Massey University
COLLEGE OF SCIENCES

159201

Week 4

Summer 2014

Te Kunenga
ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

L 13

Te Kunenga
ki Pūrehuroa

General Trees

Recall the single dimensional Data Structures:

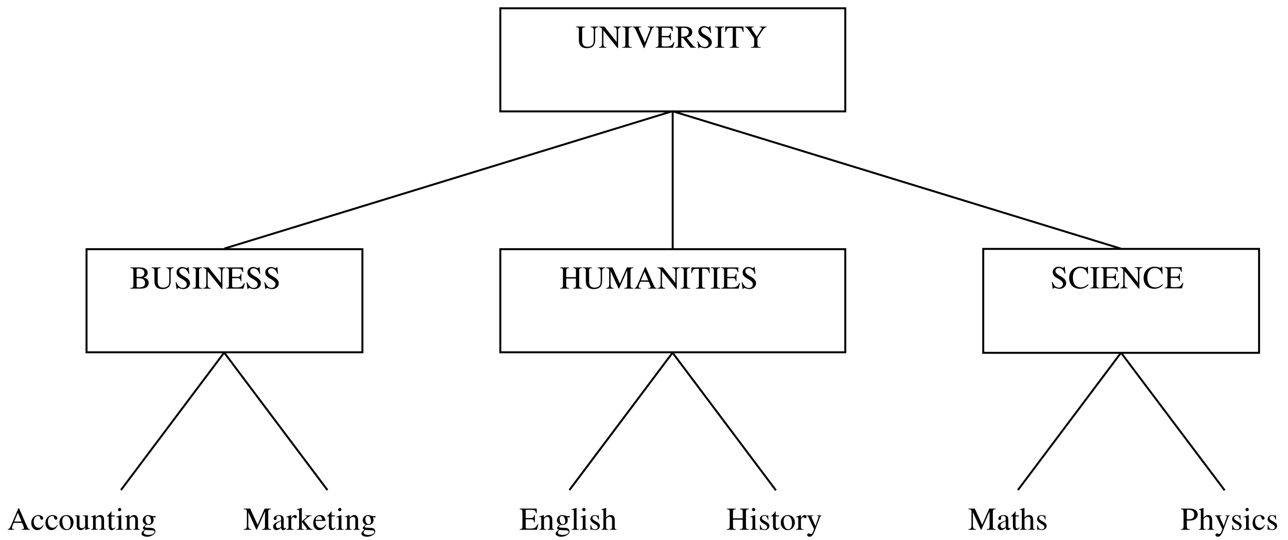
Stacks, Queues, Lists, Vectors

We move now to multi-dimensional **Data Structures**.

Trees of many types...

General Trees example

You have probably seen a tree before...



General Trees: definitions

Node – an item in a tree (any record or structure)

Branch – a link between two nodes

Root – the first node

Leaf – the last node (a tree may have many 'last' nodes)

Parent – the previous link of a node

Child – the next link of a node

General Trees: definitions (2)

Sibling – nodes with the same previous link

Levels – the root is level 1. The previous example has three levels.

Height – number of lebels in a tree (aka, ***depth***).

Null Tree – the tree of nothing, level zero.

Forest - many trees

Formal definition of a tree

A tree is either:

(a) the null tree

Or

(b) a root and several subtrees

A subtree is also a tree.

We assume trees are ordered, i.e., the order of siblings on any level is important.

We can define ***left-most child*** and ***right sibling.***

Formal definition of a tree (2)

Operations such as these allow us to navigate around a tree.

One can start with the root, and reach every other node by using a combination of these two operations.

Use of trees

Examples:

- Organisational structure
- Biology classifications
- Library books
- System decomposition e.g., parts of a car
- File systems e.g., /home, /home/user1 etc
- Family trees

Operations on trees

Insert a node (this affects the ordering of nodes)

Insert new child at the right

Insert a branch (require two nodes)

Look at a node

Delete a node (this may affect branches!)

Delete a branch

Left-most child (returns the node)

Right sibling (returns the node)

IsEmpty

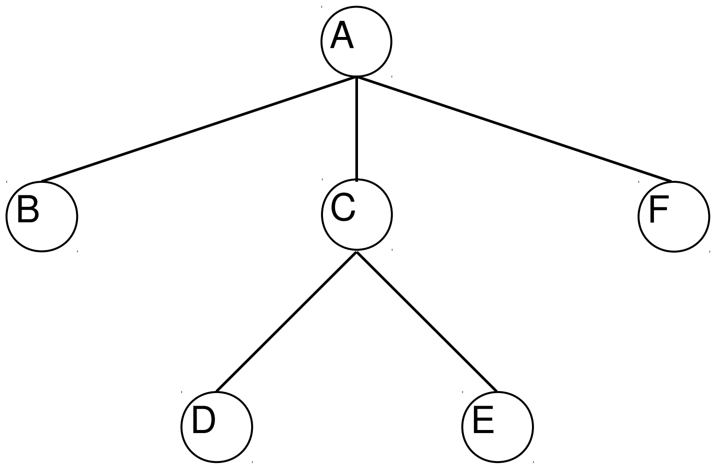
And many others...

Example of a general tree

Work through operations such as: right sibling at C is...

Left-most child of C is...

Parent of C is ...



Te Kunenga ki Pūrehuroa

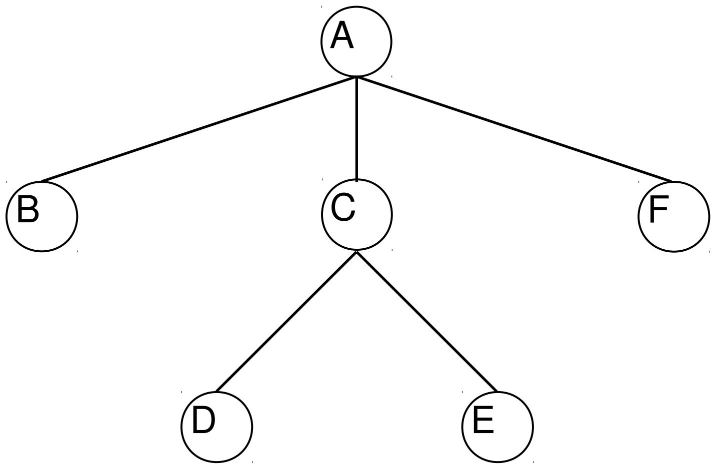
L 14

Te Kunenga
ki Pūrehuroa

Traversals and Implementation

We use letters to represent complex data

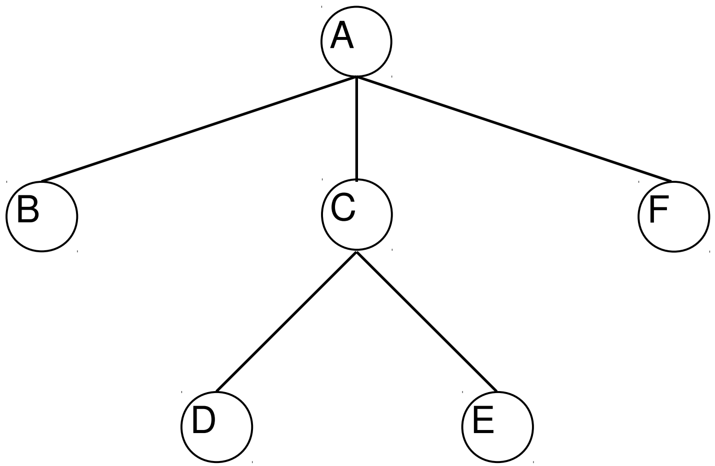
A ***tree traversal*** is an algorithm for searching every node in the tree. Each node is looked at ***once***.



