高级数据结构上机课

- 双周周二 16:00~18:00，软件学院 3-101
- 负责：董明凯（dmkaplony@sjtu.edu.cn)
- 内容：
  - Mini Basic（30%）
  - LSM K/V Store（40%）
  - Graph（15%）
  - Balanced Search Tree（15%）
- 助教：江学强、张孝东

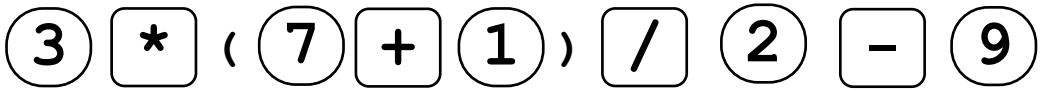
时间安排

Week 1	
Week 2	Project 1: MiniBasic
Week 3	
Week 4	Q/A
Week 5	
Week 6	Q/A + Graph
Week 7	
Week 8	DDL: Project 1 （答辩） Project 2: LSM Tree
Week 9	
Week 10	Q/A + Tree
Week 11	
Week 12	Q/A
Week 13	
Week 14	Q/A
Week 15	
Week 16	DDL: Project 2 LSM 答辩

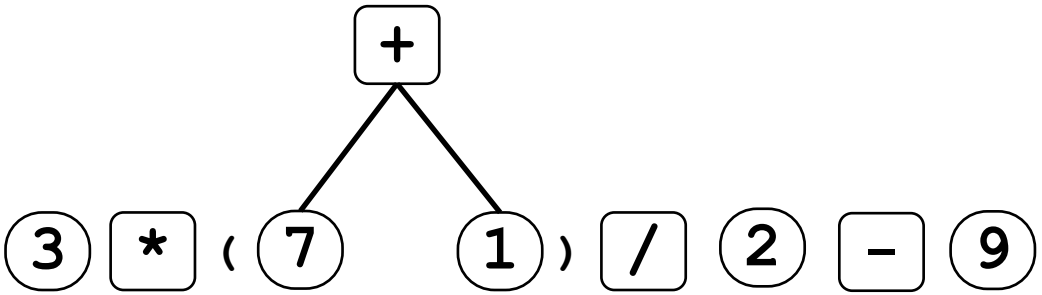
Expression Trees & Parsing & Evaluation

Most materials from Eric Roberts's lectures

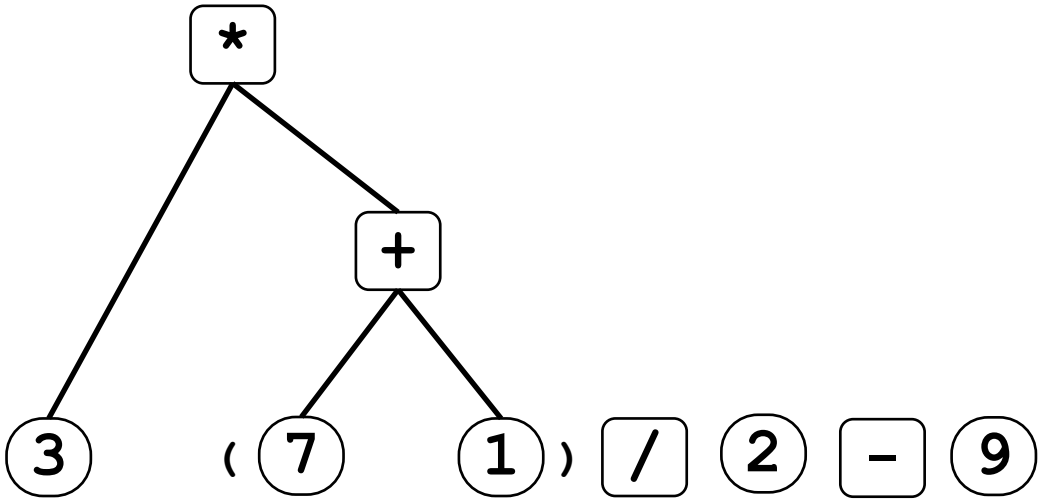
3 * (7 + 1) / 2 - 9



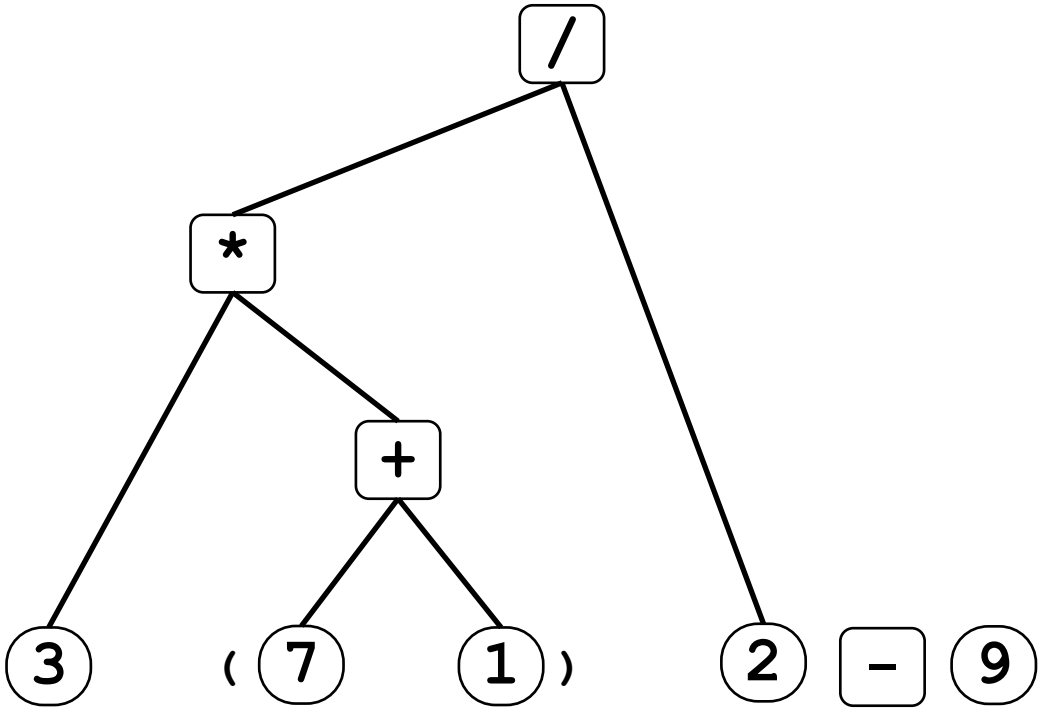
3 * (7 + 1) / 2 - 9



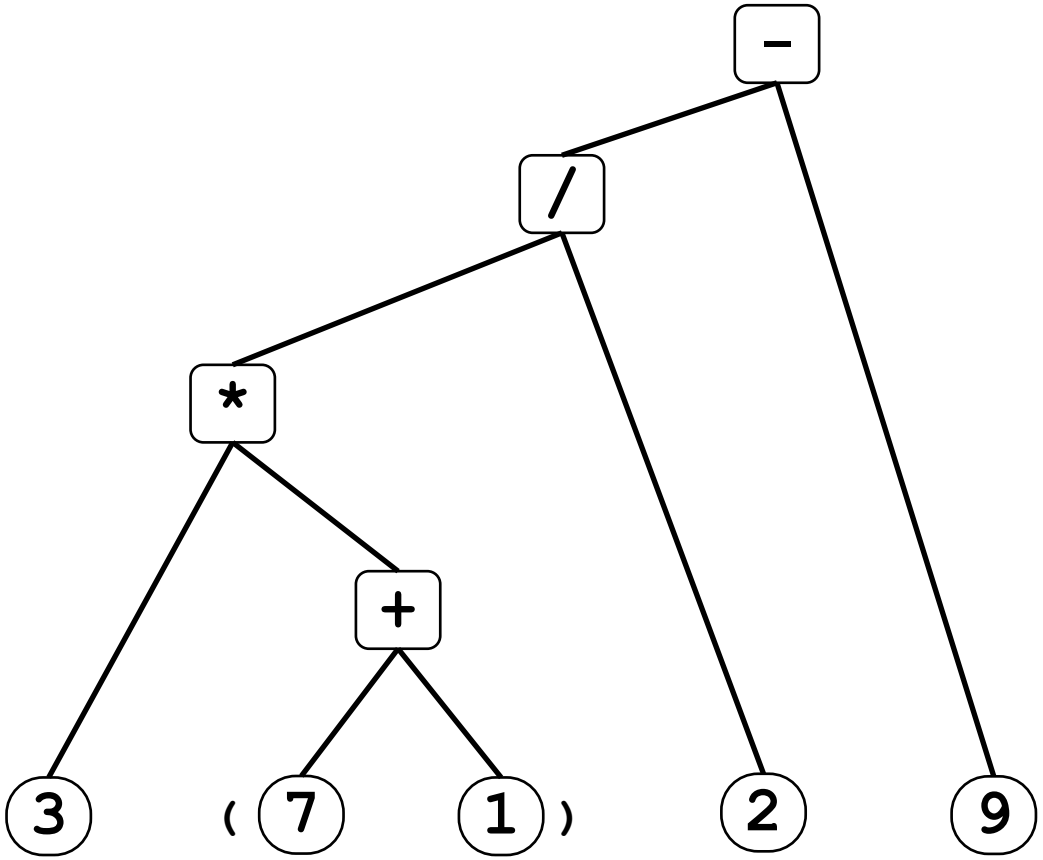

```
3 * (7 + 1) / 2 - 9
```



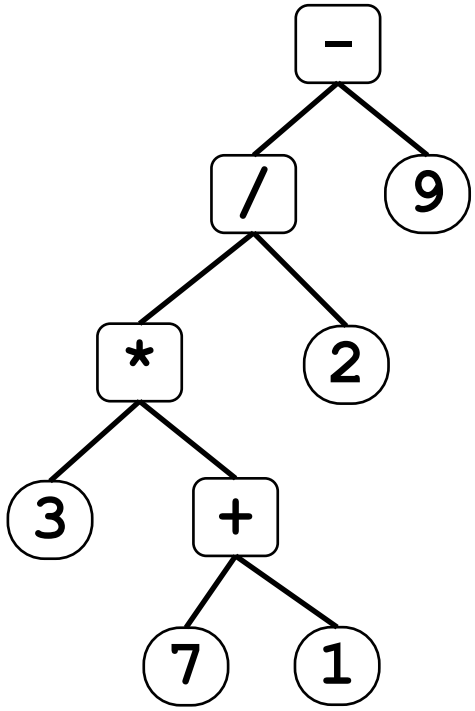
3 * (7 + 1) / 2 - 9



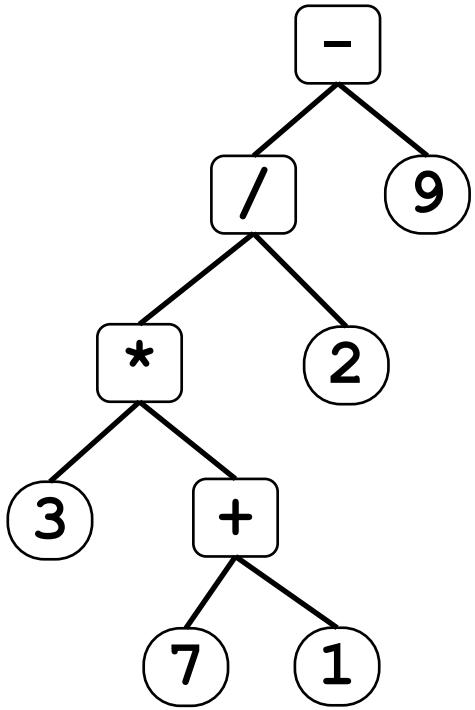
3 * (7 + 1) / 2 - 9



3 * (7 + 1) / 2 - 9

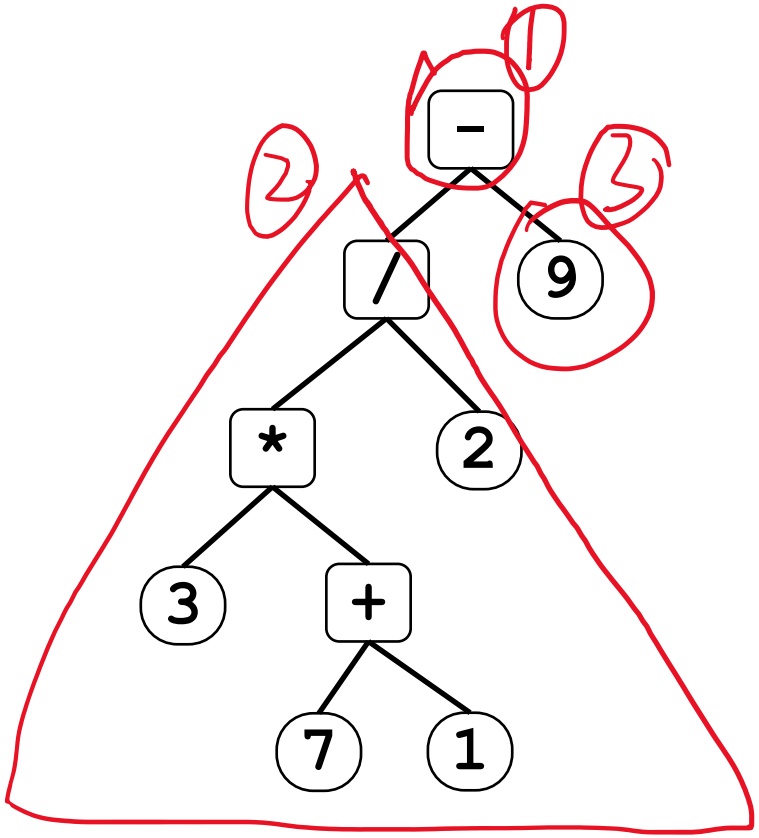



```
3 * (7 + 1) / 2 - 9
```



Pre-order

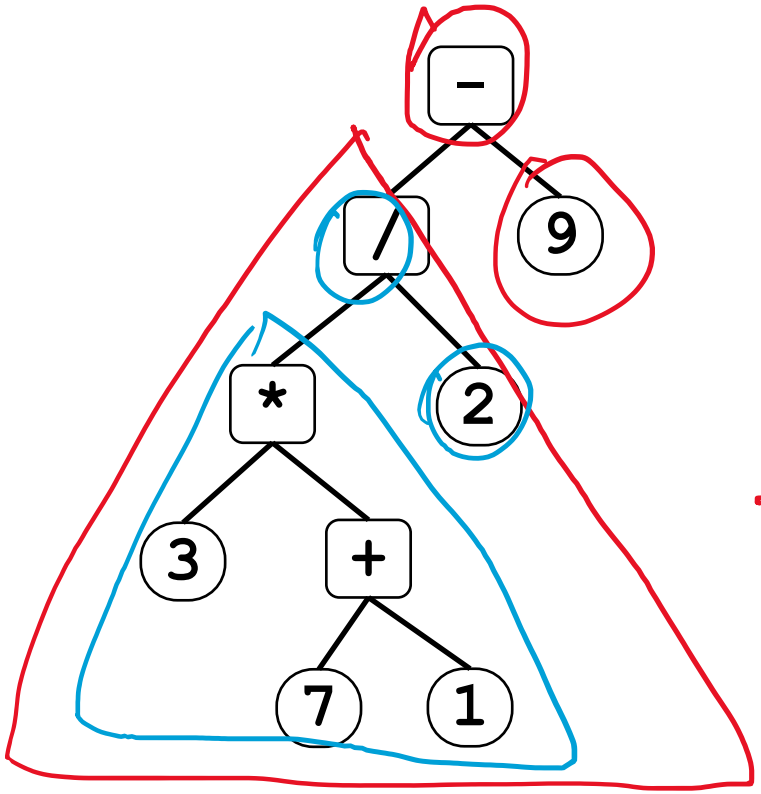
```
3 * (7 + 1) / 2 - 9
```



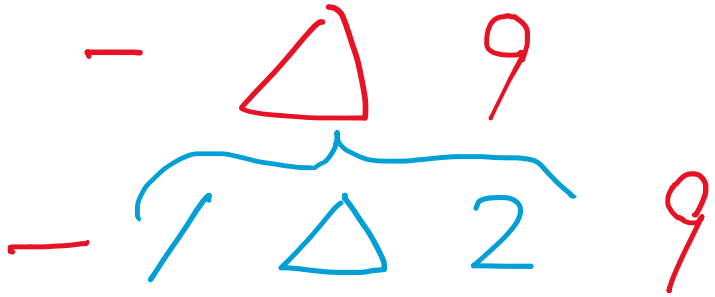
Pre-order

- △ 9

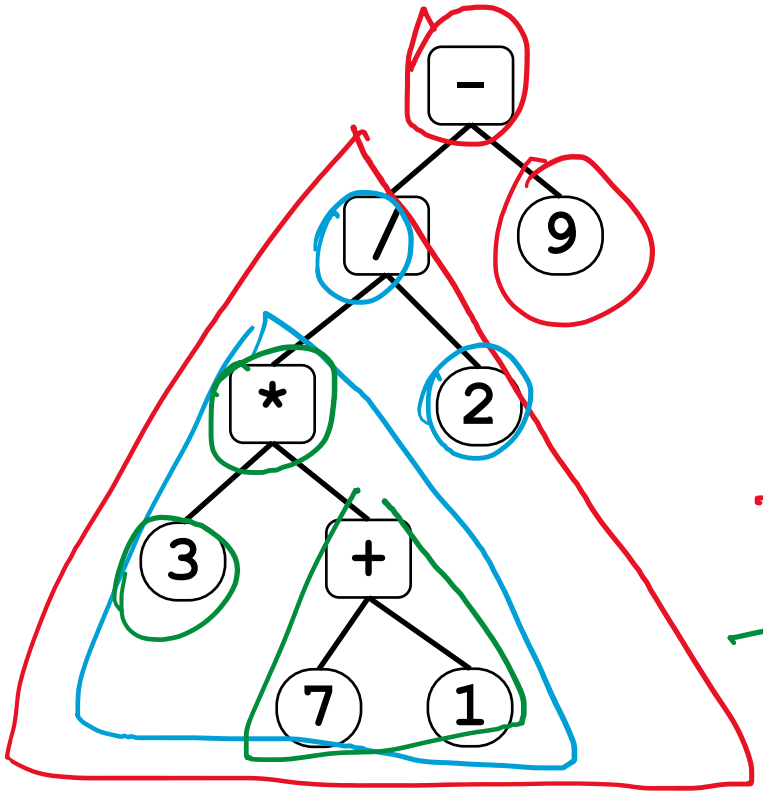
```
3 * (7 + 1) / 2 - 9
```



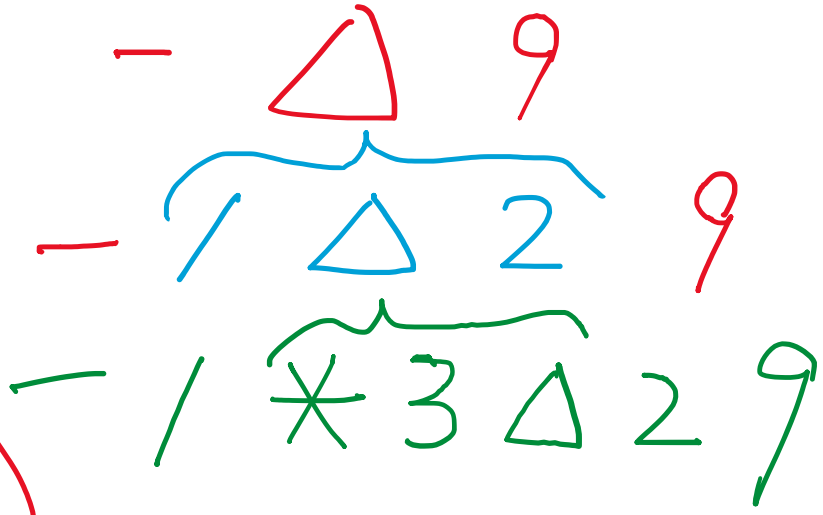
Pre-order



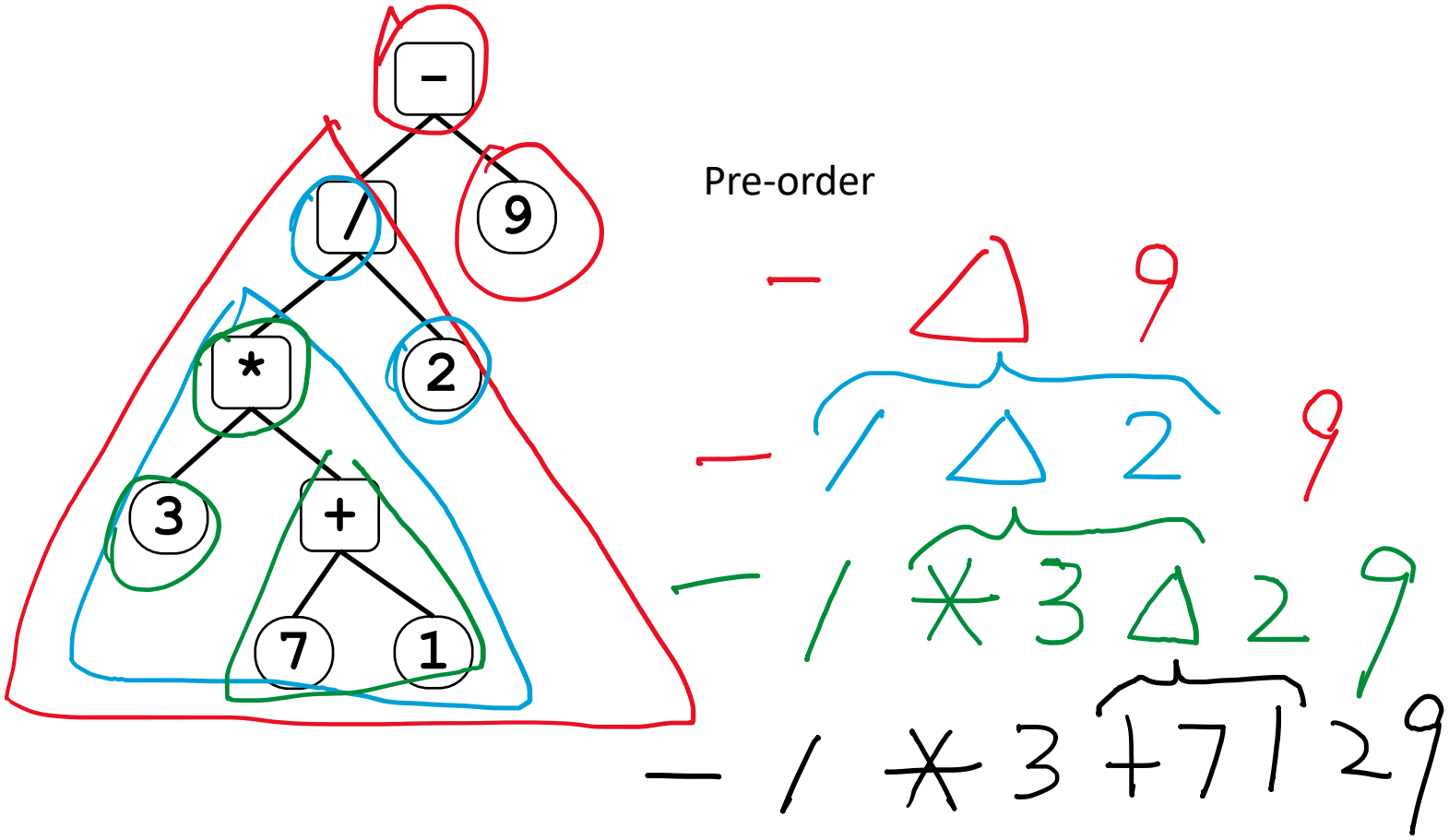
```
3 * (7 + 1) / 2 - 9
```



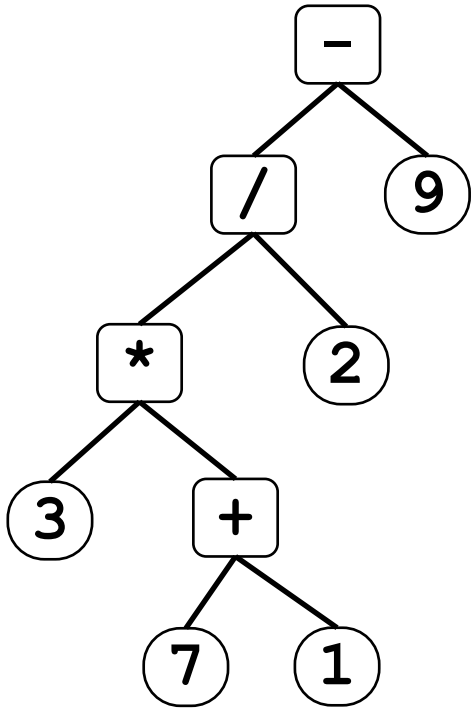
Pre-order



3 * (7 + 1) / 2 - 9



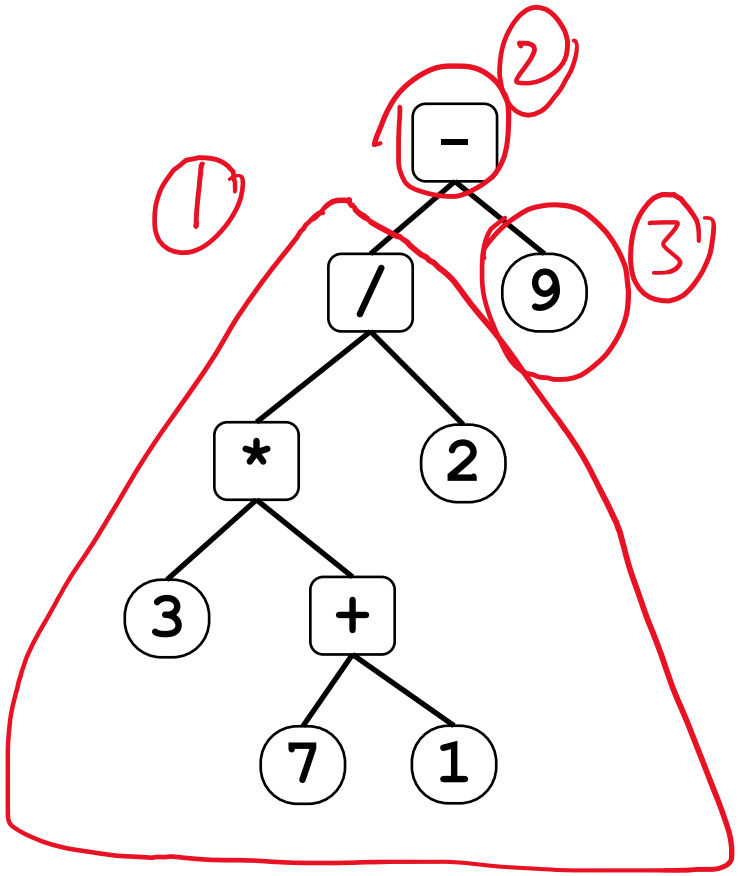
3 * (7 + 1) / 2 - 9



Pre-order **- / * 3 + 7 1 2 9**

— / * 3 + 7 1 2 9

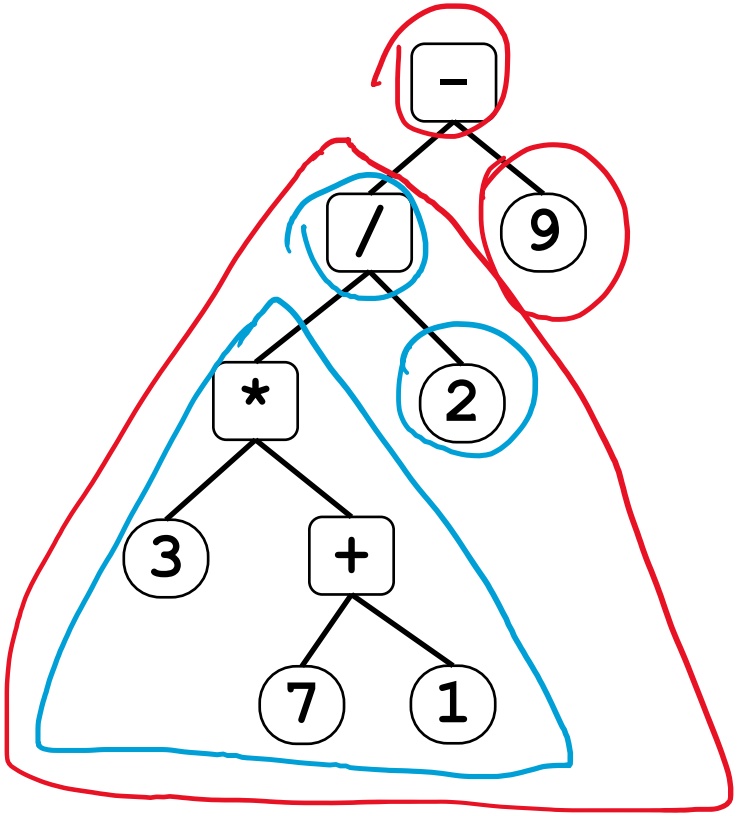
3 * (7 + 1) / 2 - 9



Pre-order - / * 3 + 7 1 2 9

In-order

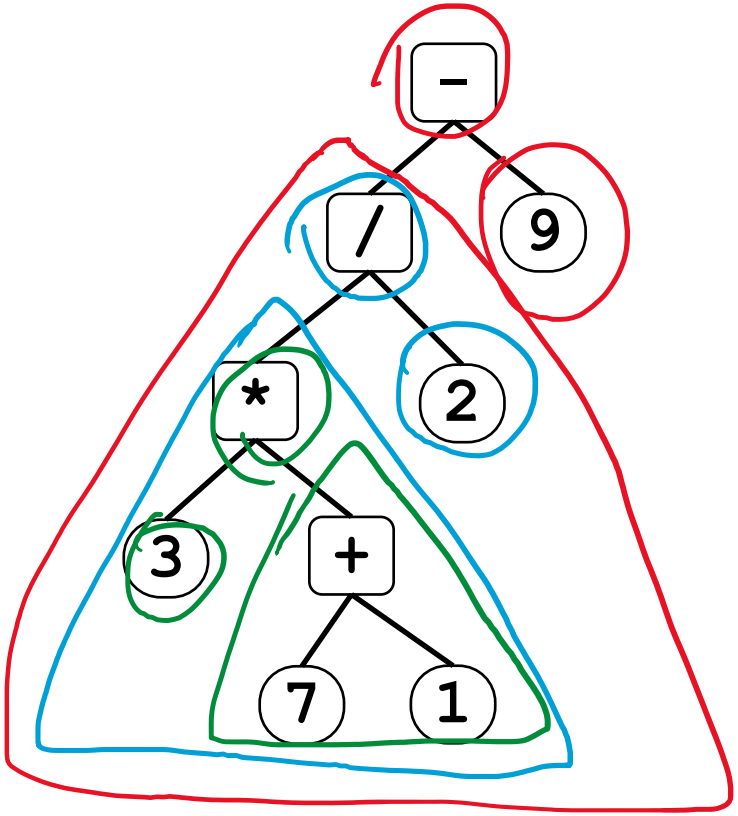
3 * (7 + 1) / 2 - 9



Pre-order - / * 3 + 7 1 2 9

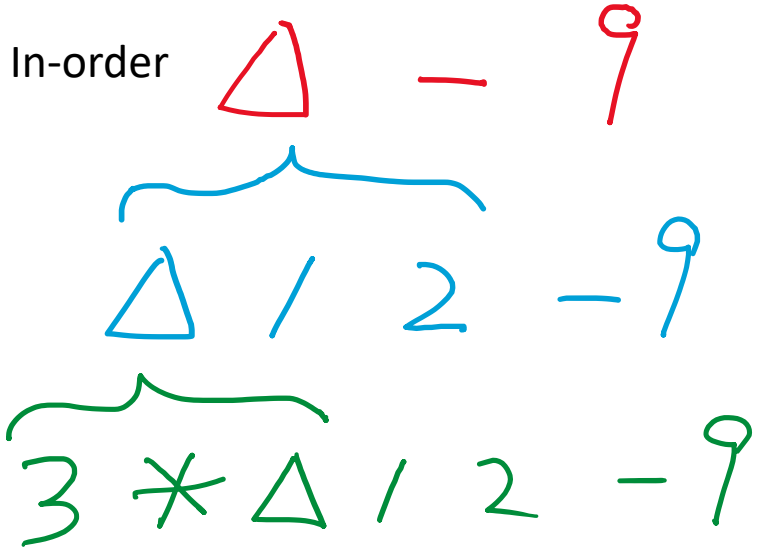
In-order

3 * (7 + 1) / 2 - 9

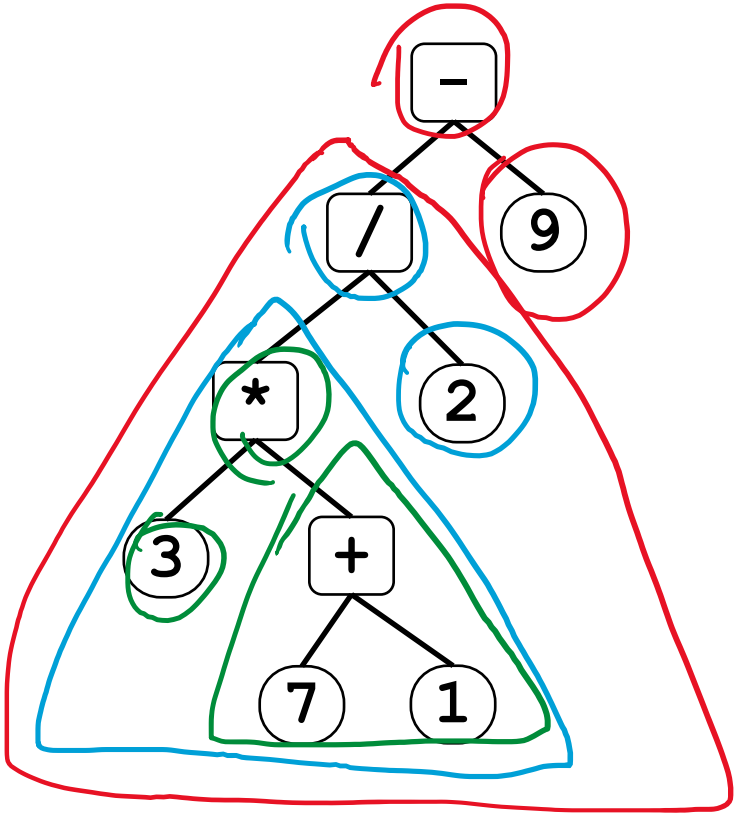


Pre-order - / * 3 + 7 1 2 9

In-order



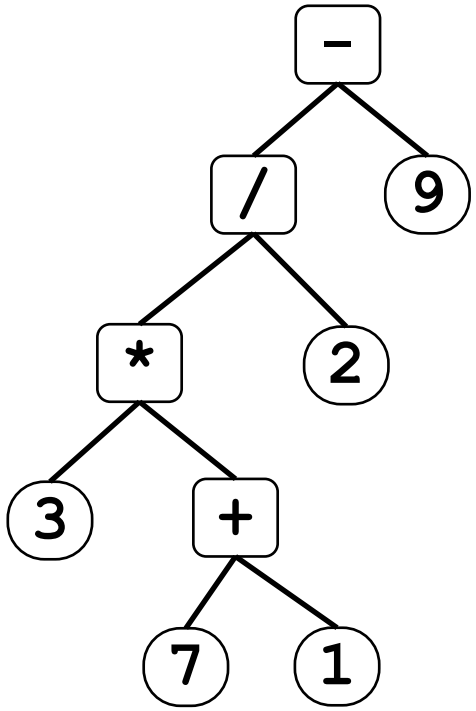
3 * (7 + 1) / 2 - 9



Pre-order - / * 3 + 7 1 2 9

In-order

3 * (7 + 1) / 2 - 9

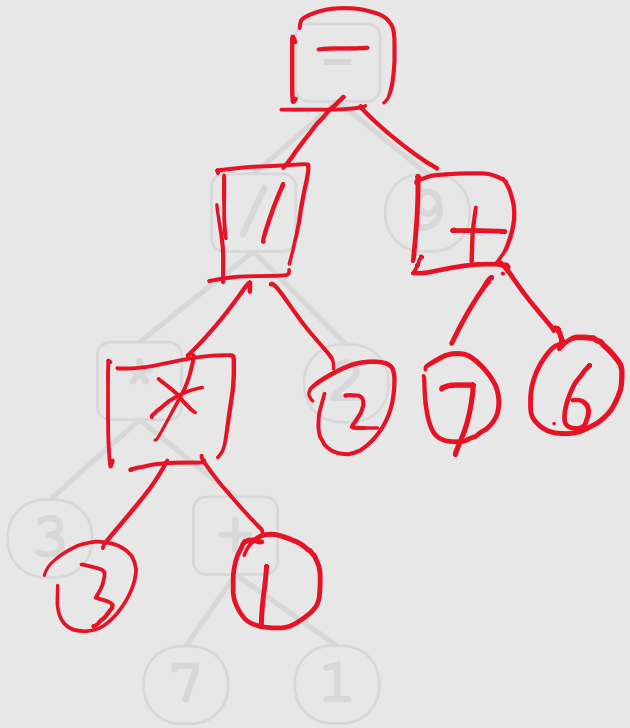


Pre-order `- / * 3 + 7 1 2 9`

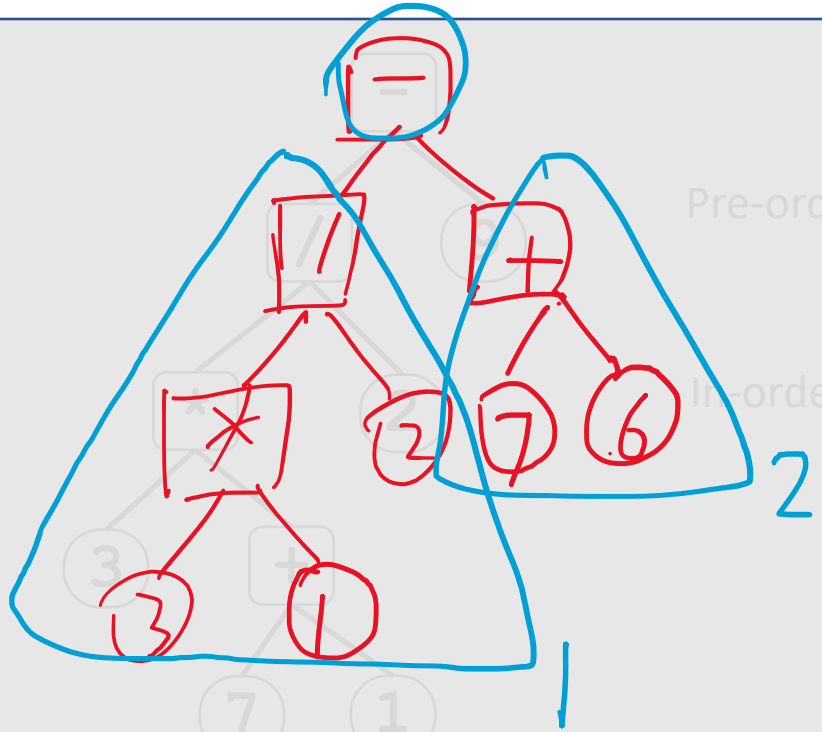
In-order `3 * 7 + 1 / 2 - 9`

3 * 7 + 1 / 2 - 9

`3 * (7 + 1) / 2 - 9`

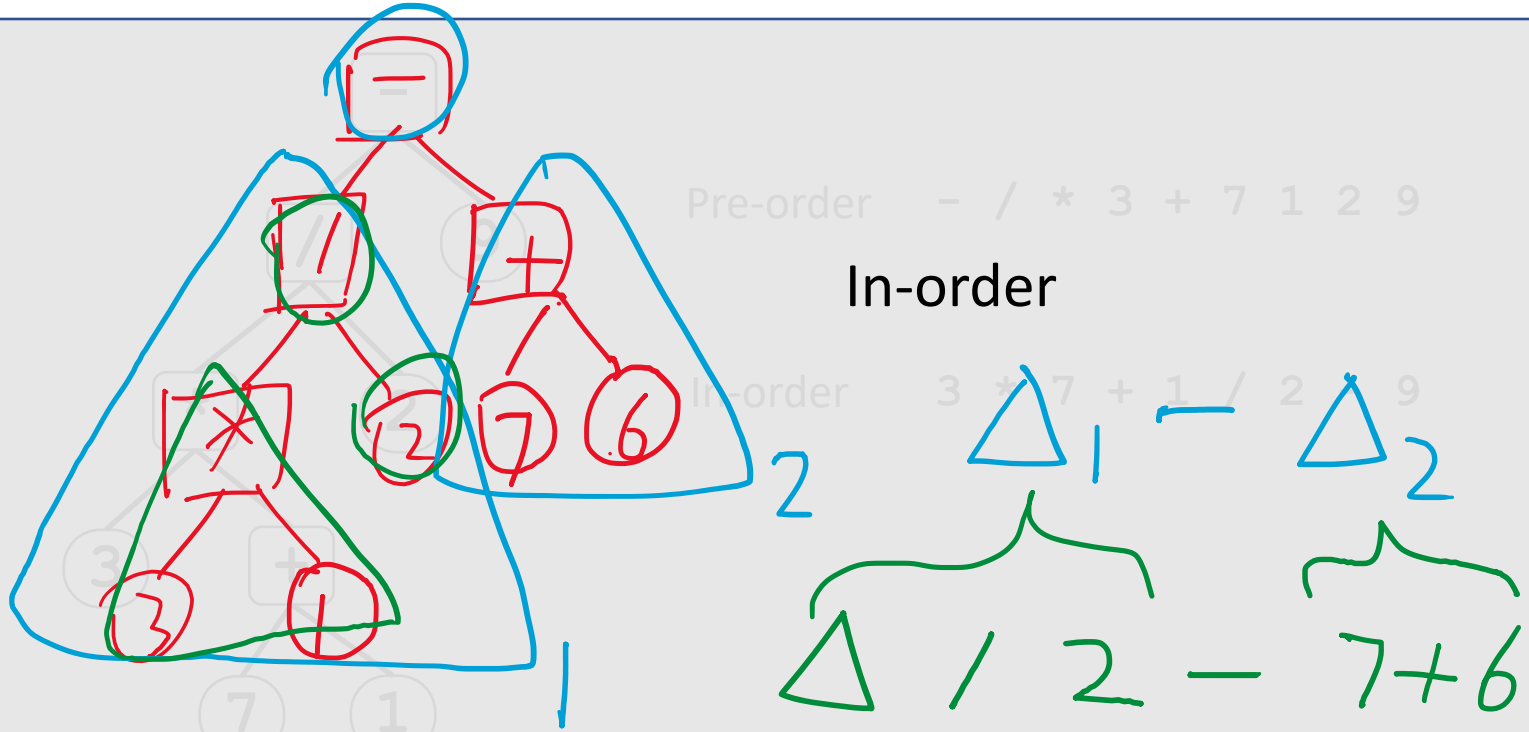


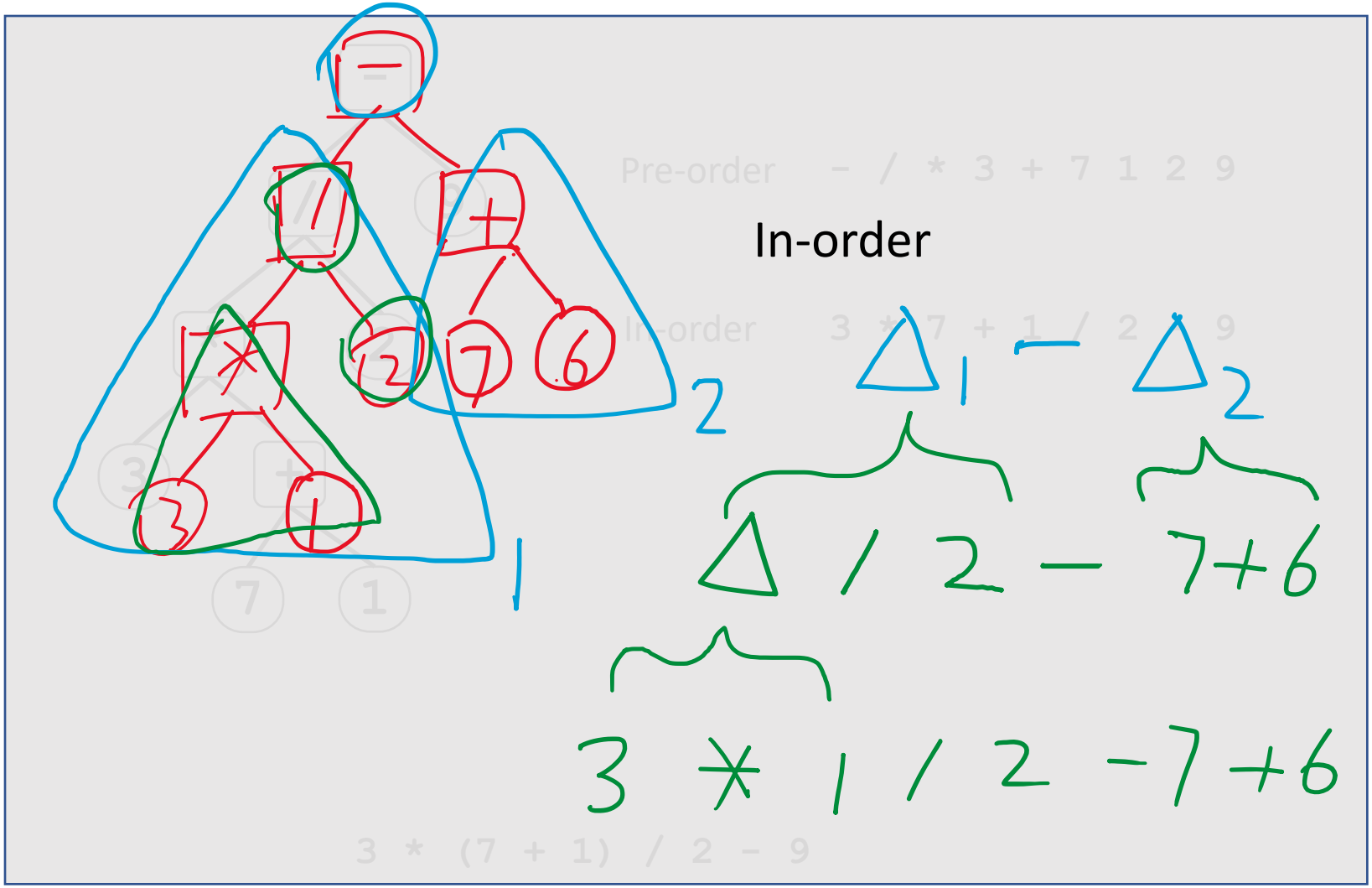
In-order

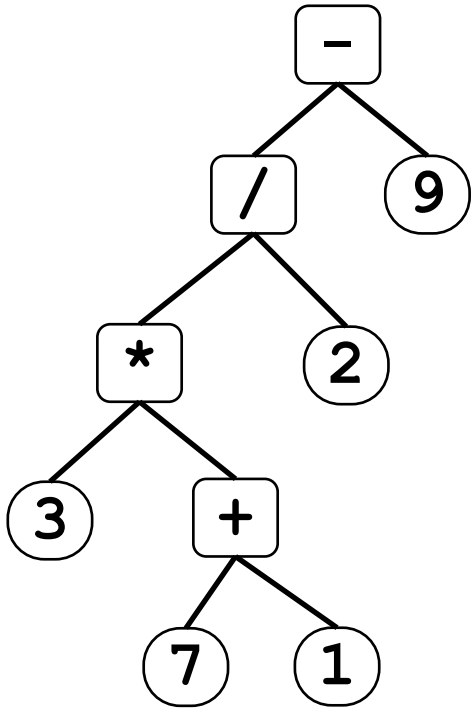


In-order

$\Delta_1 - \Delta_2$


In-order
2
1
Δ1 − Δ2
Δ / 2 − 7+6


In-order
2
1
3 * 1 / 2 - 7 + 6

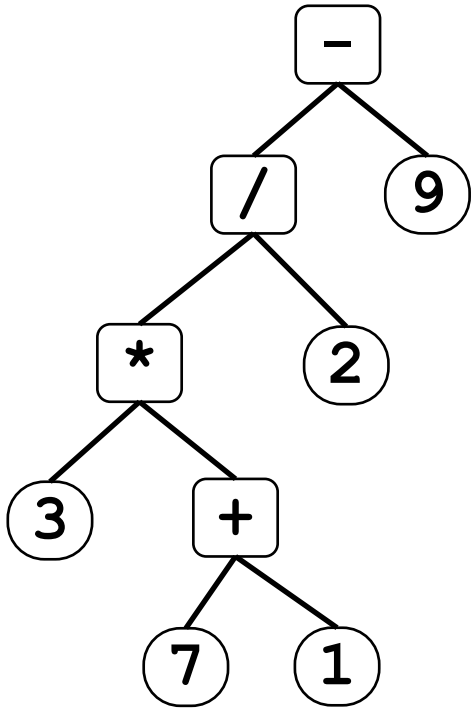


Pre-order `- / * 3 + 7 1 2 9`

In-order `3 * 7 + 1 / 2 - 9`

Post-order `3 7 1 + * 2 / 9 -`

