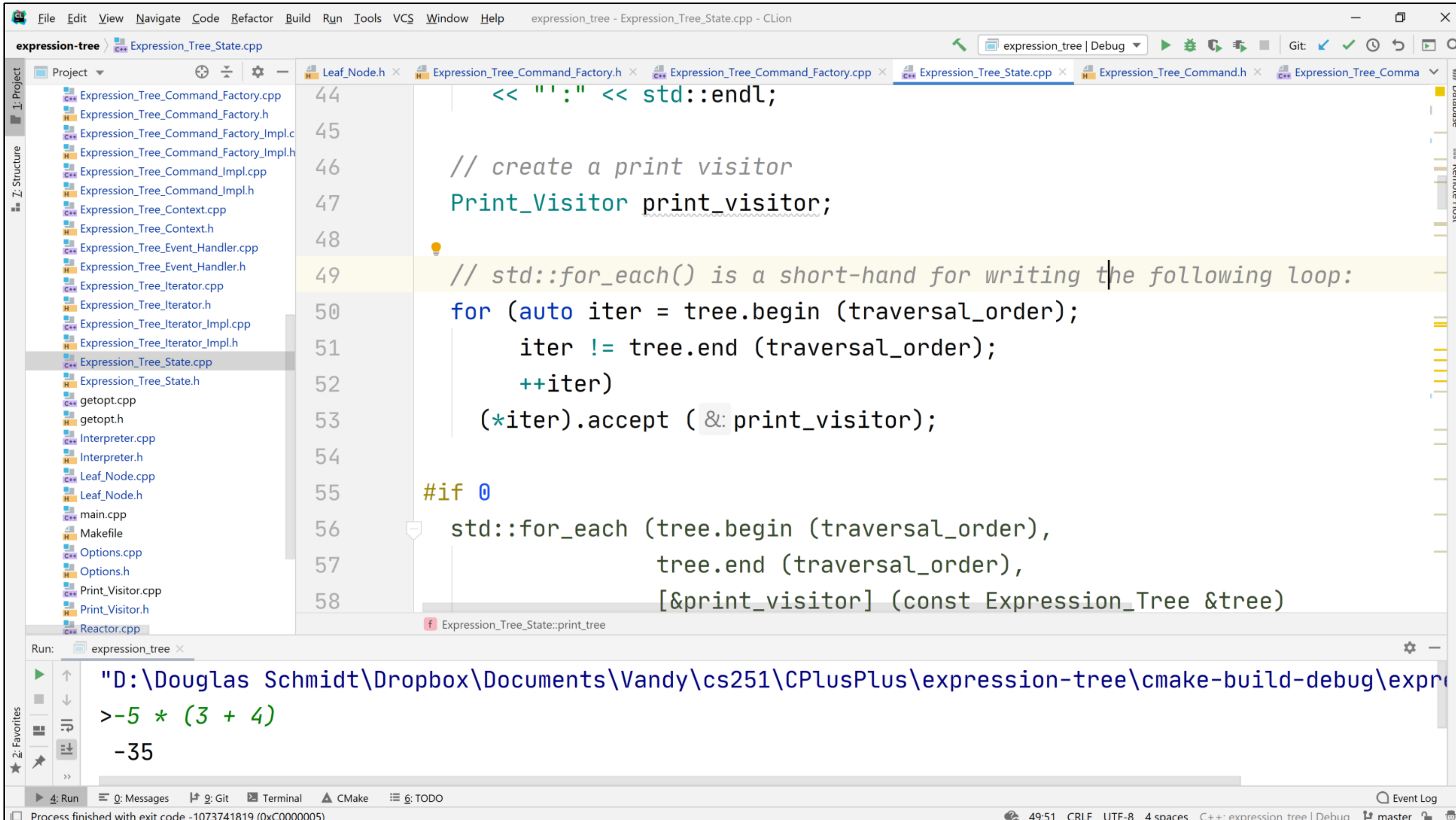
Overview of the Expression Tree Processing App Case Study (Part 1)

Douglas C. Schmidt

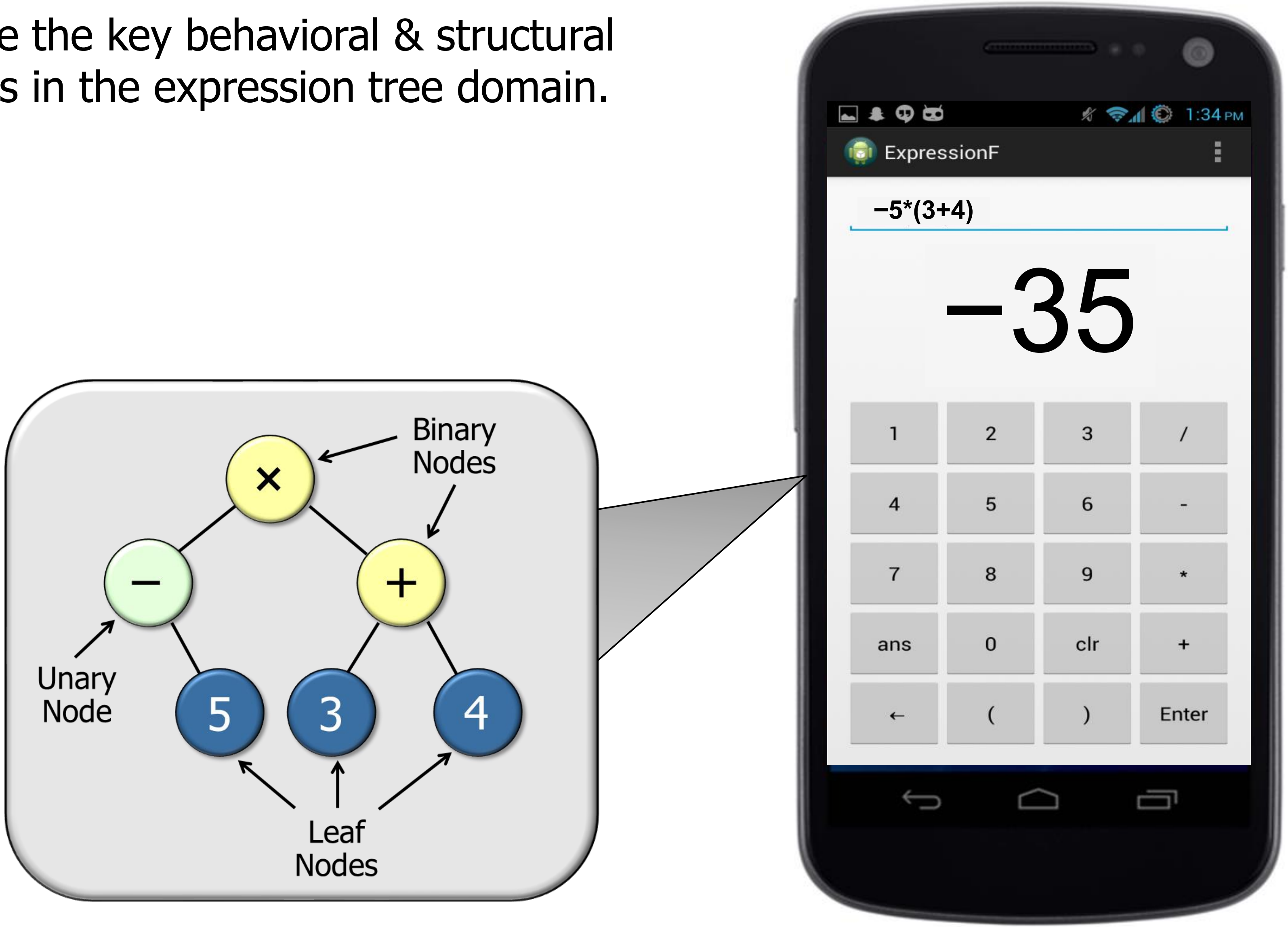
Learning Objectives in This Lesson

- Understand the goals of the object-oriented (OO) expression tree case study.