Traversals

Why?

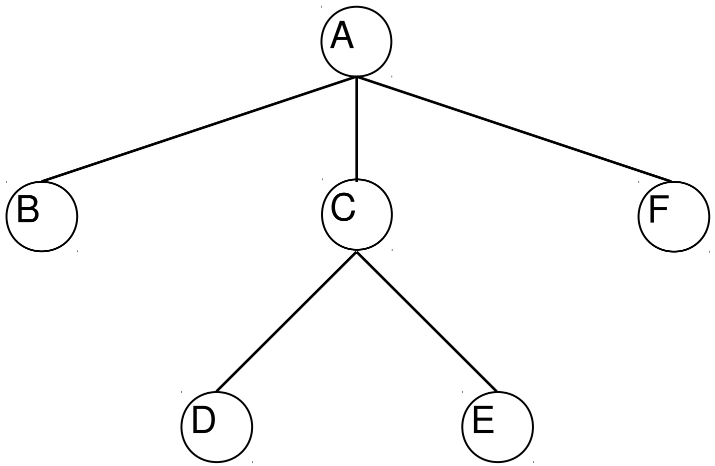
- To print out all data (in order)
- To look at a particular node
- To copy all data to another tree



Traversals

Many possible traversals, e.g.:

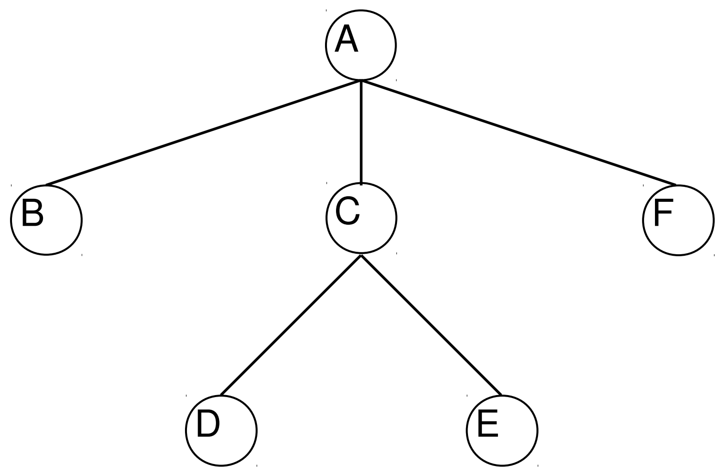
- Pre-order traversal
- In-order traversal
- Post-order traversal



Massey University
COLLEGE OF SCIENCES

Pre-order Traversals

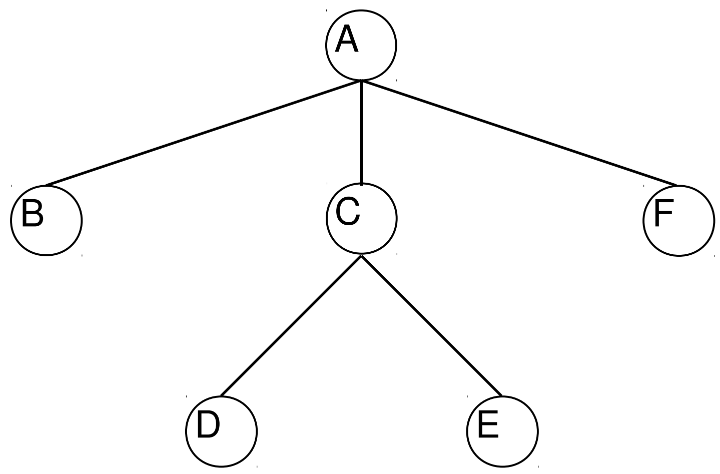
1. Visit the root
2. Pre-order traversal of all sub-trees from left to right



Te Kunenga ki Pūrehuroa

Pre-order Traversals

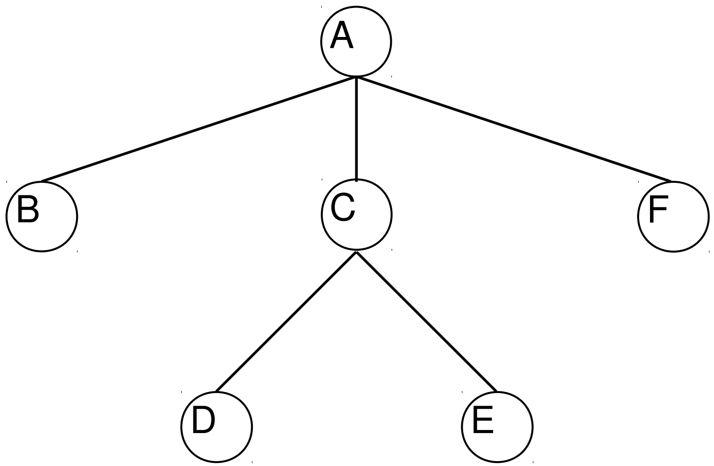
1. Visit the root
2. Pre-order traversal of all sub-trees from left to right



A B C D E F

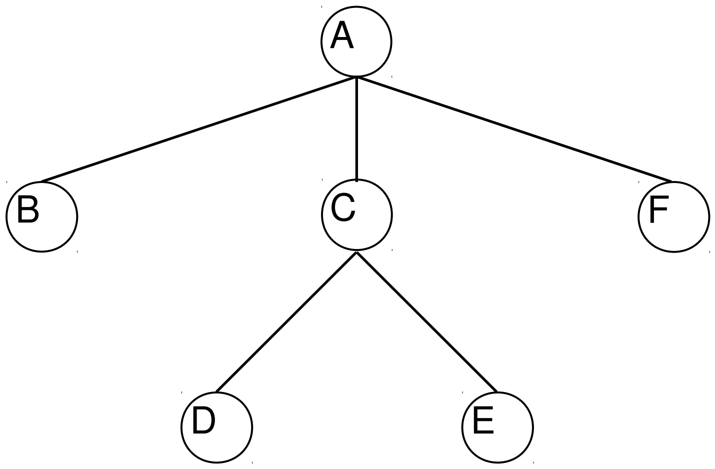
In-order Traversals

1. In-order traversal of the left sub-tree
2. Visit the root
3. In-order traversal of all remaining sub-trees from left to right



In-order Traversals

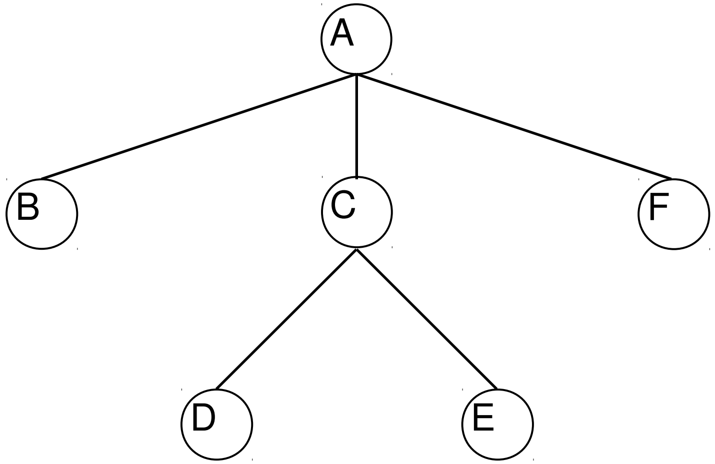
1. In-order traversal of the left sub-tree
2. Visit the root
3. In-order traversal of all remaining sub-trees from left to right