`3 * (7 + 1) / 2 - 9`



Pre-order
Prefix form
Polish Notation

```
- / * 3 + 7 1 2 9
```

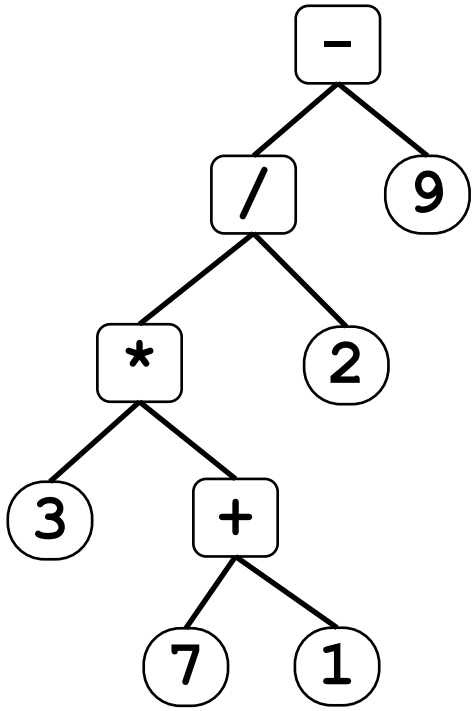
In-order
Infix form

```
3 * 7 + 1 / 2 - 9
```

Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

```
3 * (7 + 1) / 2 - 9
```



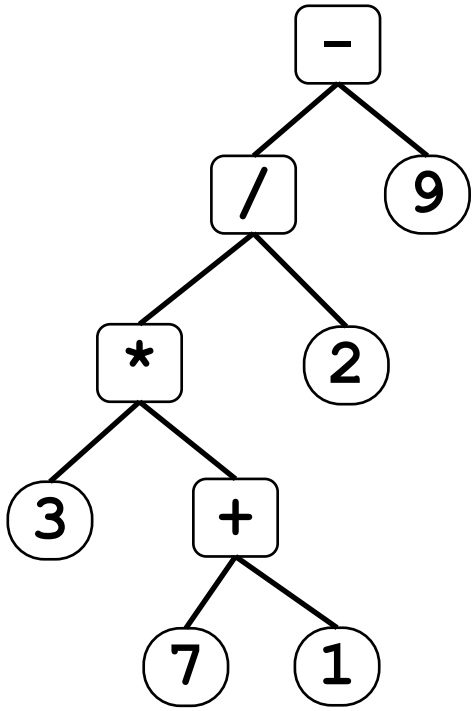
In-order
Infix form

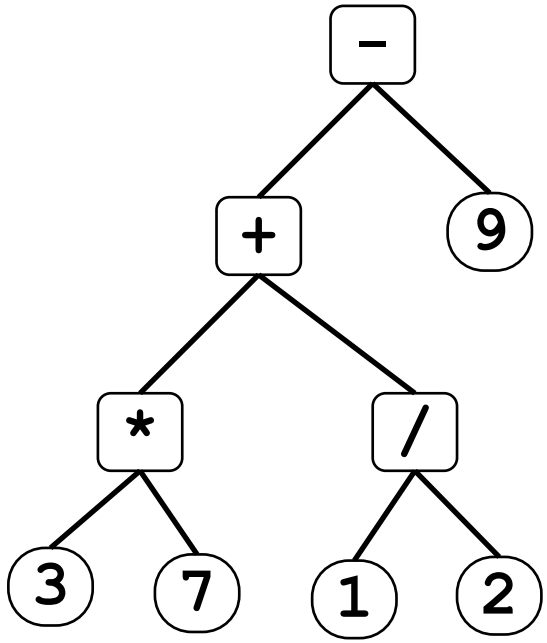
```
3 * 7 + 1 / 2 - 9
```

DIFFERENT!

```
3 * (7 + 1) / 2 - 9
```

Same infix form,
DIFFERENT shapes!

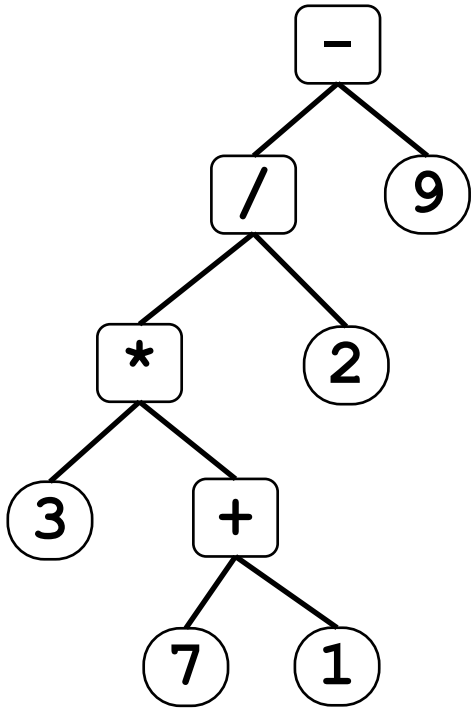




In-order
Infix form

```
3 * 7 + 1 / 2 - 9
```

```
3 * (7 + 1) / 2 - 9
```



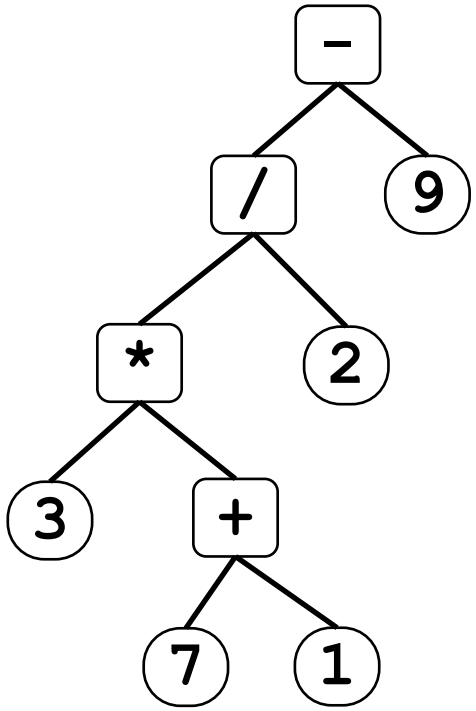
Pre-order
Prefix form
Polish Notation

```
- / * 3 + 7 1 2 9
```

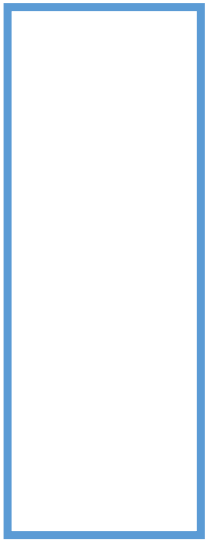
Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

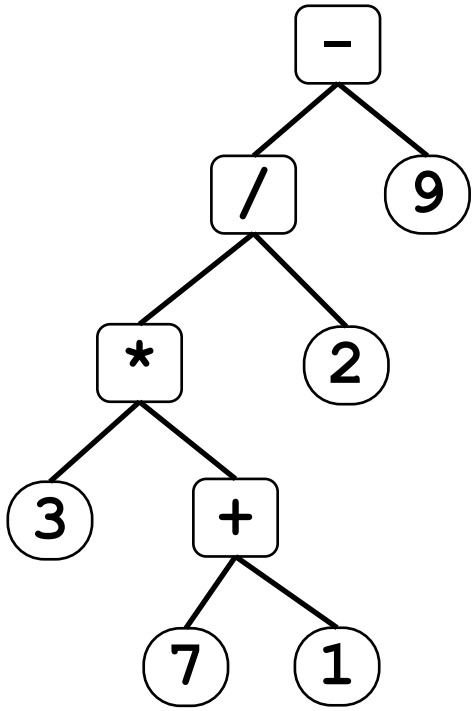
```
3 * (7 + 1) / 2 - 9
```



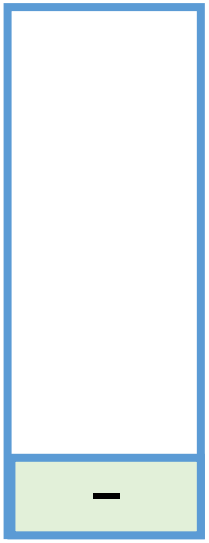
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



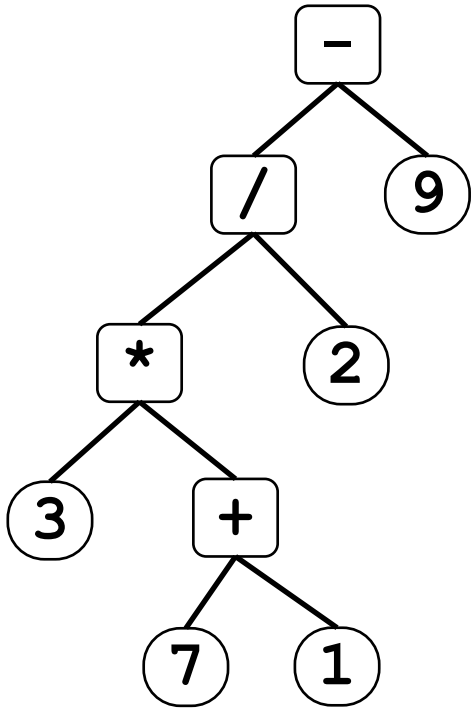
`3 * (7 + 1) / 2 - 9`



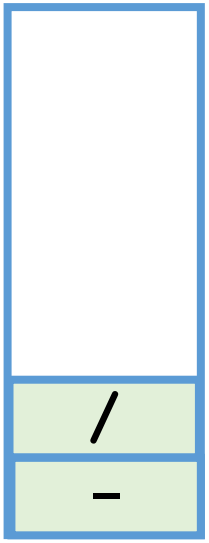
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



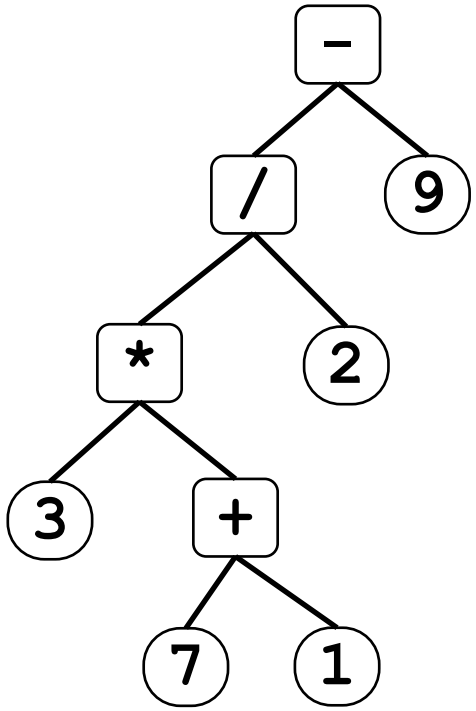
`3 * (7 + 1) / 2 - 9`



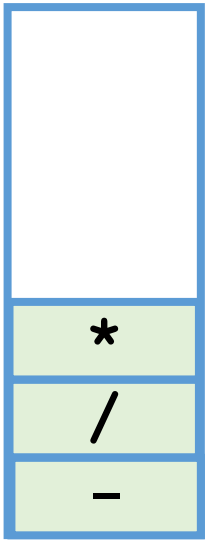
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



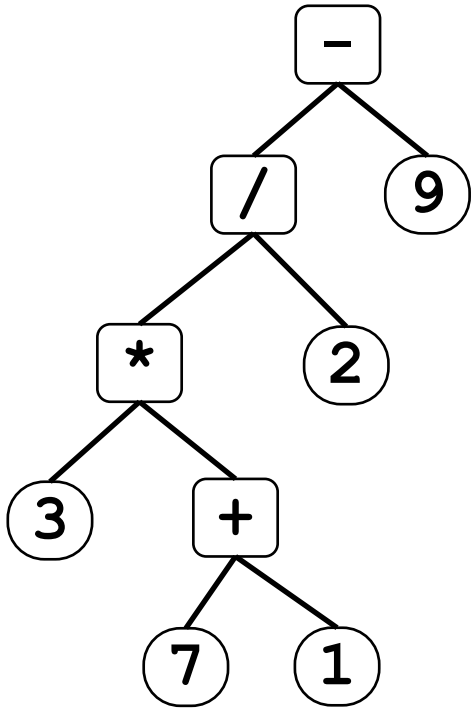
`3 * (7 + 1) / 2 - 9`



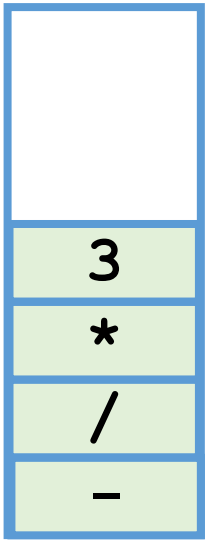
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



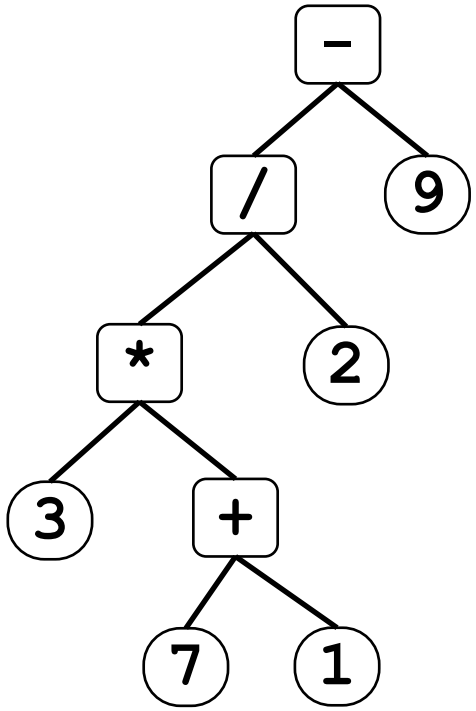
`3 * (7 + 1) / 2 - 9`



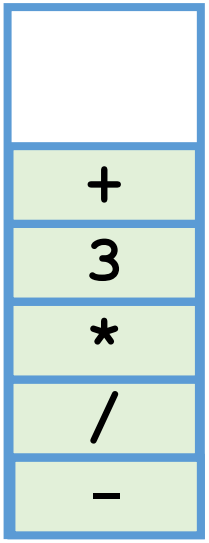
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



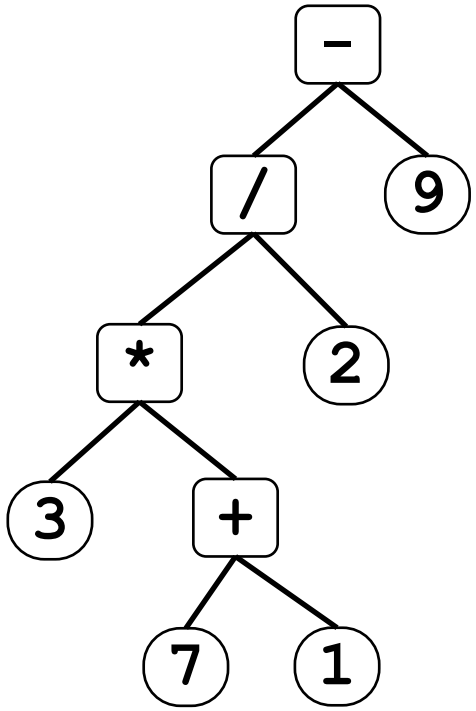
`3 * (7 + 1) / 2 - 9`



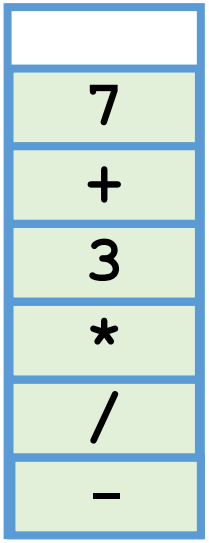
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



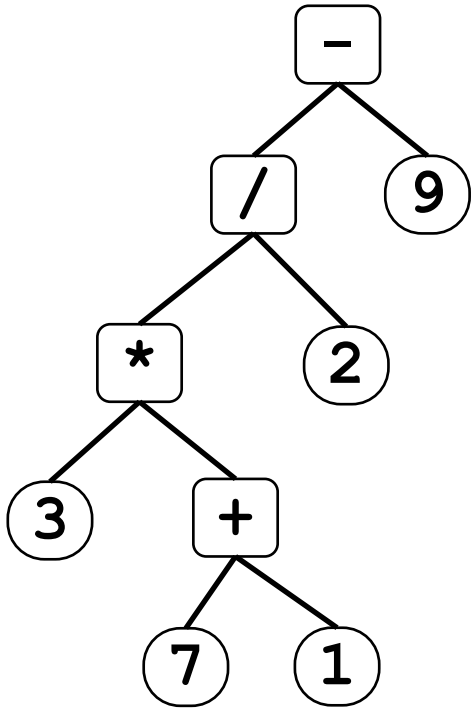
`3 * (7 + 1) / 2 - 9`



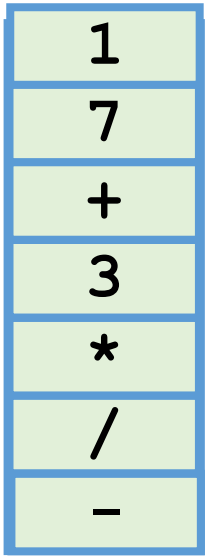
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



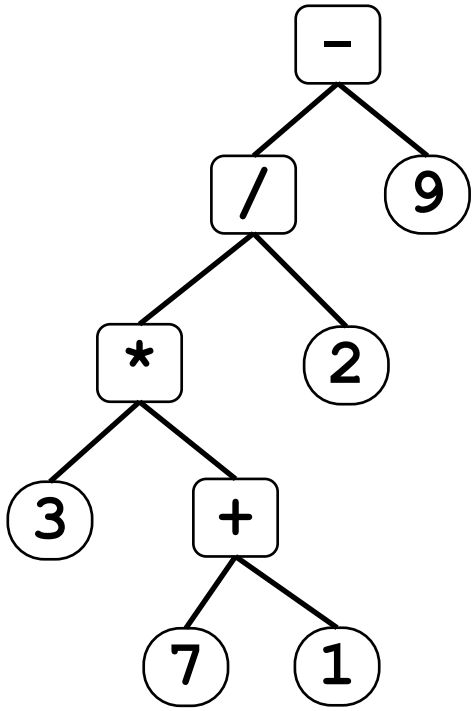
`3 * (7 + 1) / 2 - 9`



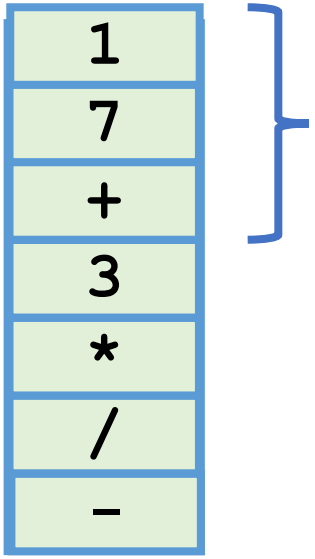
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



`3 * (7 + 1) / 2 - 9`

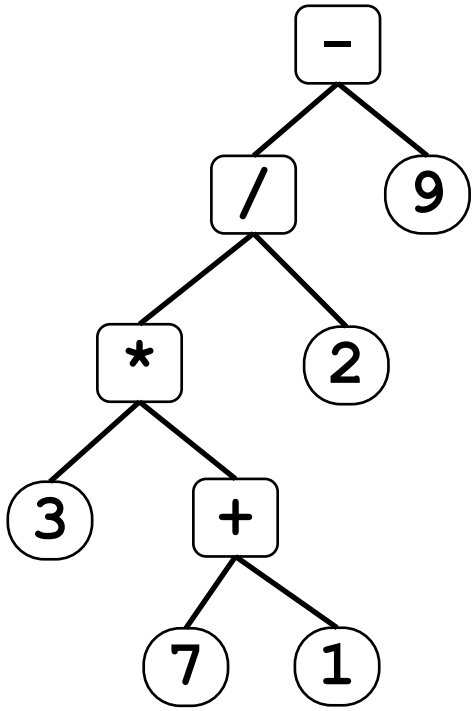


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

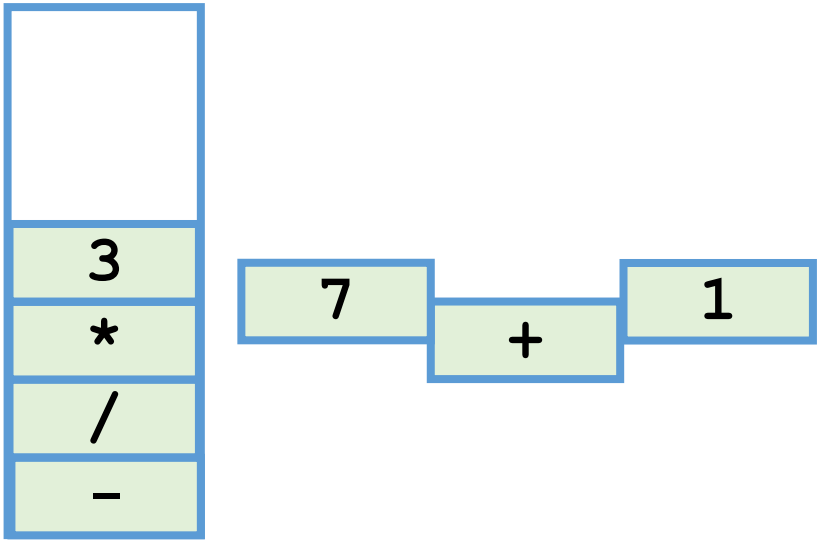


7 + 1

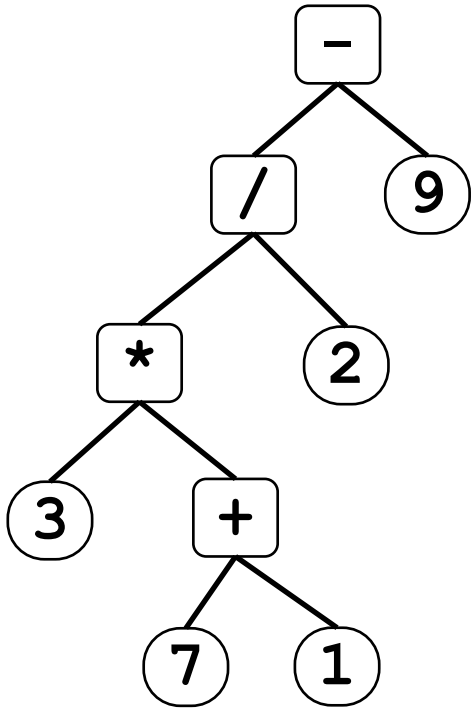
`3 * (7 + 1) / 2 - 9`



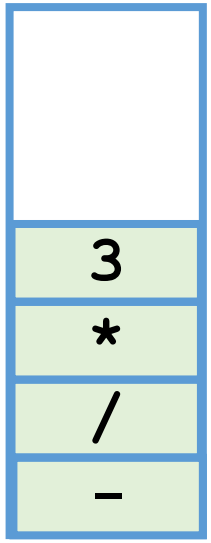
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



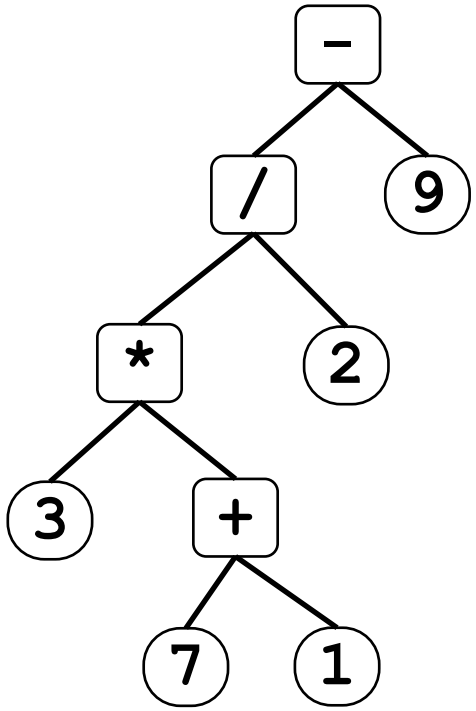
`3 * (7 + 1) / 2 - 9`



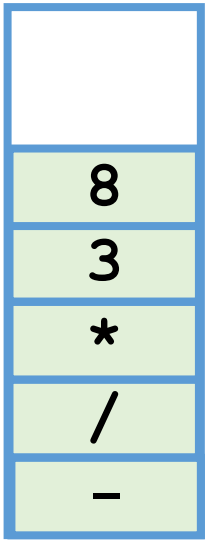
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



