**Traverse Binary Tree non-recursively In-order, Pre-Order & Post-order**

**Pre-order Traversal**

void preorder(const Node\* node) {

std::stack<const Node\*> stack;

while (node != NULL || !stack.empty()) {

if (node == NULL) {

node = stack.top();

stack.pop();

}

process(node->value);

if (node->right != NULL)

stack.push(node->right);

node = node->left;

}

}

**In-order Traversal**

void inorder(const Node\* node) {

std::stack<const Node\*> stack;

while (node != NULL || !stack.empty()) {

if (node == NULL) {

node = stack.top();

stack.pop();

process(node->value);

node = node->right;

}

if (node != NULL) {

stack.push(node);

node = node->left;

}

}

}

**Post-order Traversal**

void postorder(const Node\* node) {

std::stack<const Node\*> stack;

while (node != NULL || !stack.empty()) {

if (node == NULL) {

while (!stack.empty() && node == stack.top()->right) {

node = stack.top();

stack.pop();

process(node->value);

}

node = stack.empty() ? NULL : stack.top()->right;

}

if (node != NULL) {

stack.push(node);

node = node->left;

}

}

}

**In-Order without Stacks**

void InorderTraversal(Node \*root){

if(!root)

return;

Node \*last = NULL;

Node \*cursor = root;

while(true){

if(last){ // Going up

if(cursor->left && cursor->left == last){

last = NULL;

cout<<data<<”\n”;

if(!MoveCursorRight(&cursor)){

last = cursor;

if(!MoveCursorUp(&cursor))

return;

}

} else {

last = cursor;

if(!MoveCursorUp(&cursor))

return;

}

} else{ // Going down

while(cursor->left)

cursor=cursor->left;

cout<<data<<”\n”;

last = NULL;

if(!MoveCursorRight(&cursor)){

last = cursor;

if(!MoveCursorUp(&cursor))

return;

}

}

}

}

bool MoveCursorRight(Node \*\*cursor)

{

if(\*cursor->right){

\*cursor = \*cursor->right;

return true;

}

return false;

}

bool MoveCursorUp(Node \*\*cursor){

if(\*cursor->parent){

\*cursor = cursor->parent;

return true;

}

return false;

}