B A D C E F

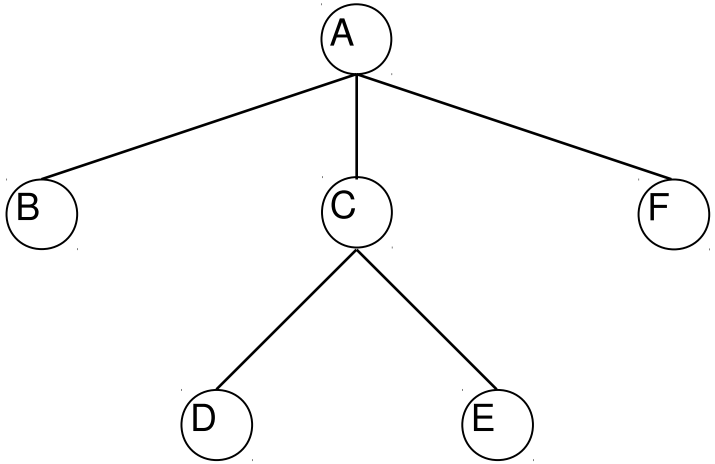
Post-order Traversals

1. Post-order traversal of all sub-trees from left to right
2. Visit the root



Post-order Traversals

1. Post-order traversal of all sub-trees from left to right
2. Visit the root

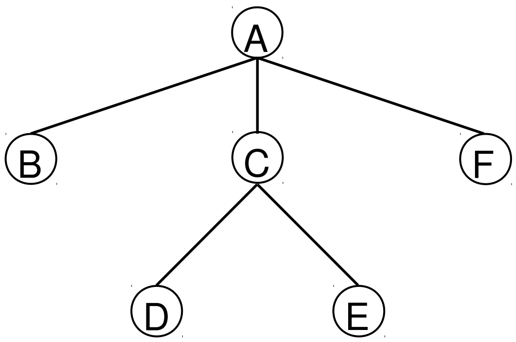


B D E C F A

Ideas for implementing trees

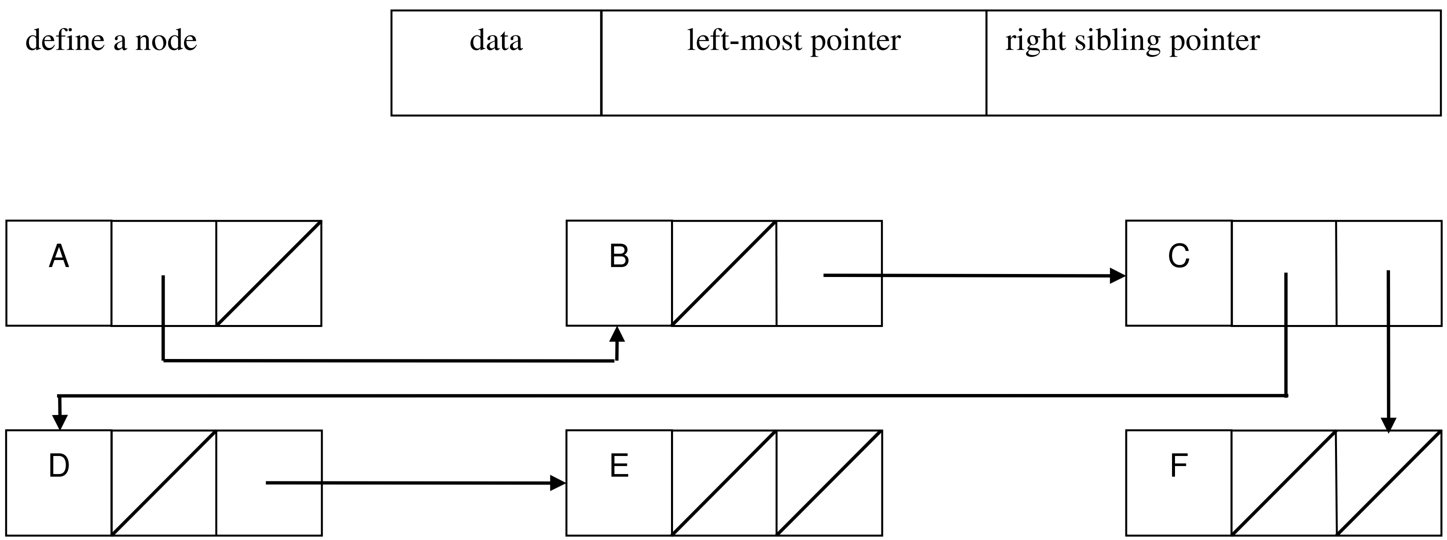
Using tables (2D arrays):

index	data	Left-most child	Right sibling
0	A	1	-1
1	B	-1	2
2	C	4	3
3	F	-1	-1
4	D	-1	5
5	E	-1	-1



Ideas for implementing trees

Using pointers:



Compare implementations

Tables:

fixed size

Structure forces a certain order

Difficulty to insert new nodes?

Pointers:

Dynamic memory allocation

Flexibility

How difficult is to insert new nodes?

About the operations...

Both implementations make the *left-most child* and *right sibling* relatively easy to implement

How about insert a *new node*?

Delete a node?

A *parent* operation?

Special cases of trees would help...

Challenge

1) Consider a different type of traversal: printing the nodes by layer (or height). Can you devise an *algorithm* to do it?

Massey University
COLLEGE OF SCIENCES

L 15

Te Kunenga
ki Pūrehuroa

Binary trees

General trees have no restrictions.

The traversal in such trees can become complex.

A special case of trees are binary trees, which have at most two children per node.

Binary trees definition

A ***binary tree*** is either:

a) the null tree

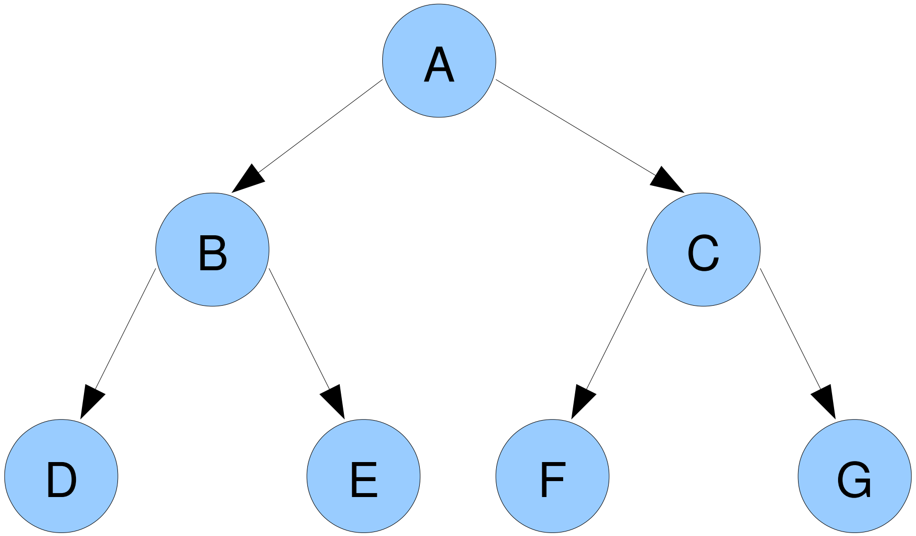
Or

b) a root and two sub-trees where each sub-tree is a binary tree.