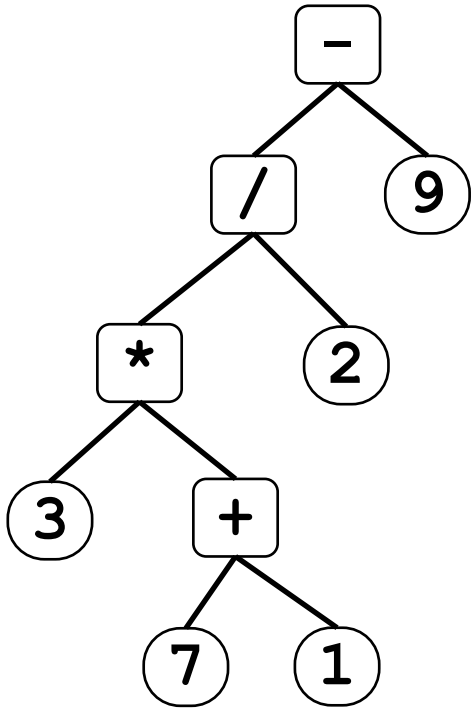
`3 * (7 + 1) / 2 - 9`



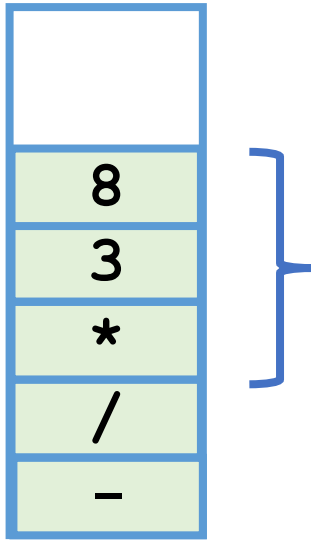
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



`3 * (7 + 1) / 2 - 9`

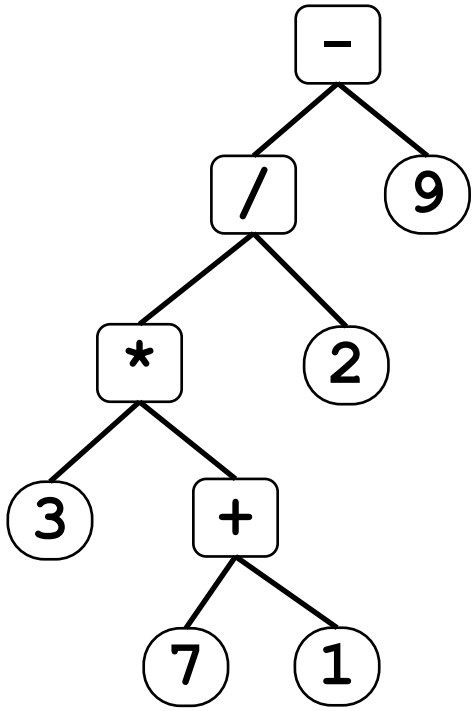


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

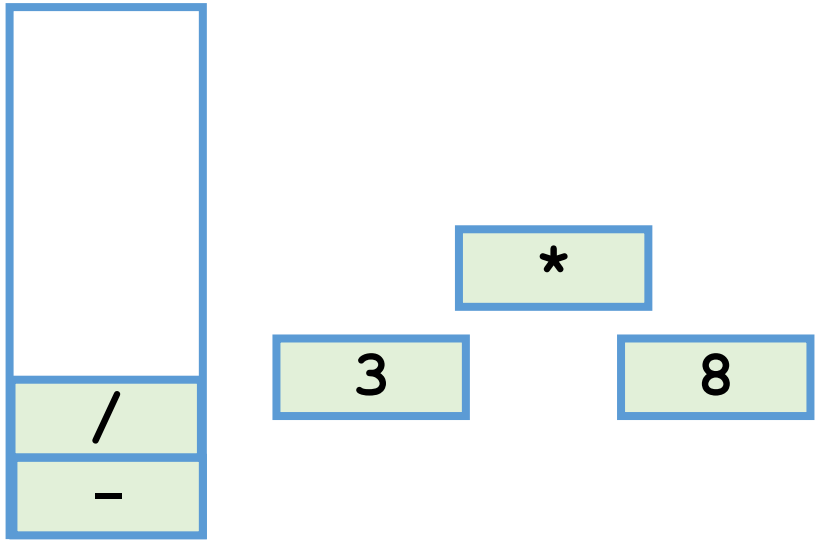


3 * 8

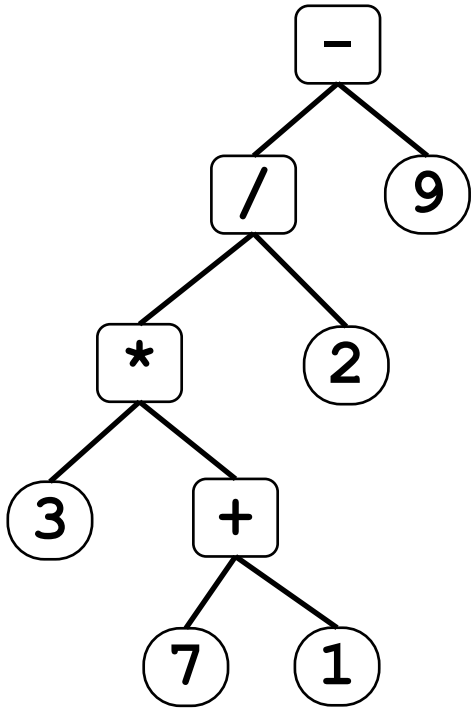
`3 * (7 + 1) / 2 - 9`



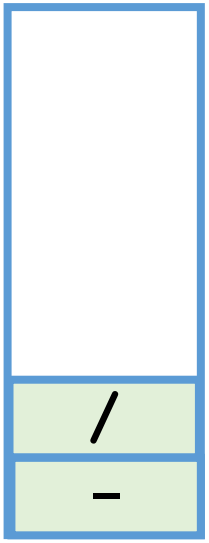
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

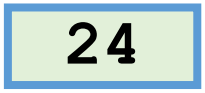


`3 * (7 + 1) / 2 - 9`

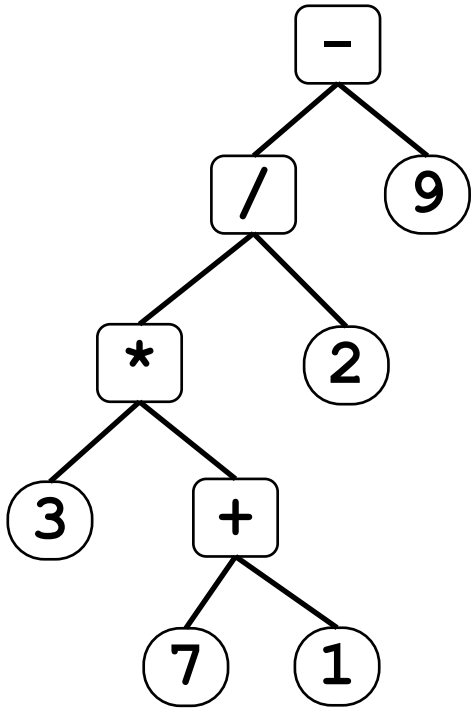


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

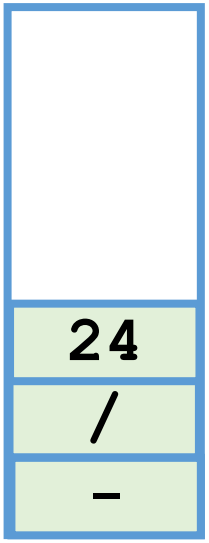




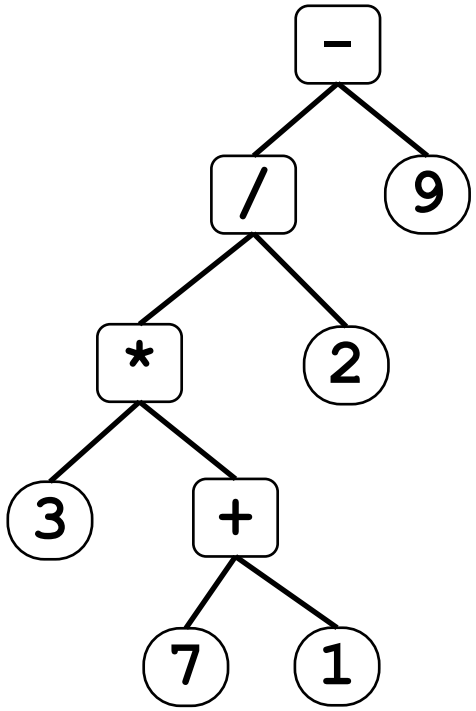
`3 * (7 + 1) / 2 - 9`



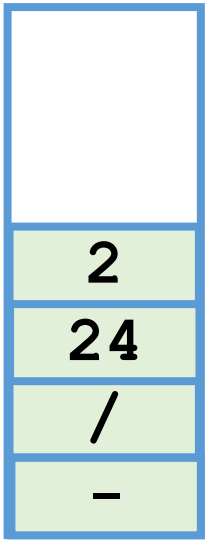
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



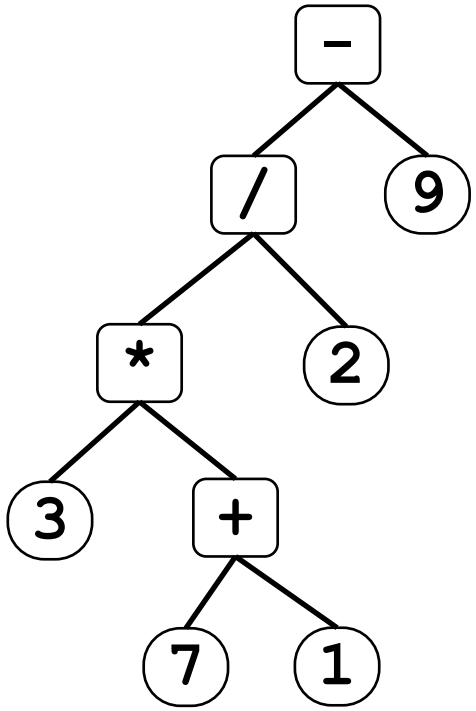
`3 * (7 + 1) / 2 - 9`



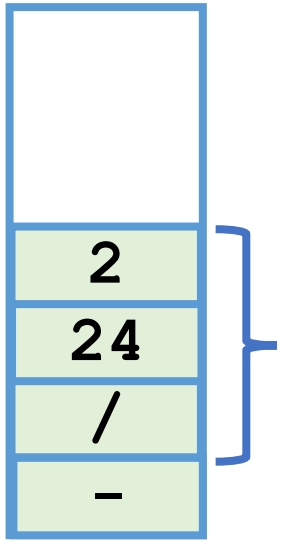
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



`3 * (7 + 1) / 2 - 9`

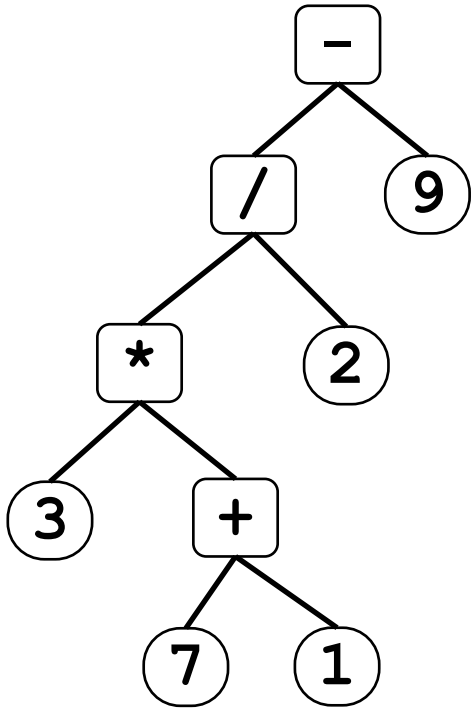


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

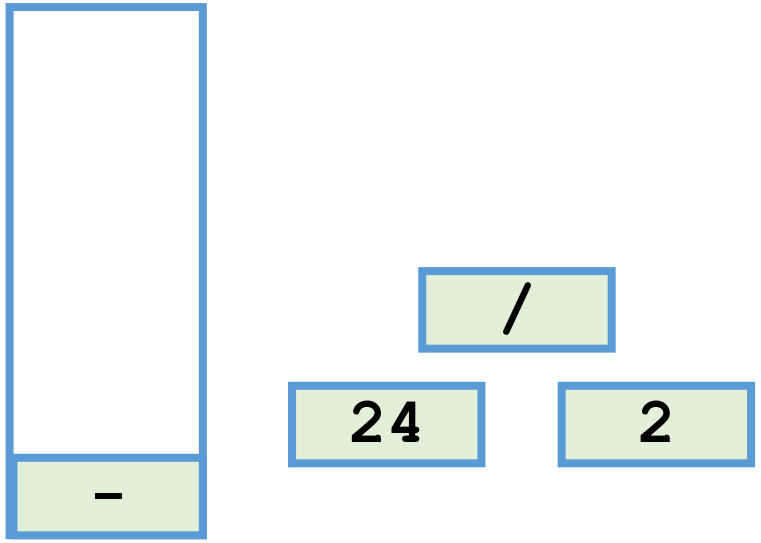


24 / 2

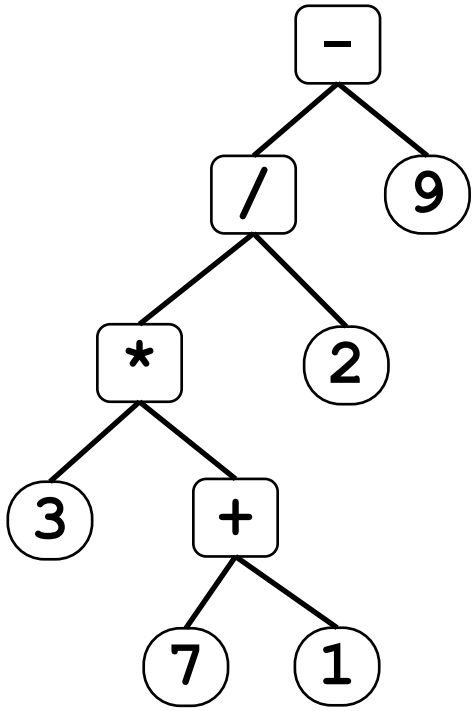
`3 * (7 + 1) / 2 - 9`



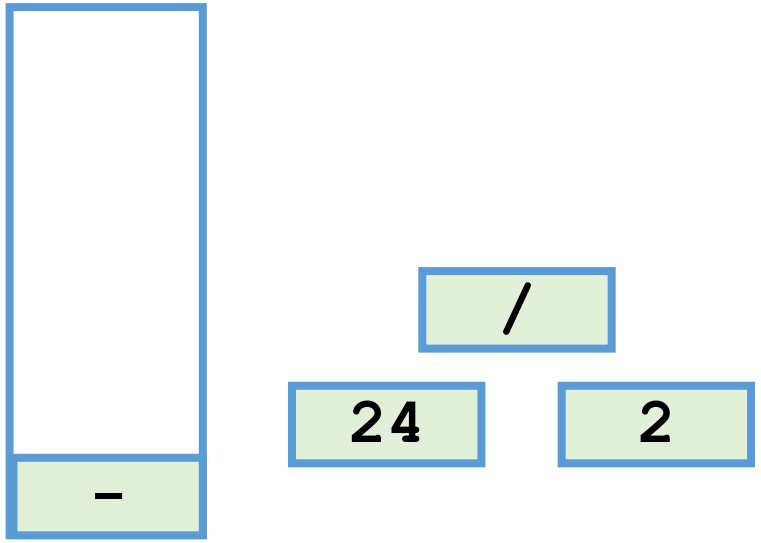
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



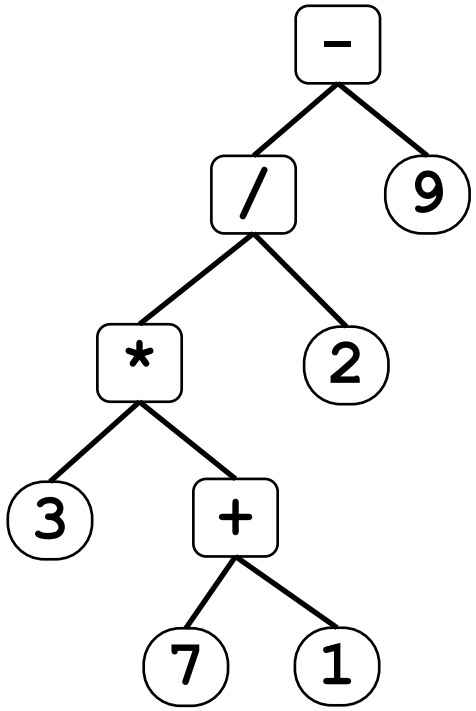
`3 * (7 + 1) / 2 - 9`



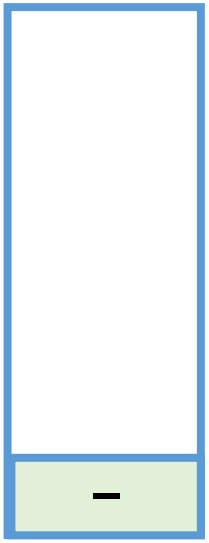
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

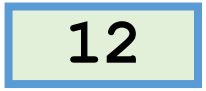


`3 * (7 + 1) / 2 - 9`

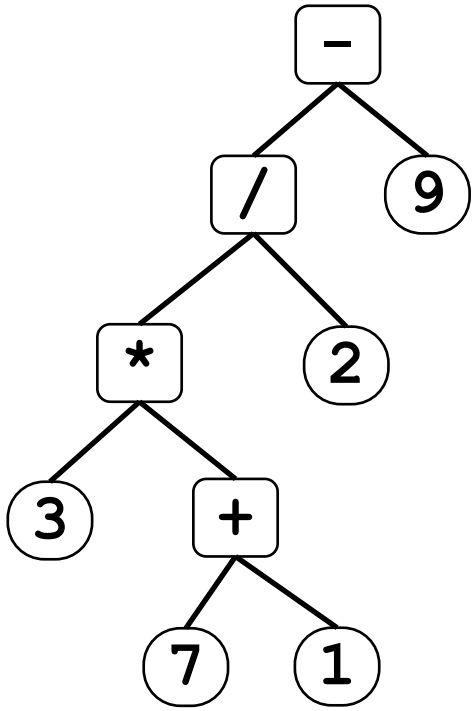


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

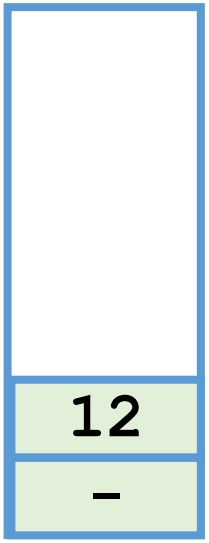




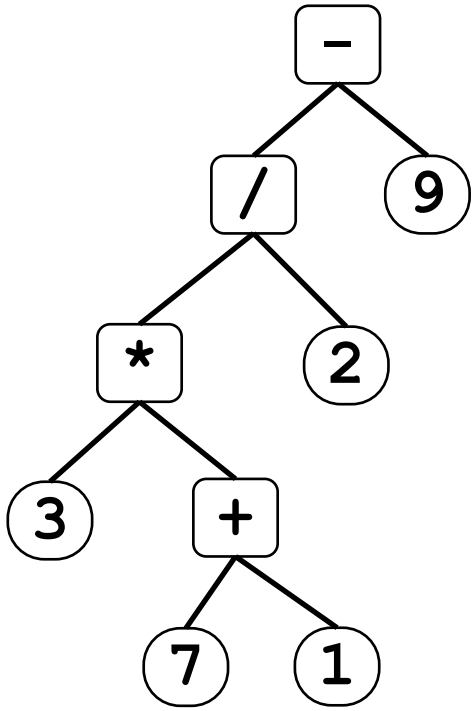
`3 * (7 + 1) / 2 - 9`



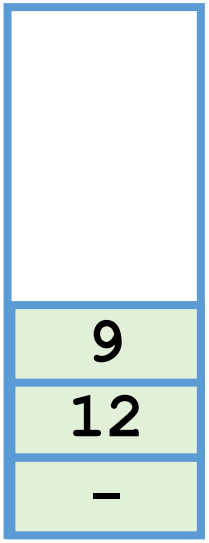
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



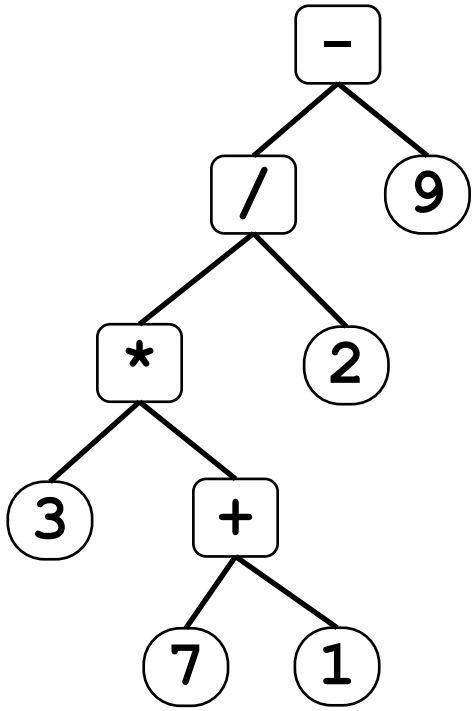
`3 * (7 + 1) / 2 - 9`



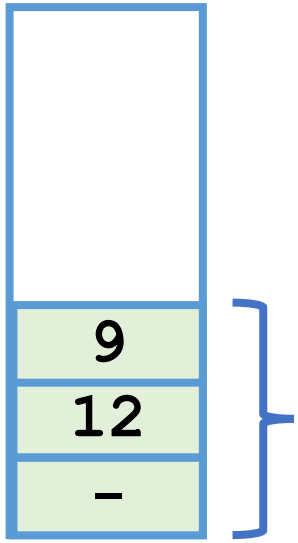
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



`3 * (7 + 1) / 2 - 9`

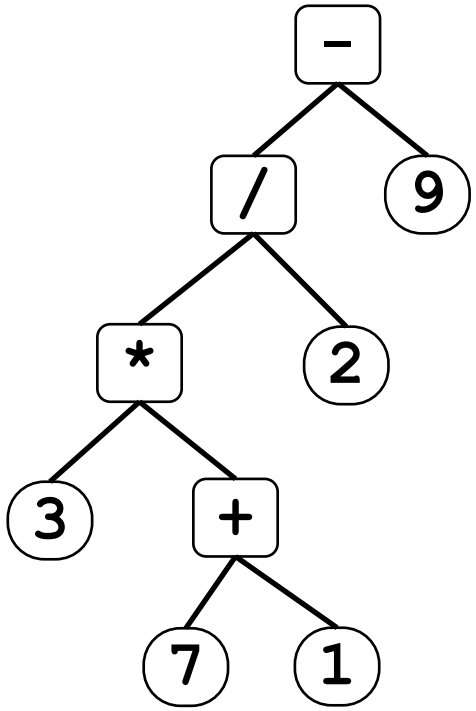


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

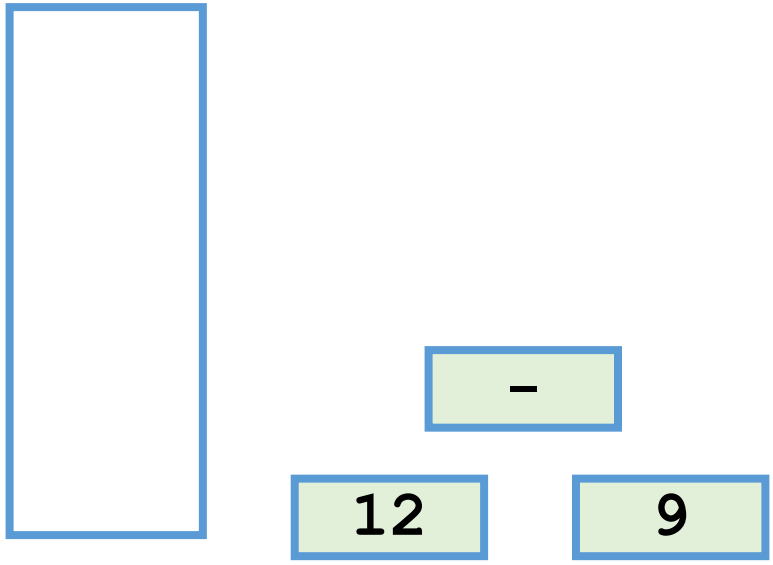


24 / 2

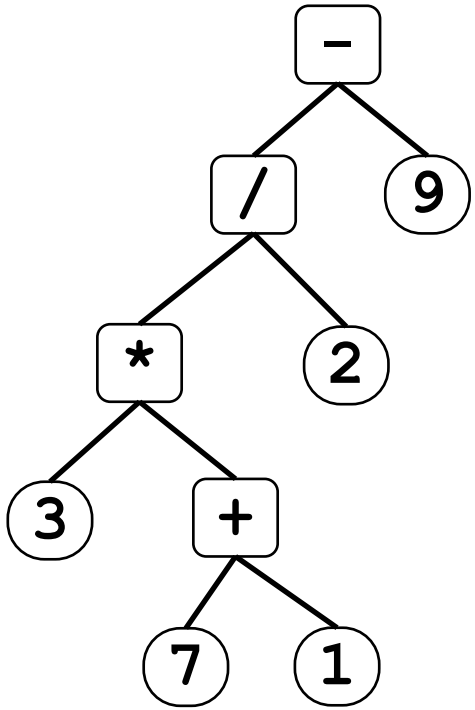
`3 * (7 + 1) / 2 - 9`



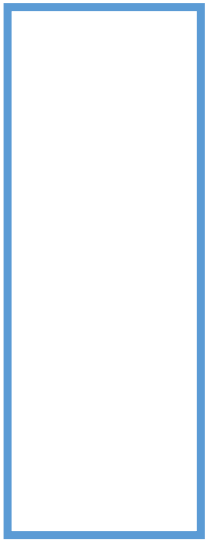
Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

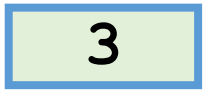


`3 * (7 + 1) / 2 - 9`

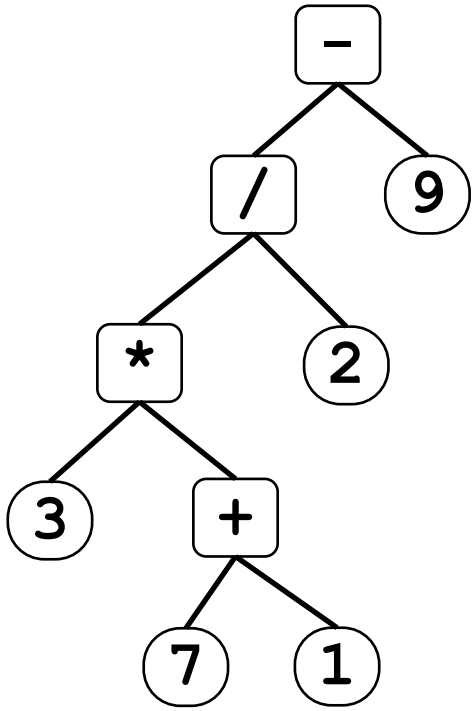


Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation

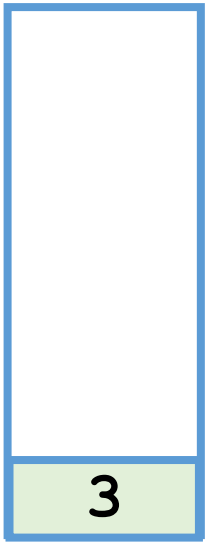




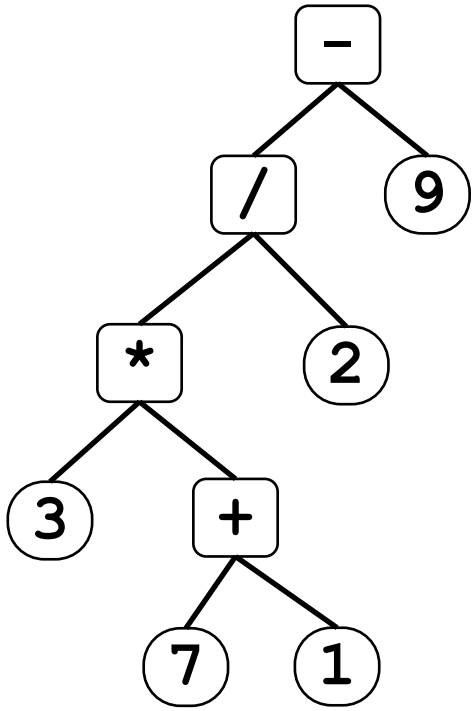
`3 * (7 + 1) / 2 - 9`



Pre-order
Prefix form `- / * 3 + 7 1 2 9`
Polish Notation



`3 * (7 + 1) / 2 - 9`



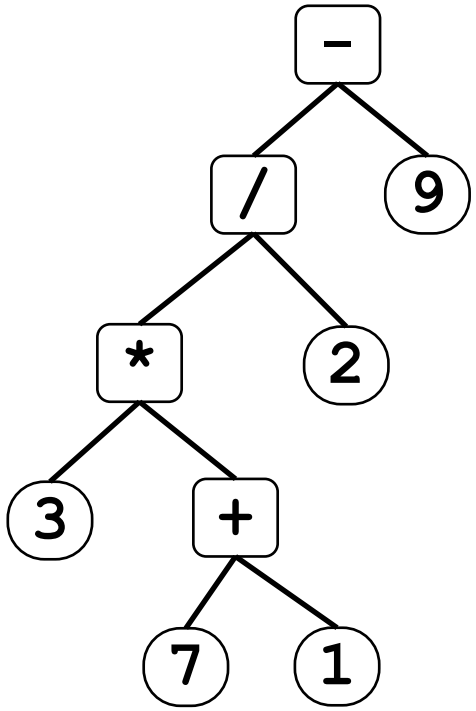
Pre-order
Prefix form
Polish Notation

```
- / * 3 + 7 1 2 9
```

Post-order
Postfix form
Reverse Polish Notation

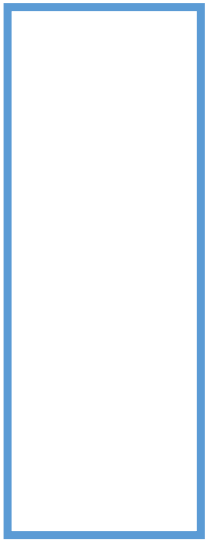
```
3 7 1 + * 2 / 9 -
```

```
3 * (7 + 1) / 2 - 9
```