Learning Objectives in This Lesson

- Understand the goals of the object-oriented (OO) expression tree case study.
- Recognize the key behavioral & structural properties in the expression tree domain.



Douglas C. Schmidt

Lesson Introduction

Lesson Introduction

- While patterns can be discussed abstractly, effective design & programming practices are not learned best by generalities.

"Sitting & thinking" is not sufficient…

Lesson Introduction

- While patterns can be discussed abstractly, effective design & programming practices are not learned best by generalities.
- Instead, it’s usually better to see how patterns can help improve nontrivial programs

Lesson Introduction

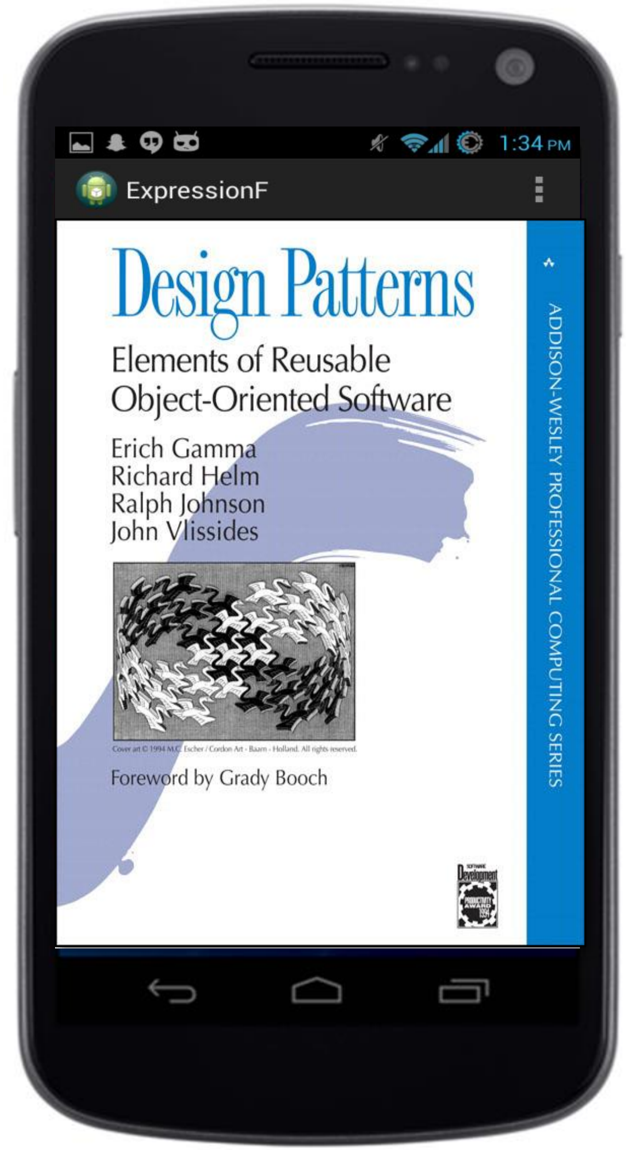
- While patterns can be discussed abstractly, effective design & programming practices are not learned best by generalities.
- Instead, it's usually better to see how patterns can help improve nontrivial programs, e.g.,
  - Easier to write & read;
  - Easier to maintain & modify;
  - More efficient & robust.

Lesson Introduction

- While patterns can be discussed abstractly, effective design & programming practices are not learned best by generalities.
- Instead, it's usually better to see how patterns can help improve nontrivial programs.
- This lesson describes a realistic—yet tractable—expression tree processing app we'll use as a case study throughout the course.

Lesson Introduction

- While patterns can be discussed abstractly, effective design & programming practices are not learned best by generalities.
- Instead, it's usually better to see how patterns can help improve nontrivial programs.
- This lesson describes a realistic—yet tractable—expression tree processing app we'll use as a case study throughout the course.
  - This case study applies many "Gang of Four" (GoF) patterns using C++ & STL.



See en.wikipedia.org/wiki/Design_Patterns

Douglas C. Schmidt

Expression Tree Processing App Case Study Goals

Expression Tree Processing App Case Study Goals

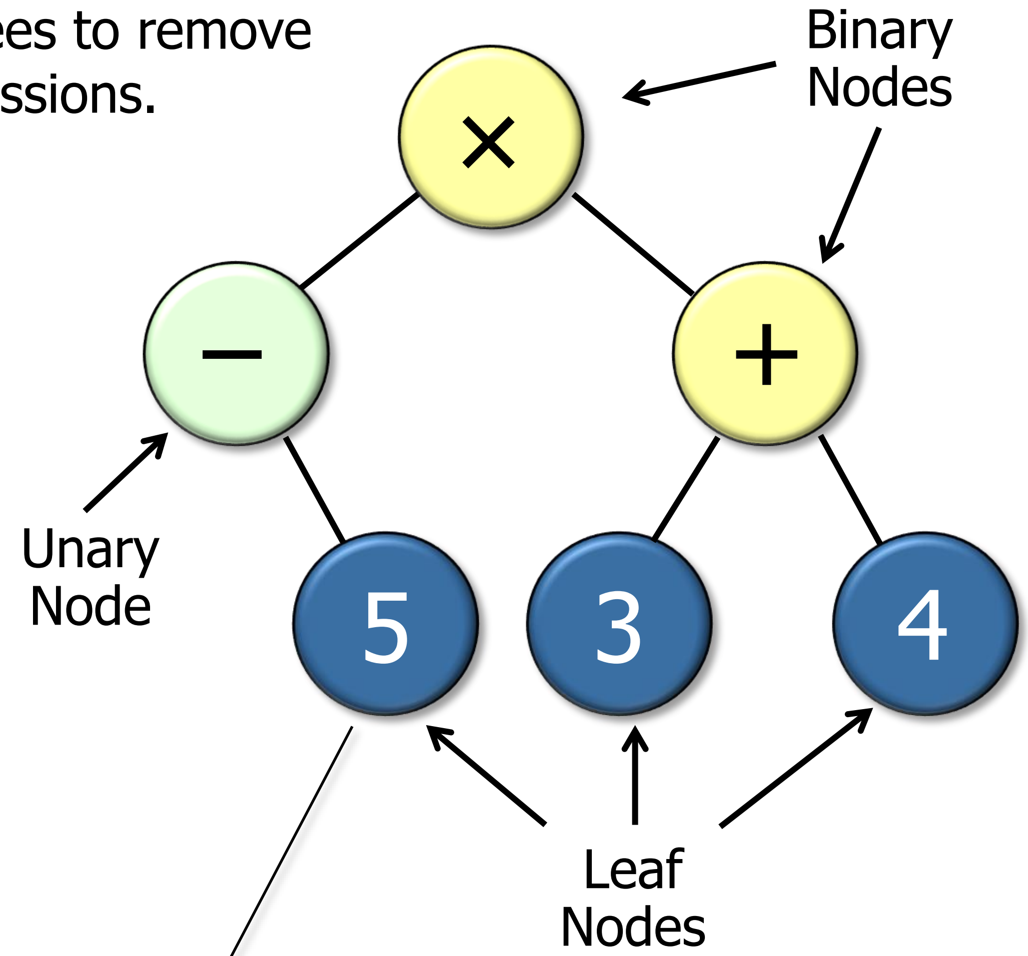
- Develop an OO expression tree processing app using *patterns* & *frameworks.*