Note: each node can have a ***left*** and a ***right*** sub-tree.

Binary tree example

Consider the following ***binary tree***



Te Kunenga ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

Representing general trees

A general tree can be stored as a binary tree as follows:

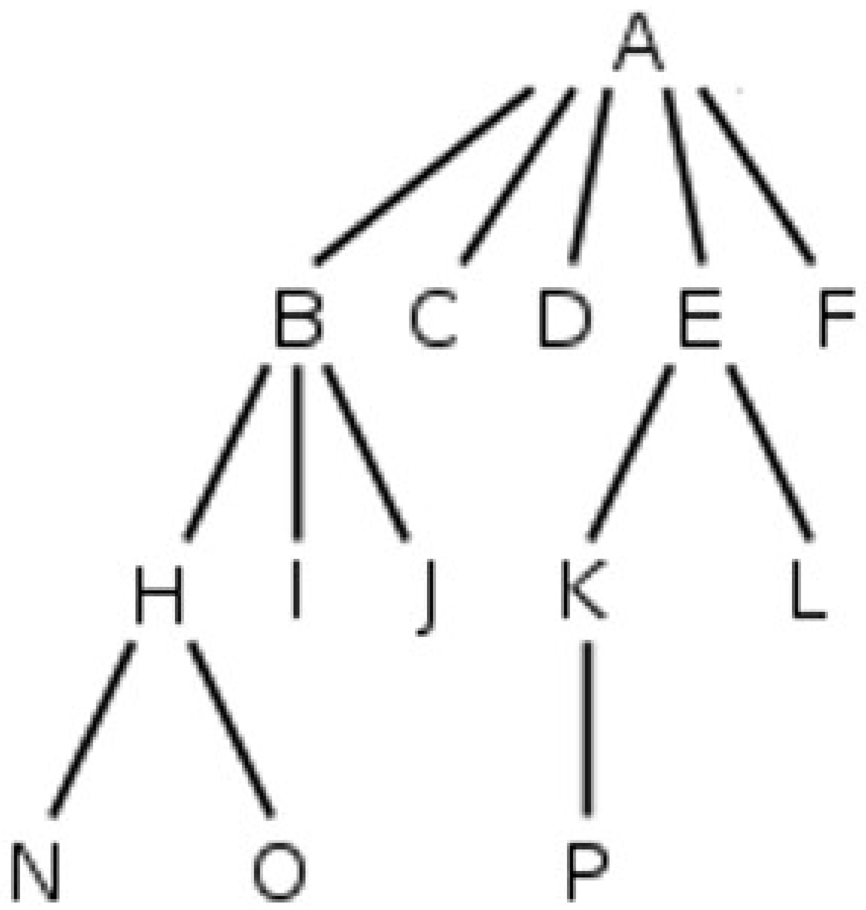
- Read every left pointer in the binary tree as left-most-child in the general tree.

- Read every right pointer in the binary tree as right-sibling in the general tree.

Te Kunenga ki Pūrehuroa

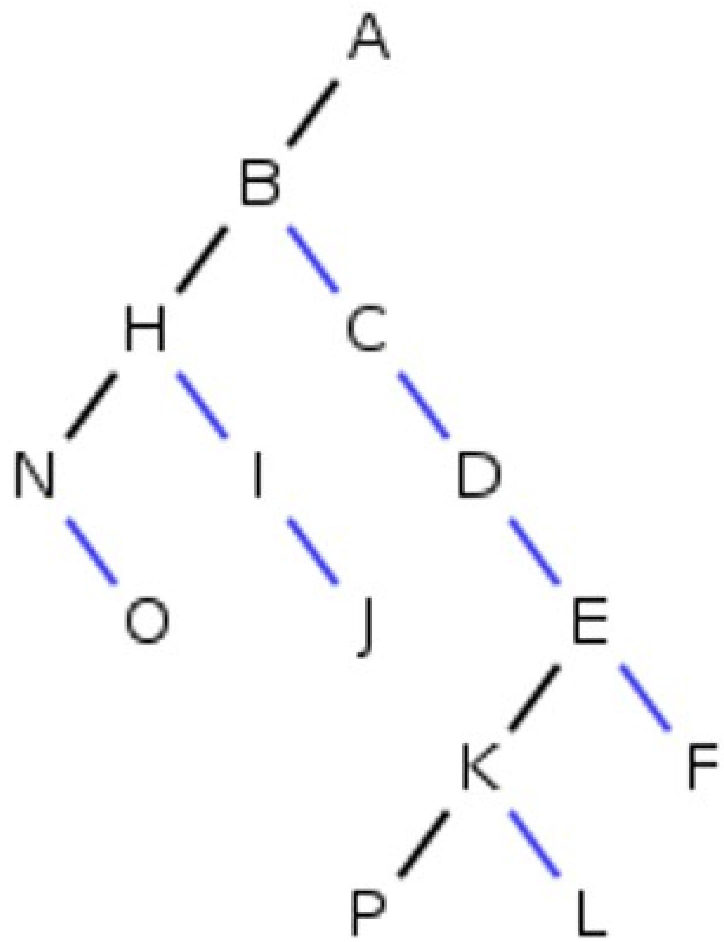
Representing general tress

The following general tree:



Representing general tress

Would be represented by the following binary tree:



C++ implementation

```
class Tree {
private:
  char data;
  Tree *leftptr, *rightptr;

public:
  Tree(char newthing, Tree* L, Tree* R);  // constructor
with parameters
  ~Tree() { }
  char RootData() { return data; }  // inline functions
  Tree* Left() { return leftptr; }
  Tree* Right() { return rightptr; }
};
```

C++ implementation

```
Tree::Tree(char newthing, Tree* L, Tree* R) {
  data = newthing;
  leftptr = L;
  rightptr = R;
}
```

Massey University
COLLEGE OF SCIENCES

C++ implementation

```
Tree *T1, *T2,...,*myTree;  // always use pointers to trees

int main() {
      T1 = new Tree('D', NULL, NULL);
      T2 = new Tree('E', NULL, NULL);
      T3 = new Tree('B', T1, T2);
      T4 = new Tree('F', NULL, NULL);
      T5 = new Tree('G', NULL, NULL);
      T6 = new Tree('C', T4, T5);
      myTree = new Tree('A', T3, T6);
 }
```

Te Kunenga ki Pūrehuroa

Binary trees traversal

Recall the standard traversals:

Pre-Order:
1. visit the root
2. pre-order traversal of the left subtree
3. pre-order traversal of the right subtree

In-Order:
1. in-order traversal of the left subtree
2. visit the root
3. in-order traversal of the right subtree

Post-Order:
1. post-order traversal of the left subtree
2. post-order traversal of the right subtree
3. visit the root

Binary trees traversal

Note: we are not changing the definition of the traversals used for general trees.

We are rewriting them using the knowledge that binary trees only have two sub-trees.

Now we can write a C++ function that performs traversals of a binary tree.