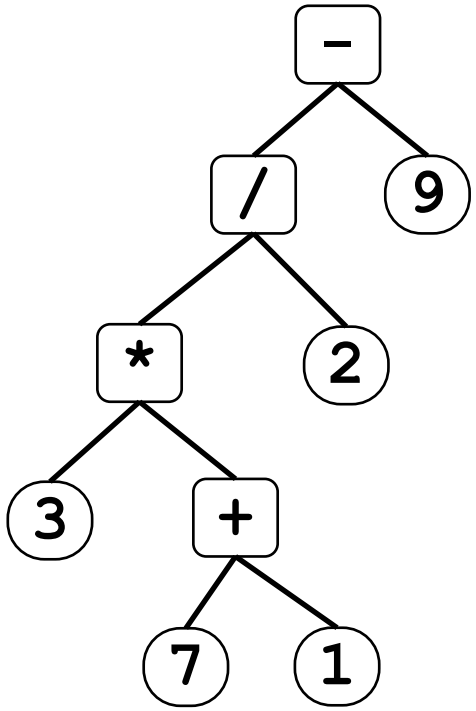


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

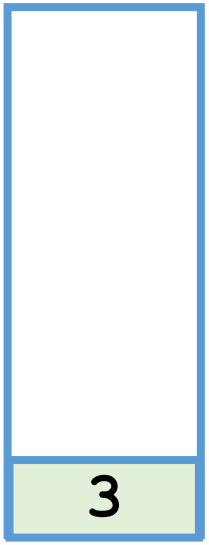


```
3 * (7 + 1) / 2 - 9
```

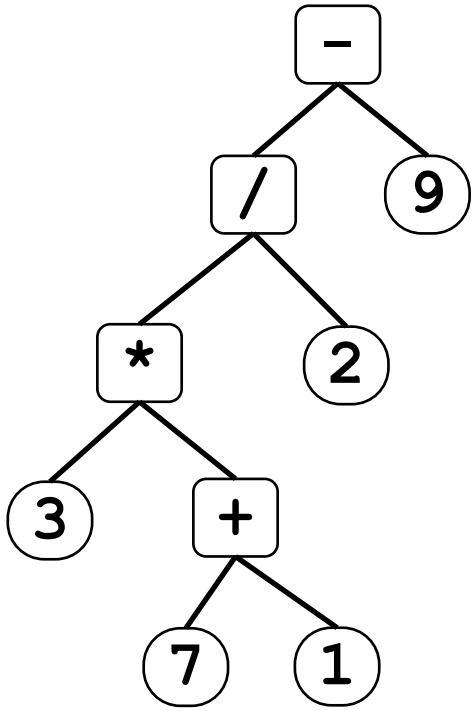


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

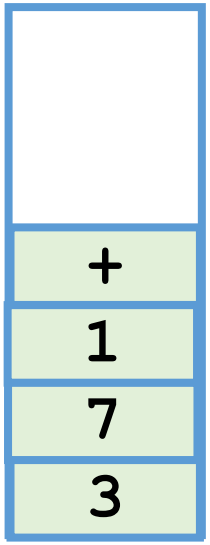



```
3 * (7 + 1) / 2 - 9
```

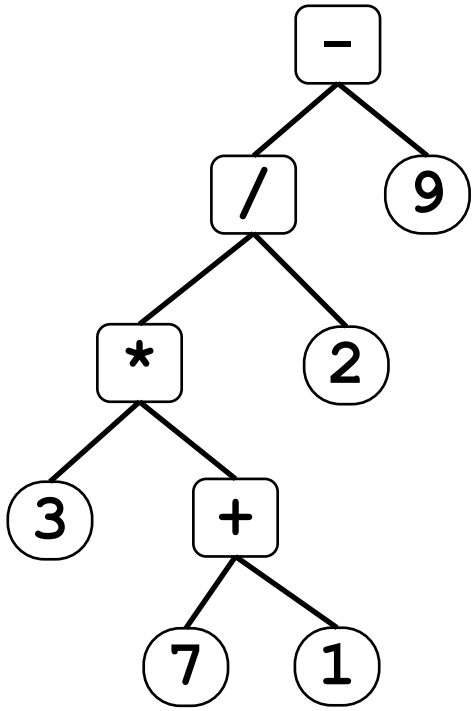


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

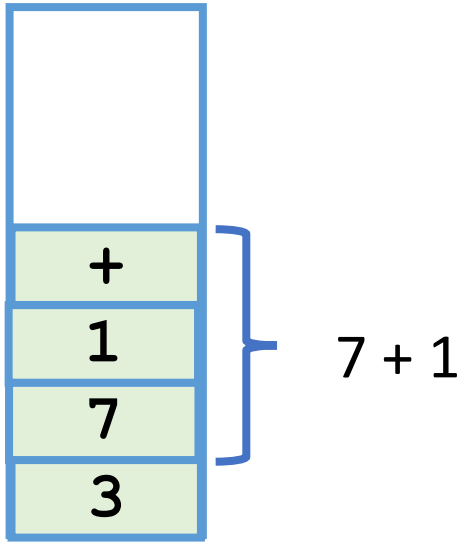



```
3 * (7 + 1) / 2 - 9
```

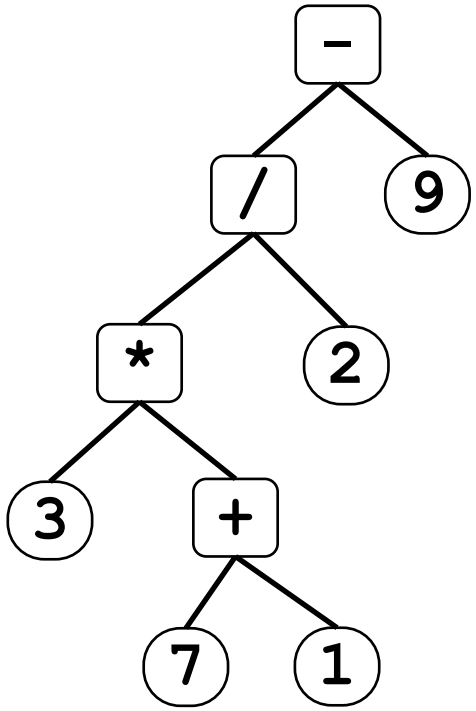


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

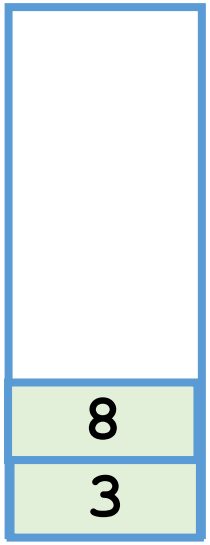



```
3 * (7 + 1) / 2 - 9
```

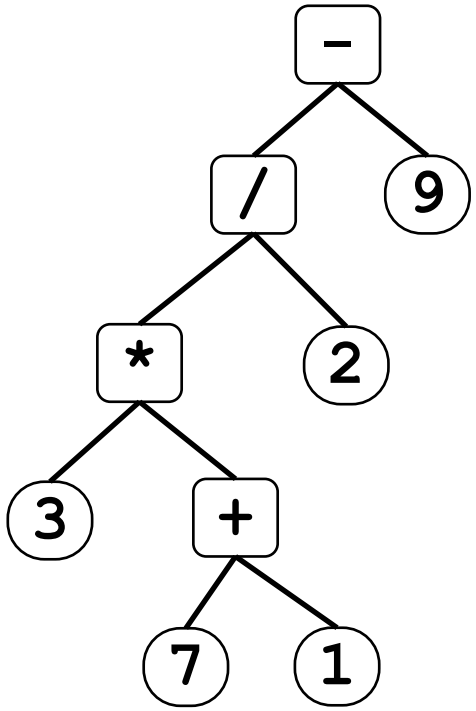


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

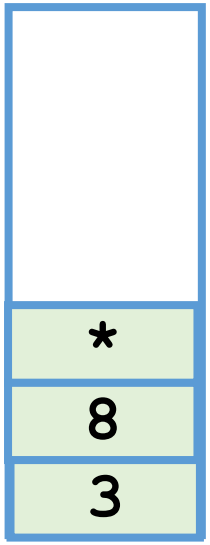



```
3 * (7 + 1) / 2 - 9
```

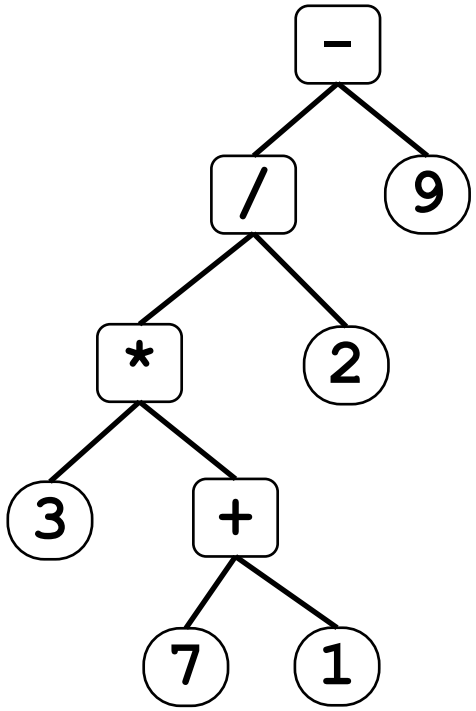


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

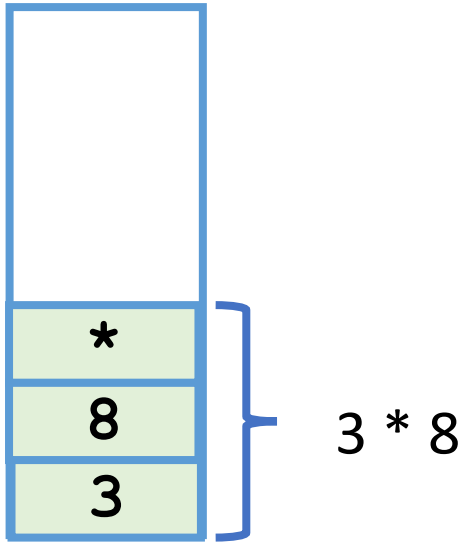



```
3 * (7 + 1) / 2 - 9
```

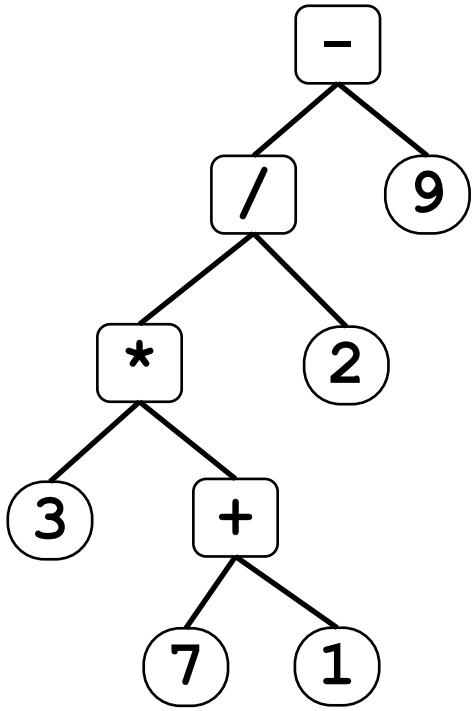


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

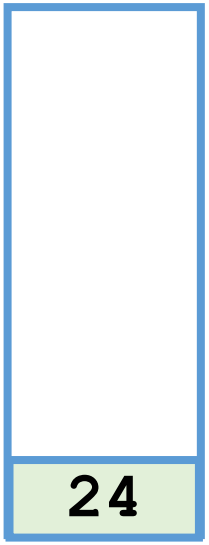



```
3 * (7 + 1) / 2 - 9
```

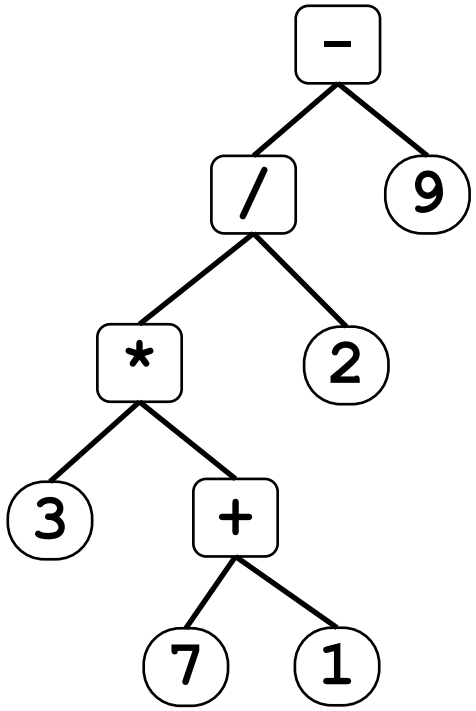


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

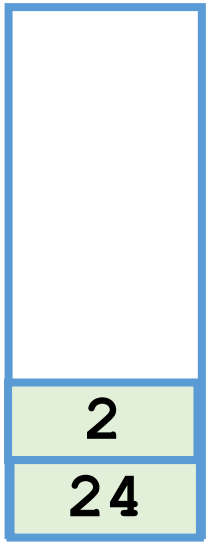



```
3 * (7 + 1) / 2 - 9
```

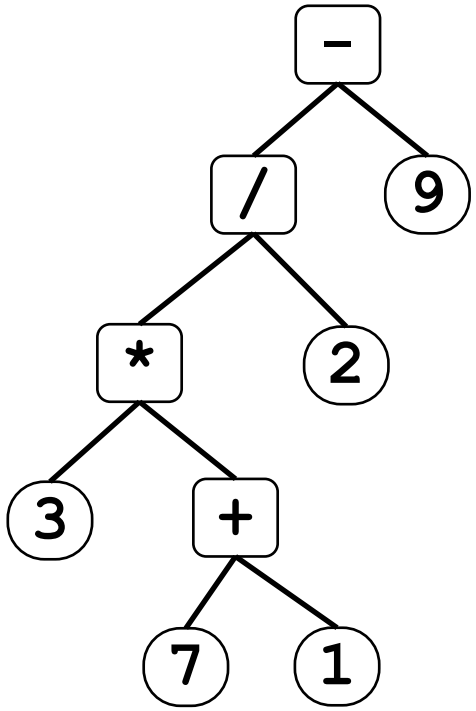


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

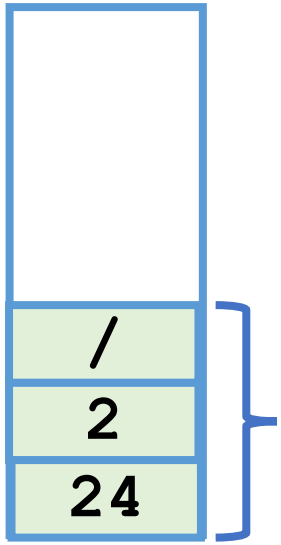



```
3 * (7 + 1) / 2 - 9
```



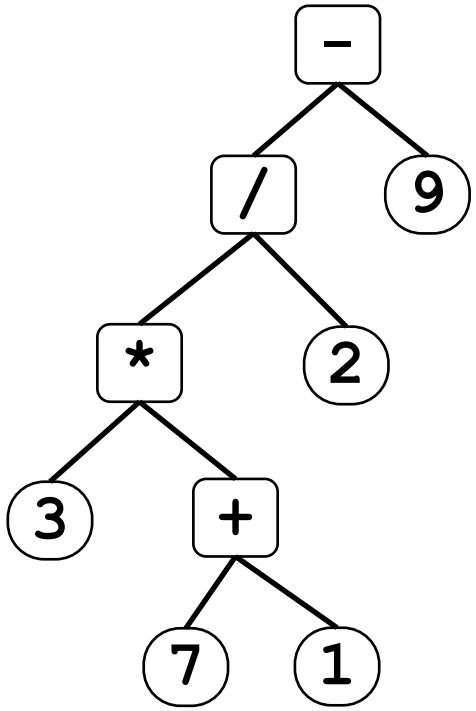
Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```



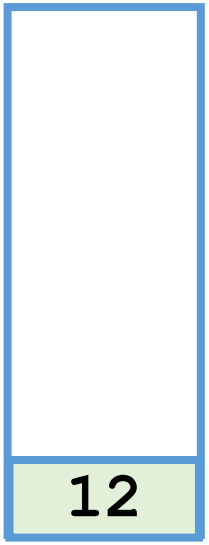
24 / 2

```
3 * (7 + 1) / 2 - 9
```

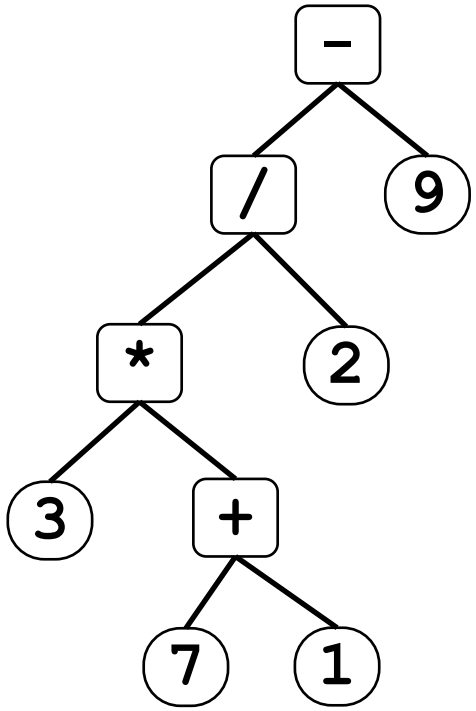


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

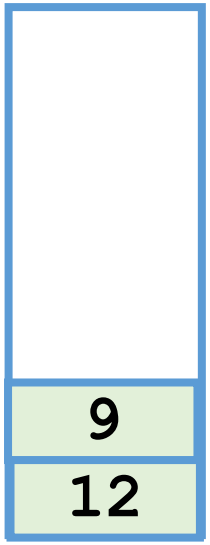



```
3 * (7 + 1) / 2 - 9
```

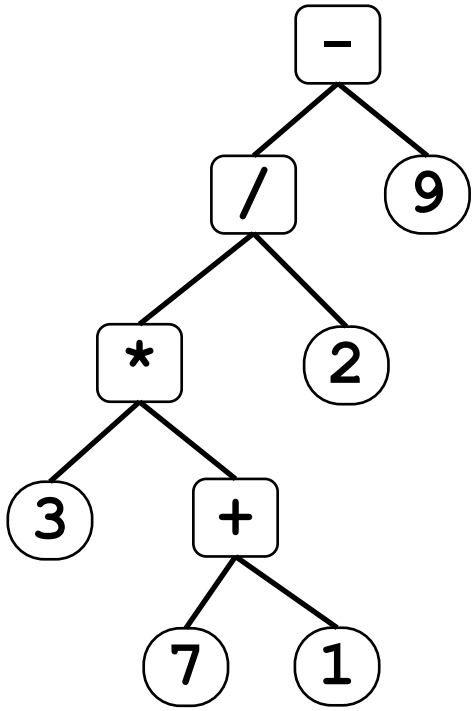


Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```

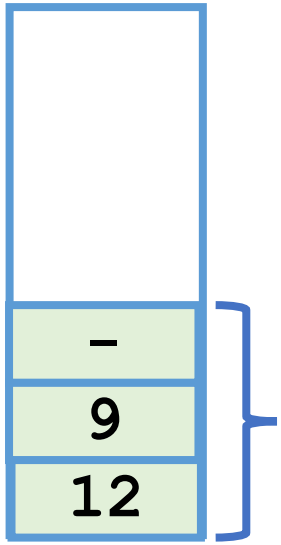



```
3 * (7 + 1) / 2 - 9
```



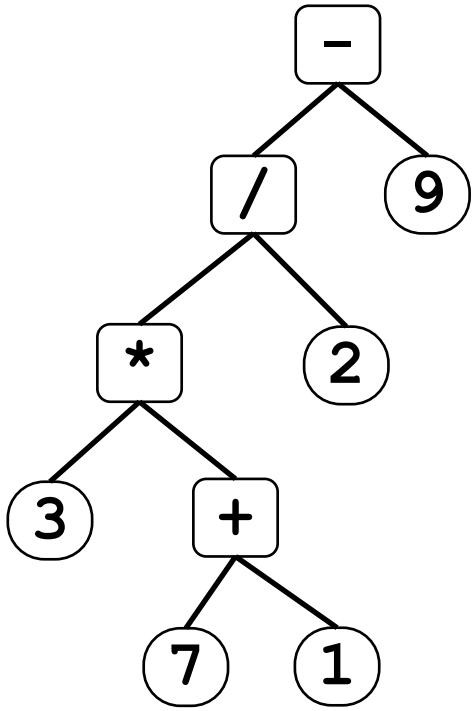
Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```



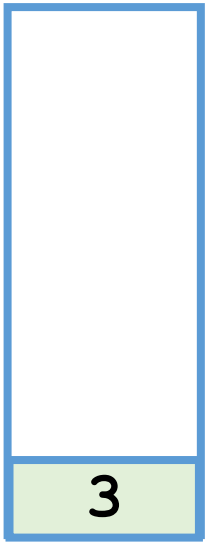
12 - 9

```
3 * (7 + 1) / 2 - 9
```



Post-order
Postfix form
Reverse Polish Notation

```
3 7 1 + * 2 / 9 -
```




```
3 * (7 + 1) / 2 - 9
```

Recursive Structure of Expressions

- In most programming languages, an ***expression*** is a recursive structure that can contain subexpressions.
- Every expression can have one of the following forms:
  - An integer constant
  - A variable name that holds an integer value
  - Two expressions joined by an operator
  - An expression enclosed in parentheses
- The rules for forming an expression can be expressed in the form of a ***grammar***, as follows:

E → *constant*	**1024**
E → *identifier*	**temp**
E → **E** *op* **E**	**temp + 1024**
E → **(E)**	**(temp + 1024)**

A Two-Level Grammar

- The problem of parsing an expression can be simplified by changing the grammar to one that has two levels:
  - An ***expression*** is either a *term* or two expressions joined by an operator.
  - A ***term*** is either a constant, an identifier, or an expression enclosed in parentheses.
- This design is reflected in the following revised grammar.

E → **T**

E → **E** *op* **E**

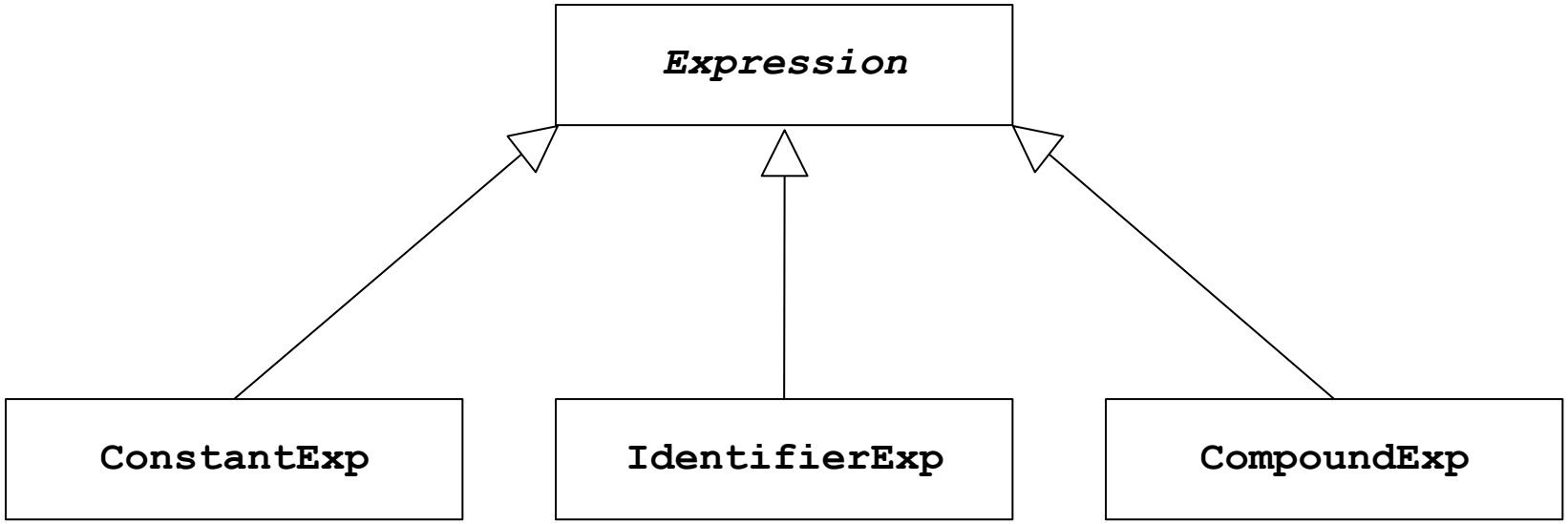
T → *constant*

T → *identifier*

T → **(E)**

The **Expression** Class Hierarchy

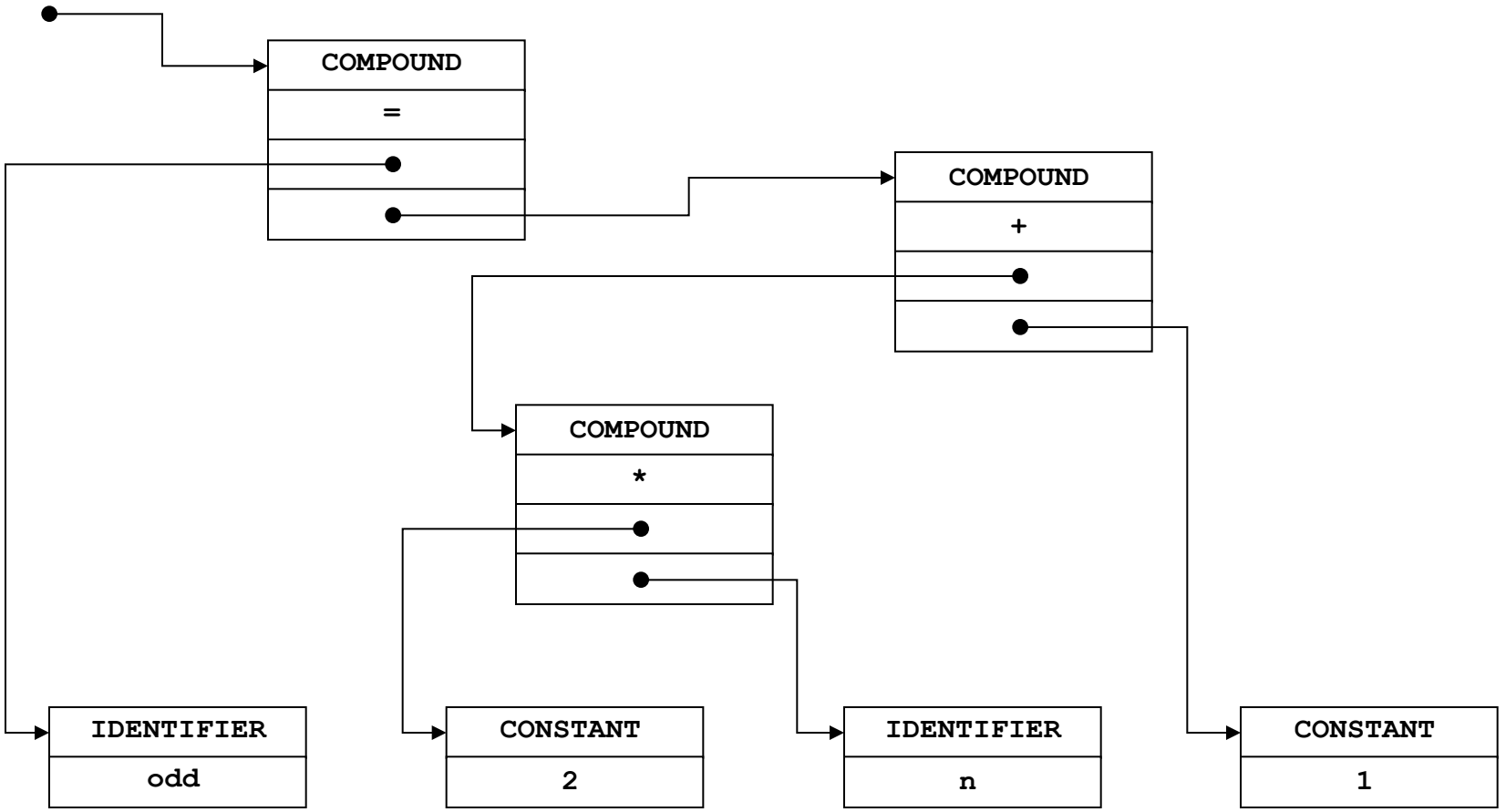
- Because expressions have more than one form, a C++ class that represents expressions can be represented most easily by a class hierarchy in which each of the expression types is a separate subclass, as shown in the following diagram:



Parsing an Expression

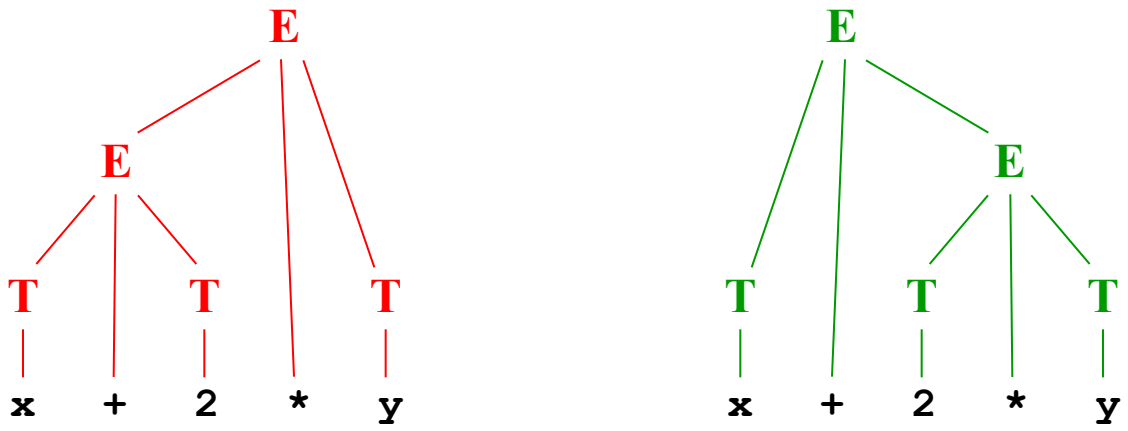
- ***Parsing***: Convert the string representation to an internal form

```
odd = 2 * n + 1
```



Ambiguity in Parse Structures

- Although the two-level grammar from the preceding slide can recognize any expression, it is ***ambiguous*** because the same input string can generate more than one parse tree.



- Ambiguity in grammars is typically resolved by providing the parser with information about the ***precedence*** of the operators.

Precedence

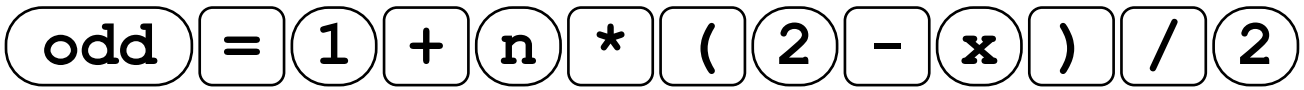
precedence	operators
3	* /
2	+ -
1	=

Parsing an expression

- Several approaches
  - Recursive parsing
  - Infix parsing using stacks

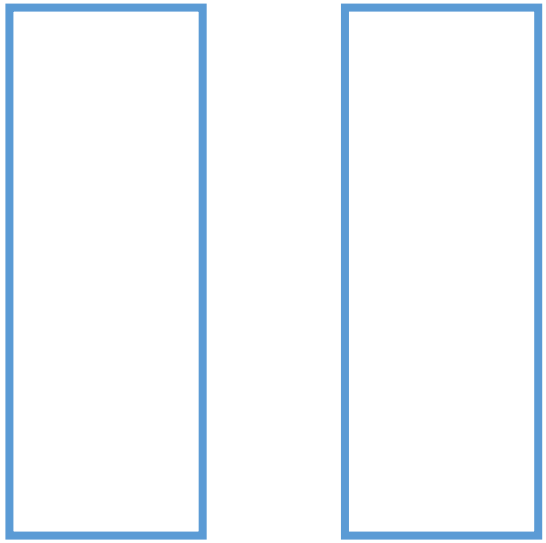
Example: Parsing using stacks

```
Odd = 1 + n * (2 - x) / 2
```




```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=

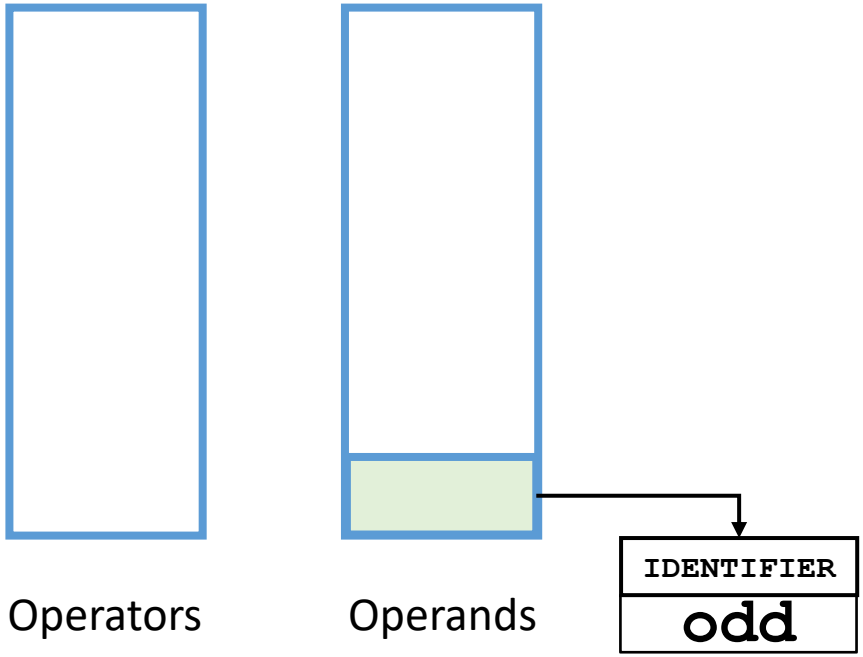


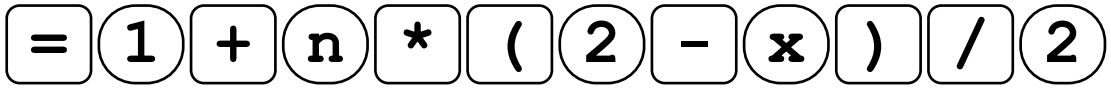
Operators Operands

odd = 1 + n * (2 - x) / 2

Odd = 1 + n * (2 - x) / 2

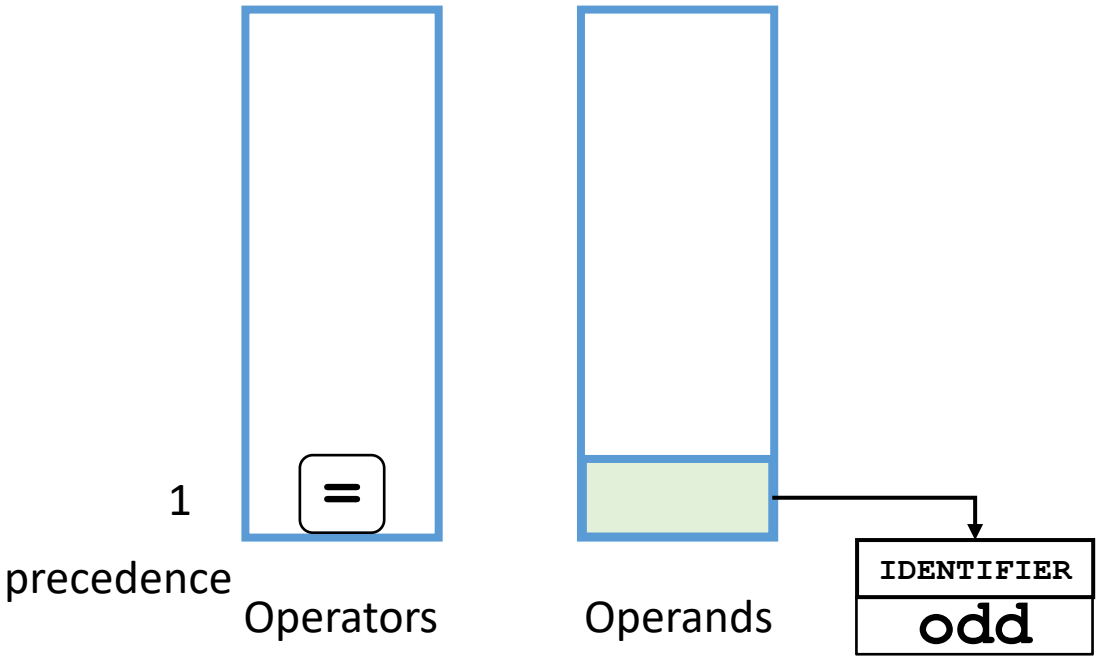
precedence	operators
3	* /
2	+ -
1	=





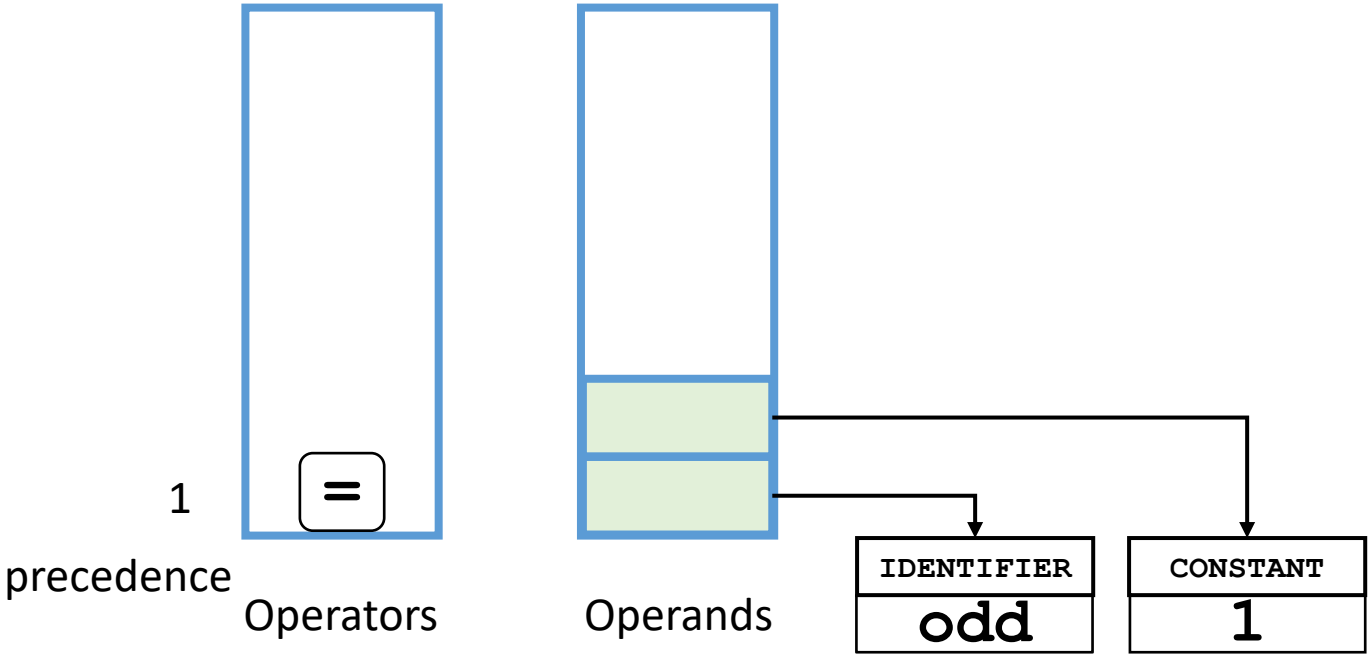

```
Odd = 1 + n * (2 - x) / 2
```

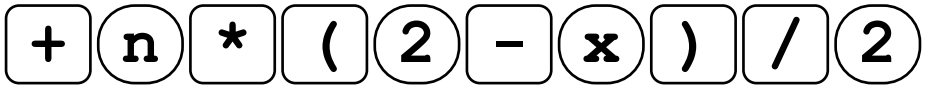
precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

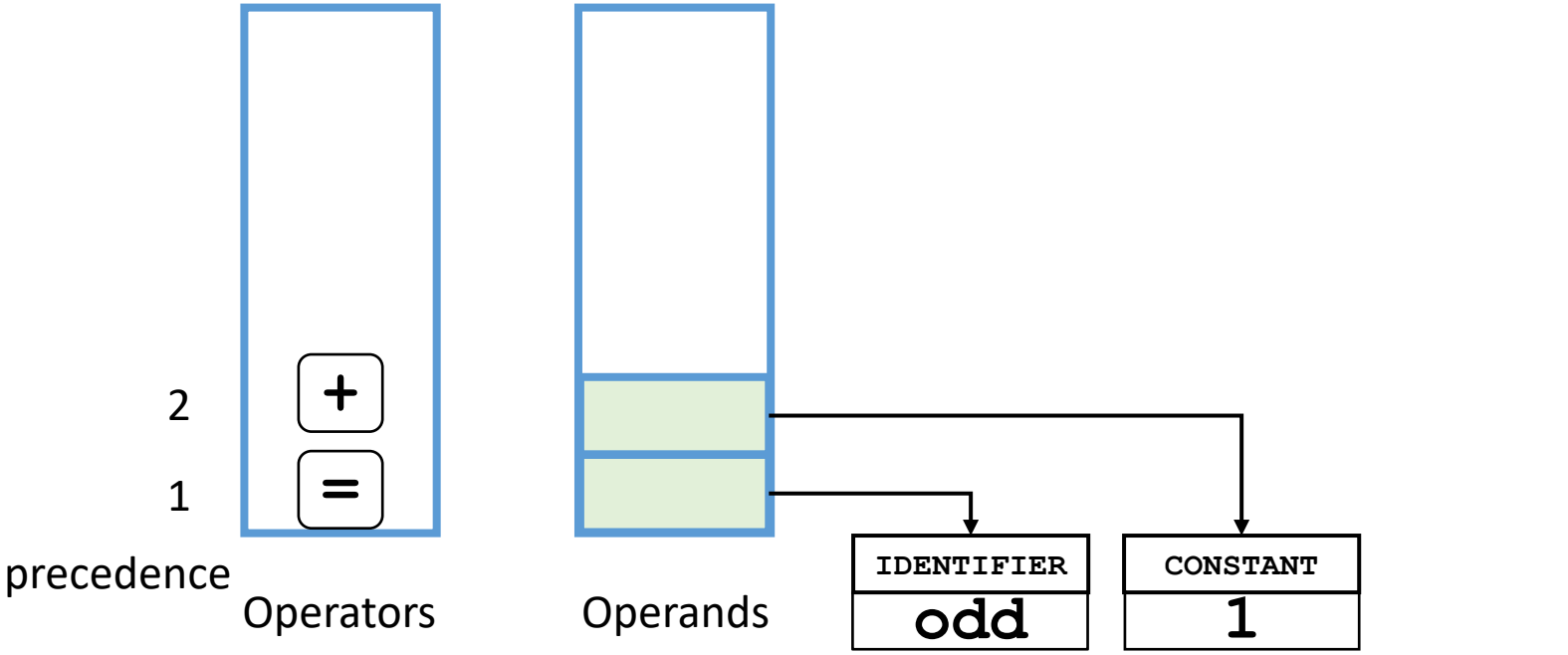
precedence	operators
3	* /
2	+ -
1	=

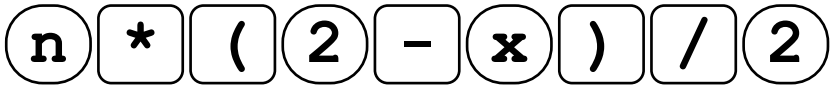





