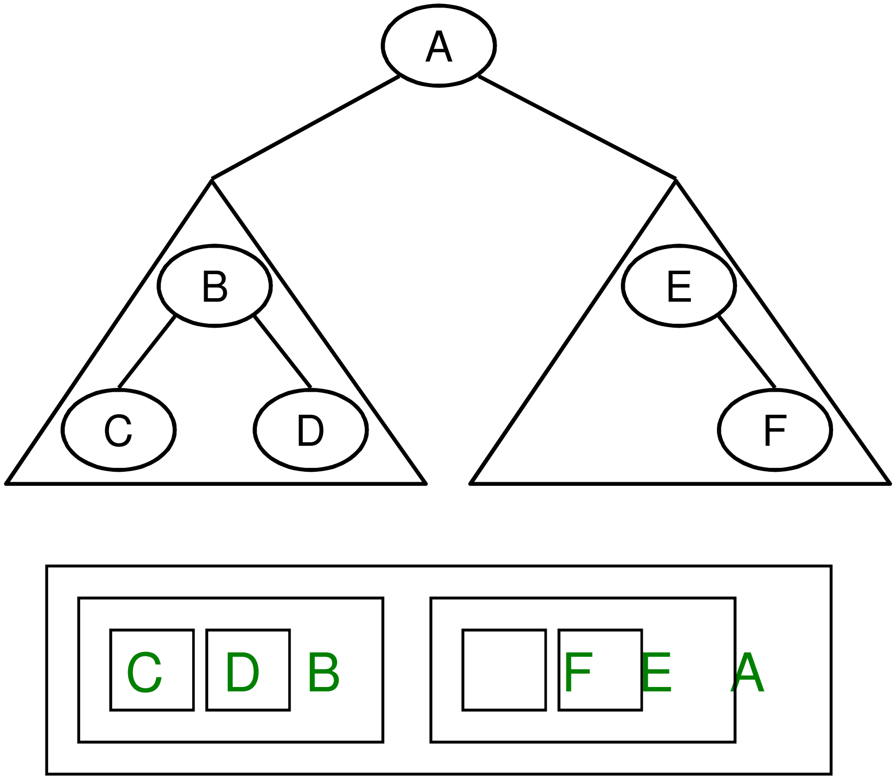


Walking order

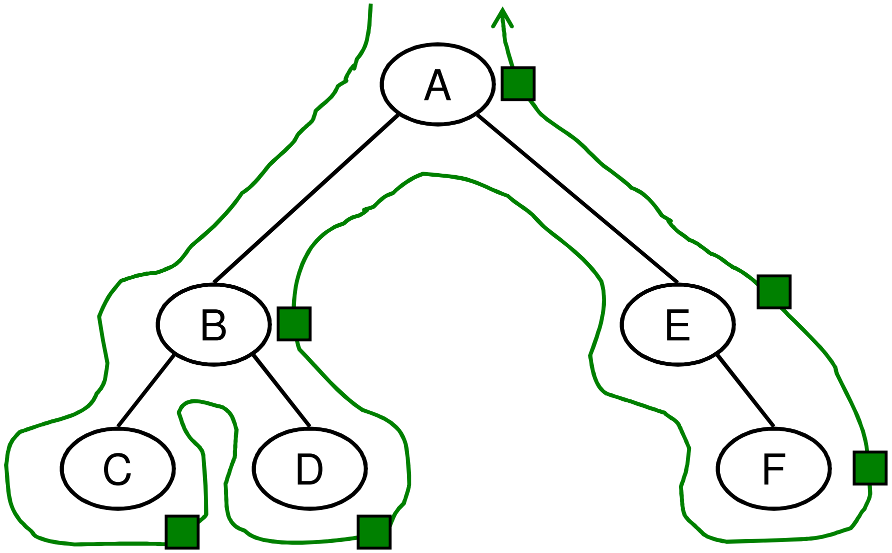
InOrder Traversal

```
Algorithm   inOrder (val root <nodePointer>)
Traverses a binary tree in left-node-right sequence
   Pre  root is the entry node of a tree/subtree
   Post       each node has been processed in order
   1   if (root is not null)
      1      inOrder (root –> leftSubTree)
      2      process (root)
      3      inOrder (root –> rightSubTree)
   4  return
End  inOrder
```