This is NOT a method in the OO sense, this is simply a function (C style)...

C++ implementation

```
void inOrder(Tree *T) {
   if (T == NULL) { return; }
   inOrder(T->Left());
   printf("%c ", T->RootData());
   inOrder(T->Right());
}
```

In-order traversal

Sequence in the tree with 7 nodes:

```
InOrder('A') InOrder('B') InOrder('D'),
InOrder(NULL)
Return, print 'D', InOrder(NULL)(right node)
Return, return, print 'B', InOrder('E')(right),
InOrder(NULL)
Return, print 'E', InOrder(NULL)(right)
Return, return, print 'A', InOrder('C')(right),
InOrder('F'), InOrder(NULL)
Return, print 'F', InOrder(NULL)(right)
Return,print 'C',InOrder('G')(right),InOrder(NULL)
Return, print 'G', InOrder(NULL)(right)
Return, return, return
```

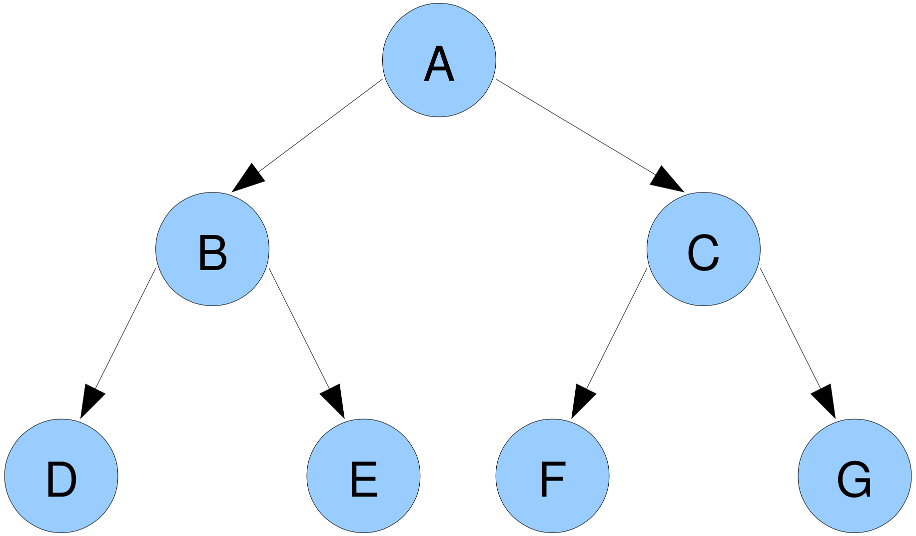
Massey University
COLLEGE OF SCIENCES

Binary tree traversals

InOrder: D B E A F C G

PreOrder: A B D E C F G

PostOrder: D E B F G C A



Te Kunenga ki Pūrehuroa

Pre-order

```
void PreOrder(Tree *T) {
  if (T == NULL) { return; }
      printf("%c \n", T->RootData());
      PreOrder(T->Left());
      PreOrder(T->Right());
}
```

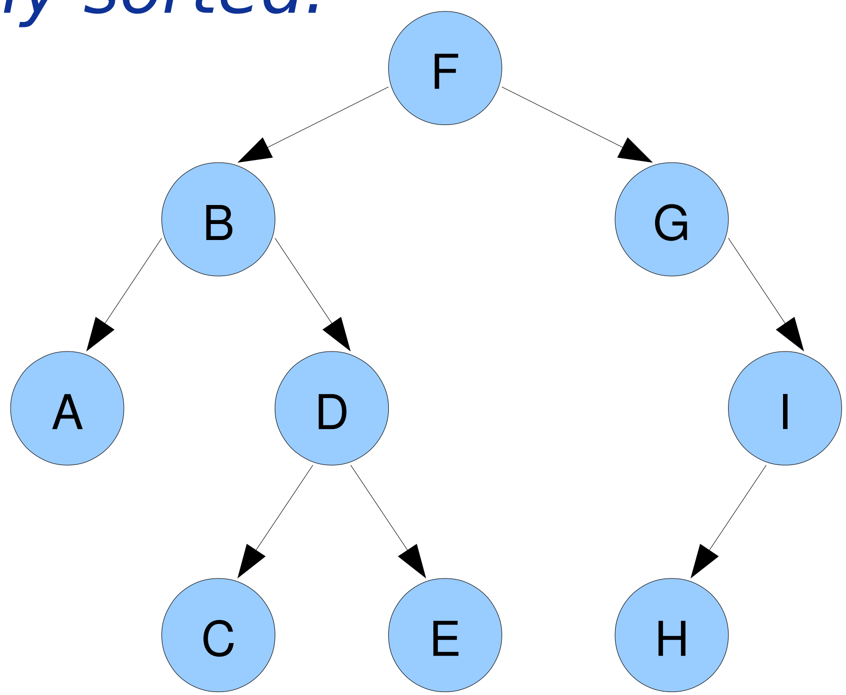
Post-order

```
void PostOrder(Tree *T) {
  if (T == NULL) { return; }
      PostOrder(T->Left());
      PostOrder(T->Right());
      printf("%c \n", T->RootData());
}
```

Test your code:

Use the following wikipedia tree to test your code.

Tip: the result of the InOrder traversal sequence is alphabetically sorted.



Source: http://en.wikipedia.org/wiki/Tree_traversal

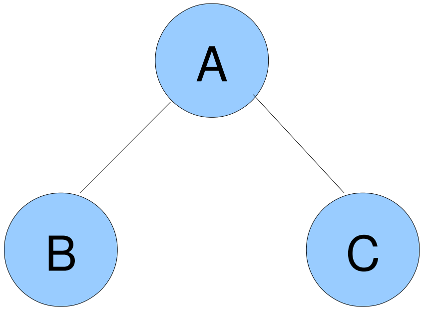
L 16

Massey University
COLLEGE OF SCIENCES

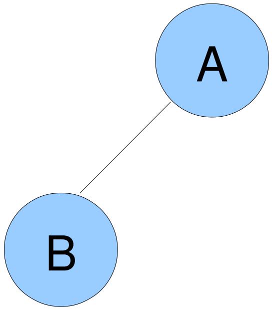
Binary Trees

A full binary tree is a binary tree in which every node has either zero or two children.

A full binary tree



This tree is not a full binary tree



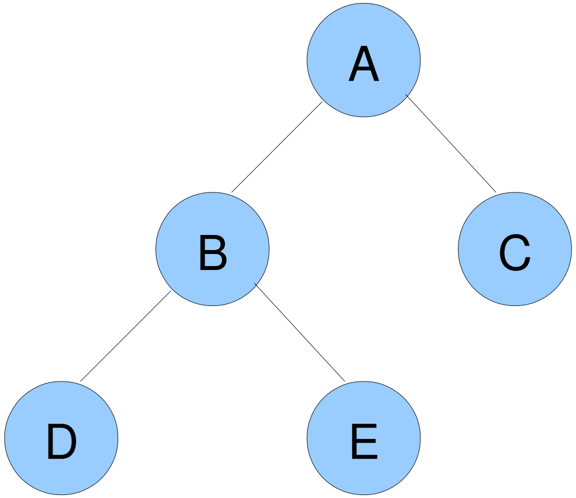
Te Kunenga ki Pūrehuroa

Complete Binary Trees

A complete (or perfect) binary tree is a full binary tree where all leaves are at the same level.