Design Problem	Pattern
Non-extensible & error-prone designs	Composite
Minimizing impact of variability	Bridge
Inflexible expression input processing	Interpreter
Inflexible interpreter output	Builder
Scattered request implementations	Command
Inflexible creation of variabilities	Factory Method
Inflexible expression tree traversal	Iterator
Obtrusive behavior changes	Strategy
Non-extensible tree operations	Visitor
Incorrect user request ordering	State
Non-extensible operating modes	Template Method
Minimizing global variable liabilities	Singleton

Naturally, these patterns apply to more than expression tree processing apps!

Expression Tree Processing App Case Study Goals

- This app uses expression trees to remove ambiguity in algebraic expressions.



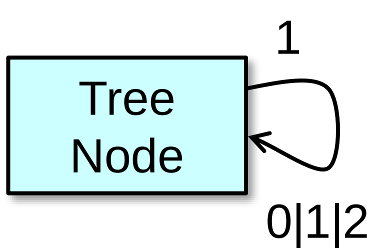
Each node of a binary expression tree has zero, one, or two children.

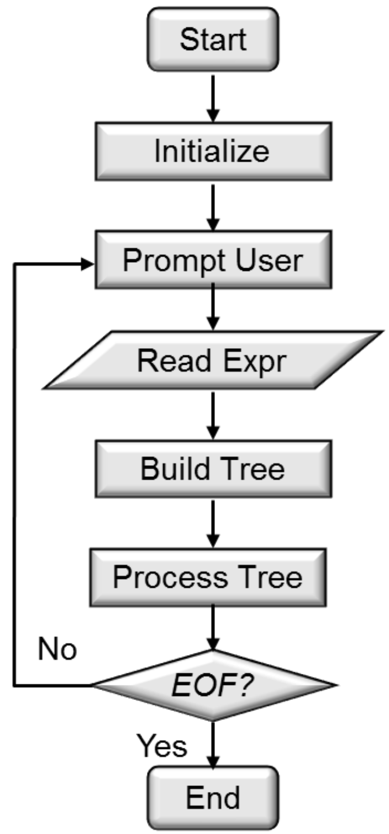
See en.wikipedia.org/wiki/Binary_expression_tree

Expression Tree Processing App Case Study Goals

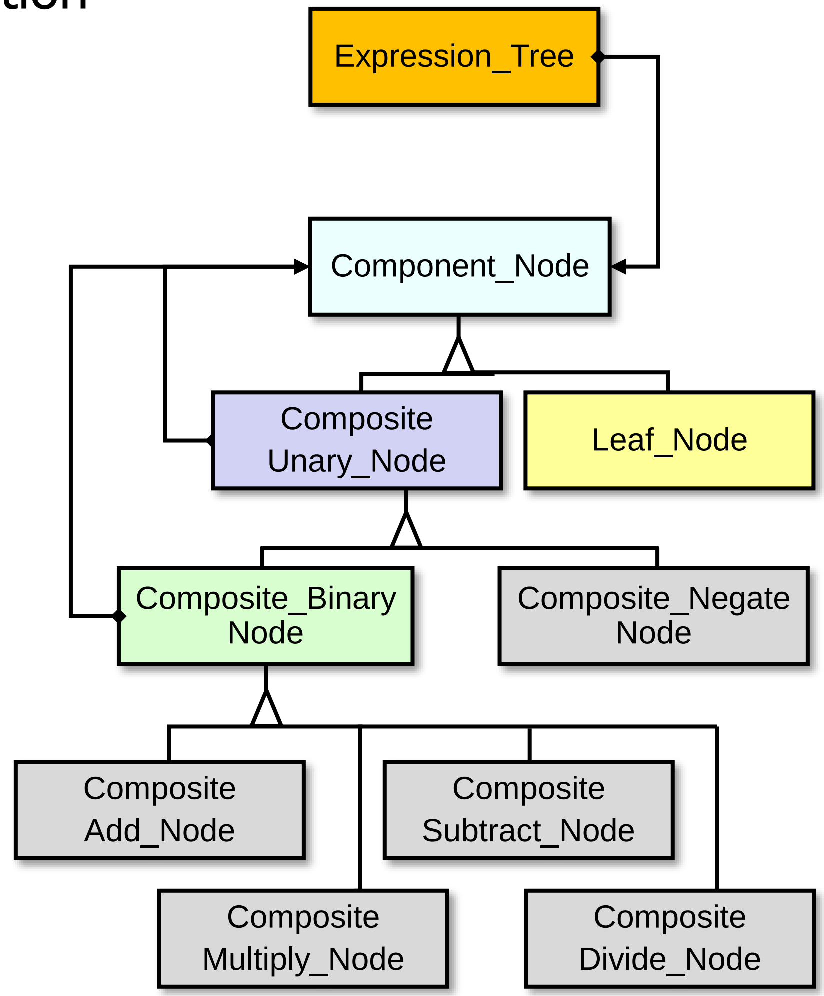
- Compare/contrast algorithmic decomposition & object-oriented (OO) approaches.

Despite decades of OO focus, algorithmic decomposition is still surprisingly common.





Algorithmic Decomposition



Pattern- & Object-Oriented Decomposition

See www.drdobbs.com/windows/software-complexity-bringing-order-to-ch/199901062

Expression Tree Processing App Case Study Goals

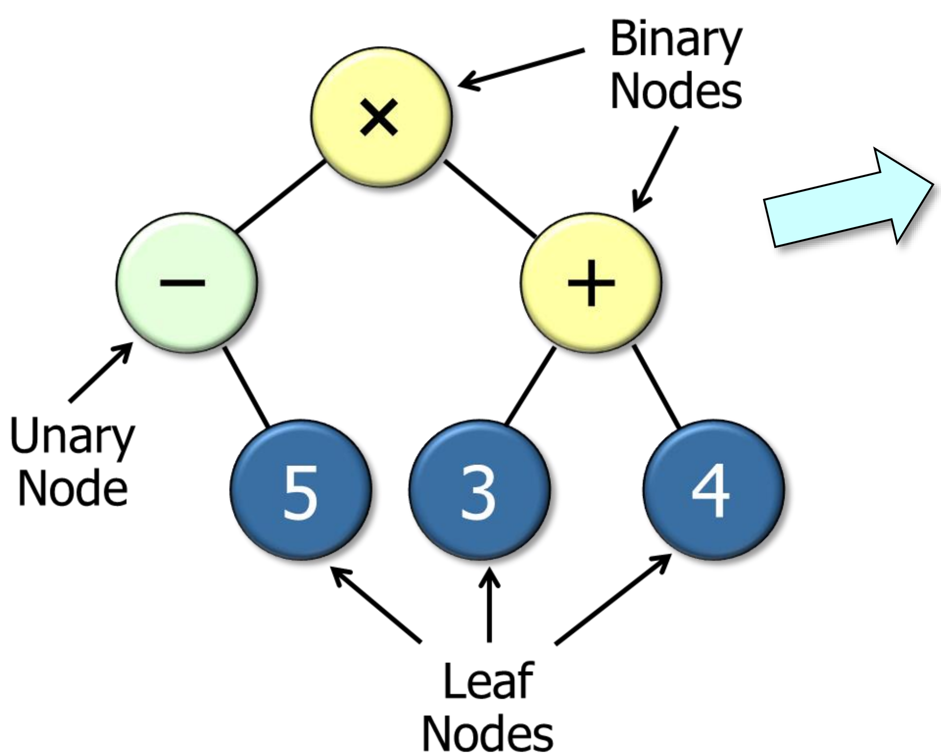
- Demonstrate *scope, commonality, & variability* (SCV) analysis as a means to achieve systematic software reuse.