```
Odd = 1 + n * (2 - x) / 2
```

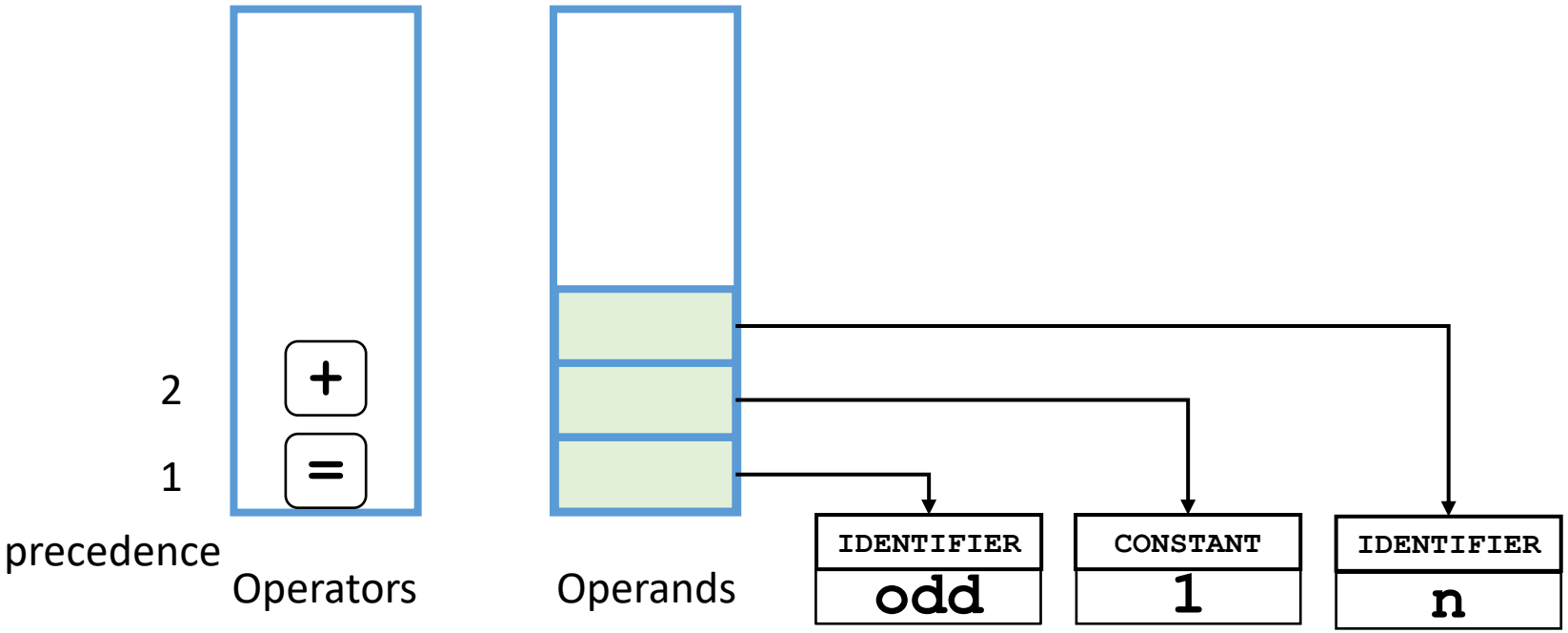
precedence	operators
3	* /
2	+ -
1	=

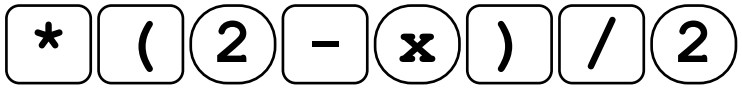





```
Odd = 1 + n * (2 - x) / 2
```

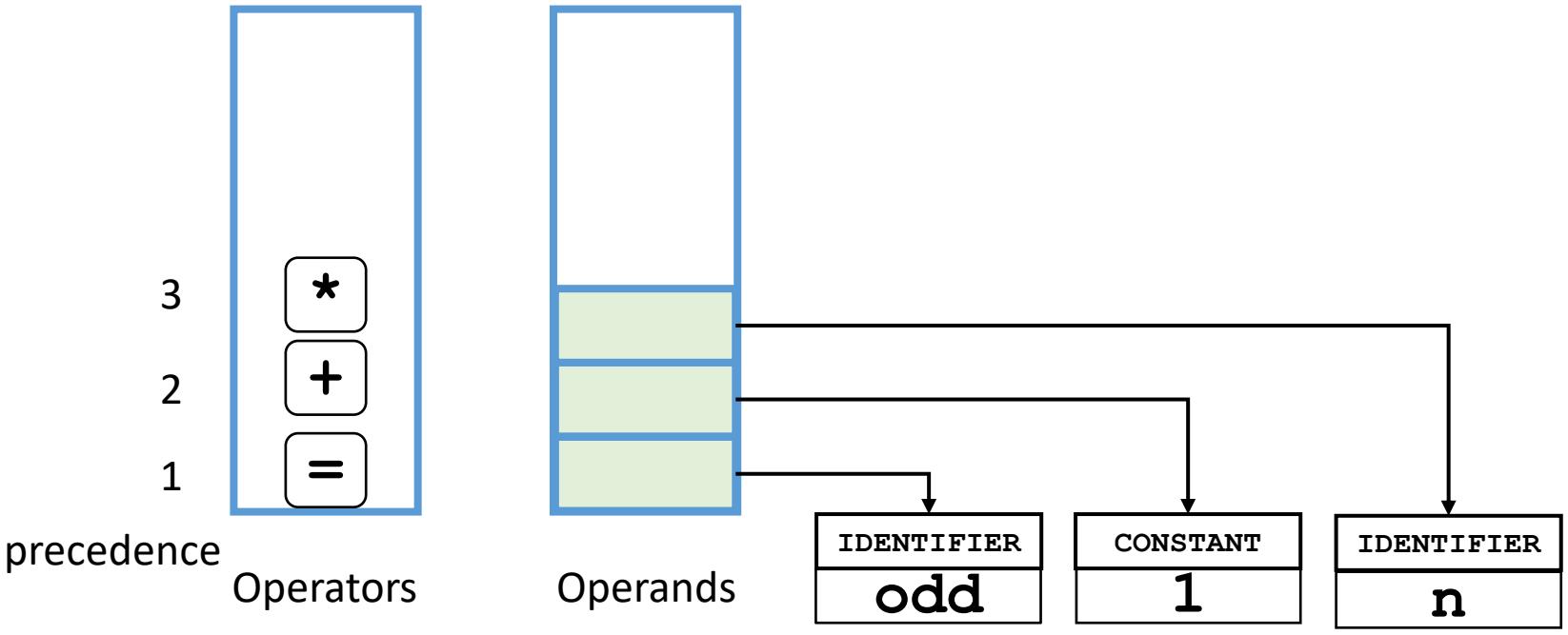
precedence	operators
3	* /
2	+ -
1	=






```
Odd = 1 + n * (2 - x) / 2
```

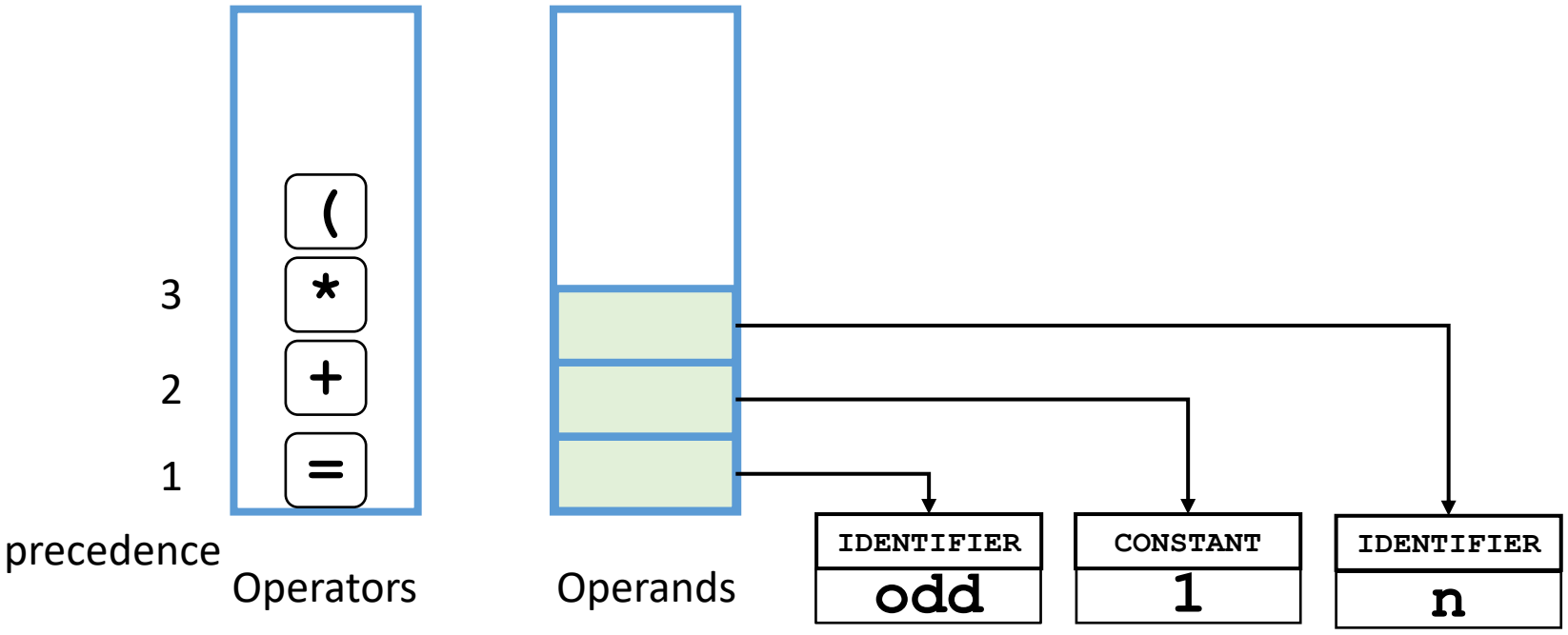
precedence	operators
3	* /
2	+ -
1	=



(2 - x) / 2

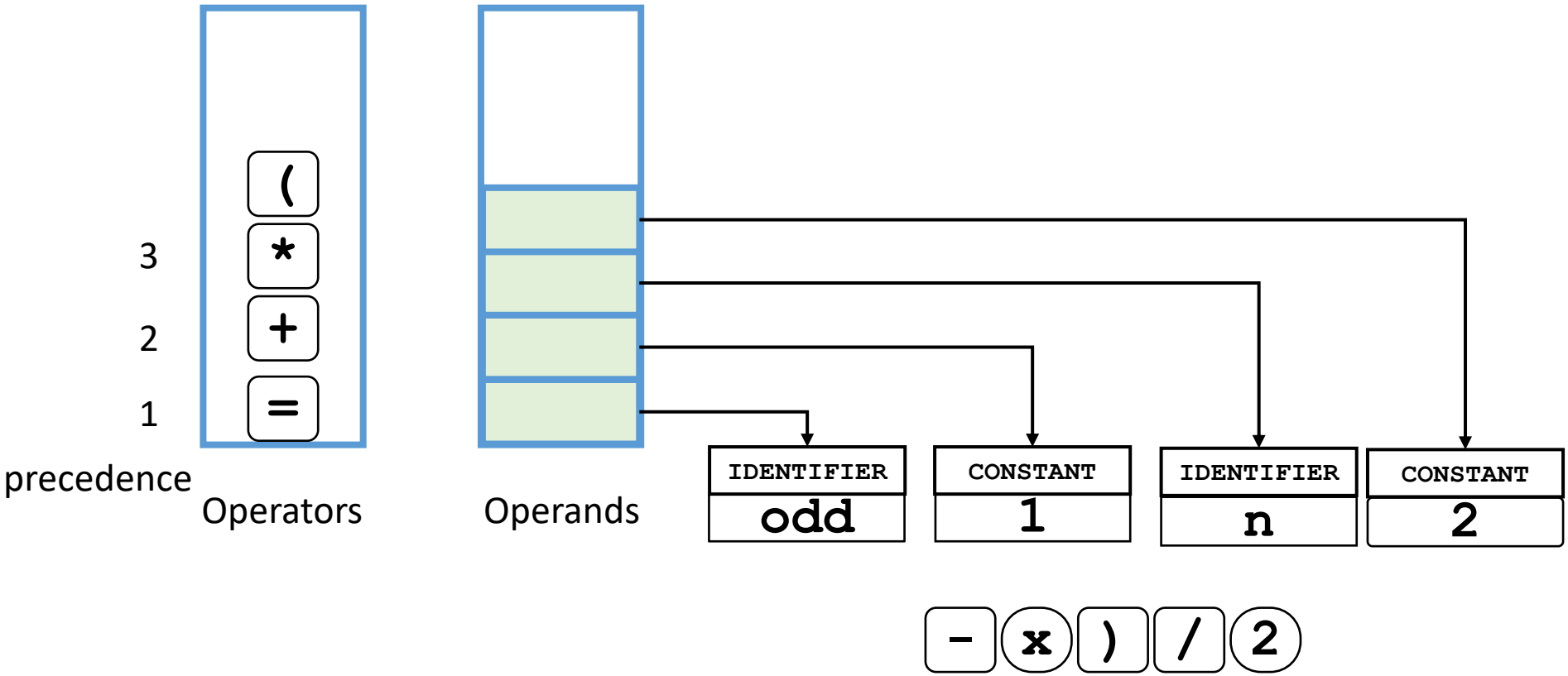
```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