This is a full binary tree.

But it is not a complete binary tree.



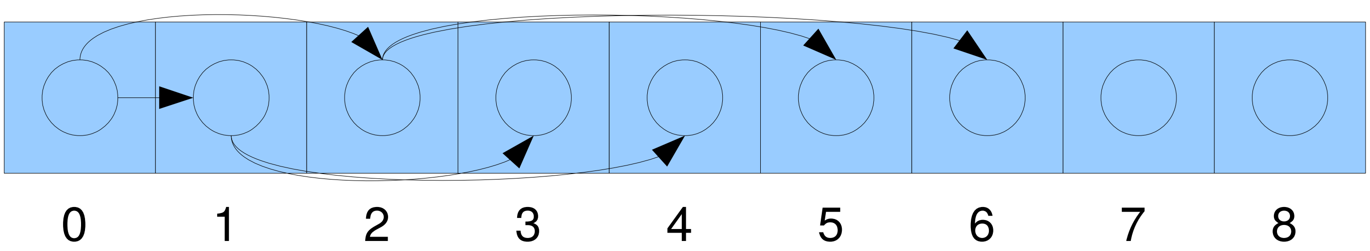
Storing Binary Trees

A different idea:

One could store it in an array

Root index = 0

Node index ***i*** has its children in ***2i+1*** and ***2i+2***



Storing Binary Trees

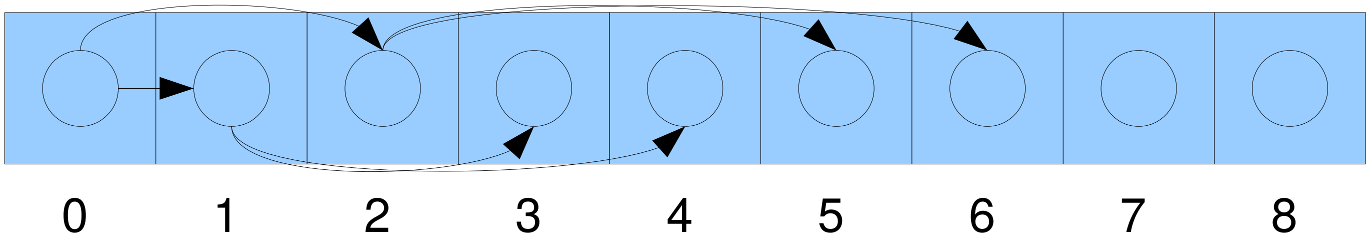
Advantage?

This uses random access and it is extremely fast.

Also, the parent of node ***i*** can be found at ***(i-1)/2***.

Disadvantage?

Waste of space if the tree is not complete.

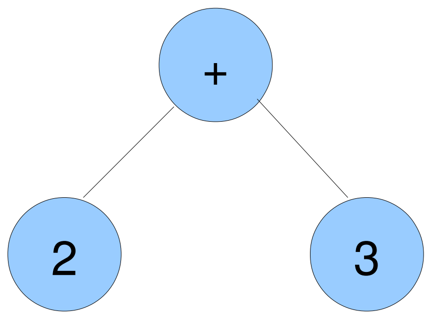


Arithmetic Trees

An arithmetic tree is a binary tree representing the structure of an arithmetic expression.

e.g.

2 + 3

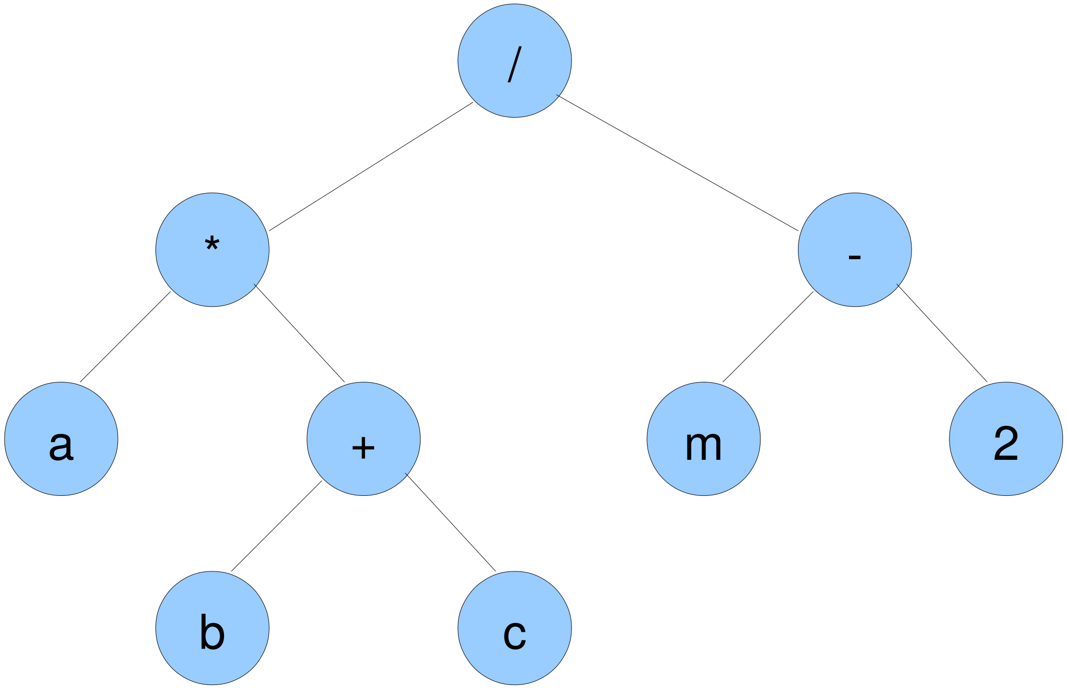


Arithmetic Trees

An arithmetic tree is a binary tree representing the structure of an arithmetic expression.

e.g.

a* (b+c)/(m-2)



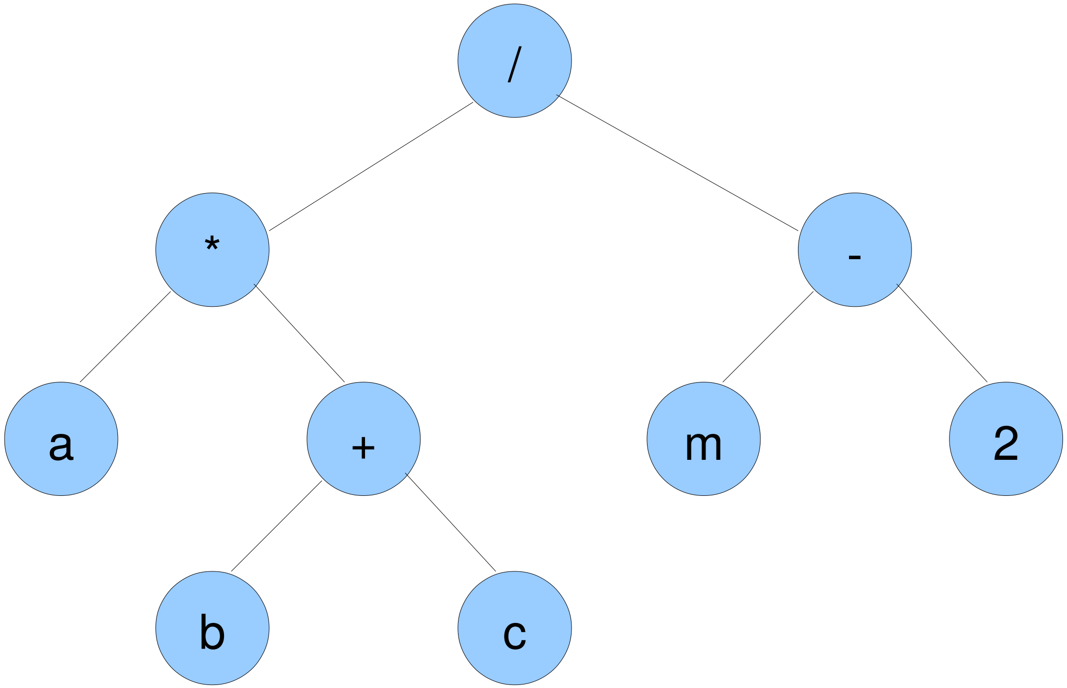
Arithmetic Trees: pre-order

Pre-order traversal:

e.g.

