Data Structures (in C++)

- Trees, Tree Traversal Algorithms, and Binary Trees -

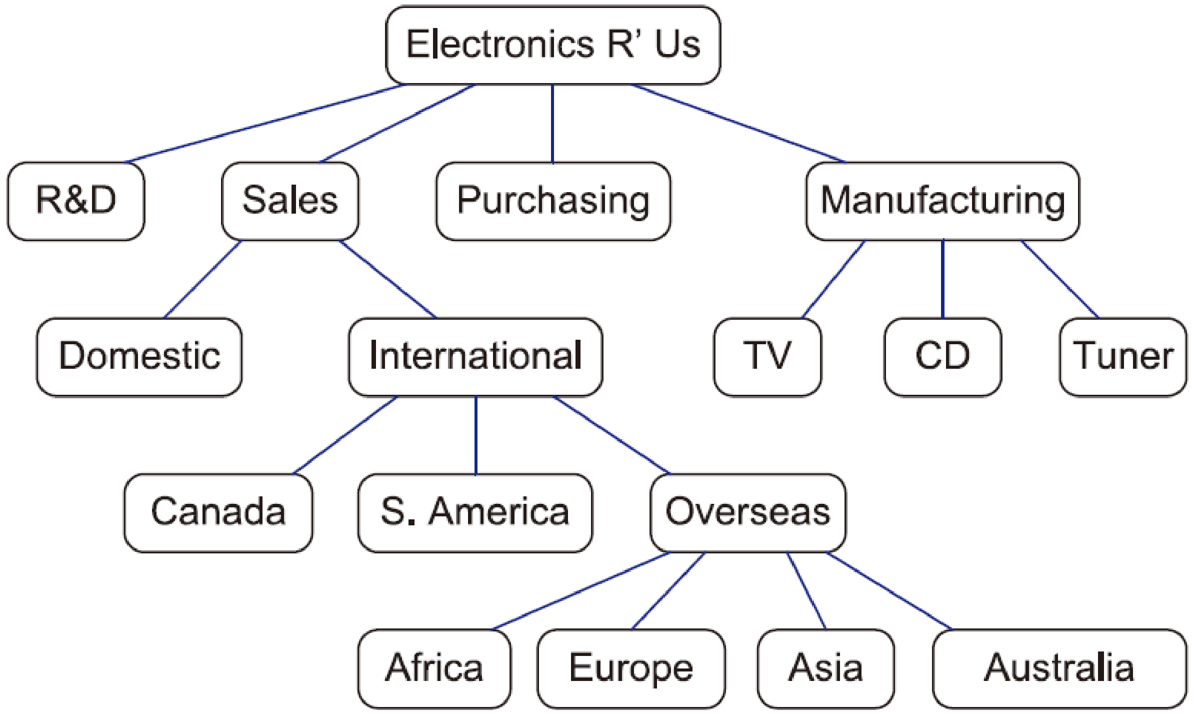
Jinsun Park

Visual Intelligence and Perception Lab., CSE, PNU

Trees

Trees

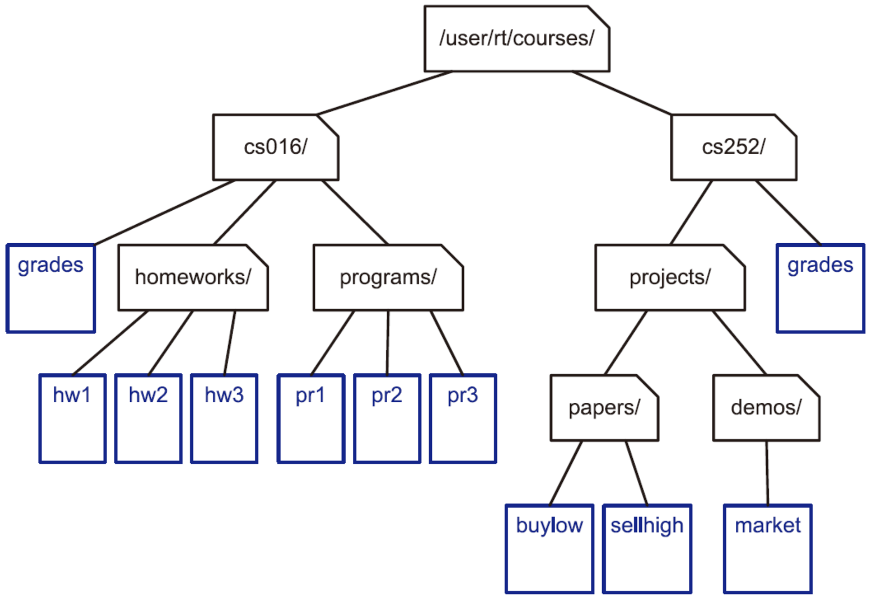
- **Tree**
  - A data type that stores elements hierarchically (*Above* and *below*)
  - Each element has a *parent* and zero or more *children* elements (except for the top element)
  - The top element is called the *root* of the tree



Trees

- **Formal Tree Definition**
  - We define tree T to be a set of *nodes* storing elements in a *parent-child* relationship with the following properties:
    - If T is nonempty, it has a special node, called the ***root*** of T, that has no parent.
    - Each node v of T different from the root has a unique ***parent*** node w; every node with parent w is a ***child*** of w.
  - Two nodes with the same parent are *siblings*
  - A node is *external* (or a *leaf*) if it has no children
  - A node is *internal* if it has one or more children