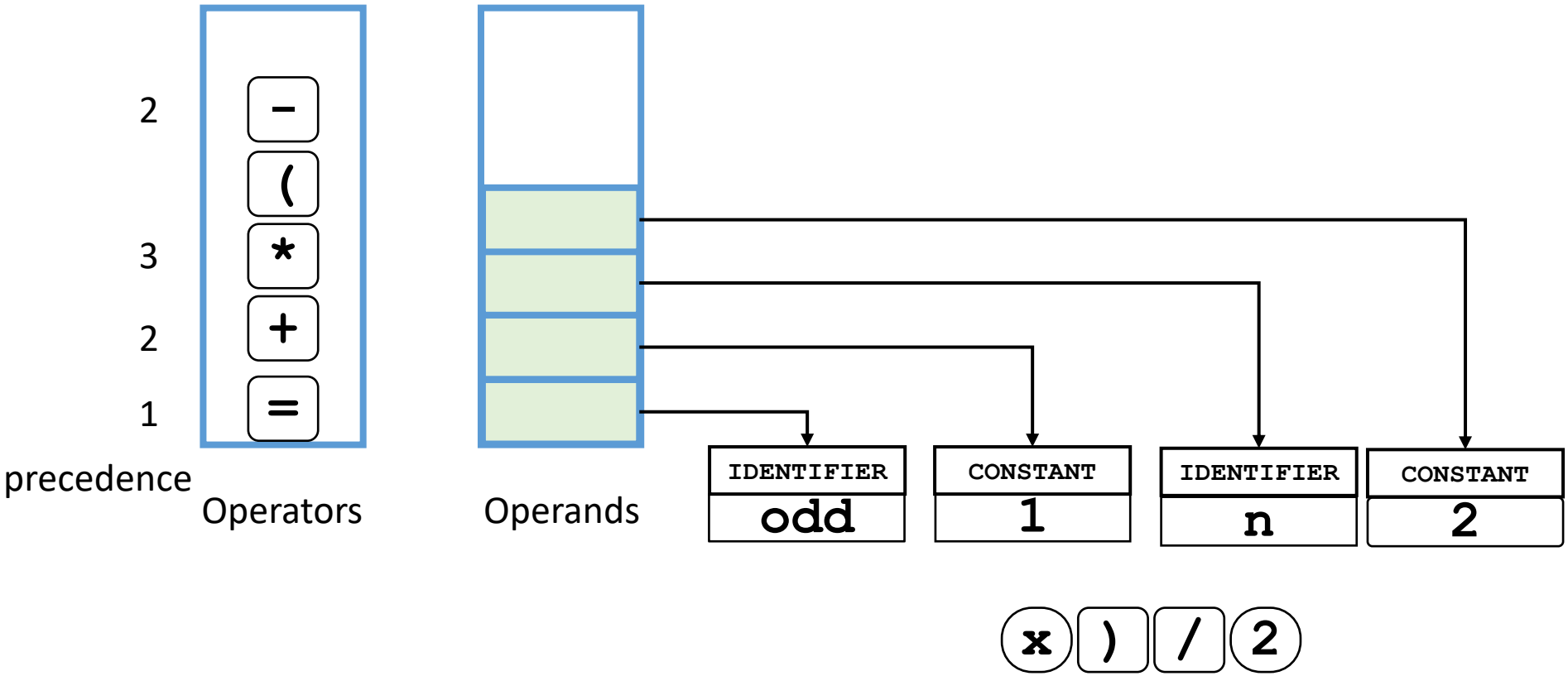

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



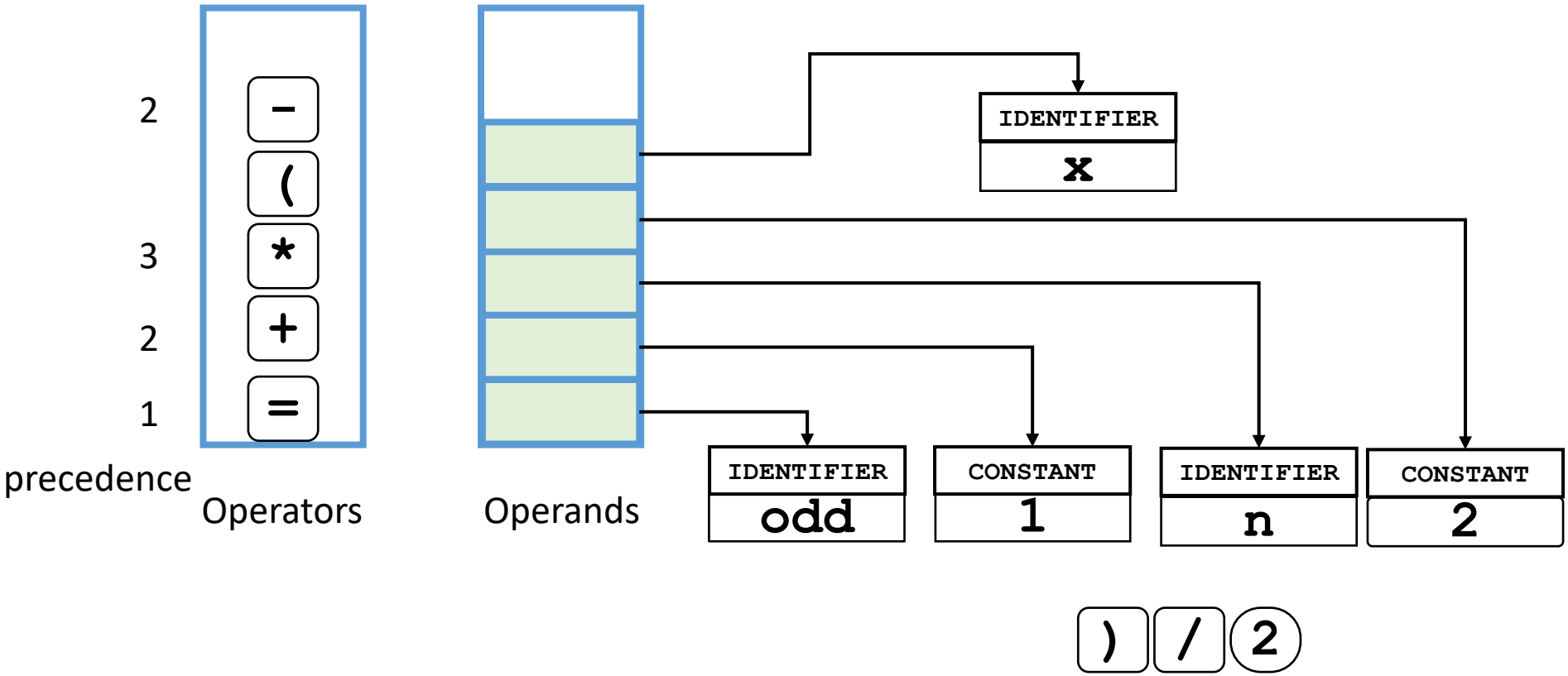

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



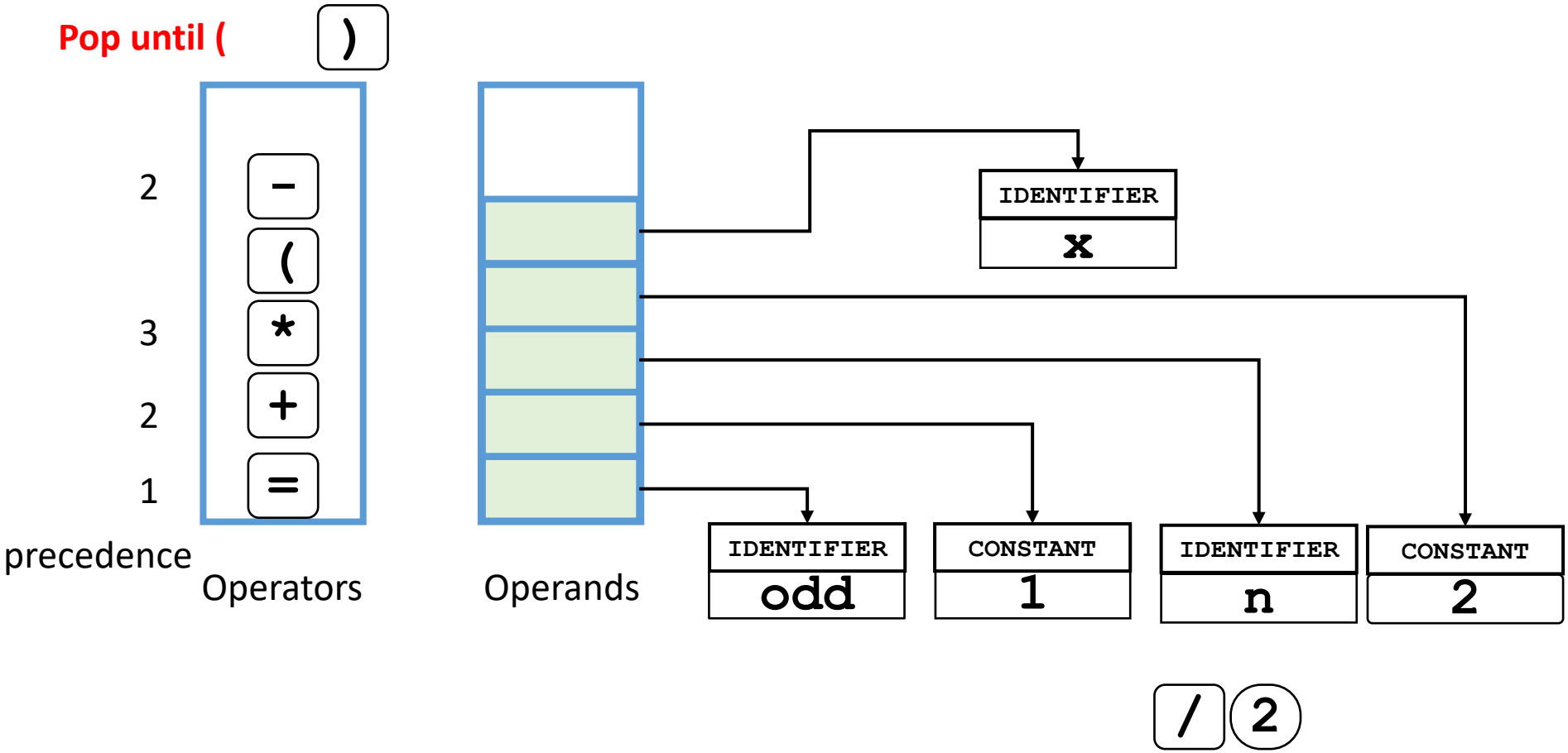

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



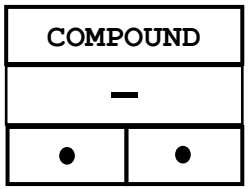

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=

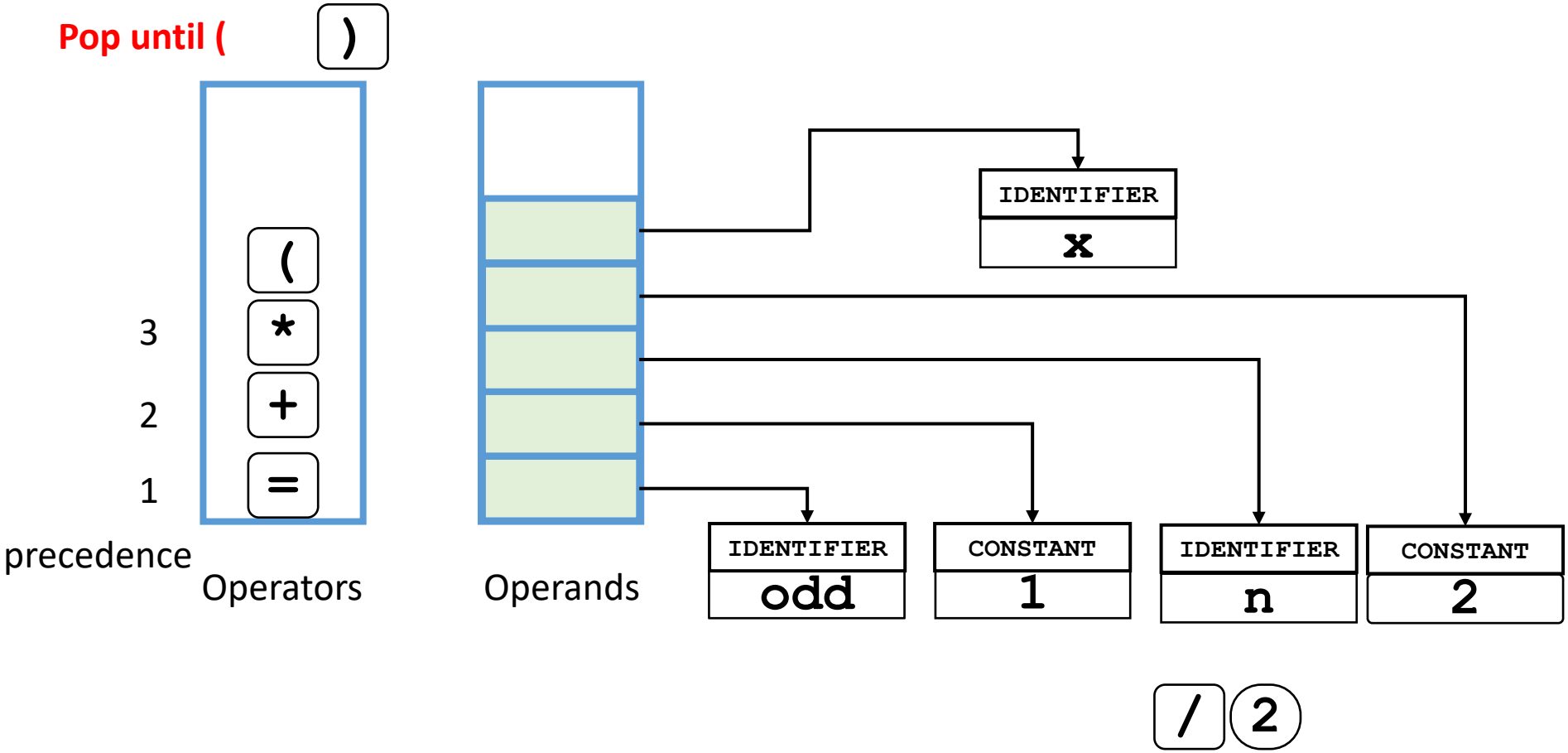



```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=

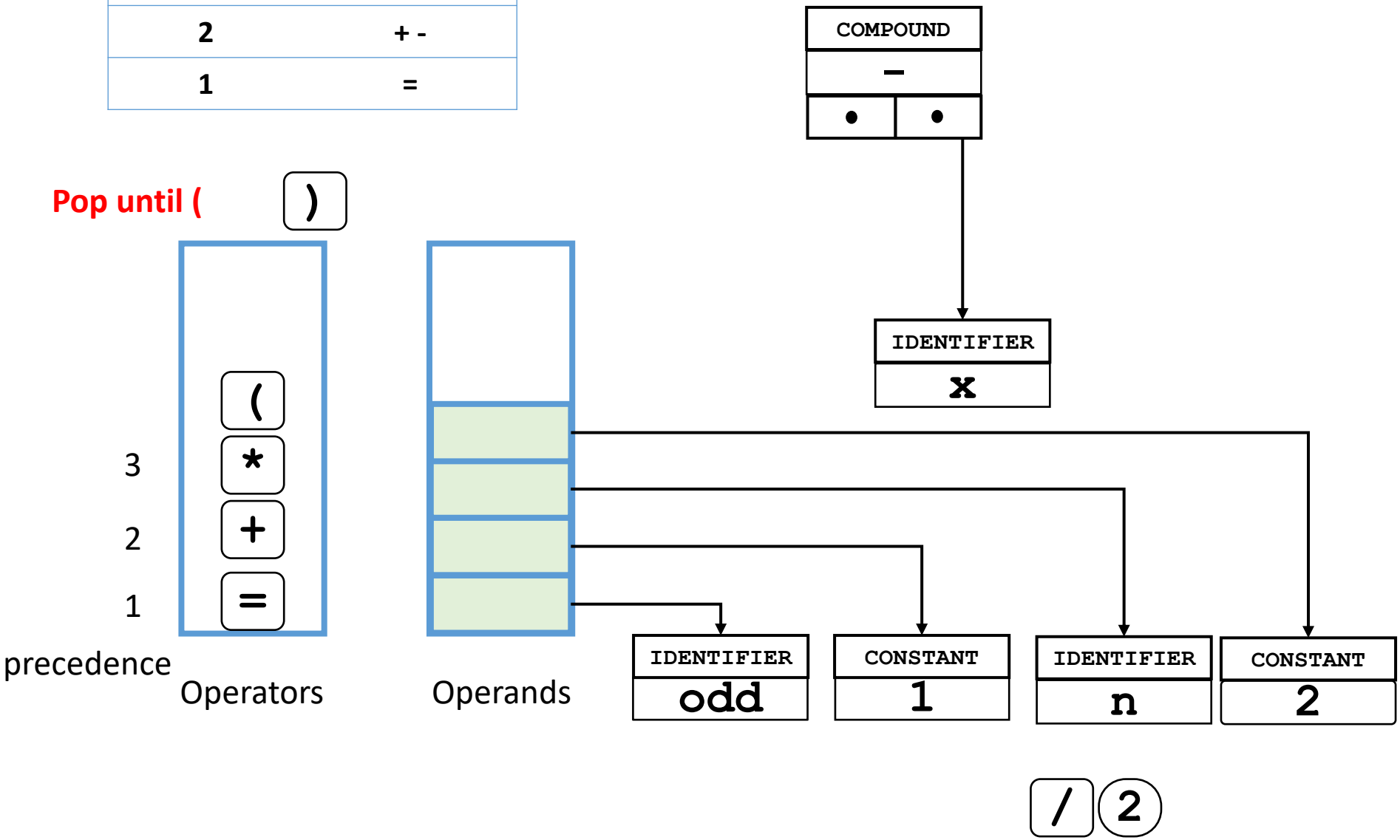


Pop until (



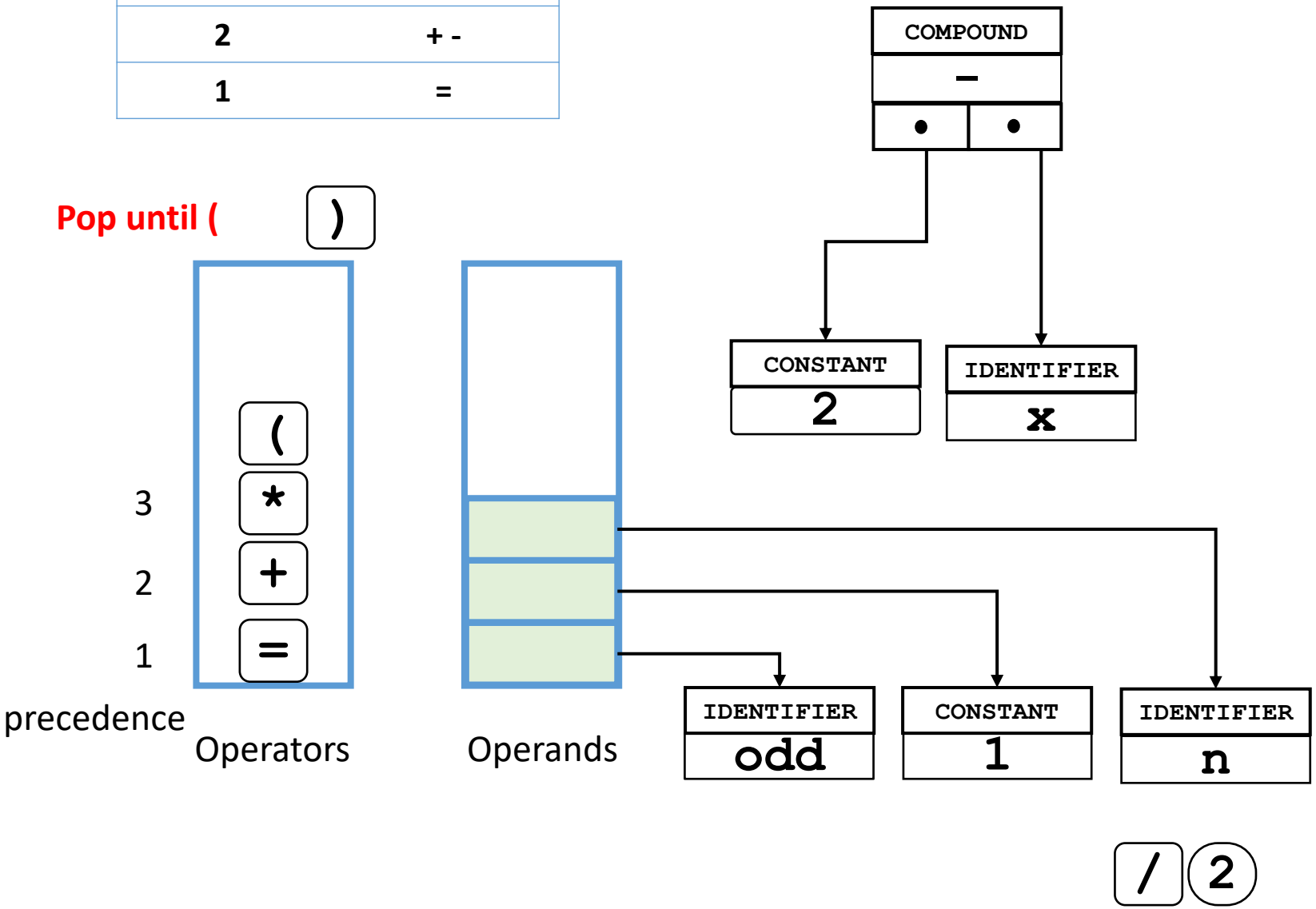

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

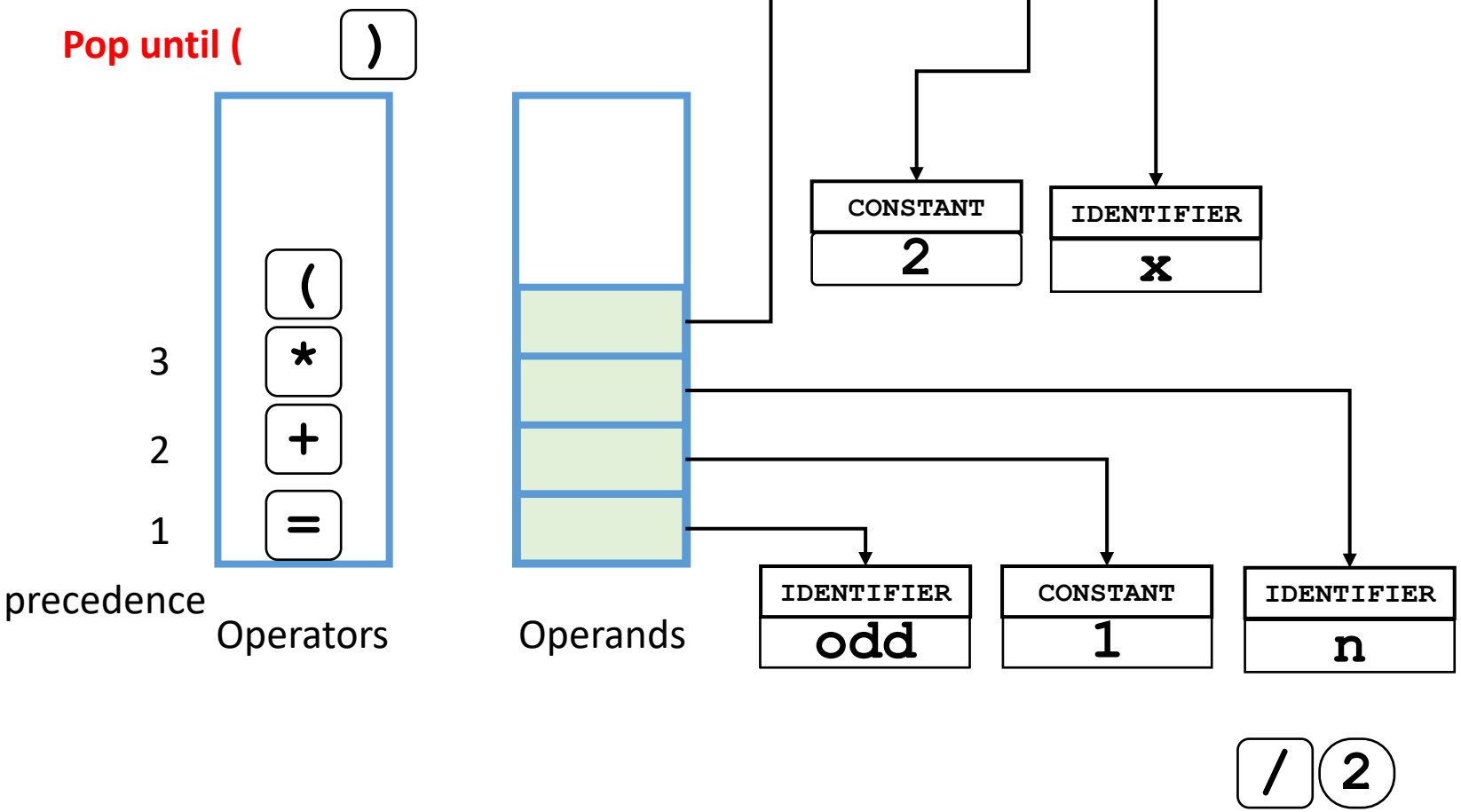
precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

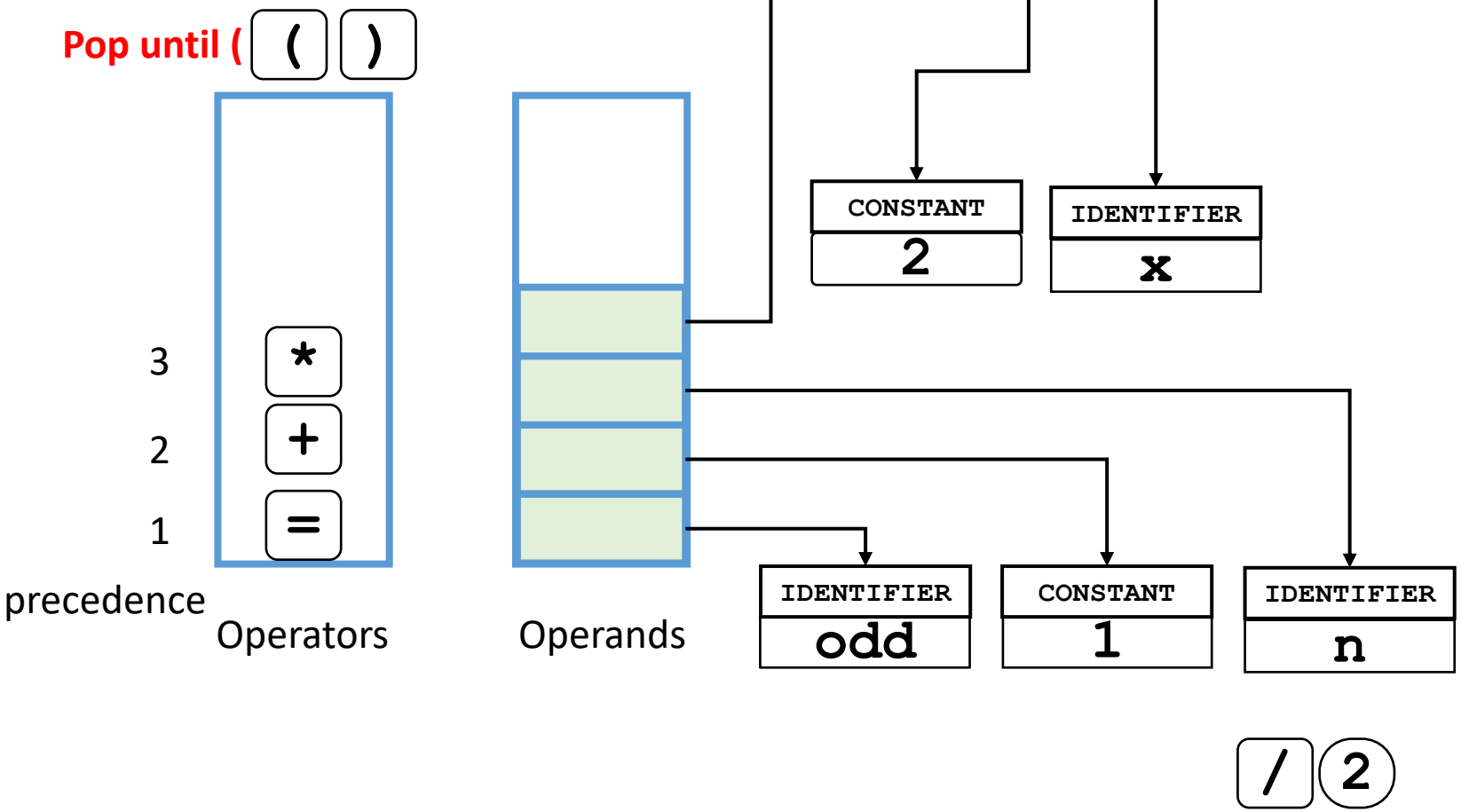
precedence	operators
3	* /
2	+ -
1	=

Pop until ( )



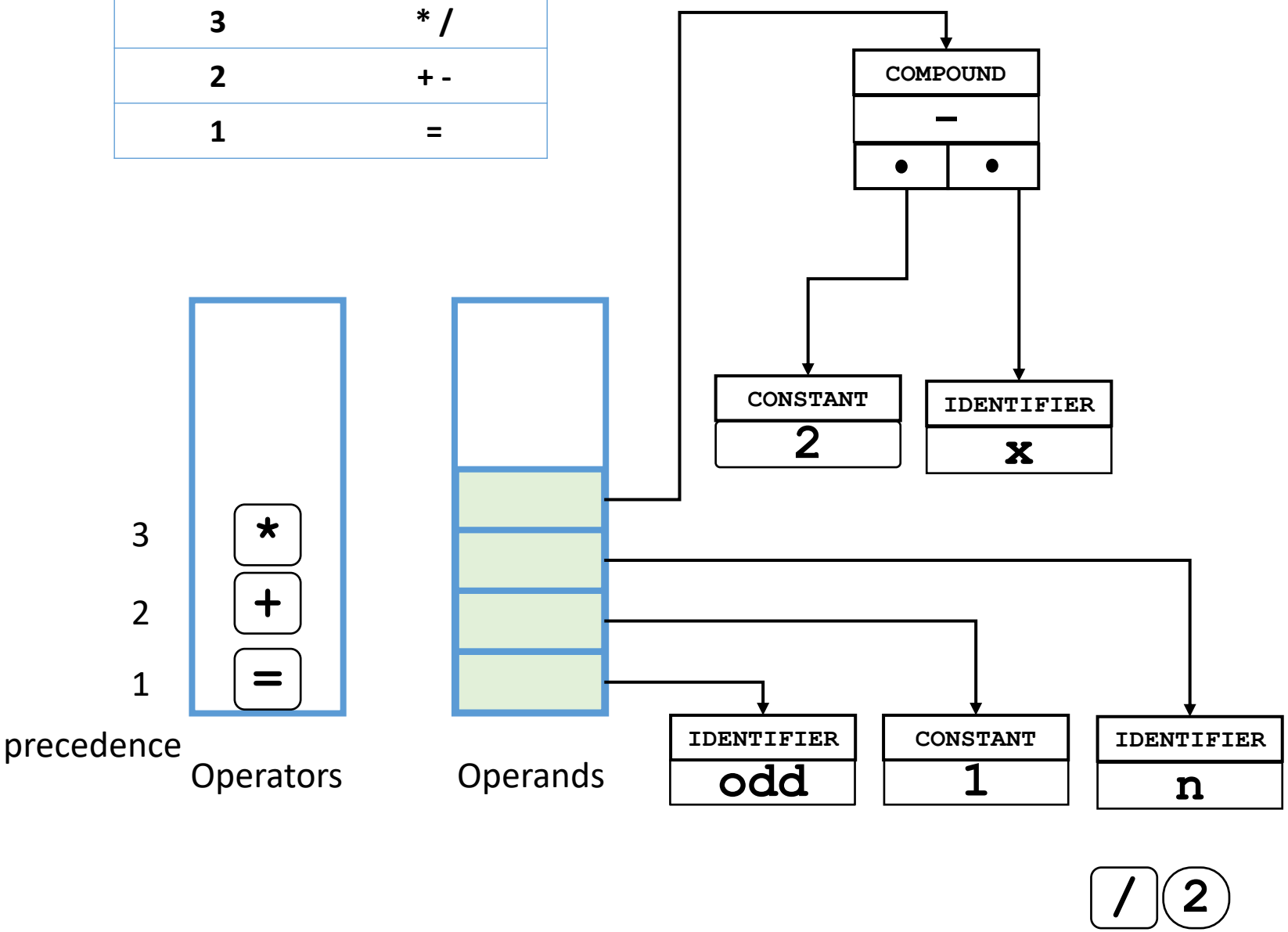

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

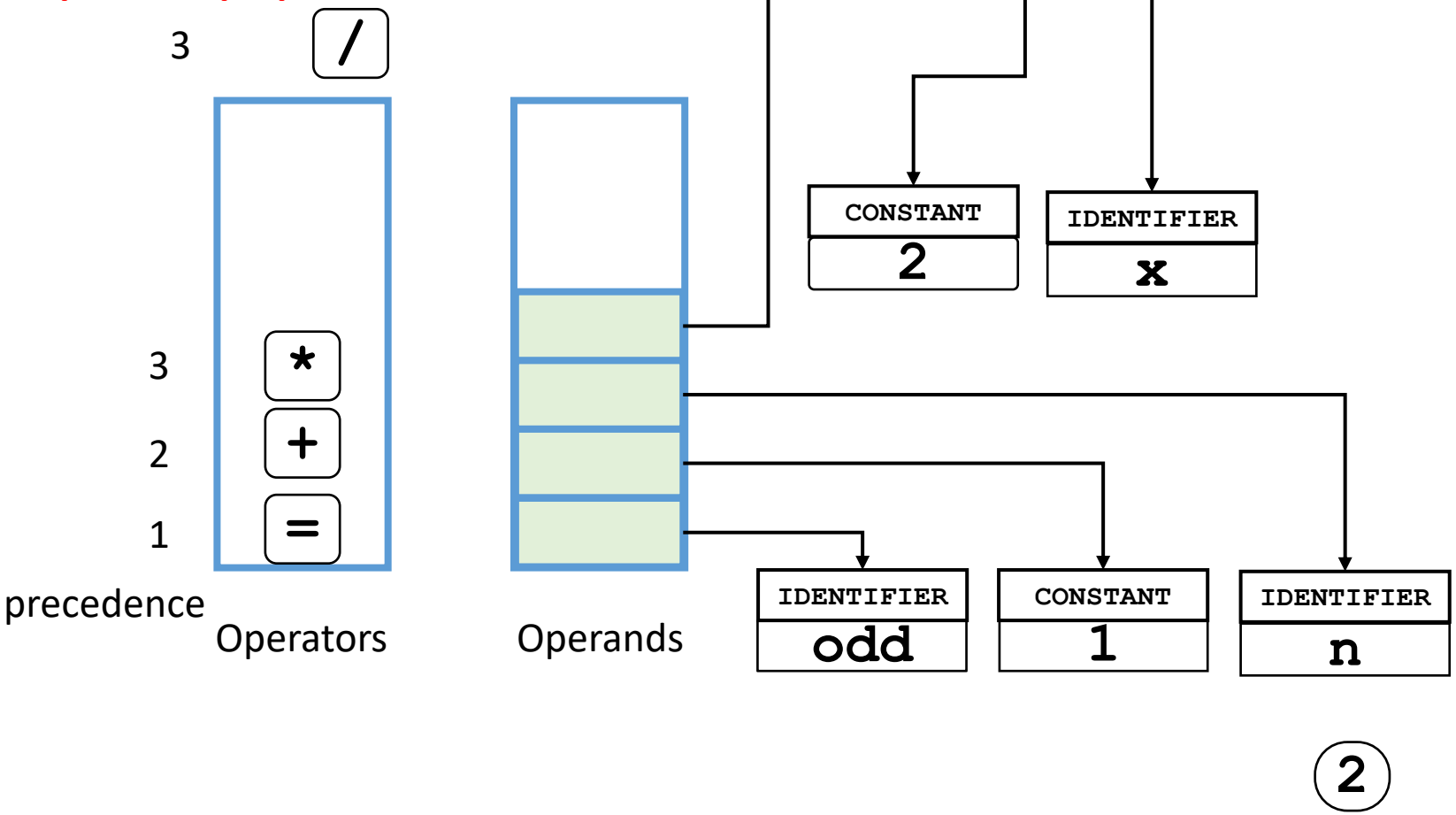
precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=

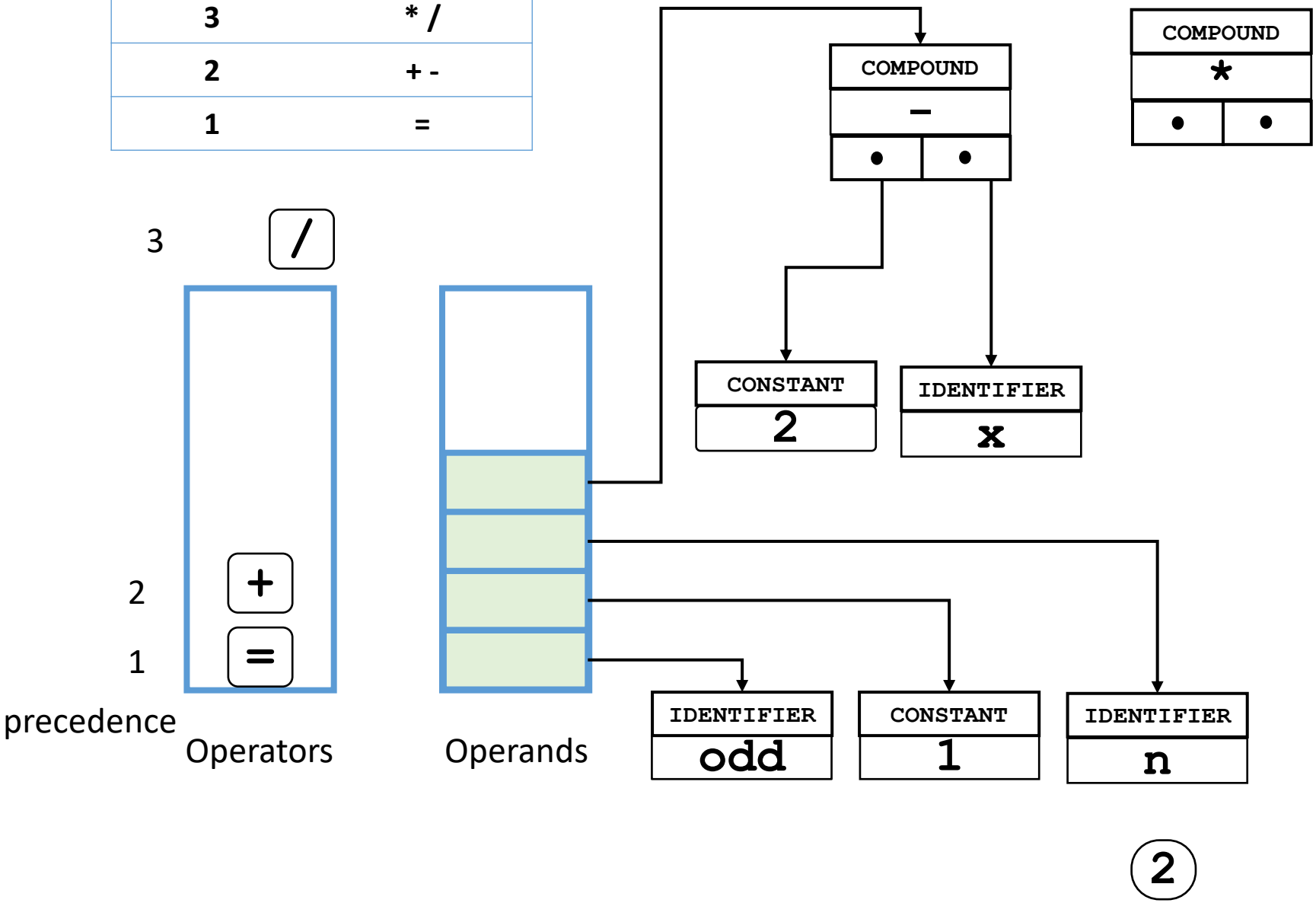
Pop until top's precedence < 3



2

```
Odd = 1 + n * (2 - x) / 2
```

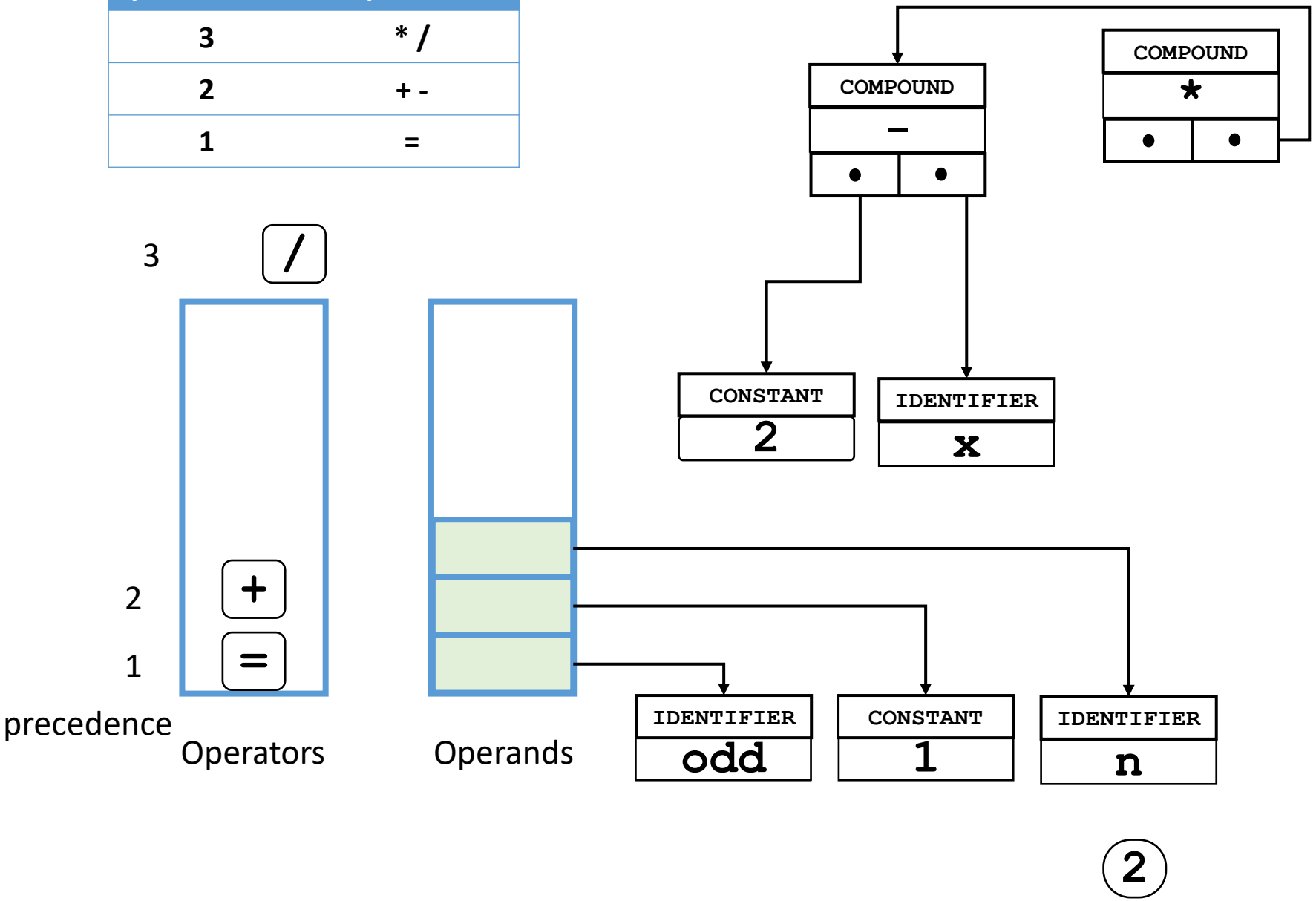
precedence	operators
3	* /
2	+ -
1	=



2

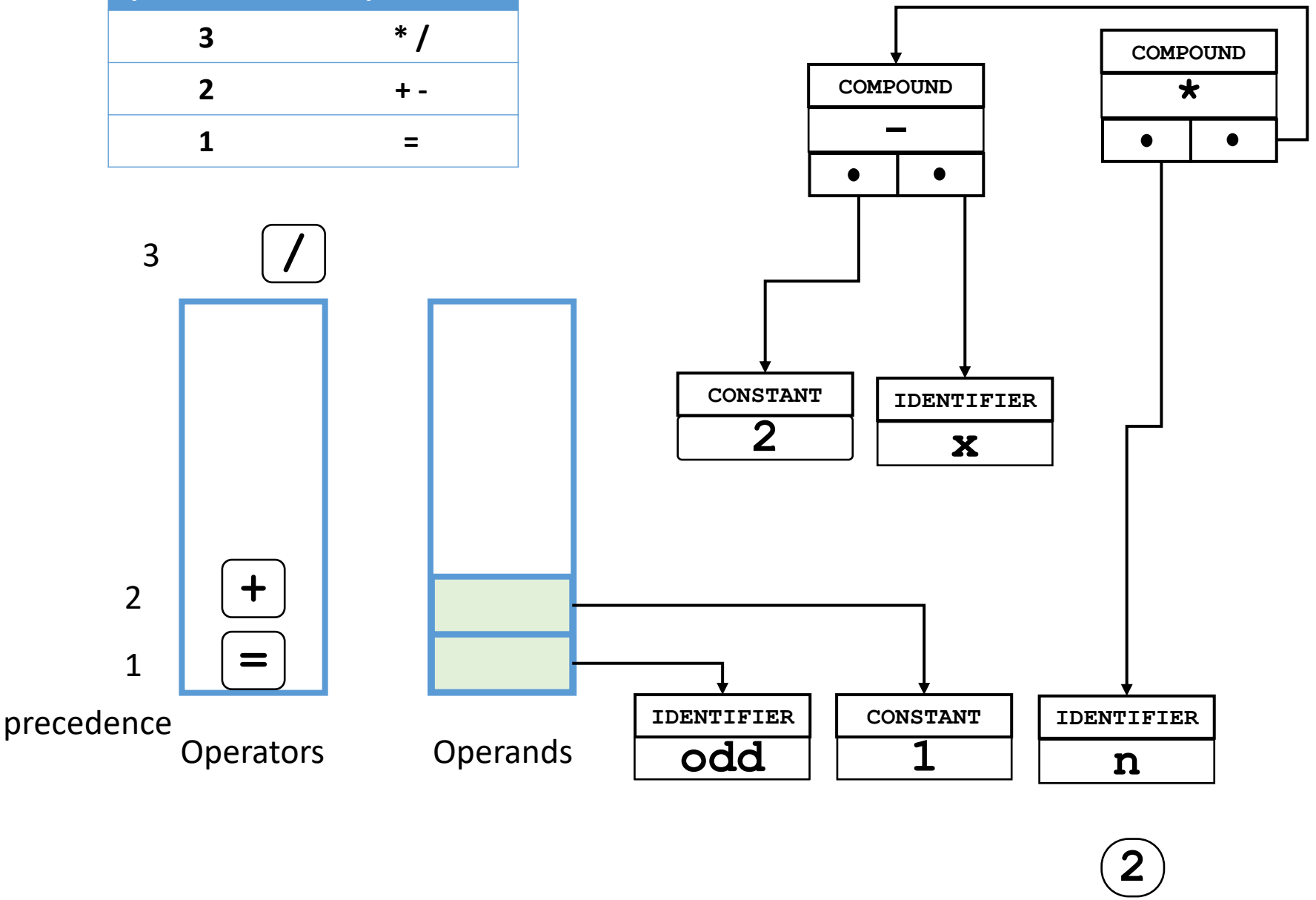
```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