- *Identify common elements of a domain & define stable interfaces.*
- *Identify variable elements of a domain & define stable interfaces.*

See www.dre.vanderbilt.edu/~schmidt/PDF/Commonality_Variability.pdf

Expression Tree Processing App Case Study Goals

- Demonstrate *scope, commonality, & variability* (SCV) analysis as a means to achieve systematic software reuse.
  - Apply SCV in the context of the expression tree processing app.



Expression Tree Processing App Case Study Goals

- Show how to implement pattern-oriented OO frameworks & generic programs in C++.

```
Expression_Tree tree = ...;
Visitor print_visitor = ...;

for (auto iter = tree.begin
                   (order);
     iter != tree.end
                   (order);
     ++iter)
   (*iter).accept
     (print_visitor);

for_each(tree.begin(order),
         tree.end(order),
         [&print_visitor](auto
                          tree) {
            tree.accept
              (print_visitor);
         });
```

Expression Tree Processing App Case Study Goals

- Show how to implement pattern-oriented OO frameworks & generic programs in C++.

```
Expression_Tree tree = ...;
Visitor print_visitor = ...;

for (auto iter = tree.begin
                   (order);
     iter != tree.end
                   (order);
     ++iter)
   (*iter).accept
     (print_visitor);
```

C++-style GoF *Iterator* pattern with for loop

```
for_each(tree.begin(order),
         tree.end(order),
         [&print_visitor](auto
                          tree) {
            tree.accept
              (print_visitor);
         });
```

See en.wikipedia.org/wiki/Iterator_pattern

Expression Tree Processing App Case Study Goals

- Show how to implement pattern-oriented OO frameworks & generic programs in C++.

```
Expression_Tree tree = ...;
Visitor print_visitor = ...;

for (auto iter = tree.begin
                    (order);
     iter != tree.end
                    (order);
     ++iter)
   (*iter).accept
     (print_visitor);

for_each(tree.begin(order),
         tree.end(order),
         [&print_visitor](auto
                          tree) {
             tree.accept
               (print_visitor);
         });
```

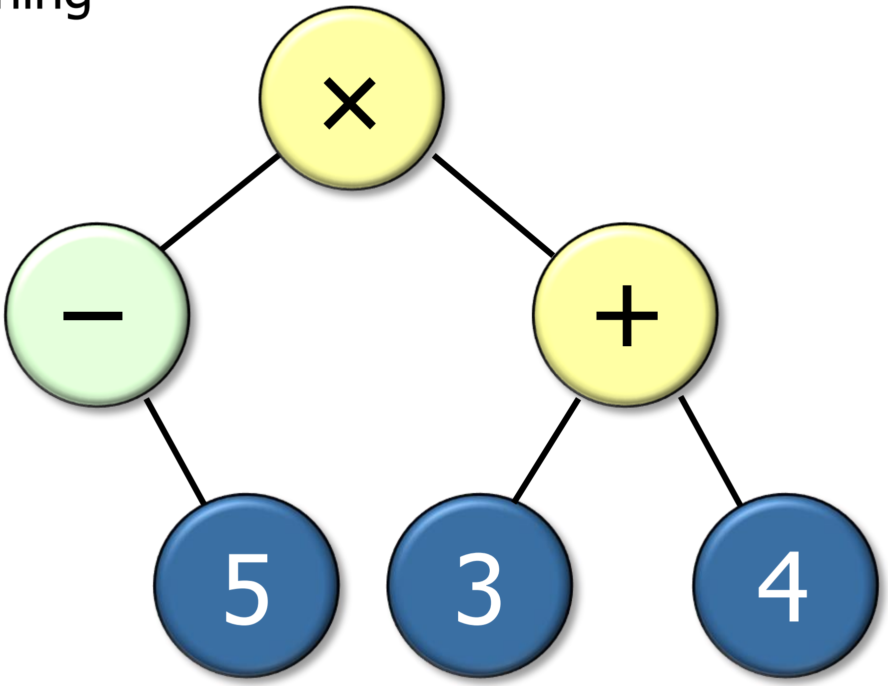
STL for_each() algorithm with C++ lambda function

Douglas C. Schmidt

Overview of the Expression Tree Processing Domain

Overview of Expression Tree Processing Domain

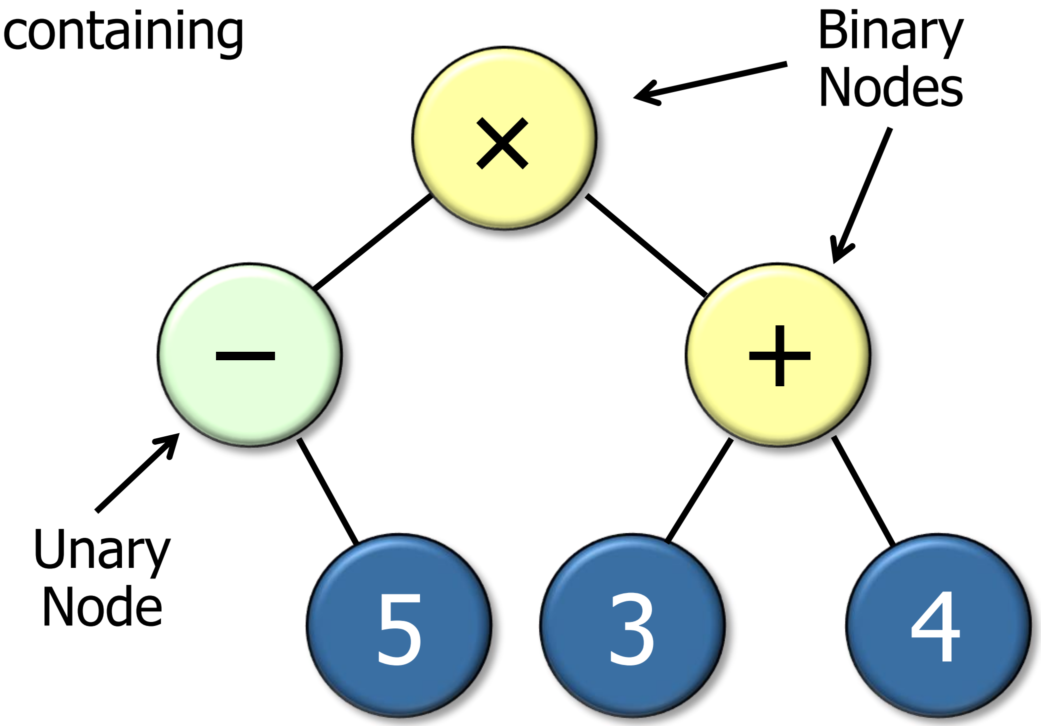
- Expression trees consist of nodes containing *operators* & *operands.*



See en.wikipedia.org/wiki/Binary_expression_tree for expression tree information.