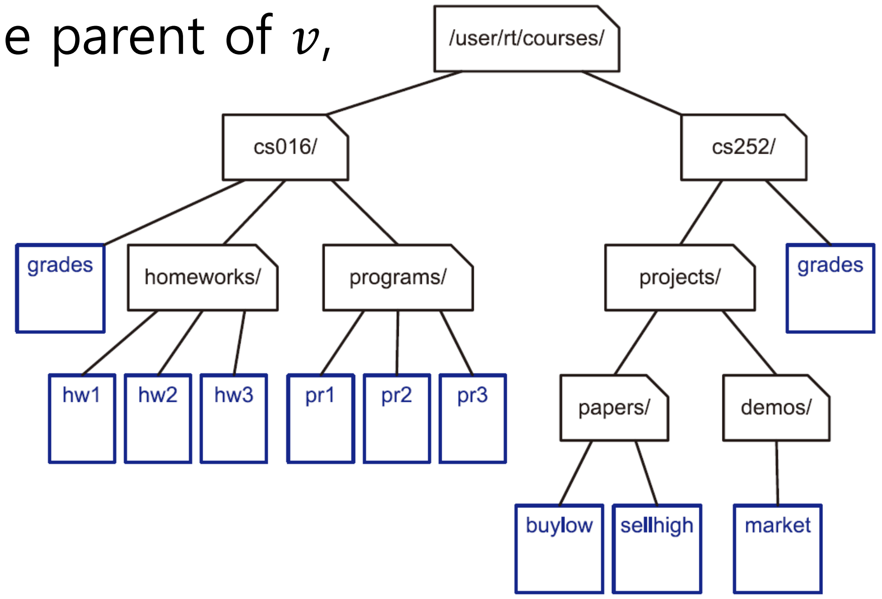
UNIX system root directory : /



Trees

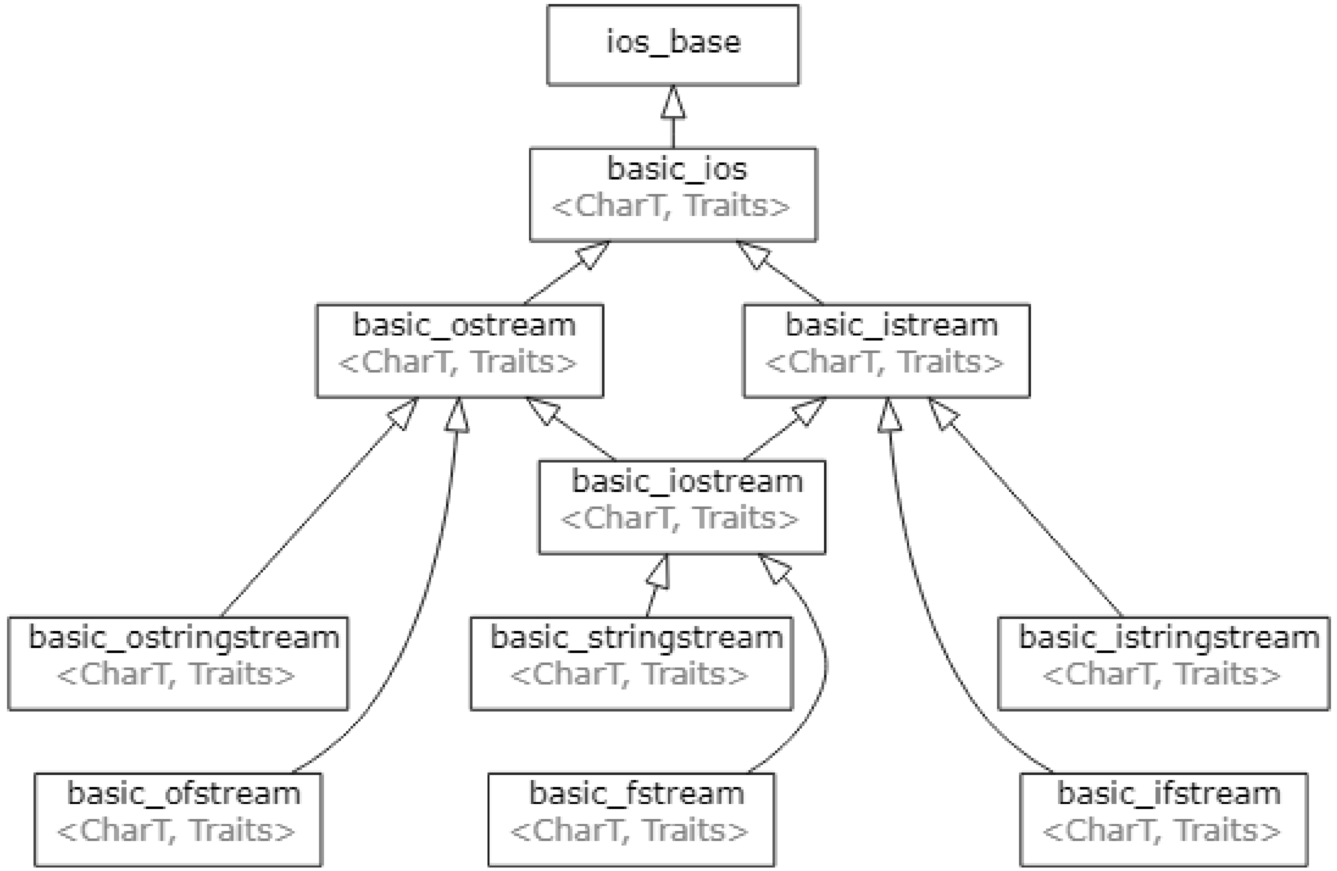
- **Formal Tree Definition**
  - A node u is an *ancestor* of a node v if $u = v$ or u is an ancestor of the parent of v
  - A node v is a *descendant* of a node u if u is an ancestor of v
  - The *subtree* of T rooted at a node v is the tree consisting of all the descendants of v (including v itself)
  - An *edge* of tree T is a pair of nodes (u, v) such that u is the parent of v, or vice versa.
  - A *path* of T is a sequence of nodes such that any two consecutive nodes in the sequence form an edge