/ * a + b c-m 2

(pre-fix notation!)



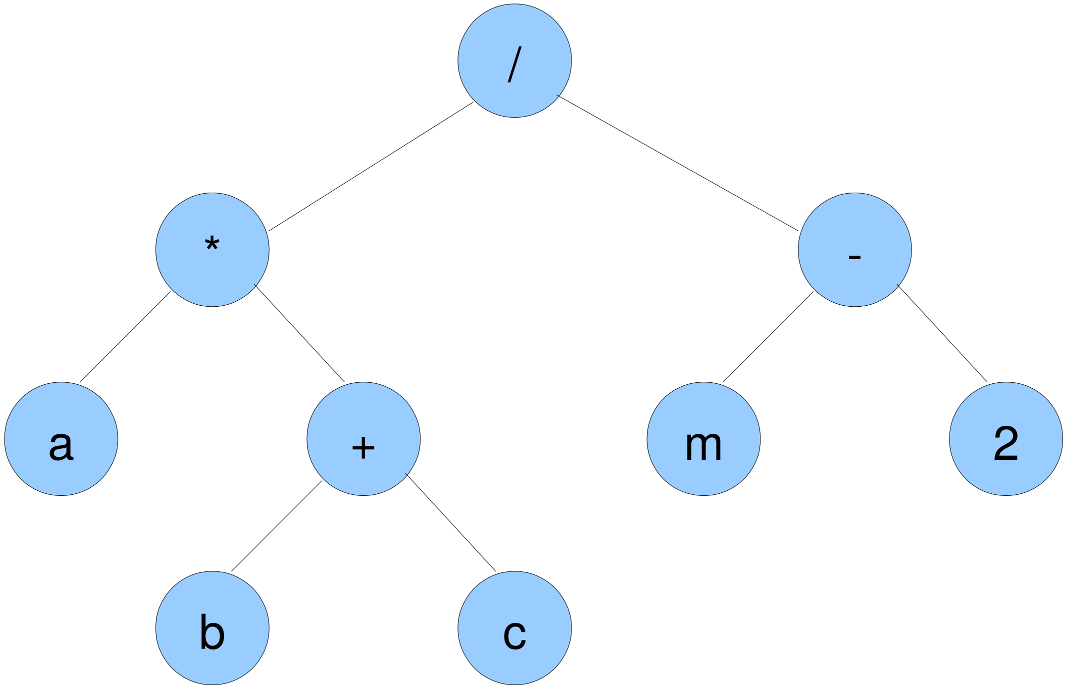
Arithmetic Trees: in order

In-order traversal:

e.g.

a*b+c/m-2

(in-fix notation!)



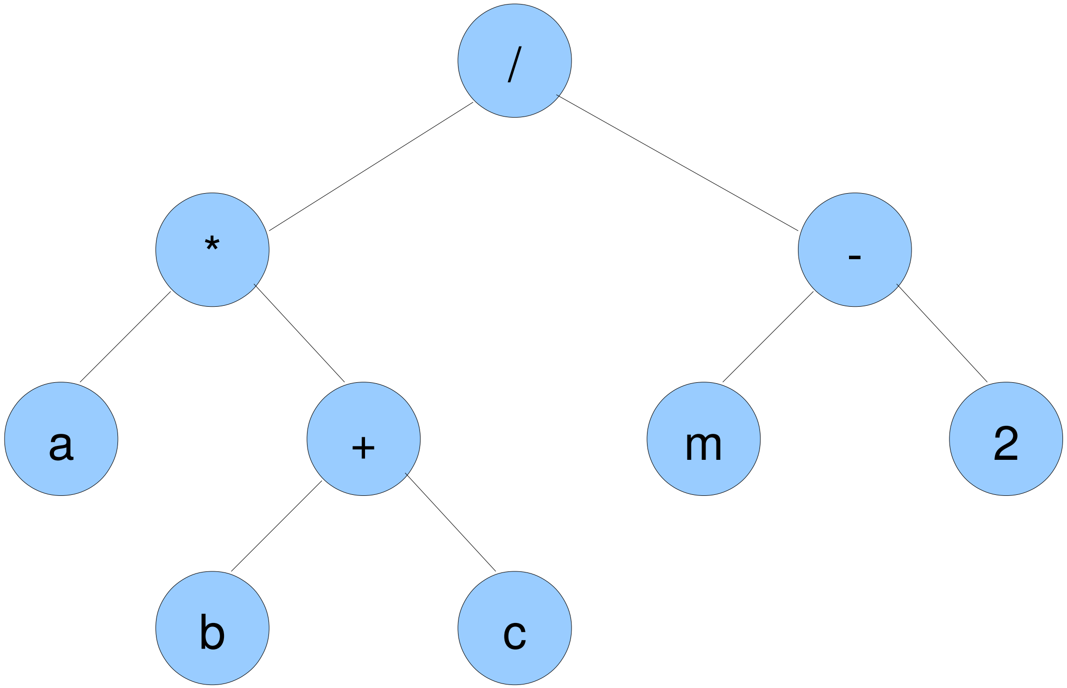
Arithmetic Trees: post-order

Post-order traversal:

e.g.

a b c + * m 2 - /

(post-fix notation!)



An algorithm: construct an arithmetic tree

1. Read in an expression that is ***already in post-fix notation.***
2. Tree *T1, *T2, *T; Stack S; *note that S is a stack of pointers to trees*
3. while (expression continues) {

   x = next item from the expression

   if (x is a number) { S.Push(new Tree(x, NULL, NULL)); }

   if (x is an operator) {

      T1 = S.Top(); S.Pop();

      T2 = S.Top(); S.Pop();

      S.Push(new Tree(x, T2, T1)); *note order of T2 and T1*

   }

   }
4. T = S.Top();