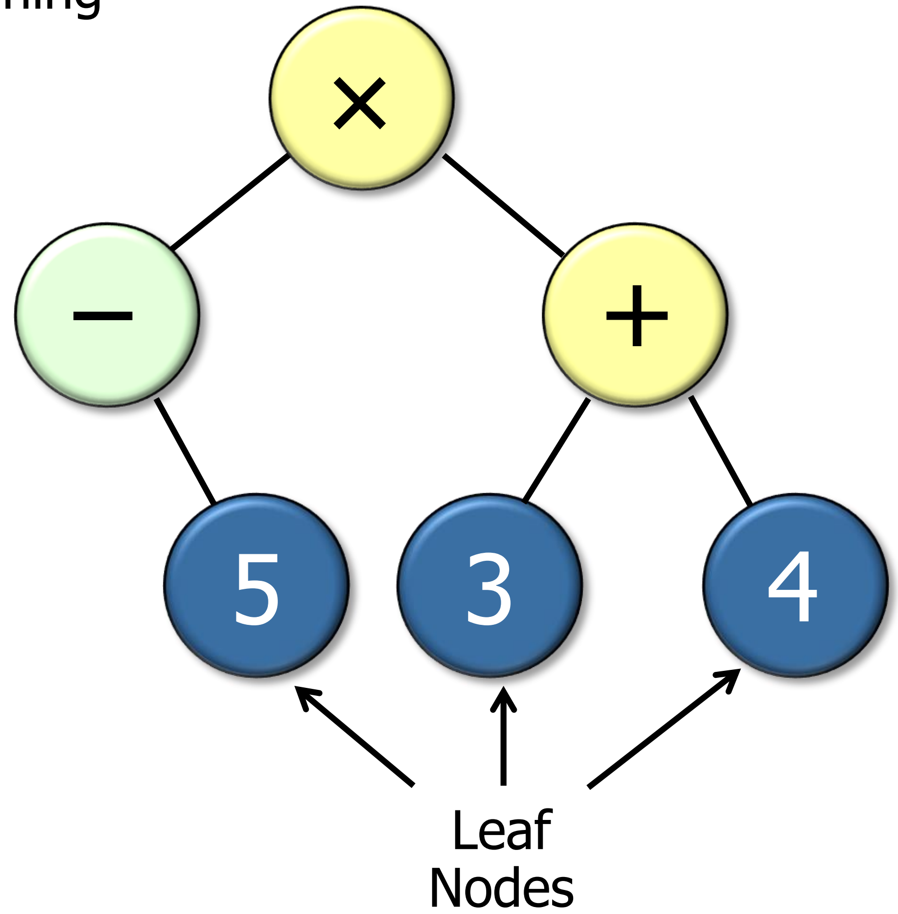
Overview of Expression Tree Processing Domain

- Expression trees consist of nodes containing *operators* & *operands.*
  - Operators are *interior nodes* in the tree, i.e.,
    - *Binary* & *unary nodes*



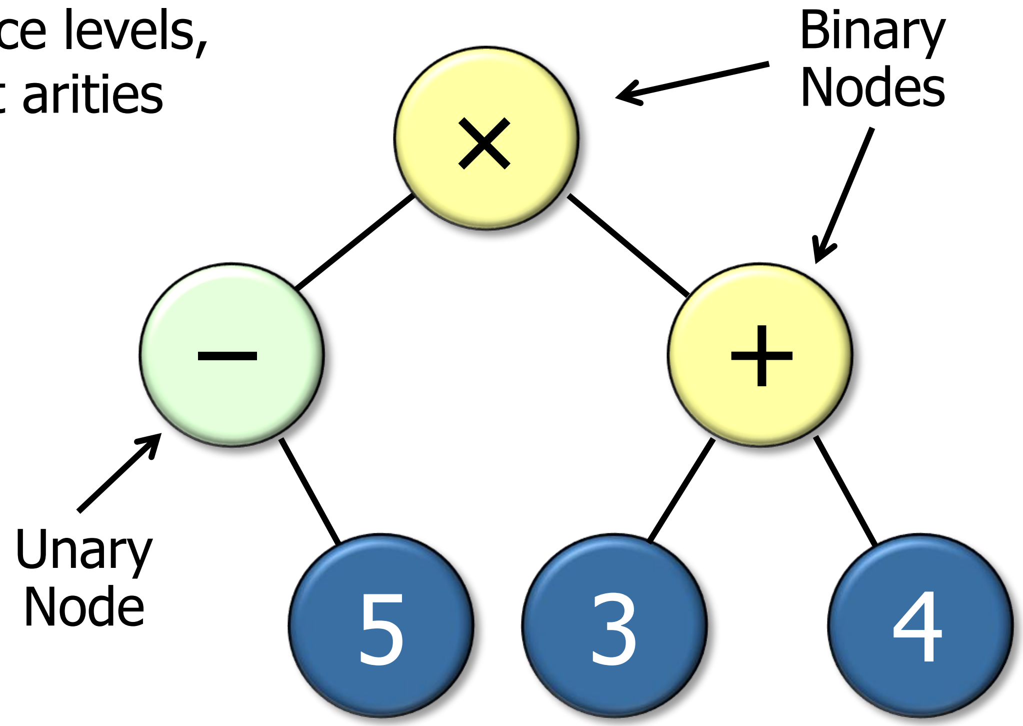
Overview of Expression Tree Processing Domain

- Expression trees consist of nodes containing *operators* & *operands.*
  - Operators are *interior nodes* in the tree
  - Operands are *exterior nodes* in the tree, i.e.,
    - *Leaf nodes*



Overview of Expression Tree Processing Domain

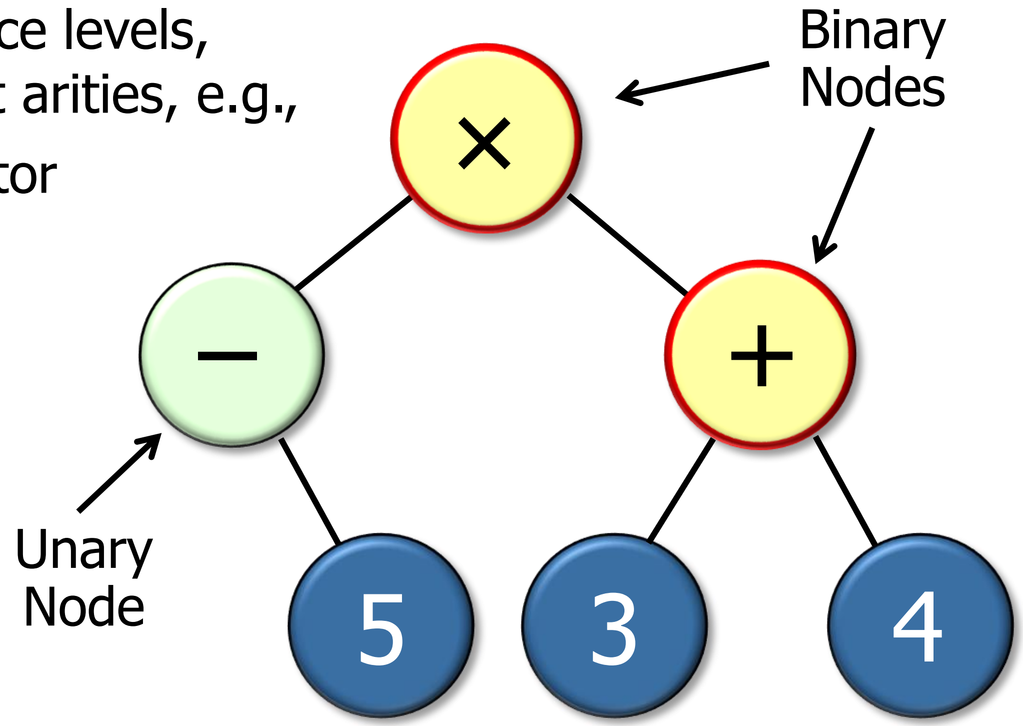
- Operators have different precedence levels, different associativities, & different arities