PostOrder Traversal



Processing order



Walking order

PostOrder Traversal

Algorithm postOrder (val root <nodePointer>)

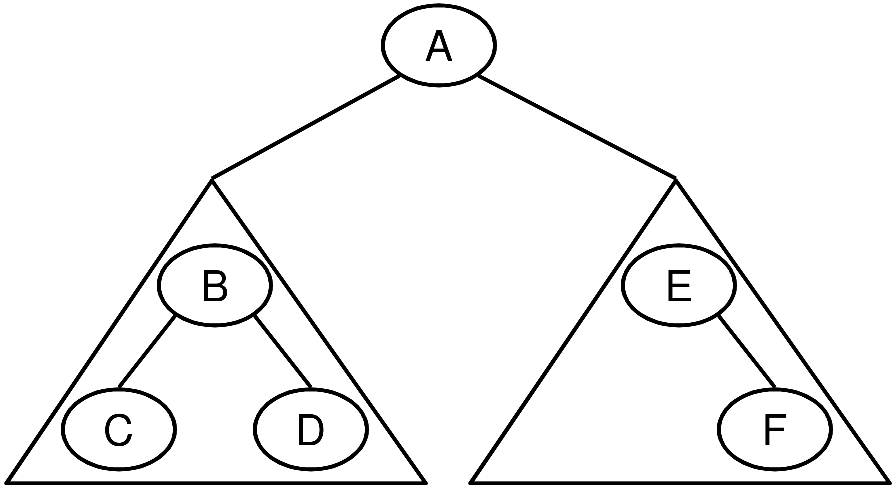
Traverses a binary tree in left-node-right sequence

```
Pre   root is the entry node of a tree/subtree
Post        each node has been processed in order

1  if (root is not null)
   1     postOrder (root –> leftSubTree)
   2     postOrder (root –> rightSubTree)
   3     process (root)
4  return

End  preOrder
```

Breadth-First Traversal

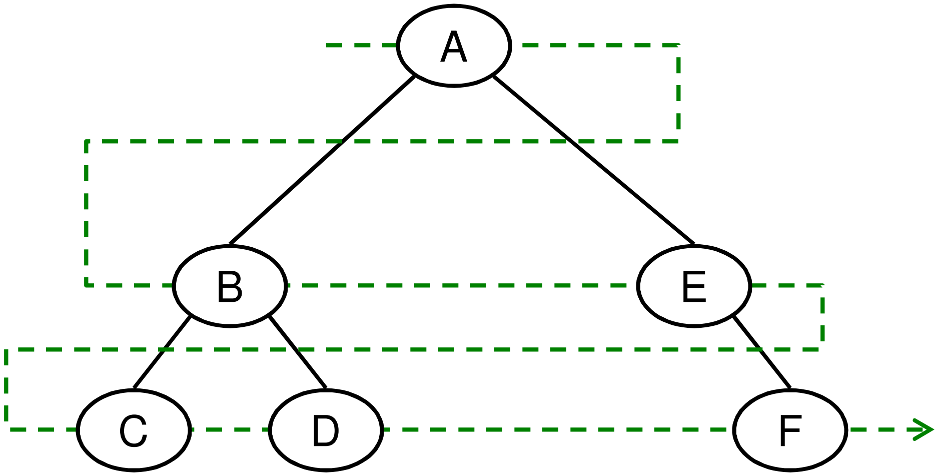


A

B E

C D F

Processing order



Walking order

Breadth-First Traversal

```
Algorithm breadthFirst (val root <nodePointer>)
1  pointer = root
2  while (pointer not null)
   1  process (pointer)
   2  if (pointer –> left not null)
      1  enqueue (pointer –> left)
   3  if (pointer –> right not null)
      1  enqueue (pointer –> right)
   4  if (not emptyQueue)
      1  dequeue (pointer)
   5  else
      1  pointer = null
3  return
End  breadthFirst
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
class BinaryTree {
public:
    BinaryTree();
    ~BinaryTree();
    void PreOrder(void (* visit)(EntryDataType &));
    void InOrder(void (* visit)(EntryDataType &));
    void PostOrder(void (* visit)(EntryDataType &));
    bool IsEmpty();
    void Clear();
     int GetSize(); int GetHeight();
```

