02393 Programming in C++

02393 Programming in C++
Module 12: Trees

Lecturer:
Alceste Scalas

(Slides based on previous versions by Andrea Vandin, Alberto Lluch Lafuente, Sebastian Mödersheim)

24 November 2020

Lecture Plan

#	Date	Topic	Book chapter *
1	01.09	Introduction	
2	08.09	Basic C++	1
3	15.09	Data Types	2
4	22.09		
5	29.09	Libraries and Interfaces	3
6	06.10	Classes and Objects	4.1, 4.2 and 9.1, 9.2
		Autumn break	
7	20.10	Templates	4.1, 11.1
8	27.10	LAB DAY	Old exams
9	03.11	Inheritance	14.3, 14.4, 14.5
10	10.11	Recursive Programming	5
11	17.11	Linked Lists	10.5
12	24.11	Trees	13
13	01.12	Summary & Exam Preparation	
	07.12	Exam	

* Recall that the book uses sometimes ad-hoc libraries that are slightly different with respect to the standard libraries (e.g., strings and vectors).

Summary

- Recursive programming... when?
  - Problem is recursively/inductively defined (e.g. Fibonacci)
  - Problem on a recursive data structure
- Examples of recursive data structures:
  - Linked lists: single- and doubly-linked lists
  - Sets, multi-sets
  - **Trees** (today!)
- Difference between **specification** vs. **implementation**:
  - **Abstract Data Type:** a type being specified, e.g. a class `Set`
  - **Concrete Data Structure:** how we implement it, e.g. `Set` implemented with linked lists, or binary trees, or...

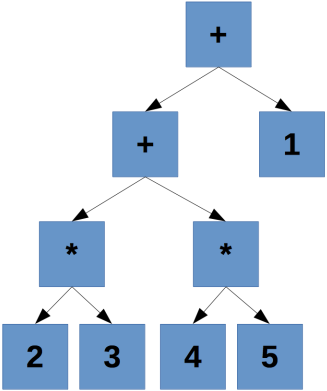
Using Trees to Represent Expressions

Representing the expression: $\Big((2*3)+(4*5)\Big)+1$

Using Trees to Represent Expressions

Representing the expression: $\Big((2 * 3) + (4 * 5)\Big) + 1$

Syntax Tree

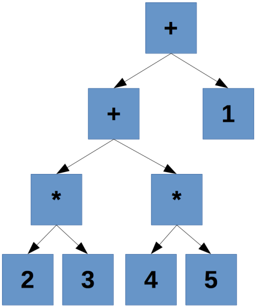


Using Trees to Represent Expressions

We develop a class `ArithmeticSyntaxTree` to represent arithmetic expressions

To represent an expression like $\big((2*3)+(4*5)\big)+1$ we need:

- **constants** (i.e., numbers)
- the **sum** of two arithmetic expressions
- the **product** of two arithmetic expressions



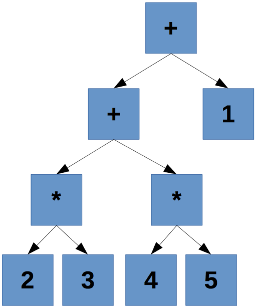
This is an example of **binary tree**

Using Trees to Represent Expressions

We develop a class `ArithmeticSyntaxTree` to represent arithmetic expressions

To represent an expression like $\Big((2 * 3) + (4 * 5)\Big) + 1$ we need:

- **constants** (i.e., numbers)
- the **sum** of two arithmetic expressions
- the **product** of two arithmetic expressions



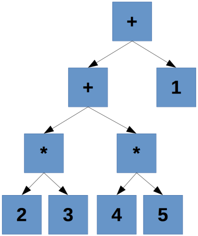
This is an example of **binary tree**

Today we address:

- the **size** of a tree
- the **height** of a tree
- the **number of leaves** of a tree
- **traversing a tree** in different ways

Arithmetic Syntax Tree

An Arithmetic Syntax Tree is made of two kind of **nodes**:



- **internal nodes** with two descendant trees (for operators)
- **leaf nodes** without descendant trees (for constants)

Some more terminology:

- **root node**: the topmost node of a tree
- **size of a tree**: the number of nodes it has
- **height of a tree**: length of the longest path from root to leaf

Representing the Nodes of a Tree

Possible implementation of tree nodes

```
enum NodeType { Const, Add, Mult };

struct Node {
    NodeType type;
    int value;      // Used if type == Const
    Tree *left;     // Used if type != Const
    Tree *right;    // Used if type != Const
}
```

A tree could be then just a pointer to a `Node`, as we did for lists

Class of Trees

Another possible recursive definition of trees

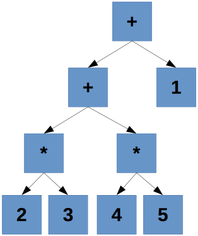
```
enum NodeType { Const, Add, Mult };

class Tree {
public:
  // Some methods...
private:
  NodeType type;
  int value;      // Used if type == Const
  Tree *left;     // Used if type != Const
  Tree *right;    // Used if type != Const
}
```

We will use this representation in the examples

Methods

Most methods we need can be easily implemented using recursion



Consider, e.g., the **size of a tree**. A recursive formulation:

- **size of a leaf node** $= 1$
- **size of an internal node** $= 1 +$ **sizes** of its descendant trees

Non-recursive implementations are possible, but more complicated!

02393 Programming in C++