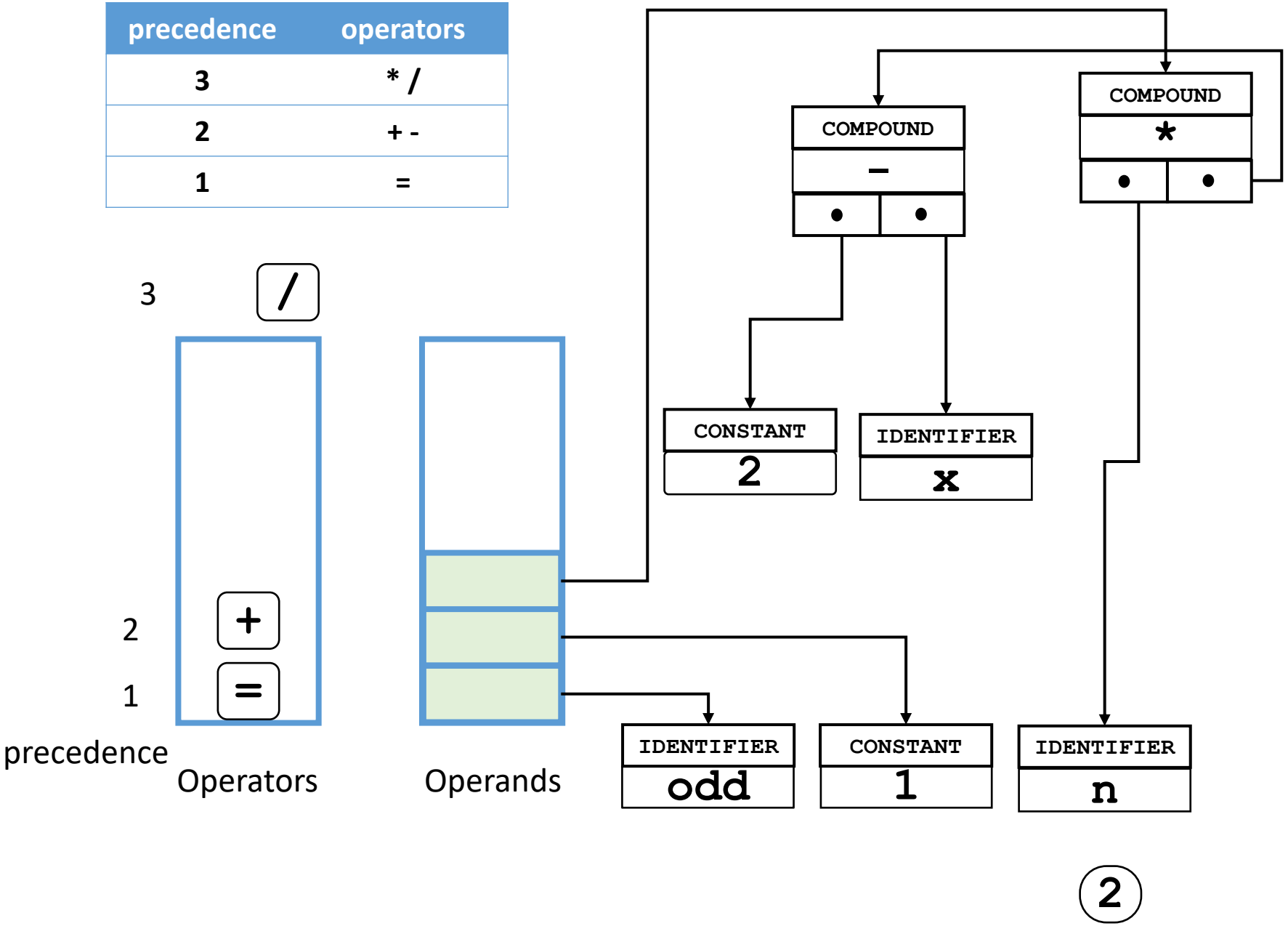

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



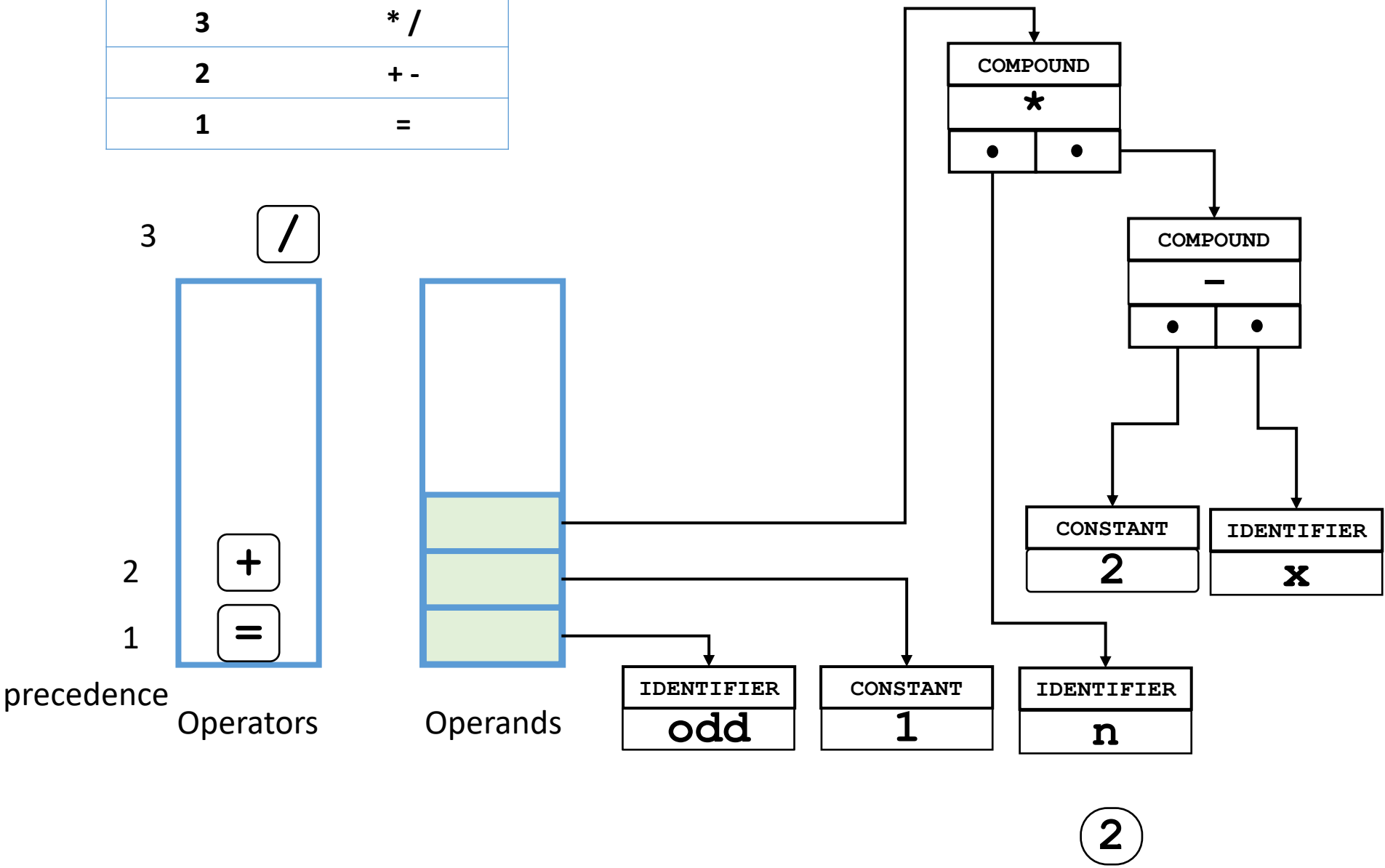

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



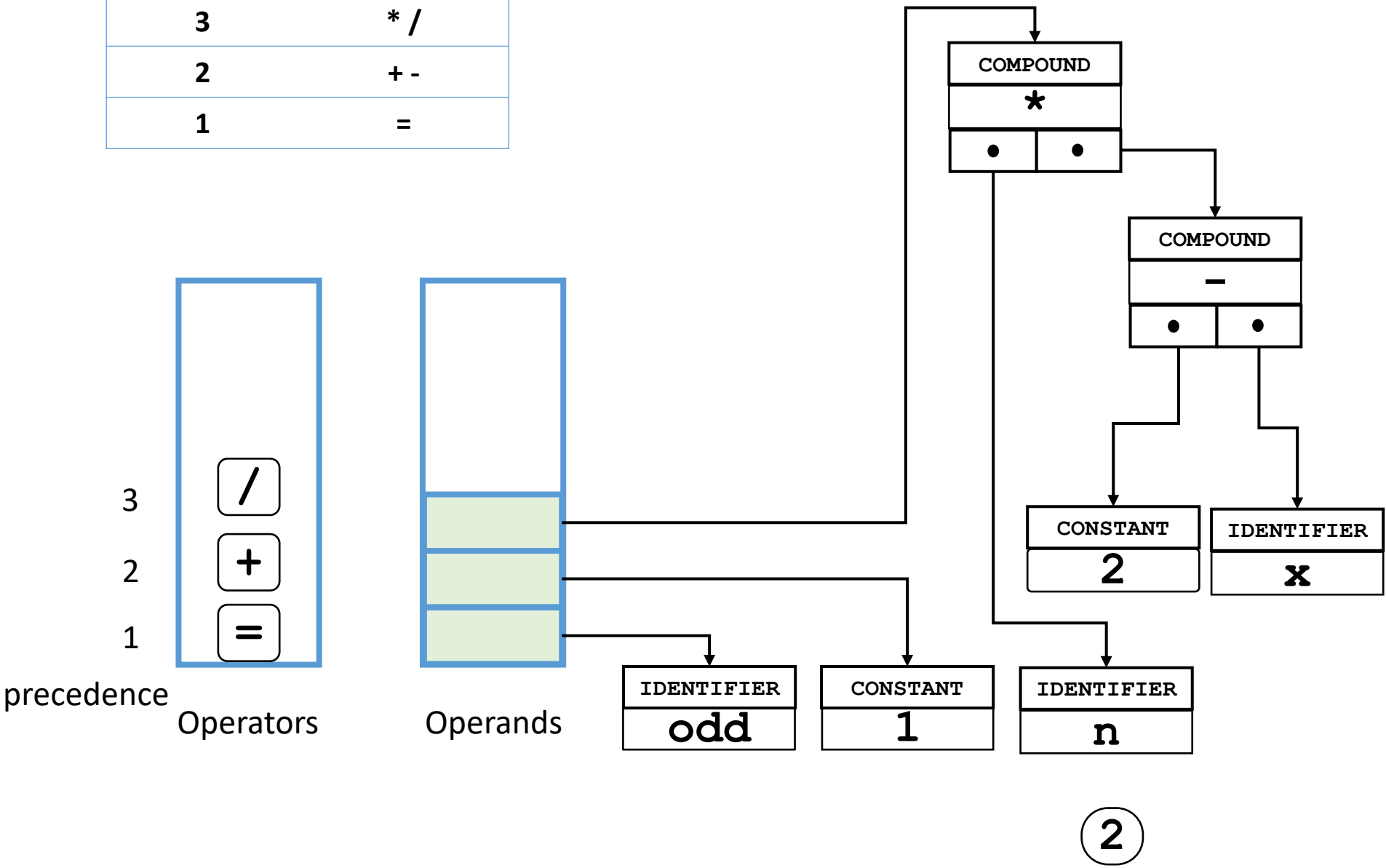

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

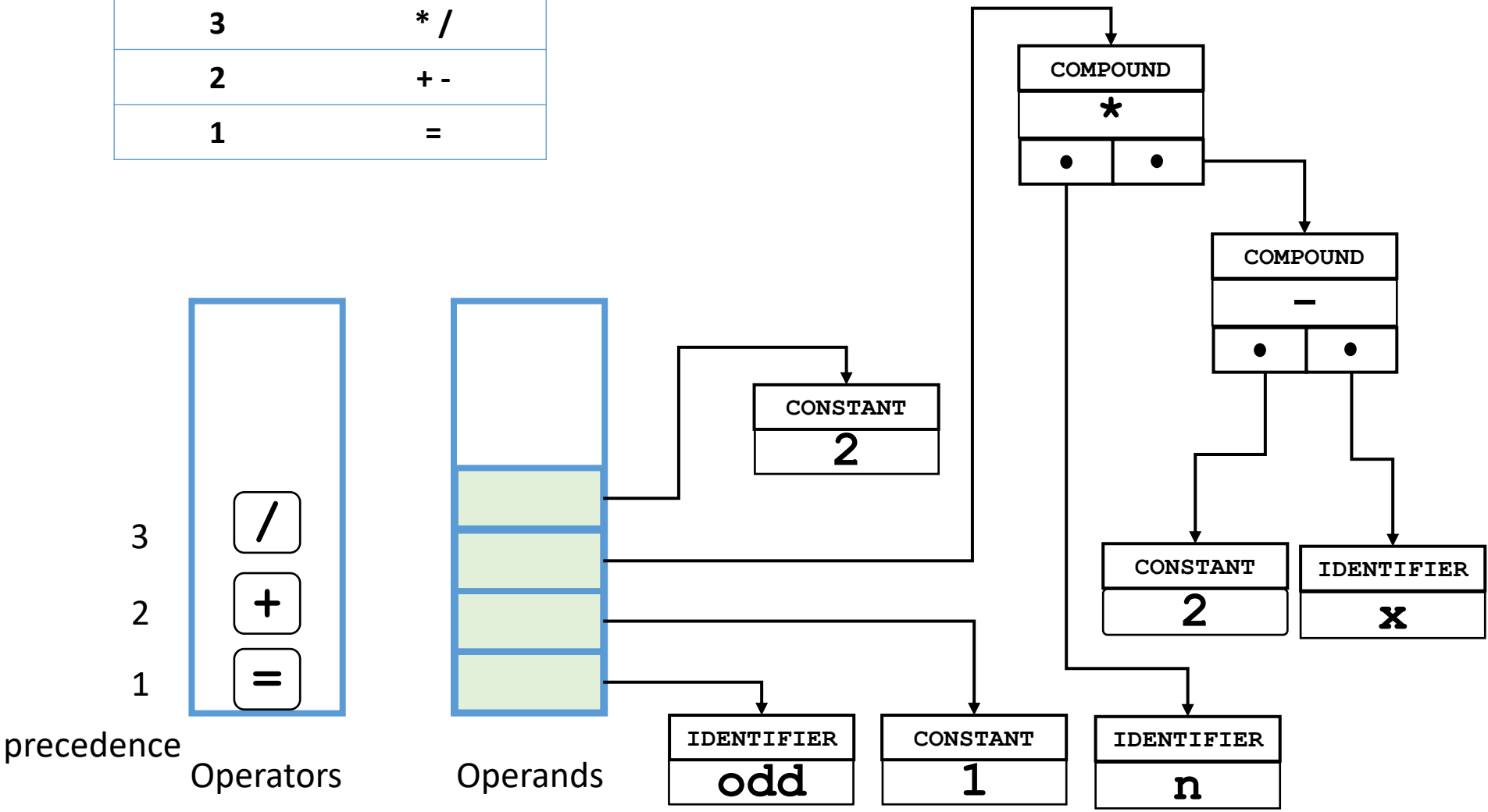
precedence	operators
3	* /
2	+ -
1	=



2

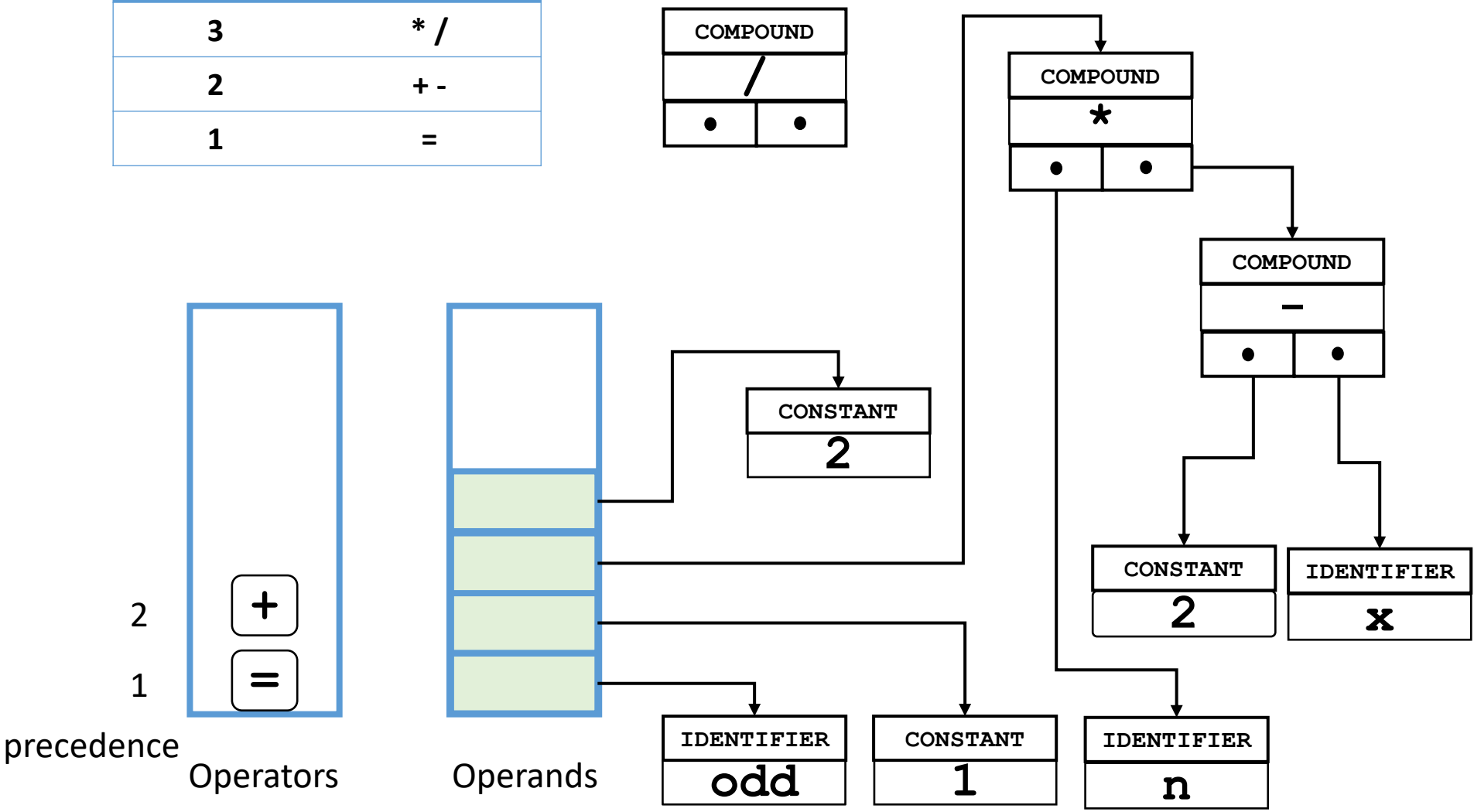
```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



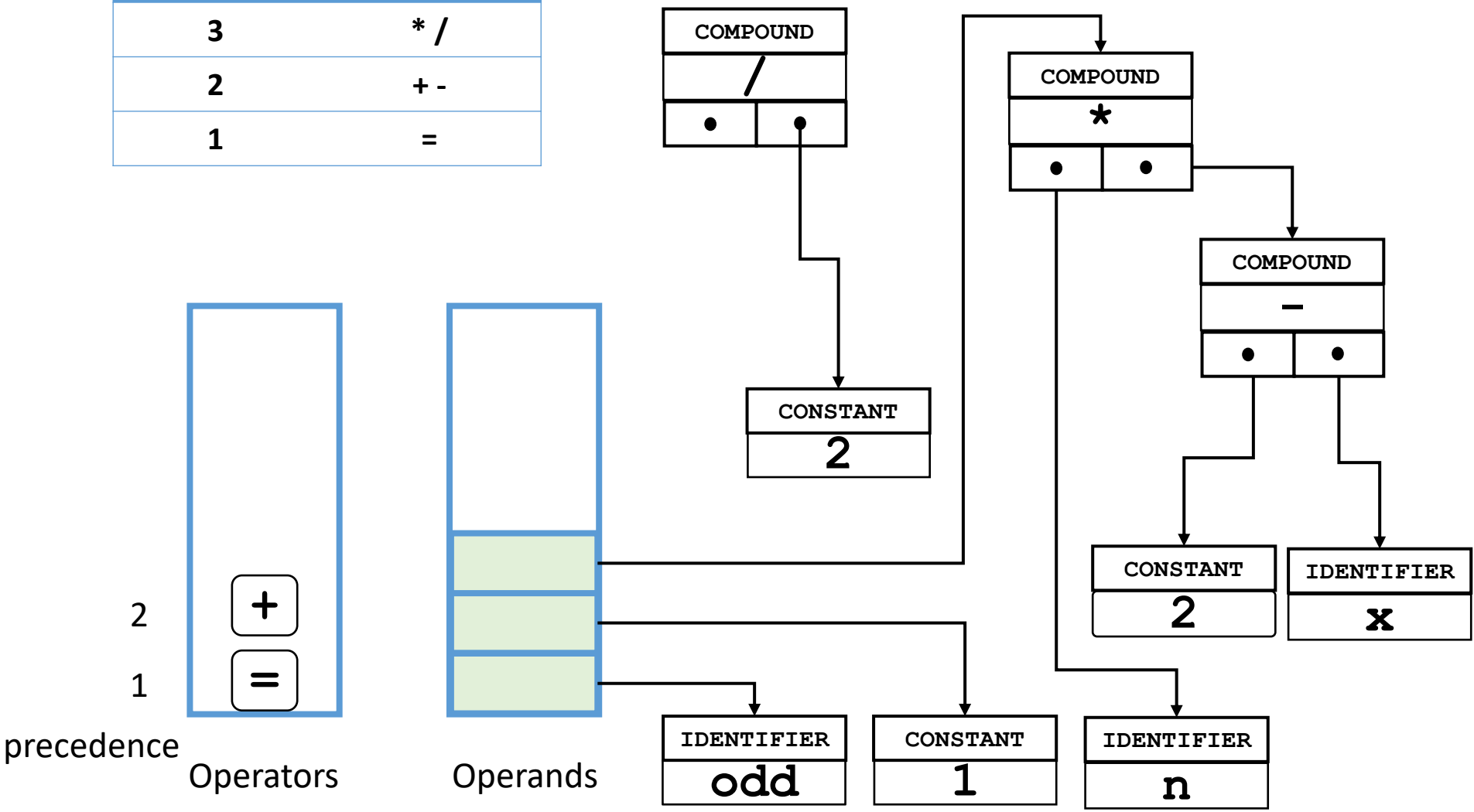

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



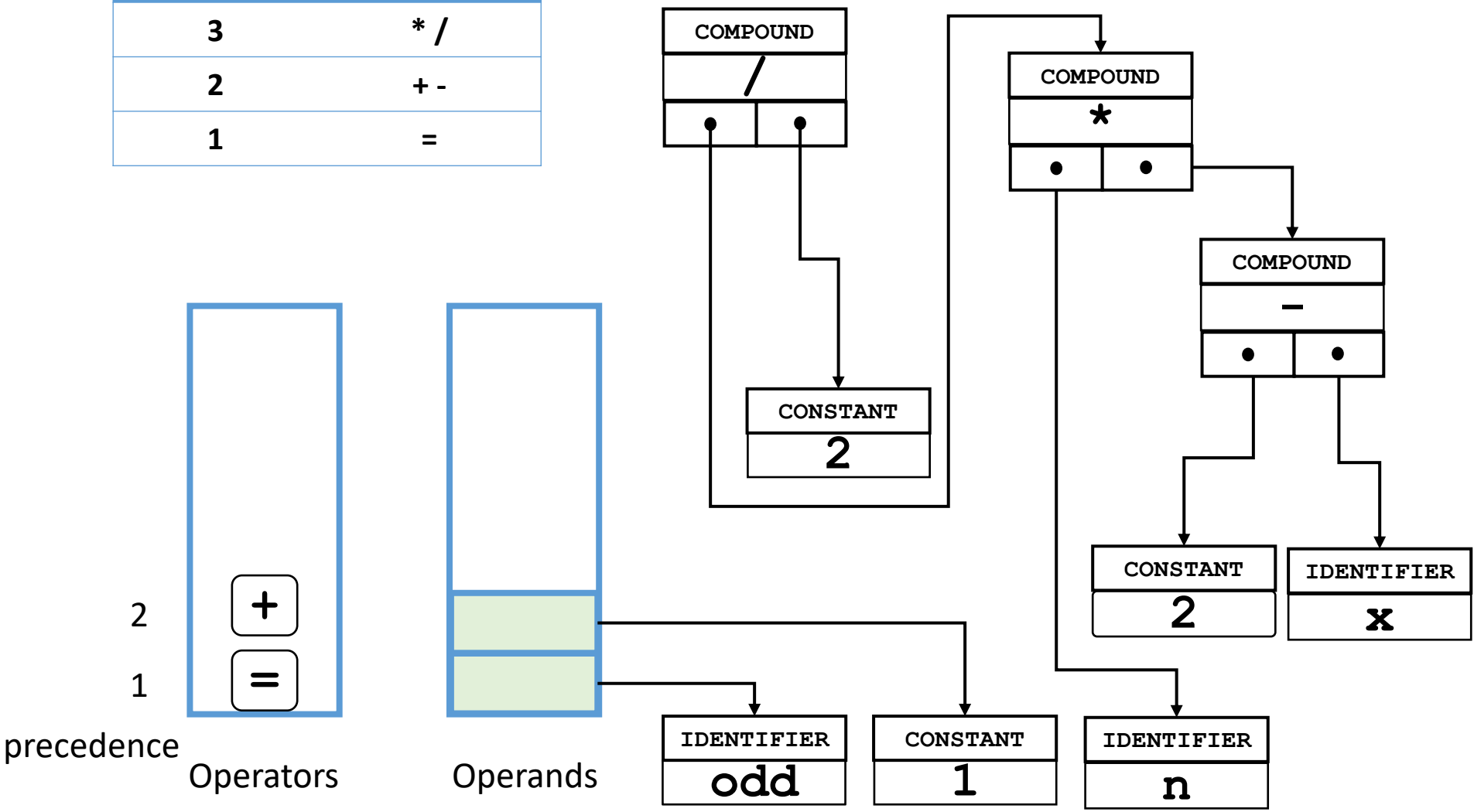

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



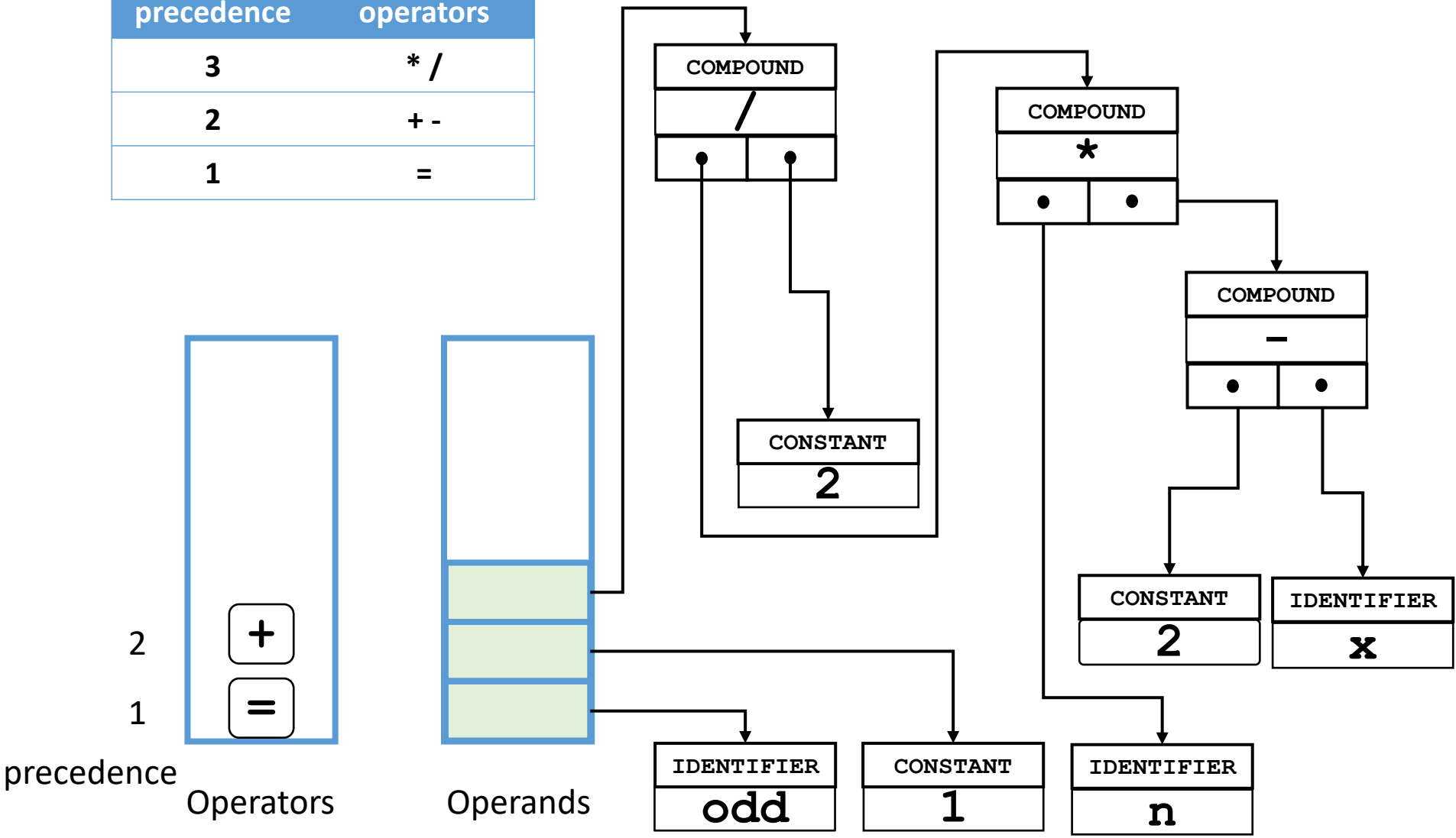

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



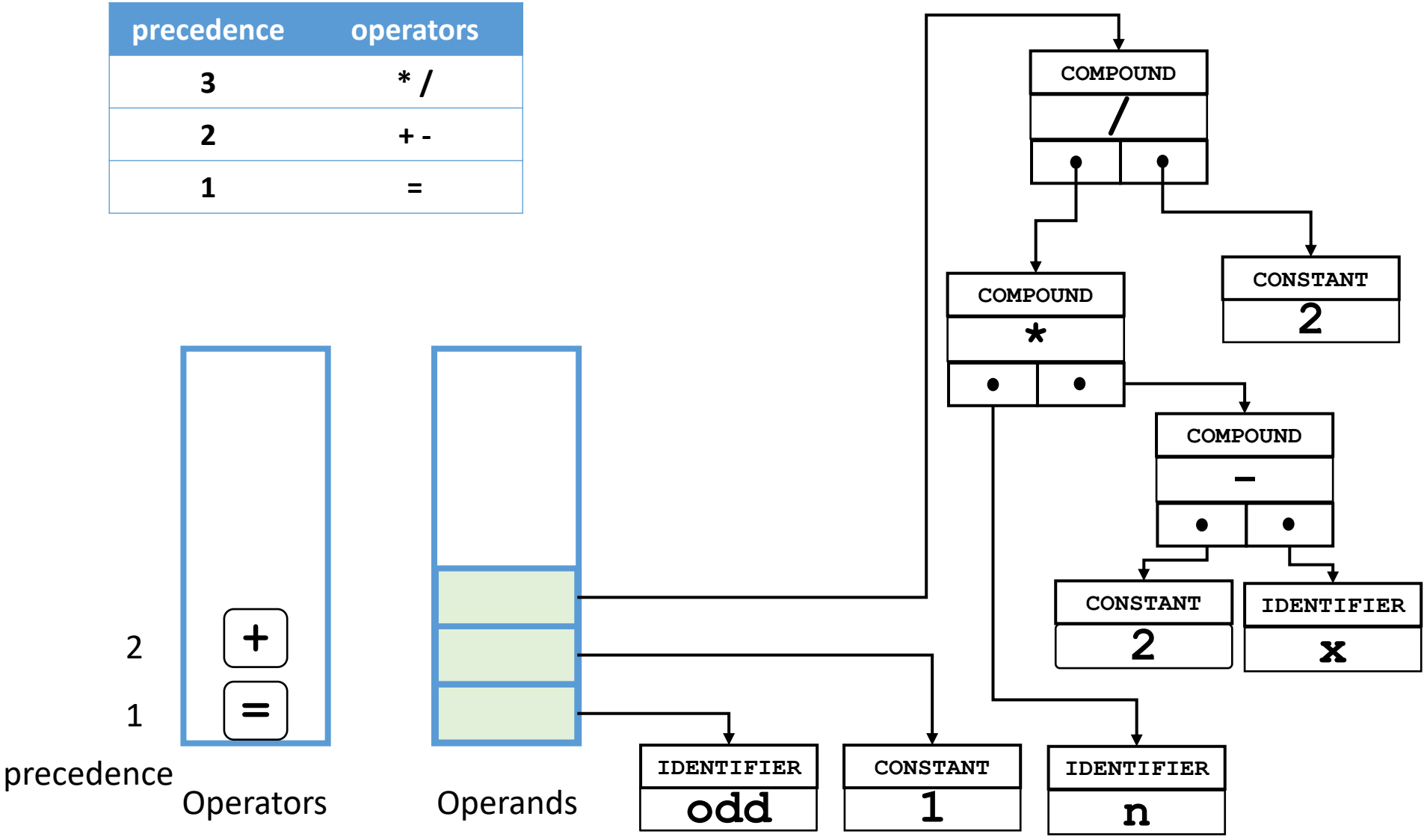

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



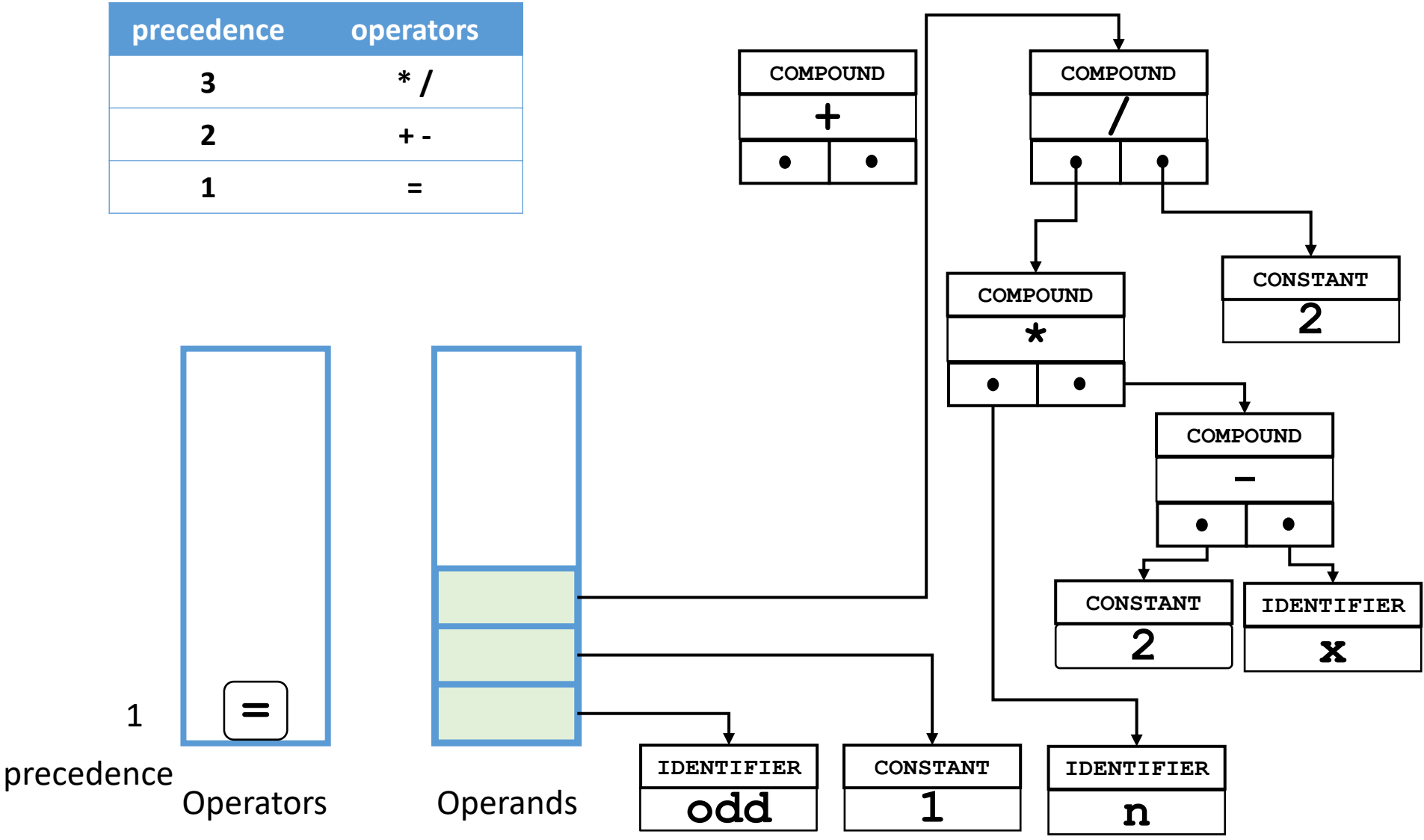

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



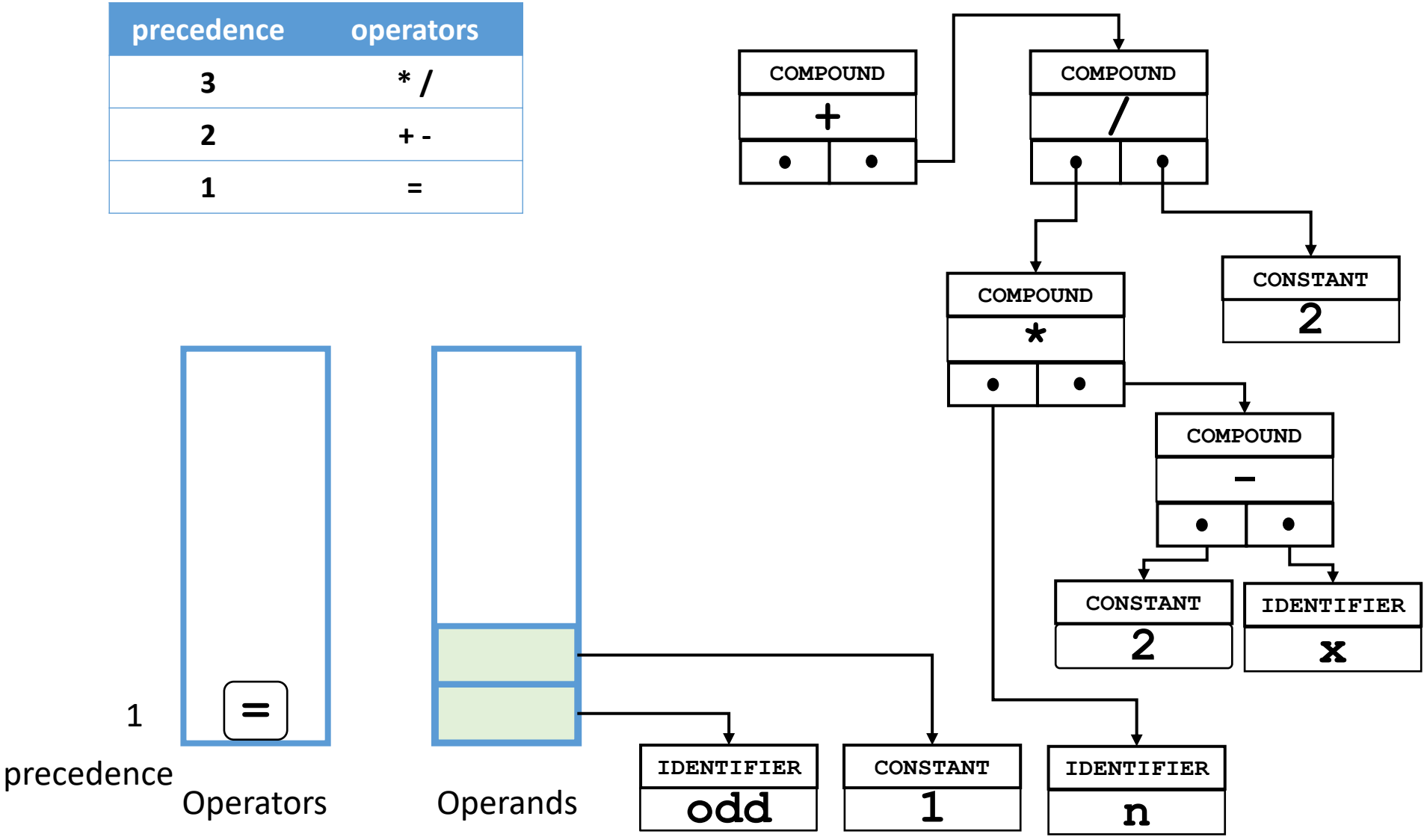

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



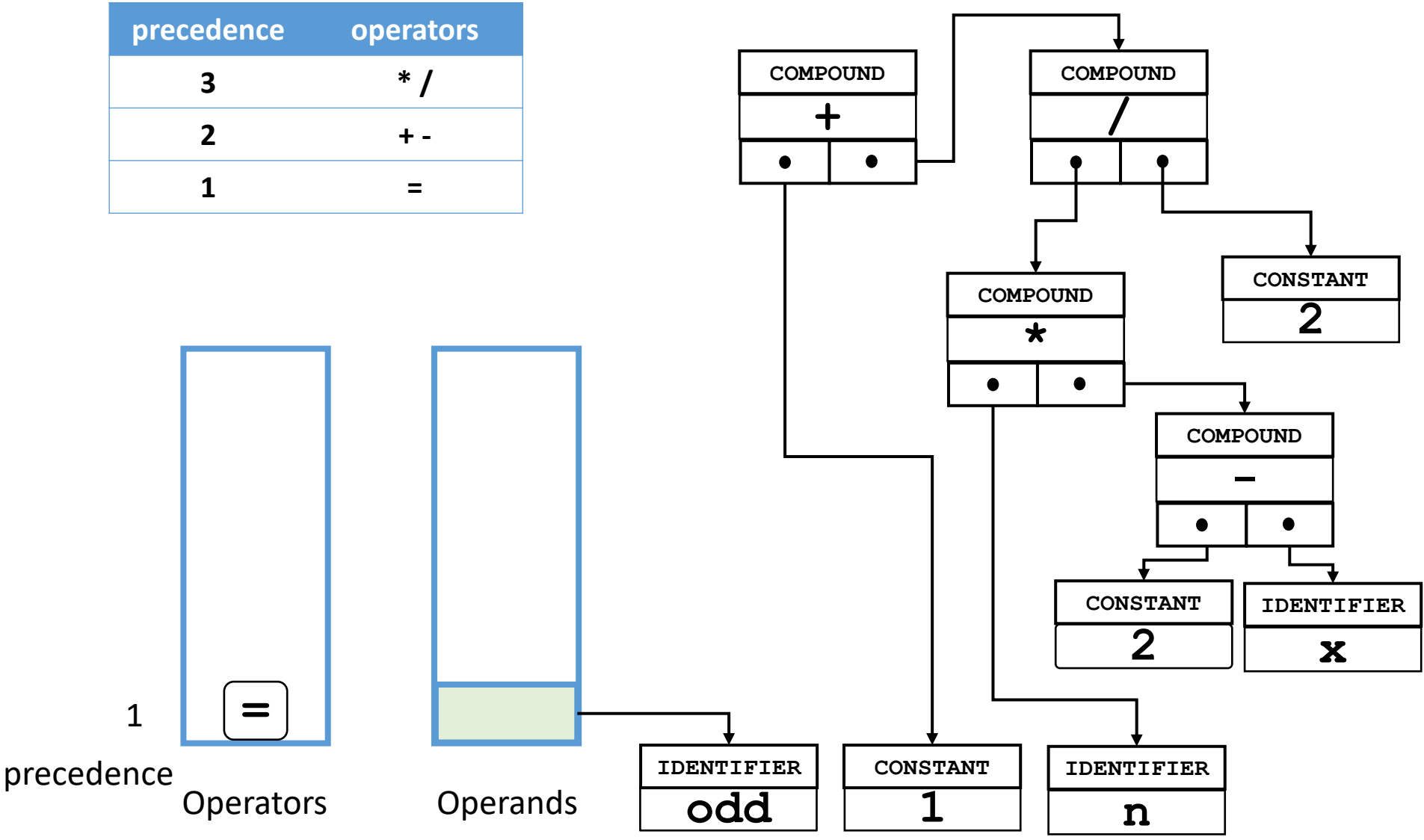

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



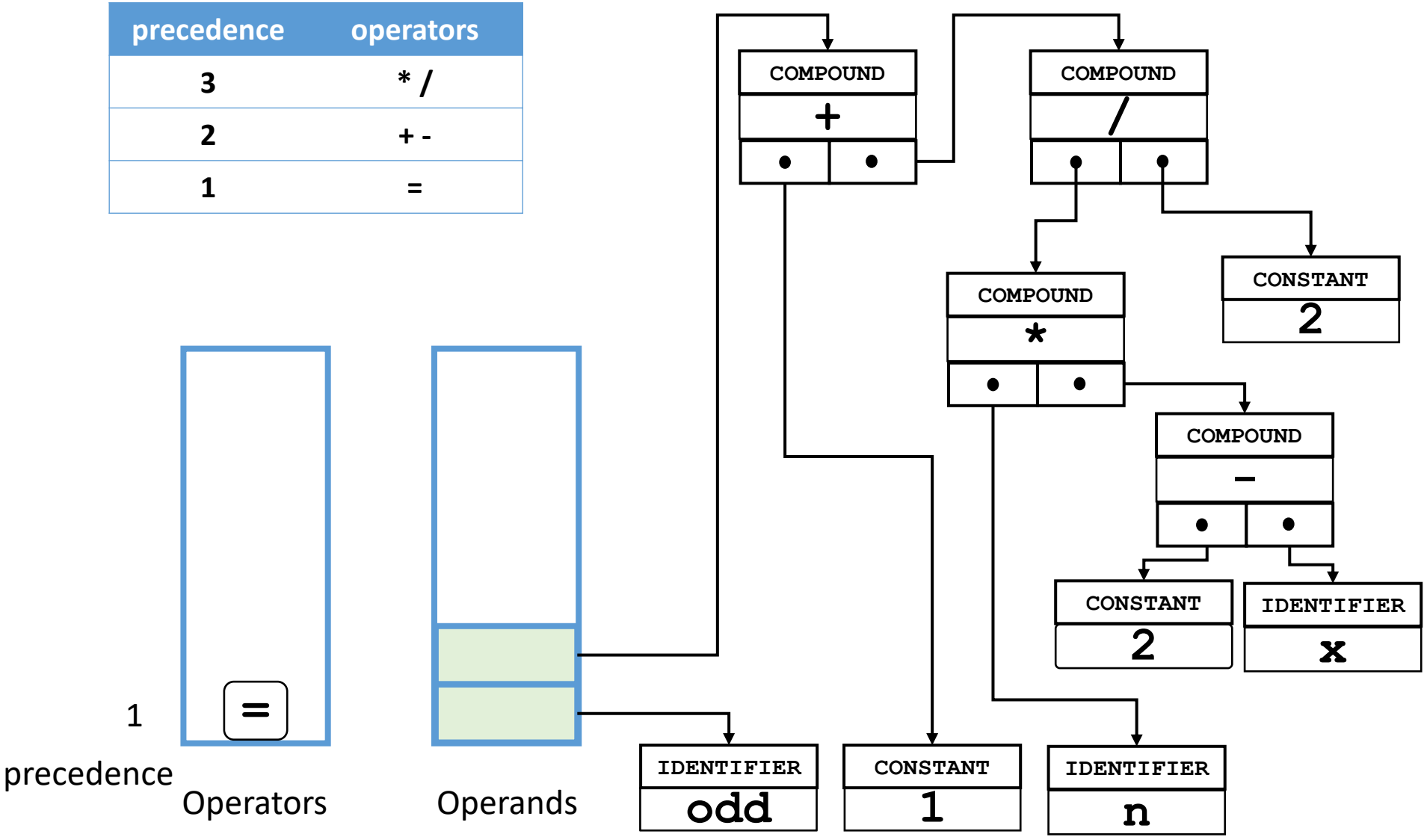

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



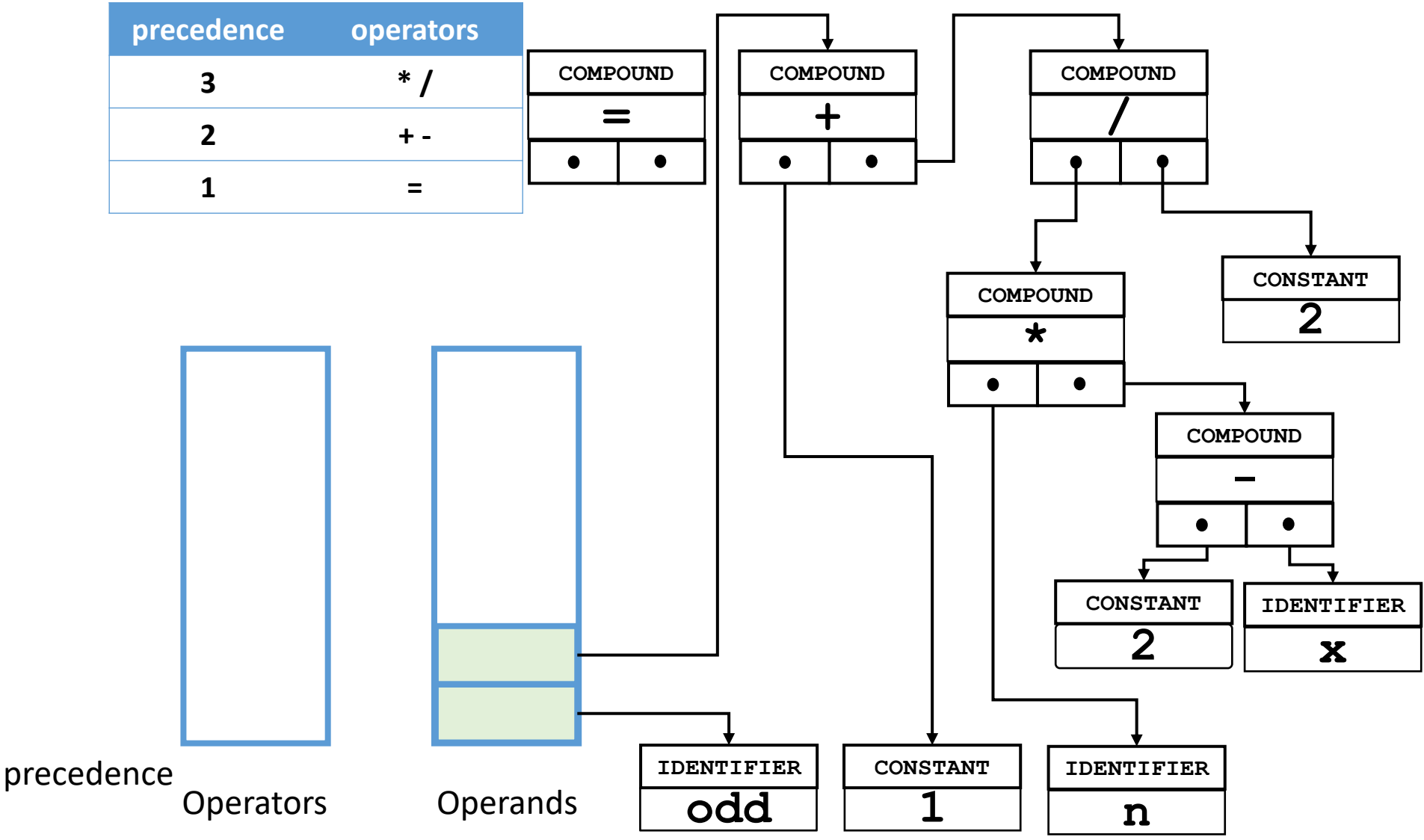

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



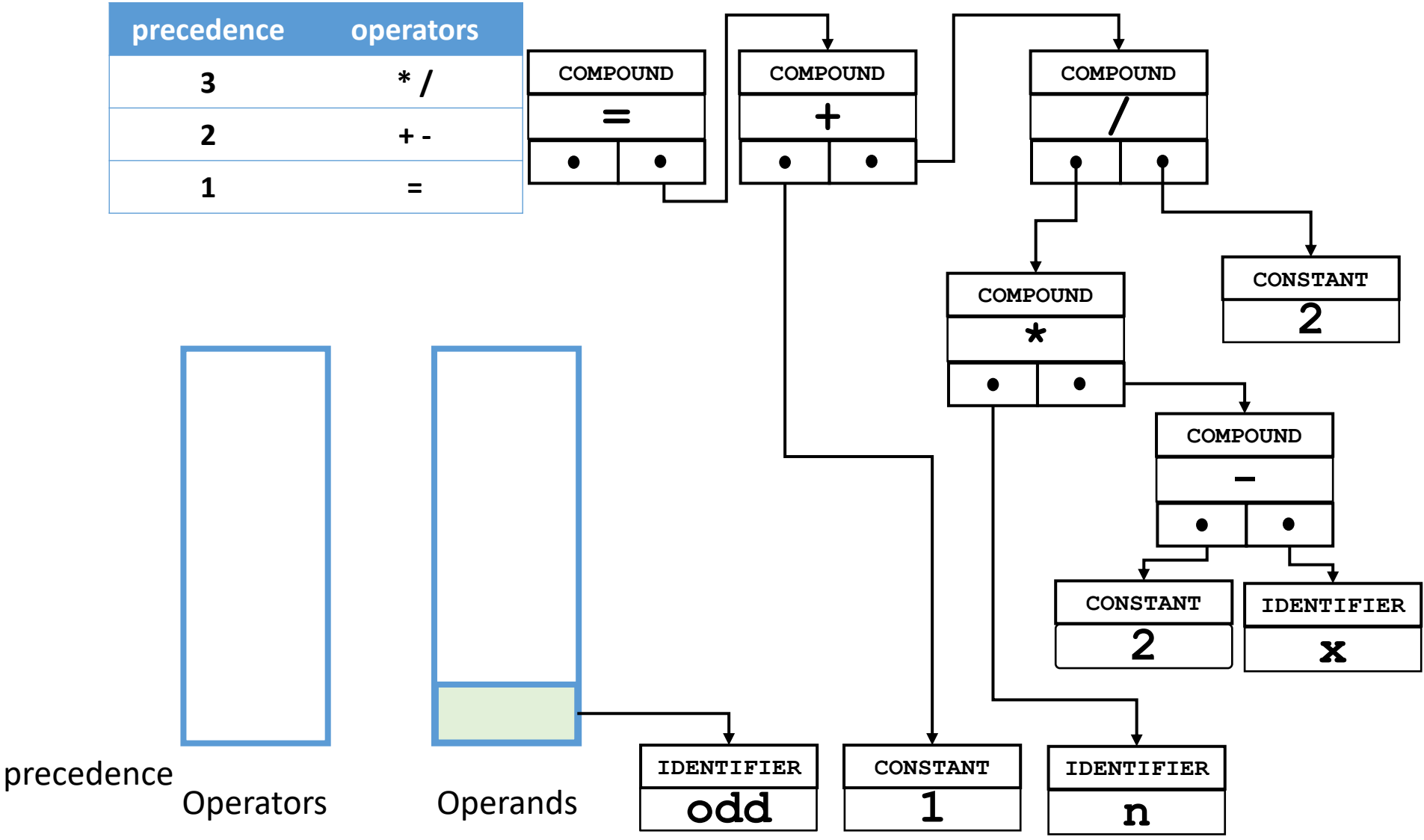

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



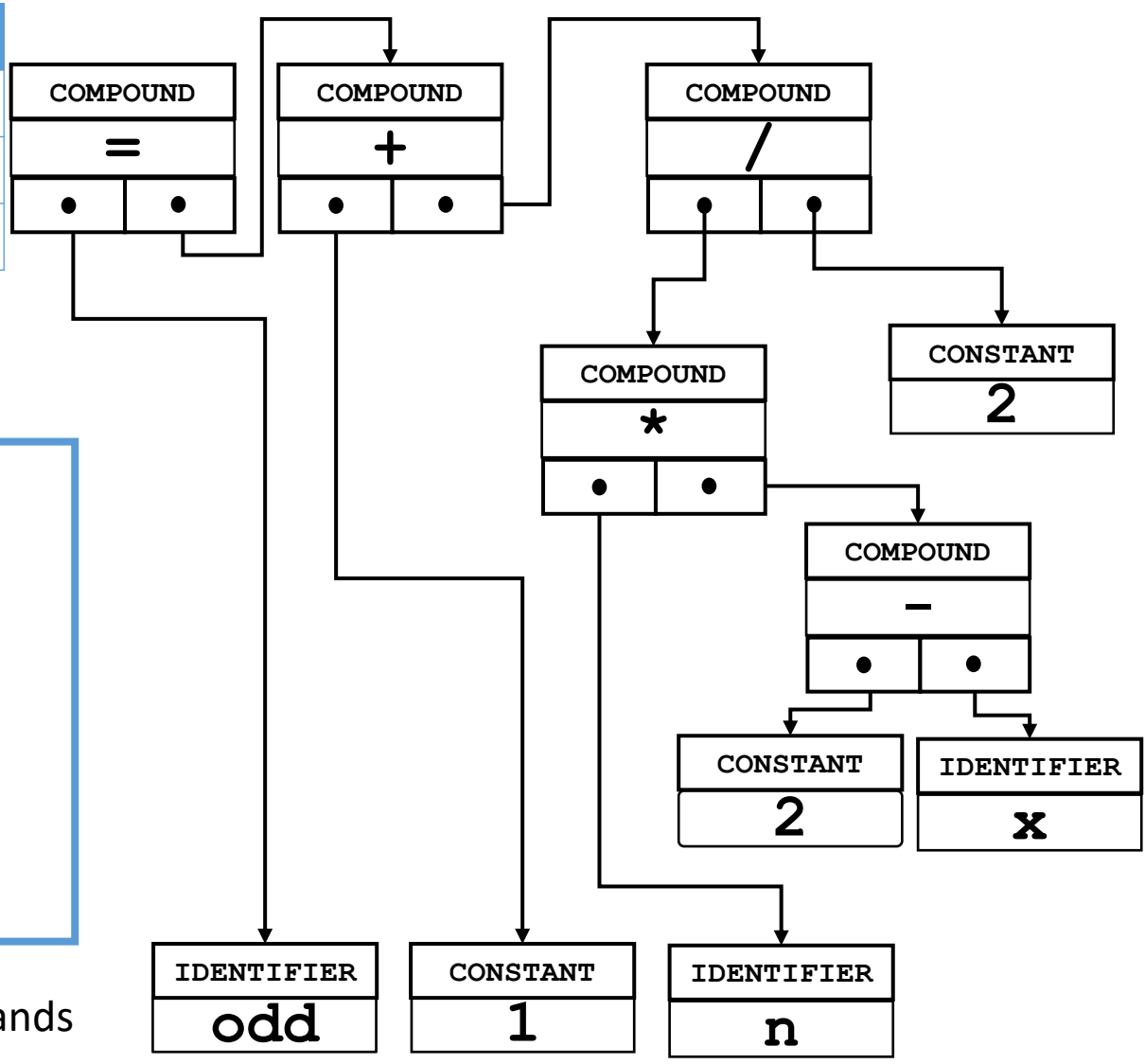