UNIX system root directory : /



Trees

- **Tree Example: C++ Inheritance**


ios_base
basic_ios <CharT, Traits>
basic_ostream <CharT, Traits>
basic_istream <CharT, Traits>
basic_iostream <CharT, Traits>
basic_ostringstream <CharT, Traits>
basic_stringstream <CharT, Traits>
basic_istringstream <CharT, Traits>
basic_ofstream <CharT, Traits>
basic_fstream <CharT, Traits>
basic_ifstream <CharT, Traits>

https://en.cppreference.com/w/cpp/io#Stream-based_I.2FO

Trees

- **Ordered Tree**
  - There is a linear ordering defined for the children of each node
  - Children of a node can be identified as the first, second, third and so on (*i.e.*, left to right)

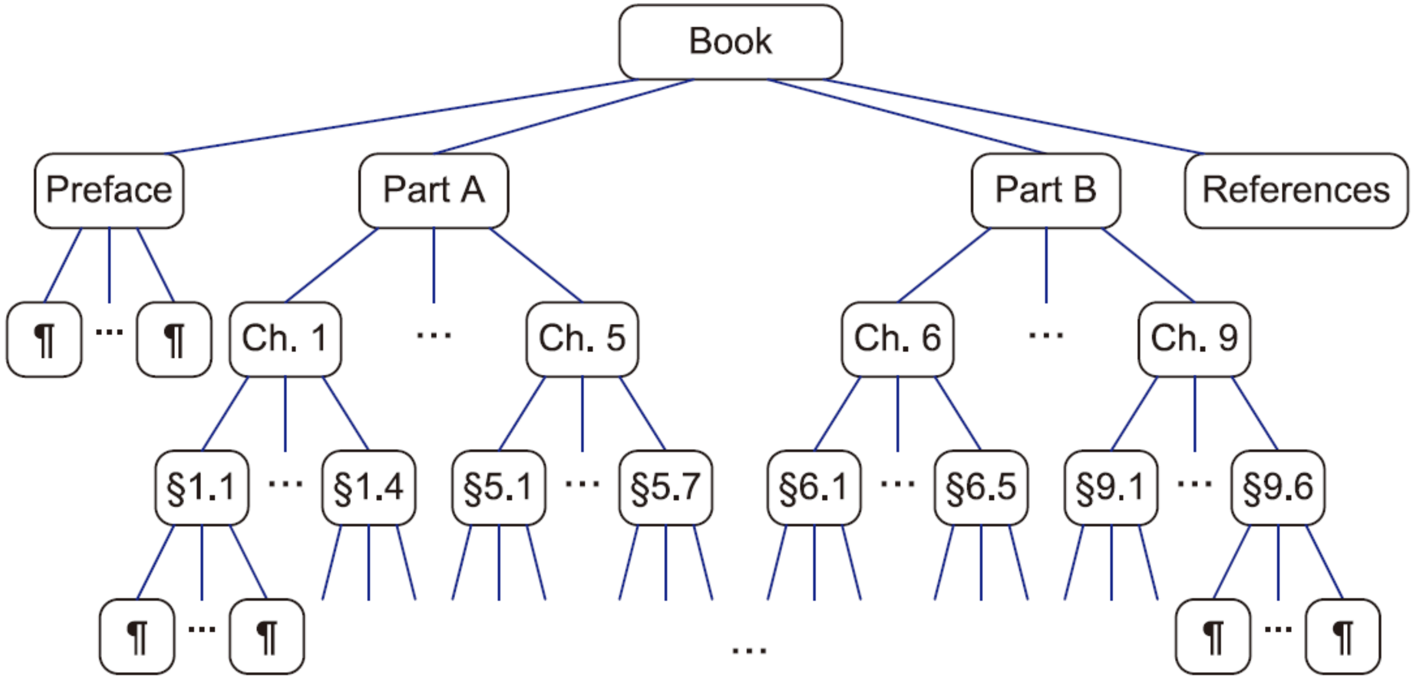


Figure 7.4: An ordered tree associated with a book.