Te Kunenga ki Pūrehuroa

Notes

1. This is an algorithm, not a program. We already know it...

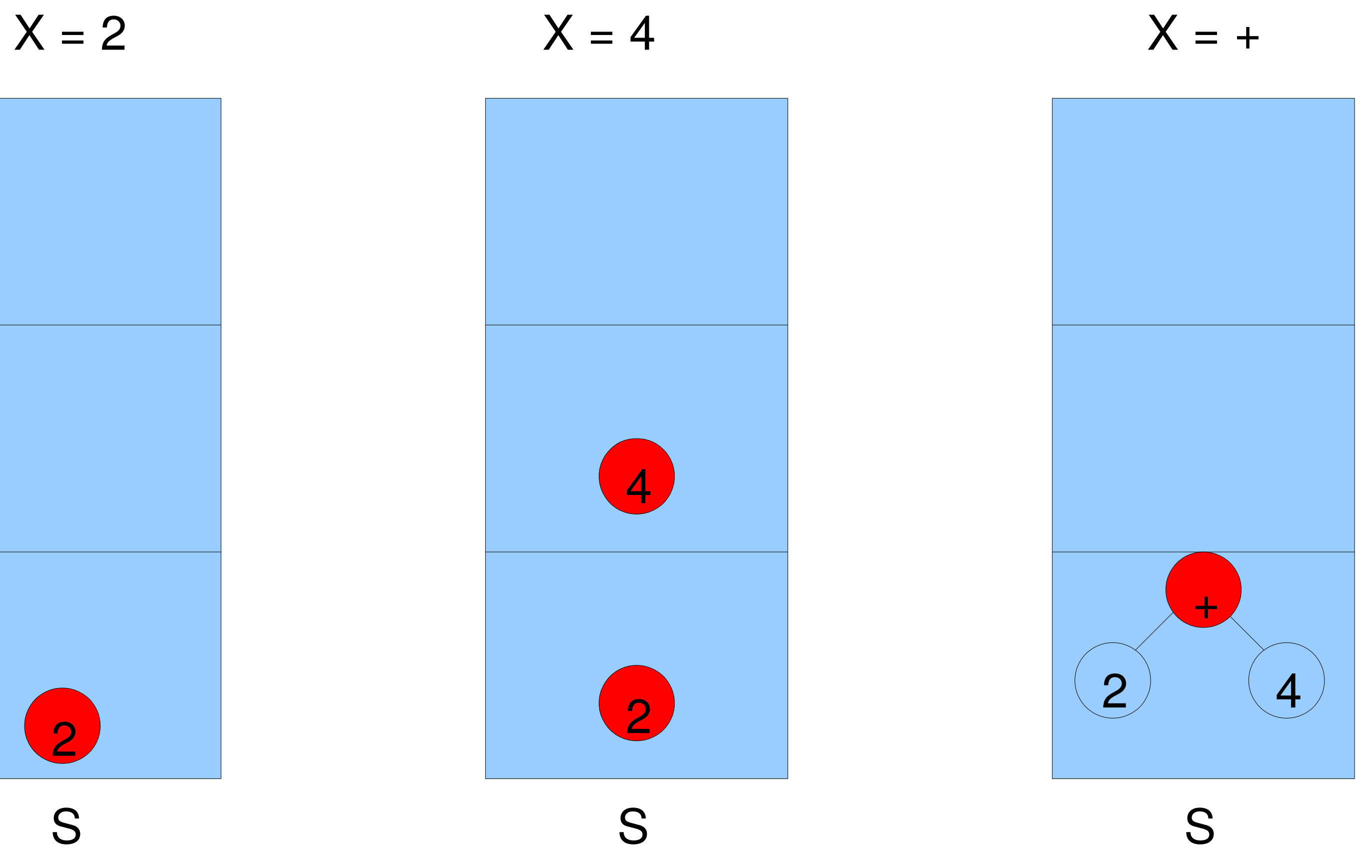
2. S is a stack of tree pointers, not elements.

3. Lets walk through an example...

Arithmetic tree example

The in-fix expression (2+4)*3 is converted to:

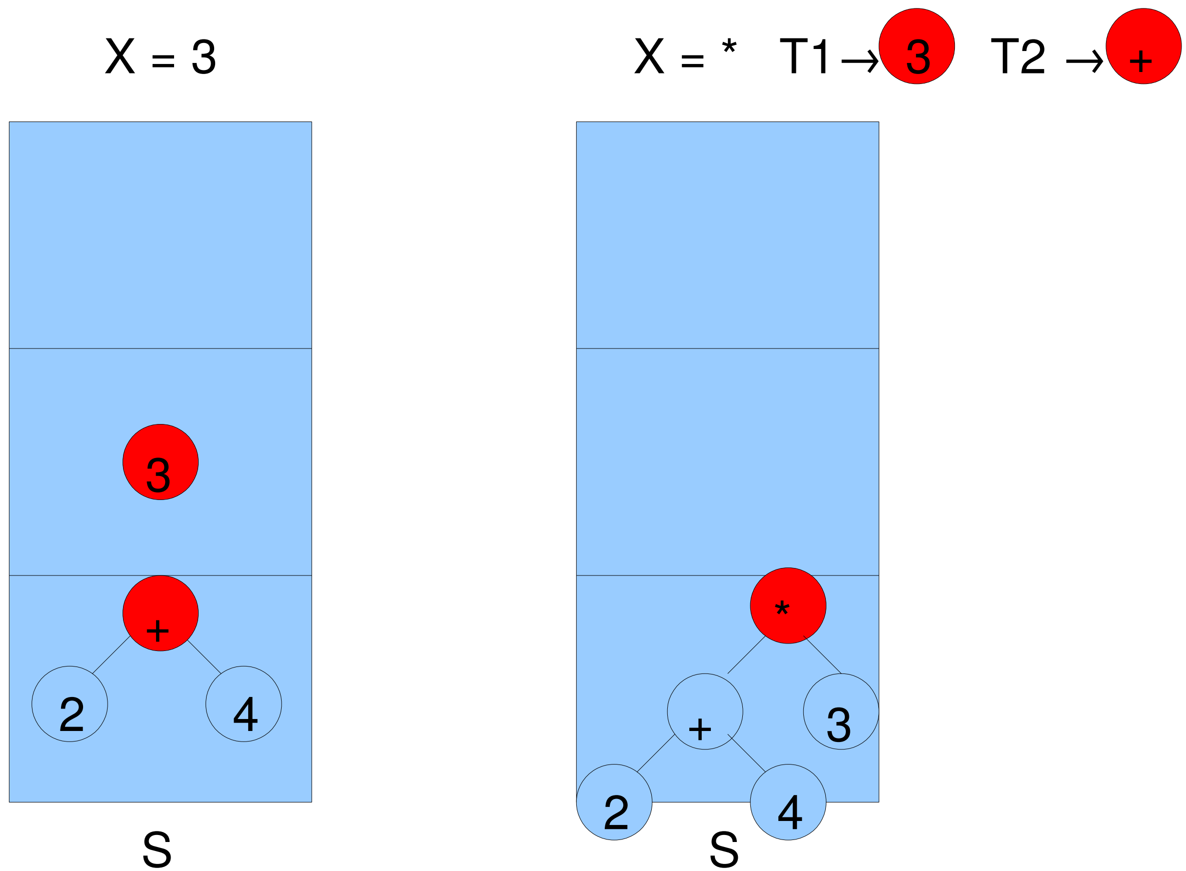
2 4 + 3 *   *(post-fix notation)*



Arithmetic tree example (cont.)

The in-fix expression (2+4)*3 is converted to:

2 4 + 3 *   *(post-fix notation)*



Massey University
COLLEGE OF SCIENCES

Challenge

1) Write a C++ function to print a binary tree by height and beautify it with symbols such as “___”, “\” and “/” like this:

```
          (20)
      _____|____
     /          \
  (30)         (40)
```

This is much easier to read, and it may be useful to help with debugging your codes.

Te Kunenga ki Pūrehuroa