See en.wikipedia.org/wiki/Operator_associativity & en.wikipedia.org/wiki/Arity

Overview of Expression Tree Processing Domain

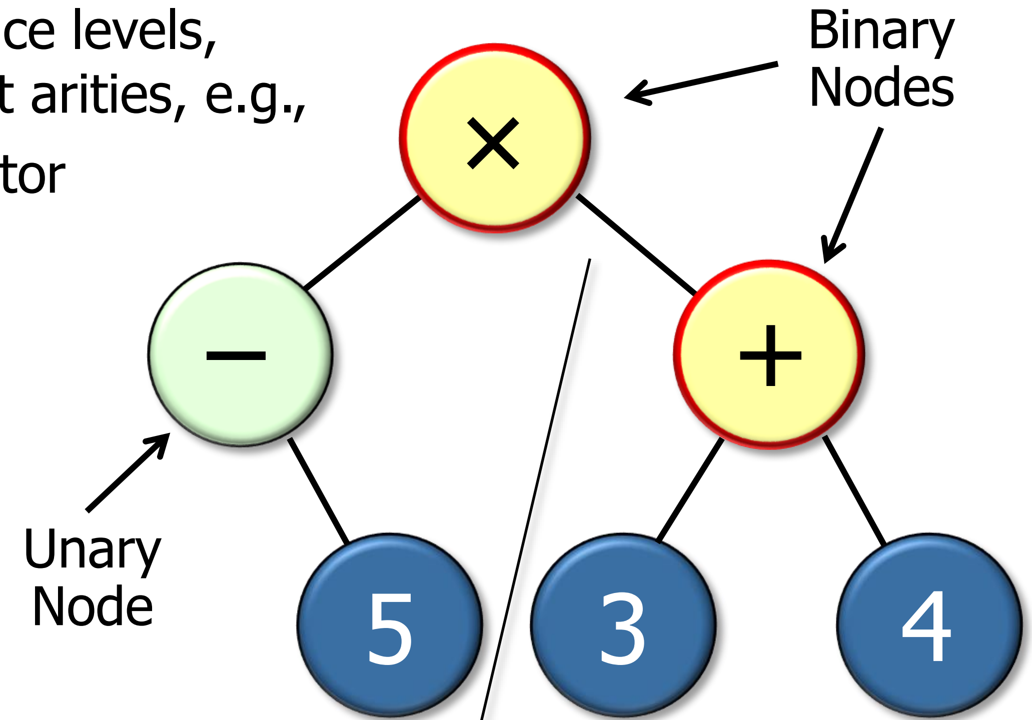
- Operators have different precedence levels, different associativities, & different arities, e.g.,
  - Precedence defines which operator to perform first to evaluate a mathematical expression.
    - Multiplication takes precedence over addition



See en.wikipedia.org/wiki/Order_of_operations

Overview of Expression Tree Processing Domain

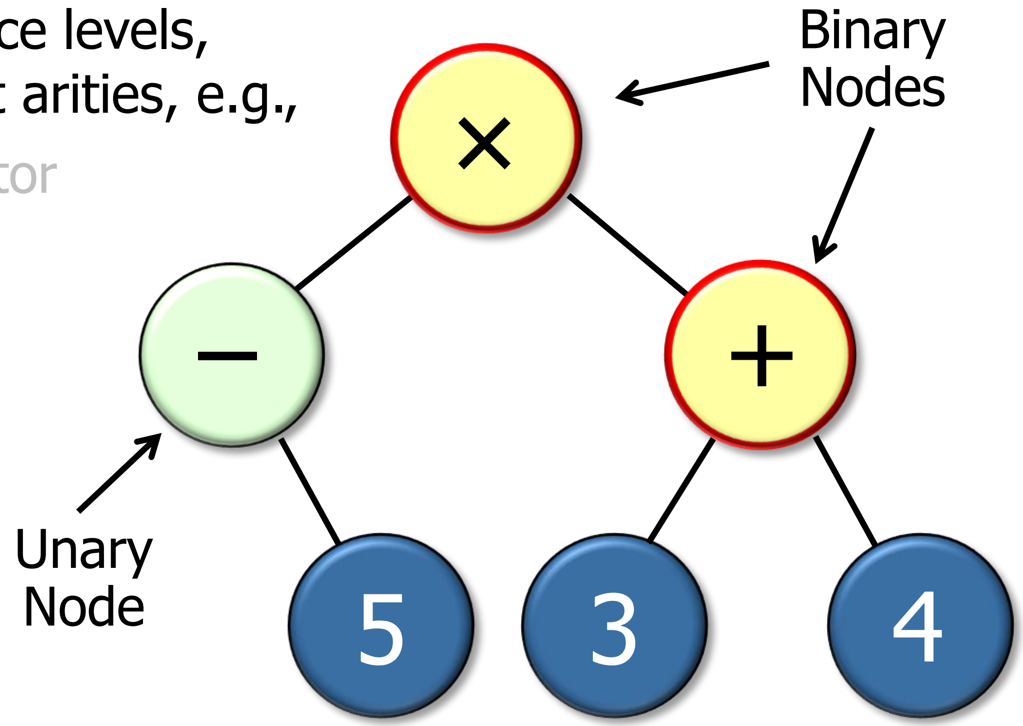
- Operators have different precedence levels, different associativities, & different arities, e.g.,
  - Precedence defines which operator to perform first to evaluate a mathematical expression.
    - Multiplication takes precedence over addition
    - Operator locations in a tree unambiguously designate precedence