Tree ADT

- It stores elements at the nodes

- Each node of the tree is associated with a *position* object
    - Provides public access to nodes

- The tree ADT supports the following operations given a position p of tree T:

p.parent():	Return the parent of p; an error occurs if p is the root.
p.children():	Return a position list containing the children of node p.
p.isRoot():	Return true if p is the root and false otherwise.
p.isExternal():	Return true if p is external and false otherwise.

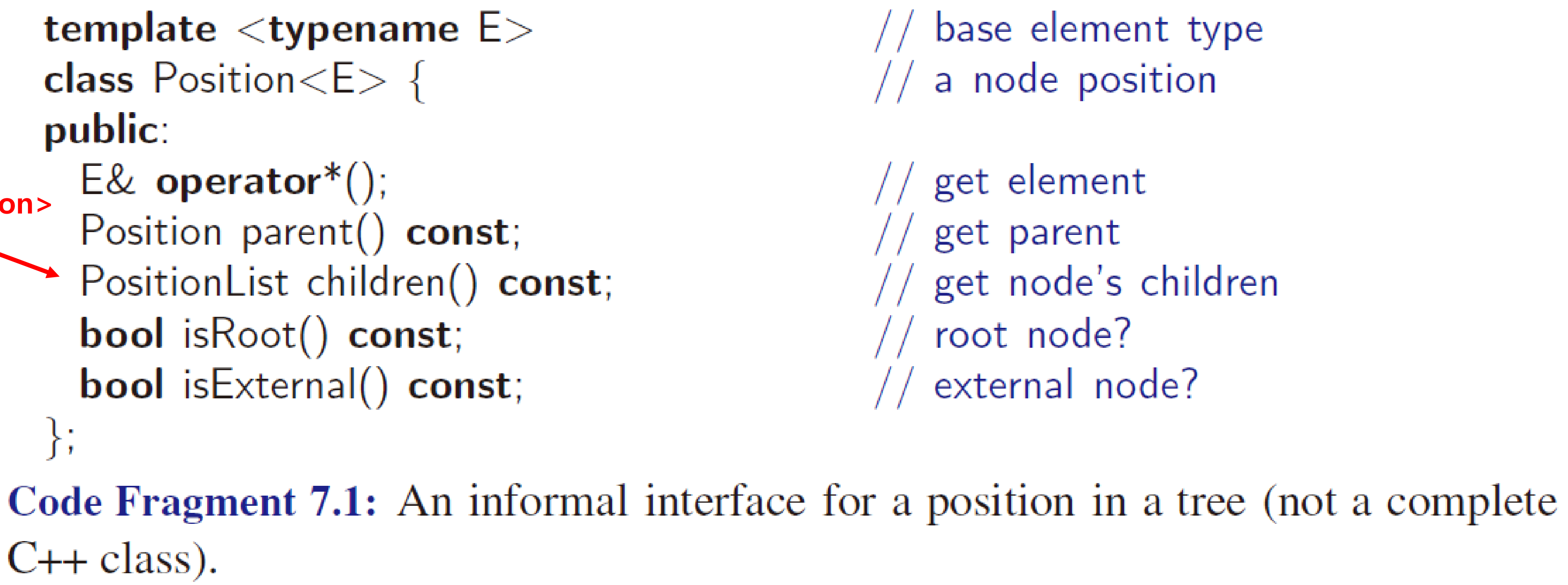
- Additional utility functions:

size():	Return the number of nodes in the tree.
empty():	Return true if the tree is empty and false otherwise.
root():	Return a position for the tree's root; an error occurs if the tree is empty.
positions():	Return a position list of all the nodes of the tree.

Tree ADT

- **C++ Implementation**
  - Position implementation

std::list<Position>



```
template <typename E>                          // base element type
class Position<E> {                            // a node position
public:
  E& operator*();                              // get element
  Position parent() const;                     // get parent
  PositionList children() const;               // get node's children
  bool isRoot() const;                         // root node?
  bool isExternal() const;                     // external node?
};
```

Code Fragment 7.1: An informal interface for a position in a tree (not a complete C++ class).

