Lab Manual

CS216L – Data Structures Lab
Lab No: 11
Topic: **Tree**

Class: **BSAI**
Semester: **III**
Session: **Fall, 2025**
Instructor: Asra Masood

Lab Date: 2nd December, 2025

Air University Islamabad
FACULTY OF COMPUTING & ARTIFICAL INTELLIGENCE
Department of Creative Technologies

Instructions

Submission: Use proper naming convention for your submission file. Name the submission file as Lab_NO_DEGREE_ROLLNUM (e.g. Lab_01_BSAI_00000). Submit the file on Google Classroom within the deadline. Failure to submit according to the above format would result in deduction of 10% marks. Submissions on the email will not be accepted.

Plagiarism: Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in the assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

Deadline: The deadlines to submit the assignment are hard. Late submission with marks deduction will be accepted according to the course policy shared by the instructor. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

Comments: Comment your code properly. Bonus marks (maximum 10%) will be awarded to well comment code. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.

Tip: For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

1. Note: Follow the given instructions to the letter, failing to do so will result in a zero.

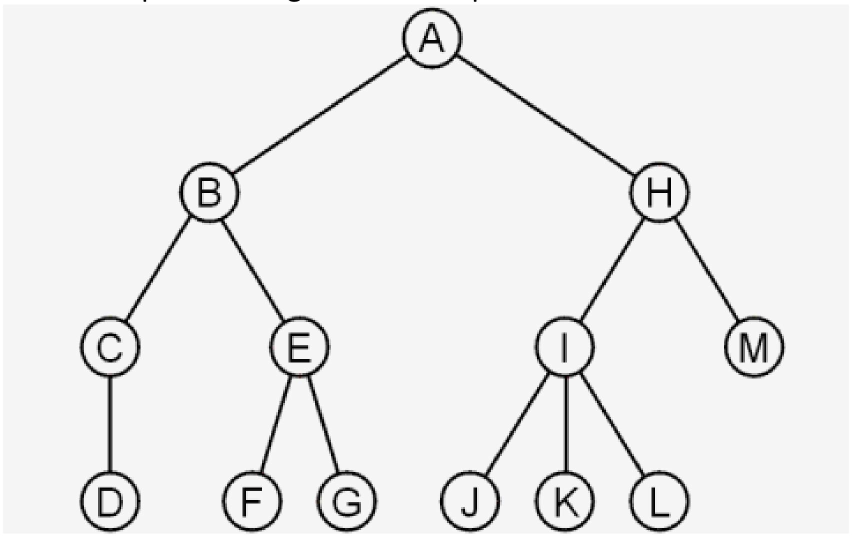
Objectives

In this lab, you will learn:

1. Trees
2. Tree Implementation using linked list
3. Tree Traversals.

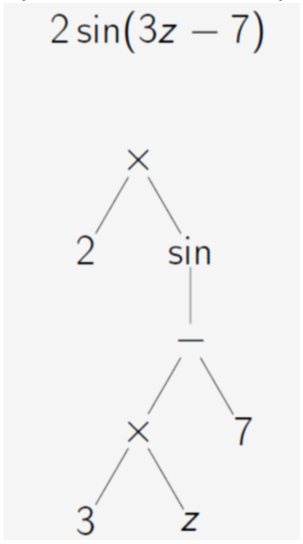
Trees:

A Tree is: empty, or a node with: a key, and a list of child trees.
Or
A tree is a finite set of nodes together with a finite set of edges that define parent-child relationships. Each edge connects a parent to its child.

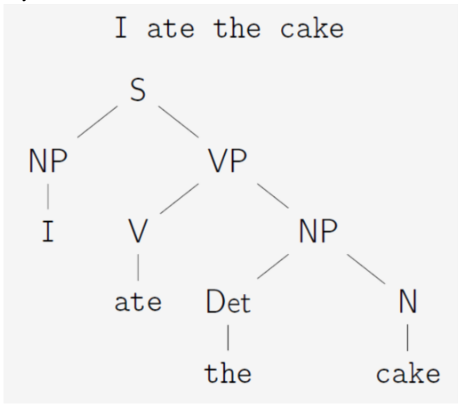


Examples:

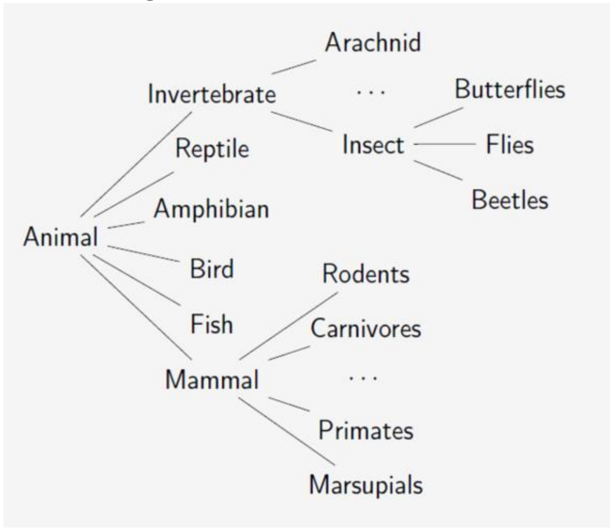
- Syntax Tree for an expression



- Syntax Tree for a sentence



- Animal Kingdom



- All nodes will have zero or more child nodes or children
- For all nodes other than the root node, there is one parent
- Nodes with the same parent are siblings

Tree Terminologies:

Node

A node is an entity that contains a key or value and pointers to its child nodes.

The last nodes of each path are called leaf nodes or external nodes that do not contain a link/pointer to child nodes.

The node having at least a child node is called an internal node.

Edge

It is the link between any two nodes.

Root

It is the topmost node of a tree.

Depth of a Node

The depth of a node is the number of edges from the root to the node.

Height of a Tree

The height of a Tree is the height of the root node or the depth of the deepest node.

Degree of a Node

The degree of a node is the total number of branches of that node.

Forest

A collection of disjoint trees is called a forest.

Types of Tree:

- Binary Tree
- Binary Search Tree
- AVL Tree

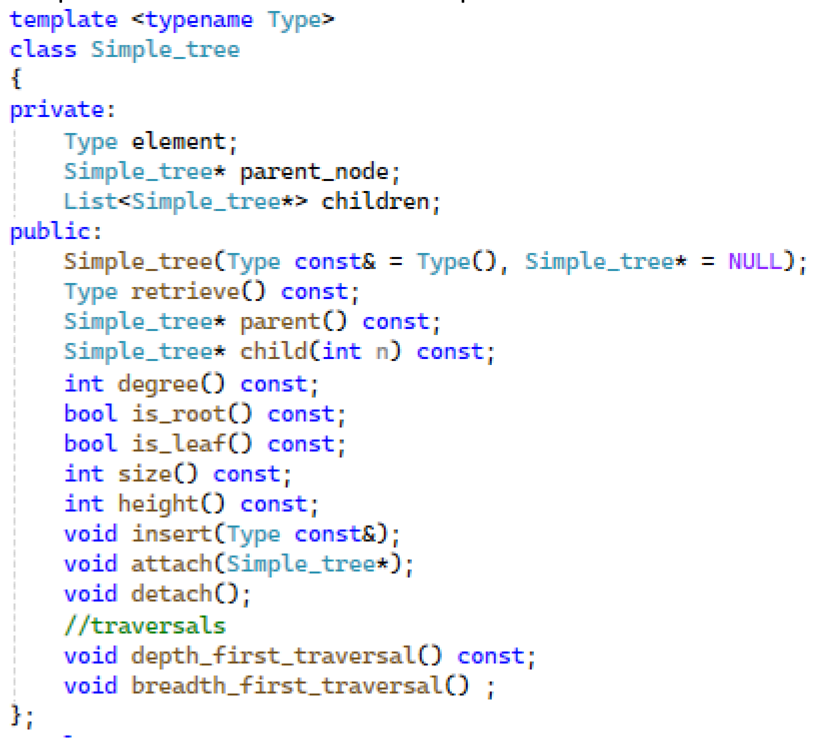
Implementation of General trees:

We can implement a general tree by using a tree class which:

Stores an element

Stores the children in a linked-list

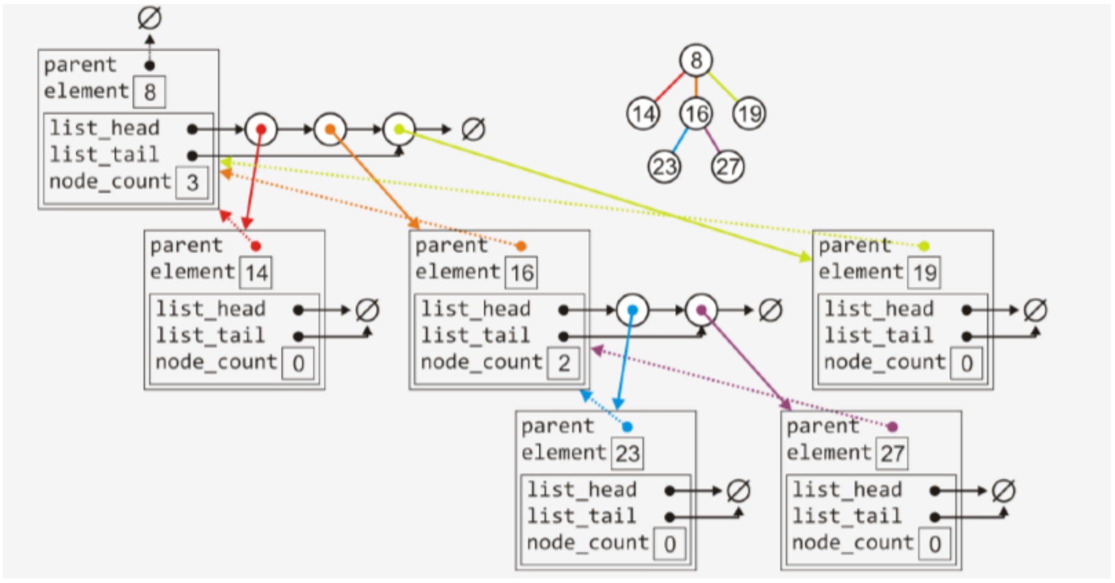
Complete definition of Tree class is provided below:



```
template <typename Type>
class Simple_tree
{
private:
    Type element;
    Simple_tree* parent_node;
    List<Simple_tree*> children;
public:
    Simple_tree(Type const& = Type(), Simple_tree* = NULL);
    Type retrieve() const;
    Simple_tree* parent() const;
    Simple_tree* child(int n) const;
    int degree() const;
    bool is_root() const;
    bool is_leaf() const;
    int size() const;
    int height() const;
    void insert(Type const&);
    void attach(Simple_tree*);
    void detach();
    //traversals
    void depth_first_traversal() const;
    void breadth_first_traversal() ;
};
```

Note:

Use the node and list class code (using templates). You also need Queue code (using templates) for traversals.



Tree Constructor:

Initialize:

Element to the data and parent node.

```
template <typename Type>
Simple_tree<Type>::Simple_tree(Type const& obj, Simple_tree* p) :element(obj), parent_node(p)
{
    // Empty constructor
}
```

Note: Initialization is performed in the order specified in the class declaration.

Retrieve:

Return the element in the node (tree).

```
template <typename Type>
Type Simple_tree<Type>::retrieve() const
{
    return element;
}
```

Parent:

Return parent node.

```
template <typename Type>
Simple_tree<Type>* Simple_tree<Type>::parent() const
{
    return parent_node;
}
```

Is root:

Check whether tree node is root or not (if no parent, then root)

```
template <typename Type>
bool Simple_tree<Type>::is_root() const
{
    return (parent() == NULL);
}
```

Child:

Find nth child of the tree:

```
template <typename Type>
Simple_tree<Type>* Simple_tree<Type>::child(int n) const
{
    if (n<0 || n> degree())
    {
        return NULL;
        //throw underflow();
    }
    Node<Simple_tree*>* ptr = children.head();
    for (int i = 1; i < n; ++i)
    {
        ptr = ptr->next();
    }
    return ptr->retrieve();
}
```

Degree:

Return the number of children (size of children list).

```
template <typename Type>
int Simple_tree<Type>::degree() const
{
    return children.size();
}
```

Is leaf:

Check whether tree node is leaf or not (check if node has children: if degree is zero then leaf)

```
template <typename Type>
bool Simple_tree<Type>::is_leaf() const
{
    return (degree() == 0);
}
```

Insert:

Add a child node to the tree

```
template <typename Type>
void Simple_tree<Type>::insert(Type const& obj)
{

    children.push_back(new Simple_tree(obj, this));
    //cout<< children.back()->retrieve();
}
```

Size:

Count the size of tree.

```
template <typename Type>
int Simple_tree<Type>::size() const
{
    int s = 1;
    for (Node<Simple_tree*>* ptr = children.head();ptr != NULL; ptr = ptr->next())
    {
        s += ptr->retrieve()->size();
    }
    return s;
}
```

Height:

Return height of the tree.

```
template <typename Type>
int Simple_tree<Type>::height() const
{
    int h = 0;
    for (Node<Simple_tree*>* ptr = children.head();ptr != NULL; ptr = ptr->next())
    {
        h = std::max(h, 1 + ptr->retrieve()->height());
    }
    return h;
}
```

Attach:

Attach/add the tree to the parent node/tree.

```
template <typename Type>
void Simple_tree<Type>::attach(Simple_tree<Type>* tree)
{
    if (!tree->is_root())
    {
        tree->detach();
    }
    tree->parent_node = this;
    children.push_back(tree);
}
```

Note: if we are try to attach a sub tree (node which has parent) to another tree we need to detach it from its parent first.

Detach:

Detach the tree from its parent node.

```
template <typename Type>
void Simple_tree<Type>::detach()
{
    if (is_root())
    {
        return;
    }
    parent()->children.erase(this);
    parent_node = NULL;
}
```

Traversals

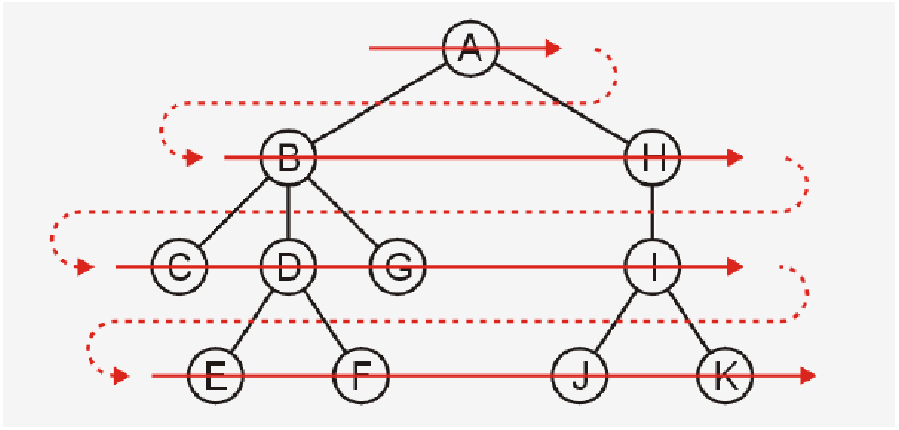
Depth first Traversal:

Depth first traversal visit always go as deep as possible before visiting other siblings.

```
template <typename Type>
void Simple_tree<Type>::depth_first_traversal() const
{
    cout << retrieve() << "\t";
    for (Node <Simple_tree *> * ptr = children.head();ptr != NULL;ptr = ptr->next())
    {
        ptr->retrieve()->depth_first_traversal();
    }
}
```

Breadth first Traversal:

The breadth first traversal visits all nodes at depth k before proceeding onto depth k + 1



Create a queue and push the root node onto the queue

While the queue is not empty:

Push all of its children of the front node onto the queue

Pop the front node

```
template <typename Type>
void Simple_tree <Type>::breadth_first_traversal()
{
    Queue<Simple_tree*> q;
    q.push(this);
    while (!q.empty())
    {
        Simple_tree* p = q.pop();
        cout << p->retrieve() << "\t";
        for (Node <Simple_tree*>* ptr = p->children.head(); ptr != NULL; ptr = ptr->next())
        {
            q.push(ptr->retrieve());
        }
    }
}
```

Task 1:

Complete the above implementation and test it by making a tree of your name and finding traversals.