Binary Tree Implementation in C++

```
protected:
    BinaryNode<EntryDataType>* root;

    void recursive_PreOrder(BinaryNode<EntryDataType>*,
    void (*visit) (EntryDataType &));
    void recursive_InOrder(BinaryNode<EntryDataType>*,
    void (*visit) (EntryDataType &));
    void recursive_PostOrder(BinaryNode<EntryDataType>*,
    void (*visit) (EntryDataType &));
    void clear(BinaryNode<EntryDataType>* &);
    int getSize(BinaryNode<EntryDataType>* p);
    int getHeight(BinaryNode<EntryDataType>* p);
};
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
BinaryTree<EntryDataType>::BinaryTree(){
   root = NULL;
}
template <class EntryDataType>
BinaryTree<EntryDataType>::~BinaryTree(){
   Clear();
}
template <class EntryDataType>
bool BinaryTree<EntryDataType>::IsEmpty(){
   return root == NULL;
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::PreOrder(
    void (*visit) (EntryDataType &) ) {
    recursive_PreOrder(root, visit);
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::recursive_PreOrder(
    BinaryNode<EntryDataType> *p,
    void (*visit) (EntryDataType &)) {
    if (p != NULL) {
        (*visit)(p->data);
        recursive_PreOrder(p->left, visit);
        recursive_PreOrder(p->right, visit);
    }
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::InOrder(
   void (*visit) (EntryDataType &)) {
   recursive_InOrder(root, visit);
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::recursive_InOrder(
   BinaryNode<EntryDataType> *p,
   void (*visit) (EntryDataType &)) {
   if (p != NULL){
        recursive_InOrder(p->left, visit);
        (*visit)(p->data);
        recursive_InOrder(p->right, visit);
   }
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::PostOrder(
    void (*visit) (EntryDataType &)) {
    recursive_PostOrder(root, visit);
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::recursive_PostOrder(
    BinaryNode<EntryDataType> *p,
    void (*visit) (EntryDataType &)) {
    if (p != NULL){
            recursive_PostOrder(p->left, visit);
            recursive_PostOrder(p->right, visit);
            (*visit)(p->data);
    }
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::Clear() {
   clear(root);
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
void BinaryTree<EntryDataType>::clear(
                                        BinaryNode<EntryDataType>* &p) {
    if (p != NULL) {
          clear(p->left);
          clear(p->right);
          delete p;
    }
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
int BinaryTree<EntryDataType>::GetSize() {
    return getSize(root);
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
int BinaryTree<EntryDataType>::getSize(
    BinaryNode<EntryDataType>* p) {
    if (p == NULL)
            return 0;
    return (1 + getSize(p->Left) + getSize(p->right));
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
int BinaryTree<EntryDataType>::GetHeight() {
   return getHeight(root);
}
```

Binary Tree Implementation in C++

```
template <class EntryDataType>
int BinaryTree<EntryDataType>::getHeight(
    BinaryNode<EntryDataType>* p) {
    if (p == NULL)
        return 0;
    int left = getHeight(p->left);
    int right = getHeight(p->right);
    return 1 + (left > right ? left : right);
}
```

Exercise

- Write a method and a recursive function to count the leaves of a linked binary tree
- Write Clone() method for BinaryTree class
- Check whether 2 binary trees are equal
- Interchange all left and right subtrees in a linked binary tree
- Find the width of a linked binary tree

Binary Tree Implementation in C++

- Input data for binary tree

```
void main() {
    BinaryTree* tree = new BinaryTree();
    tree->Input();
}
class BinaryTree {
public: //...
    Input();
private: //...
    recursive_input(TreeNode* &root );
};
void BinaryTree::Input() {
    recursive_input(this->root);
}
```

Binary Tree Implementation in C++

```
void BinaryTree::recursive_input(TreeNode* &root) {
    int data;
    printf("Key 0 is NULL: "); scanf("%d", &data);
    if (data == 0)
        root = NULL;
    else {
        root = new Node(); root->data = data;
        printf("Left child of %d:\n", data);
        recursive_input(root->left);
        printf("Right child of %d:\n", data);
        recursive_input(root->right);
    }
}
```