Tree ADT

- **C++ Implementation**
  - Tree implementation

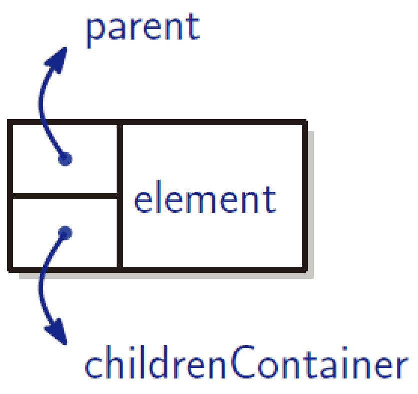
```
template <typename E>                        // base element type
class Tree<E> {
public:                                      // public types
  class Position;                            // a node position
  class PositionList;                        // a list of positions
public:                                      // public functions
  int size() const;                          // number of nodes
  bool empty() const;                        // is tree empty?
  Position root() const;                     // get the root
  PositionList positions() const;            // get positions of all nodes
};
```

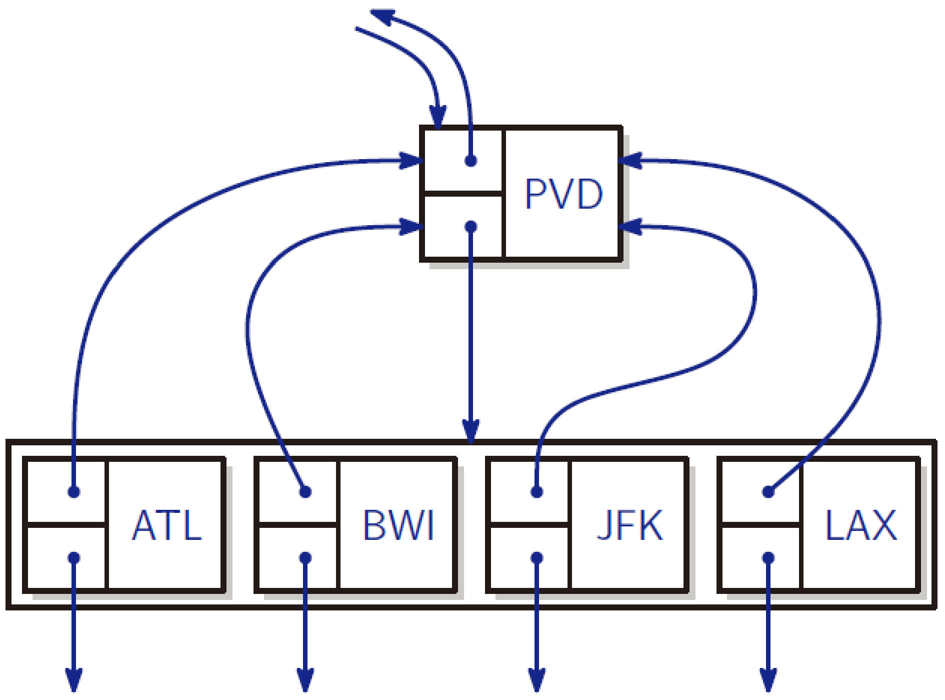
Code Fragment 7.2: An informal interface for the tree ADT (not a complete class).

  - There is no C++ STL container *std::tree* but *std::map* is implemented as a red/black tree

Tree ADT

- **A Linked Structure for General Trees**
  - Each node of T by a position object p has:
    - A reference to the node's element
    - A link to the node's parent
    - A collection to store links to the node's children (*e.g.*, PositionList)





using iterator

Operation	***Time***
isRoot, isExternal	$O(1)$
parent	$O(1)$
children(p)	$O(c_p)$
size, empty	$O(1)$
root	$O(1)$
positions	$O(n)$