*e.g., 3 + 4 is performed before –5 * 7.*

Overview of Expression Tree Processing Domain

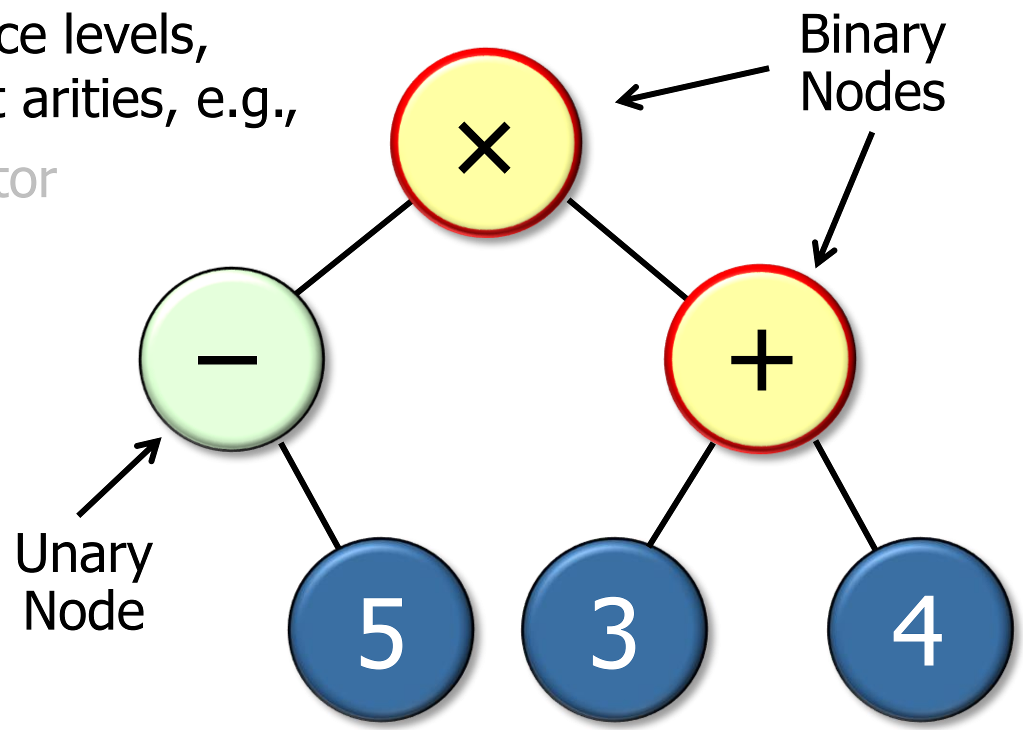
- Operators have different precedence levels, different associativities, & different arities, e.g.,
  - Precedence defines which operator to perform first to evaluate a mathematical expression.
  - Associativity determines how operators of the same level of precedence are grouped in the absence of parentheses.
    - 5 + 3 – 4 == (5 + 3) – 4



See en.wikipedia.org/wiki/Operator_associativity

Overview of Expression Tree Processing Domain

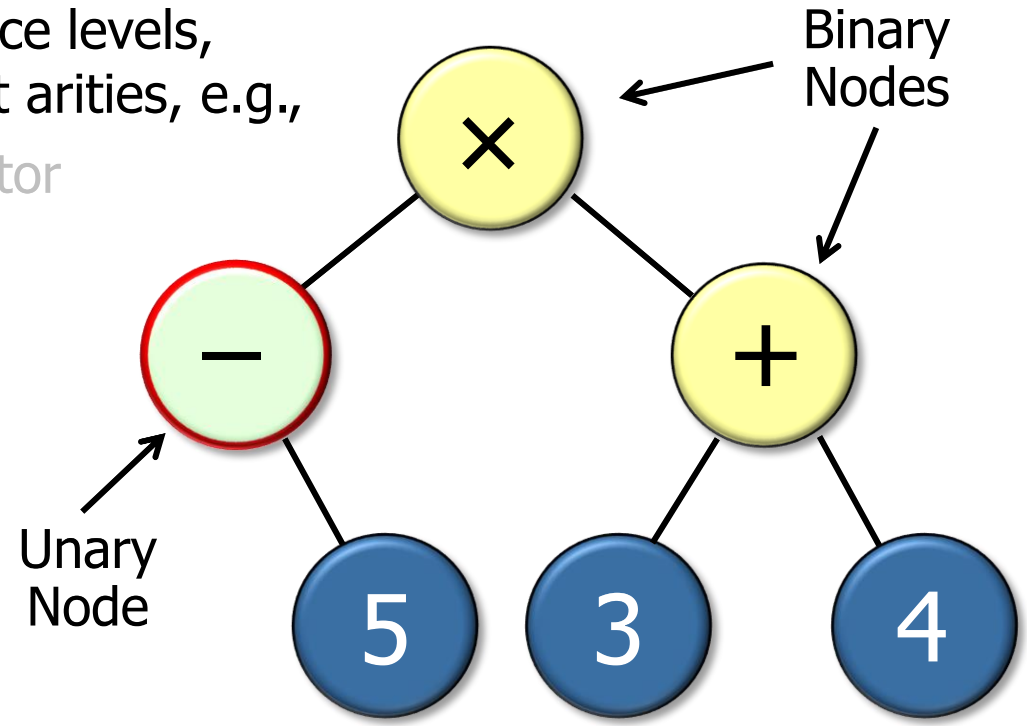
- Operators have different precedence levels, different associativities, & different arities, e.g.,
  - Precedence defines which operator to perform first to evaluate a mathematical expression.
  - Associativity determines how operators of the same level of precedence are grouped in the absence of parentheses.
  - Arity defines the number of operands an operator takes.
    - Multiplication & addition operators have two arguments (arity == 2)



See en.wikipedia.org/wiki/Arity

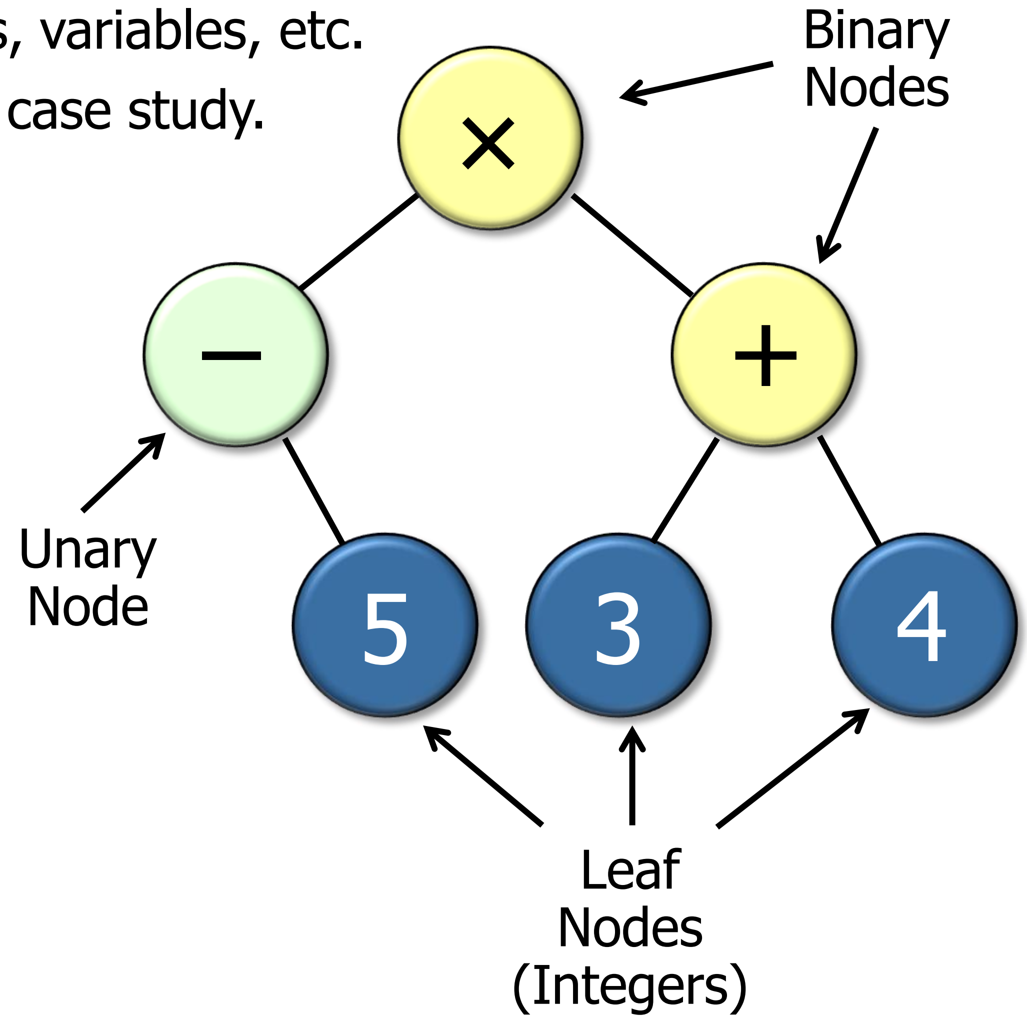
Overview of Expression Tree Processing Domain

- Operators have different precedence levels, different associativities, & different arities, e.g.,
  - Precedence defines which operator to perform first to evaluate a mathematical expression.
  - Associativity determines how operators of the same level of precedence are grouped in the absence of parentheses.
  - Arity defines the number of operands an operator takes.
    - Multiplication & addition operators have two arguments (arity == 2)
    - The unary minus operator has one argument (arity == 1)



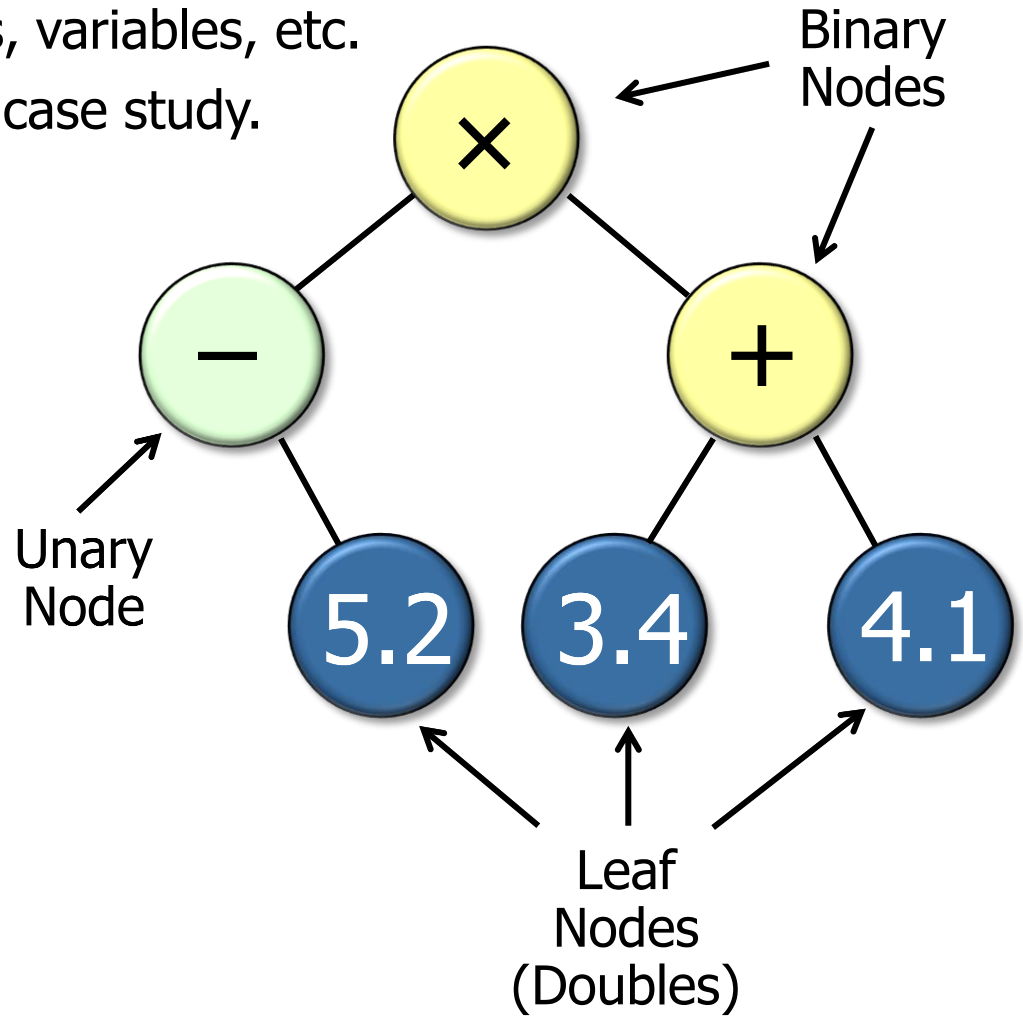
Overview of Expression Tree Processing Domain

- Operands can be integers, doubles, variables, etc.
  - We'll just handle integers in this case study.



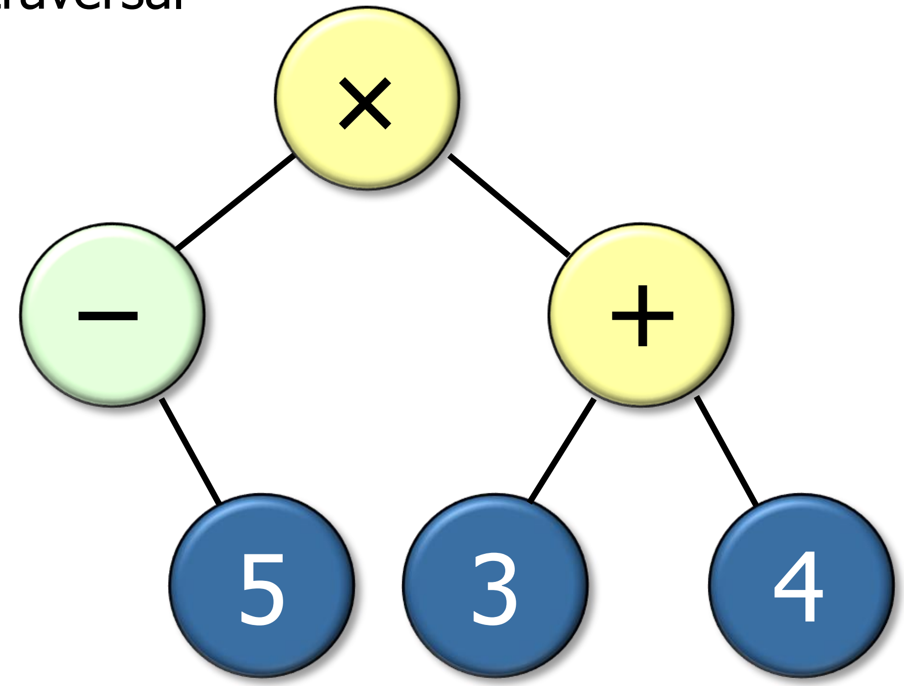
Overview of Expression Tree Processing Domain

- Operands can be integers, doubles, variables, etc.
  - We'll just handle integers in this case study.
    - It’s easy to extend the app to handle other types.



Overview of Expression Tree Processing Domain

- Trees may be "evaluated" via different traversal orders, e.g.,
  - "In-order traversal" = **`-5×(3+4)`**
  - "Pre-order traversal" = **`×-5+34`**
  - "Post-order traversal" = **`5-34+×`**
  - "Level-order traversal" = **`×-+534`**



See en.wikipedia.org/wiki/Binary_expression_tree#Traversal for more information.