```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```

precedence	operators
3	* /
2	+ -
1	=



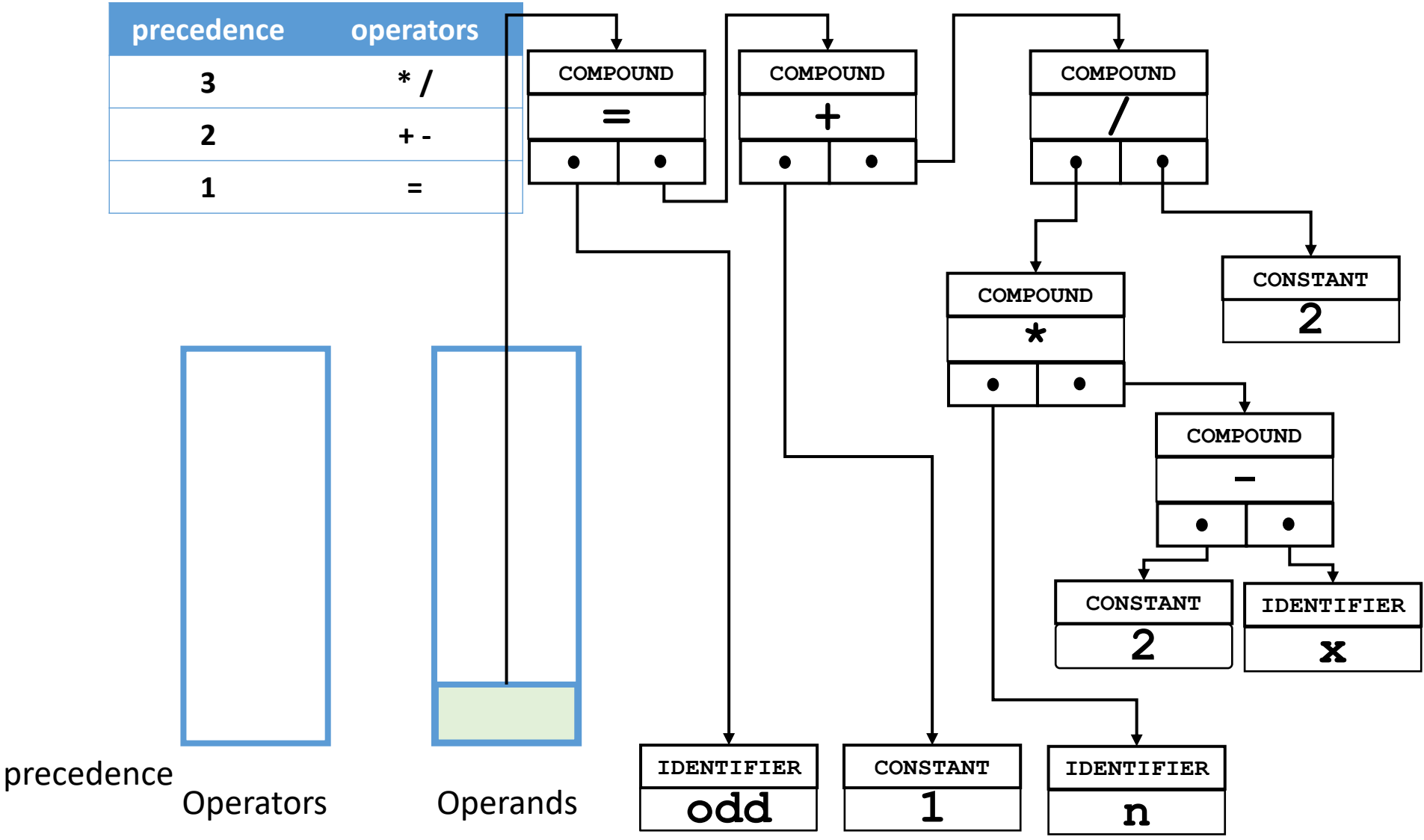
precedence

Operators

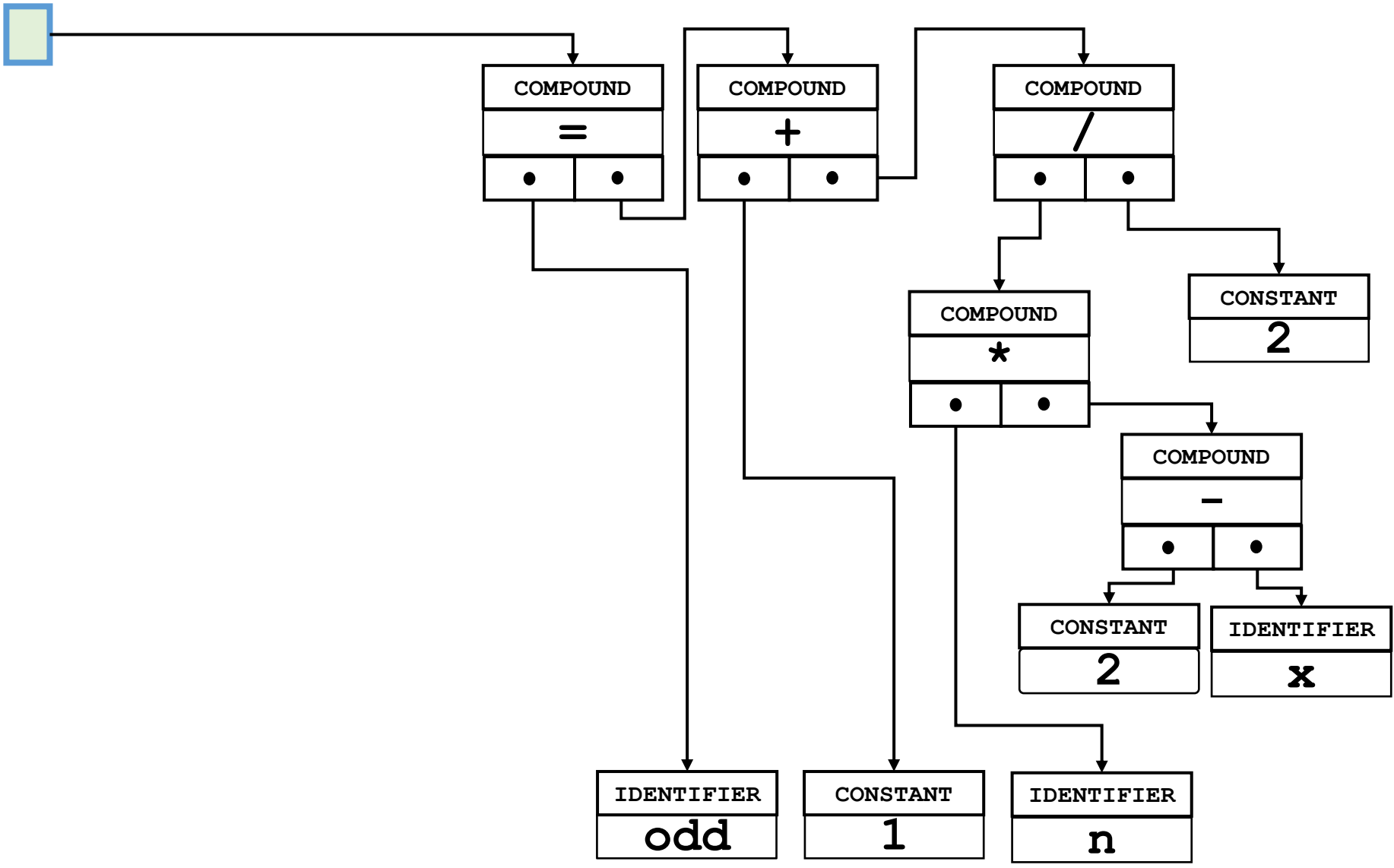
Operands

```
Odd = 1 + n * (2 - x) / 2
```

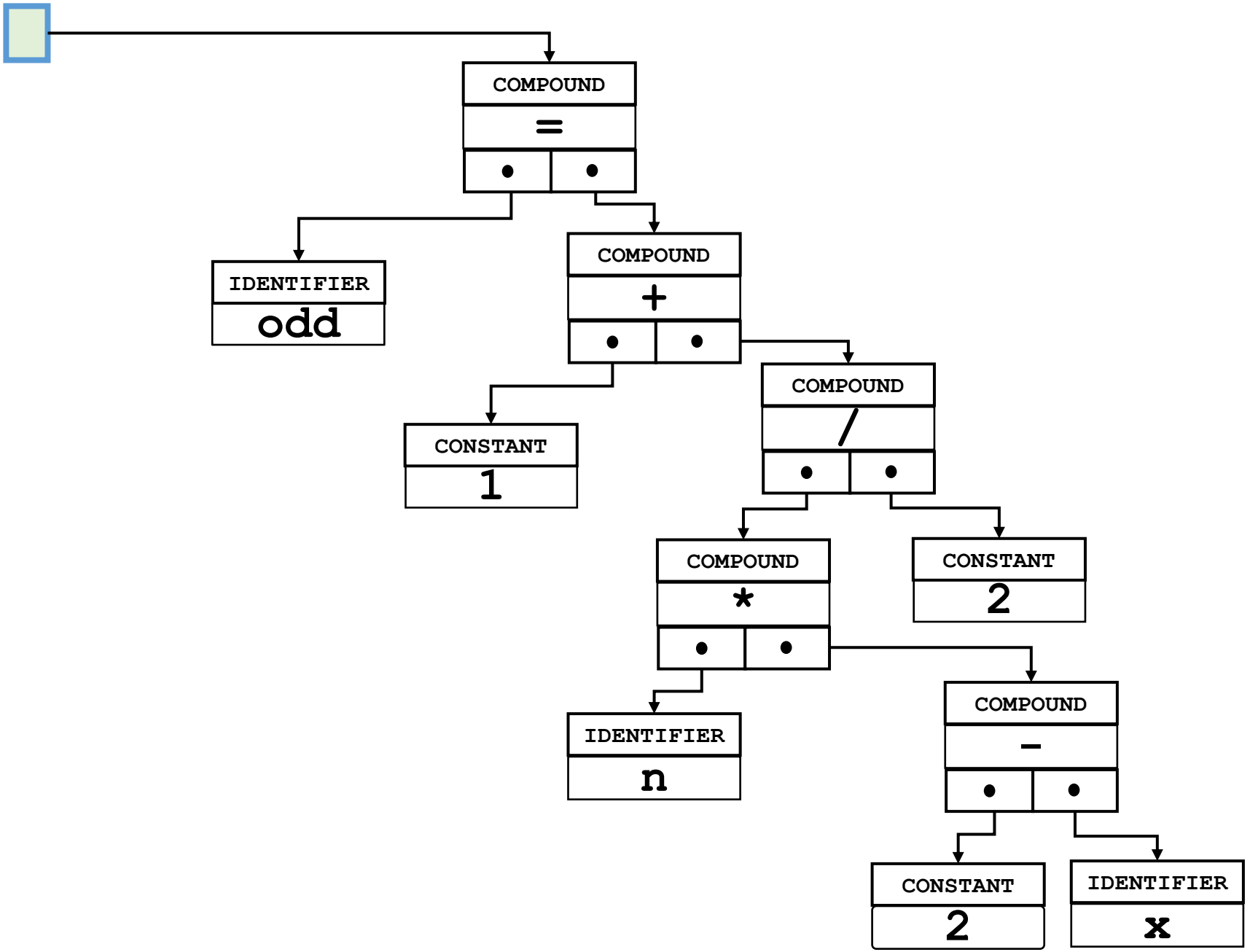
precedence	operators
3	* /
2	+ -
1	=




```
Odd = 1 + n * (2 - x) / 2
```


COMPOUND
=
COMPOUND
+
COMPOUND
/
COMPOUND
*
CONSTANT
2
COMPOUND
-
CONSTANT
2
IDENTIFIER
x
IDENTIFIER
odd
CONSTANT
1
IDENTIFIER
n

```
Odd = 1 + n * (2 - x) / 2
```


COMPOUND
=
IDENTIFIER
odd
COMPOUND
+
CONSTANT
1
COMPOUND
/
COMPOUND
*
CONSTANT
2
IDENTIFIER
n
COMPOUND
-
CONSTANT
2
IDENTIFIER
x

Odd = 1 + n * (2 - x) / 2

The `exp.h` Interface

```
/*
 * File: exp.h
 * -----------
 * This interface defines a class hierarchy for arithmetic expressions.
 */

#ifndef _exp_h
#define _exp_h

#include <string>
#include "map.h"
#include "tokenscanner.h"

/* Forward reference */

class EvaluationContext;

/*
 * Type: ExpressionType
 * --------------------
 * This enumerated type is used to differentiate the three different
 * expression types: CONSTANT, IDENTIFIER, and COMPOUND.
 */

enum ExpressionType { CONSTANT, IDENTIFIER, COMPOUND };
```

The `exp.h` Interface

```
/*
 * Class: Expression
 * -----------------
 * This class is used to represent a node in an expression tree.
 * Expression itself is an abstract class.  Every Expression object
 * is therefore created using one of the three concrete subclasses:
 * ConstantExp, IdentifierExp, or CompoundExp.
 */

class Expression {

public:

   Expression();
   virtual ~Expression();
   virtual int eval(EvaluationContext & context) = 0;
   virtual std::string toString() = 0;
   virtual ExpressionType type() = 0;

/* Getter methods for convenience */

   virtual int getConstantValue();
   virtual std::string getIdentifierName();
   virtual std::string getOperator();
   virtual Expression *getLHS();
   virtual Expression *getRHS();

};
```

The `exp.h` Interface

```
/*
 * Class: ConstantExp
 * ------------------
 * This subclass represents a constant integer expression.
 */

class ConstantExp: public Expression {

public:

   ConstantExp(int val);

   virtual int eval(EvaluationContext & context);
   virtual std::string toString();
   virtual ExpressionType type();

   virtual int getConstantValue();

private:

   int value;

};
```

The `exp.h` Interface

```
/*
 * Class: IdentifierExp
 * --------------------
 * This subclass represents a expression corresponding to a variable.
 */

class IdentifierExp: public Expression {

public:

   IdentifierExp(string name);

   virtual int eval(EvaluationContext & context);
   virtual std::string toString();
   virtual ExpressionType type();

   virtual string getIdentifierName();

private:

   std::string name;

};
```

The `exp.h` Interface

```
/*
 * Class: CompoundExp
 * ------------------
 * This subclass represents a compound expression.
 */

class CompoundExp: public Expression {

public:

   CompoundExp(string op, Expression *lhs, Expression *rhs);
   virtual ~CompoundExp();

   virtual int eval(EvaluationContext & context);
   virtual std::string toString();
   virtual ExpressionType type();

   virtual std::string getOperator();
   virtual Expression *getLHS();
   virtual Expression *getRHS();

private:

   std::string op;
   Expression *lhs, *rhs;

};
```

The `exp.h` Interface

```
/*
 * Class: EvaluationContext
 * ------------------------
 * This class encapsulates the information that the evaluator needs to
 * know in order to evaluate an expression.
 */

class EvaluationContext {

public:

   void setValue(std::string var, int value);
   int getValue(std::string var);
   bool isDefined(std::string var);

private:

   Map<std::string,int> symbolTable;

};

#endif
```

Selecting Fields from Subtypes

- The **`Expression`** class exports several methods that allow clients to select fields from one of the concrete subtypes:

```
virtual int getConstantValue();
virtual std::string getIdentifierName();
virtual std::string getOperator();
virtual Expression *getLHS();
virtual Expression *getRHS();
```

- The implementation of these methods in the **`Expression`** class always generates an error. Each subclass overrides that implementation with code that returns the appropriate instance variable.
- These methods exist primarily for the convenience of the client. A more conventional strategy would be to have each subclass export only the getters that apply to that subtype. Adopting this strategy, however, forces clients to include explicit type casting, which quickly gets rather tedious.

Code for the `eval` Method

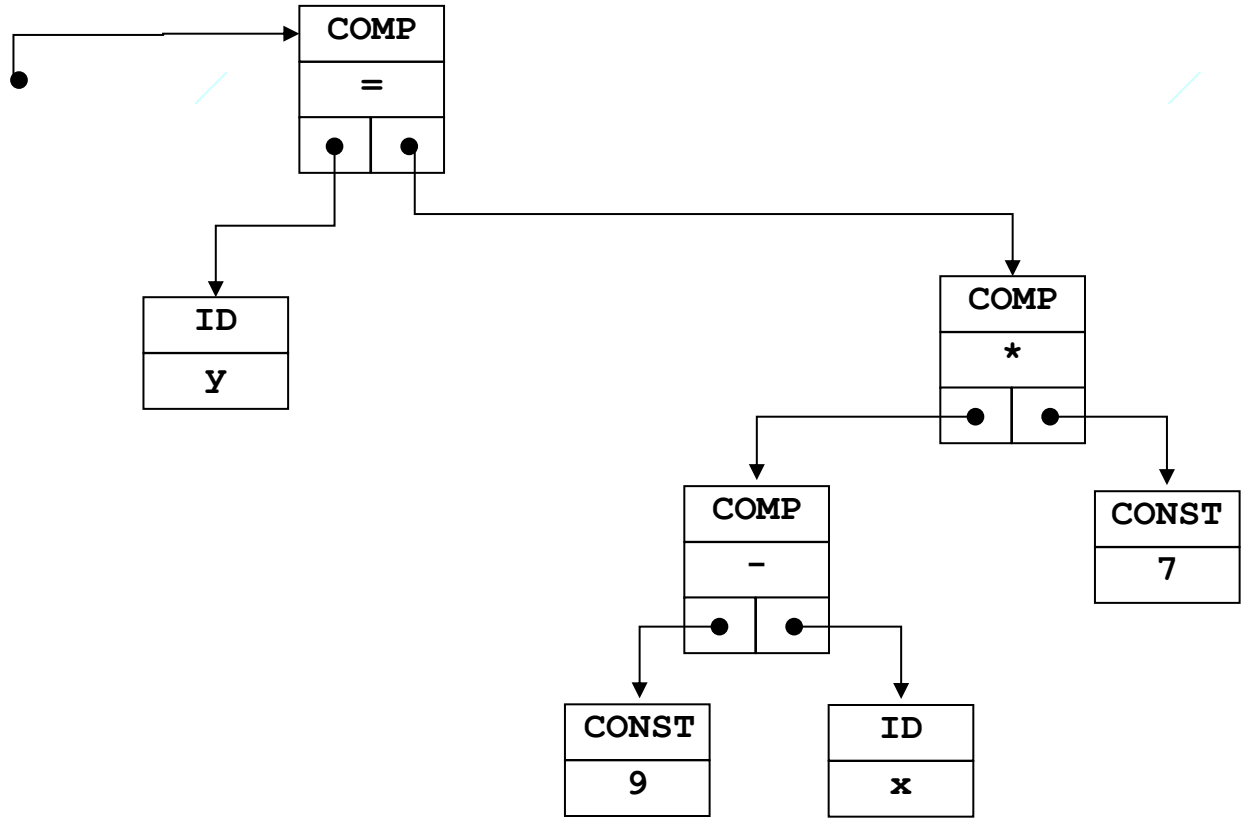
```
int ConstantExp::eval(EvaluationContext & context) {
   return value;
}

int IdentifierExp::eval(EvaluationContext & context) {
   if (!context.isDefined(name)) error(name + " is undefined");
   return context.getValue(name);
}

int CompoundExp::eval(EvaluationContext & context) {
   int right = rhs->eval(context);
   if (op == "=") {
      context.setValue(lhs->getIdentifierName(), right);
      return right;
   }
   int left = lhs->eval(context);
   if (op == "+") return left + right;
   if (op == "-") return left - right;
   if (op == "*") return left * right;
   if (op == "/") {
      if (right == 0) error("Division by 0");
      return left / right;
   }
   error("Illegal operator in expression");
   return 0;
}
```

Exercise: Evaluating an Expression

- Trace through the steps involved in evaluating the expression

```
y = ( 9 - x ) * 7
```

in a context in which **x** has the value 3. The representation of this expression looks like this:



To: BASIC Development Group

From: Bill Gates

Subject: C++ reimplementation

Date: April 1, 1981

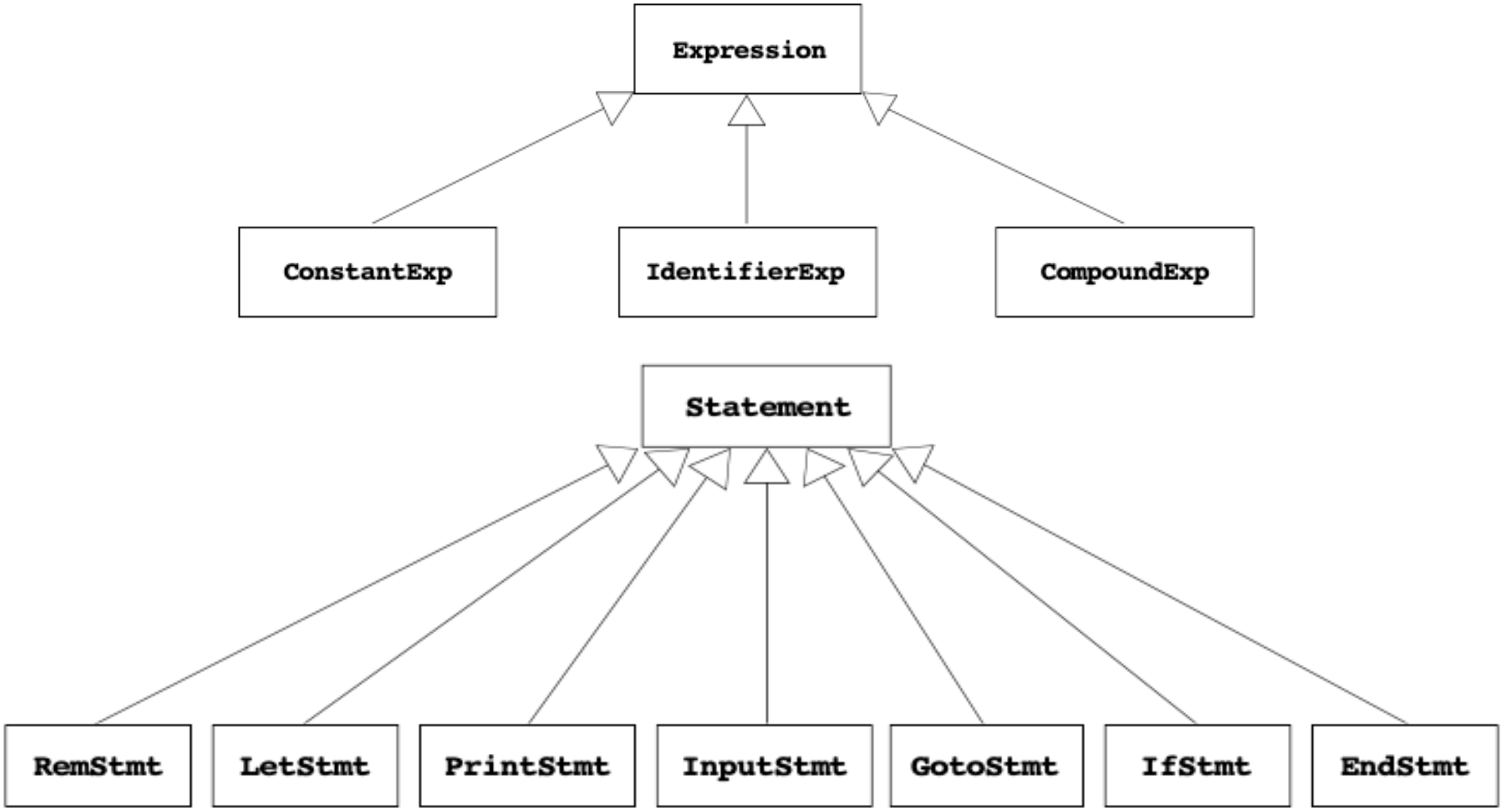
This guy from Bell Labs, Bjarne Stroustroup, just sent me a parser program written in a new language he's calling C++. His parser is much simpler than ours but still seems quite efficient. The code is much easier to read as well.

I think it's time to move away from assembly language for our version of BASIC, and C++ may be just the right tool.

Please get going on this project as soon as possible.

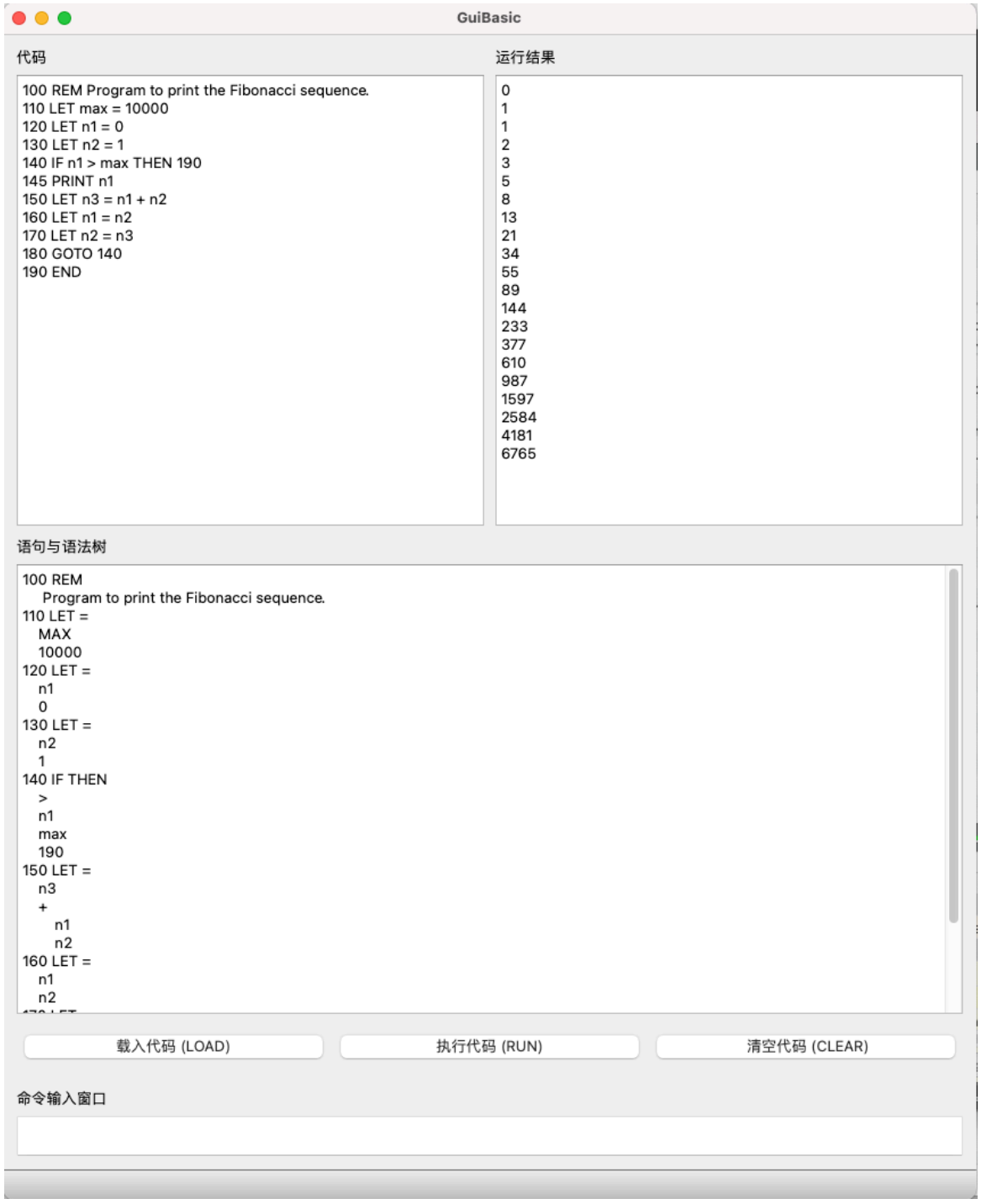
Bill Gates

The `Expression/Statement` Class Hierarchy

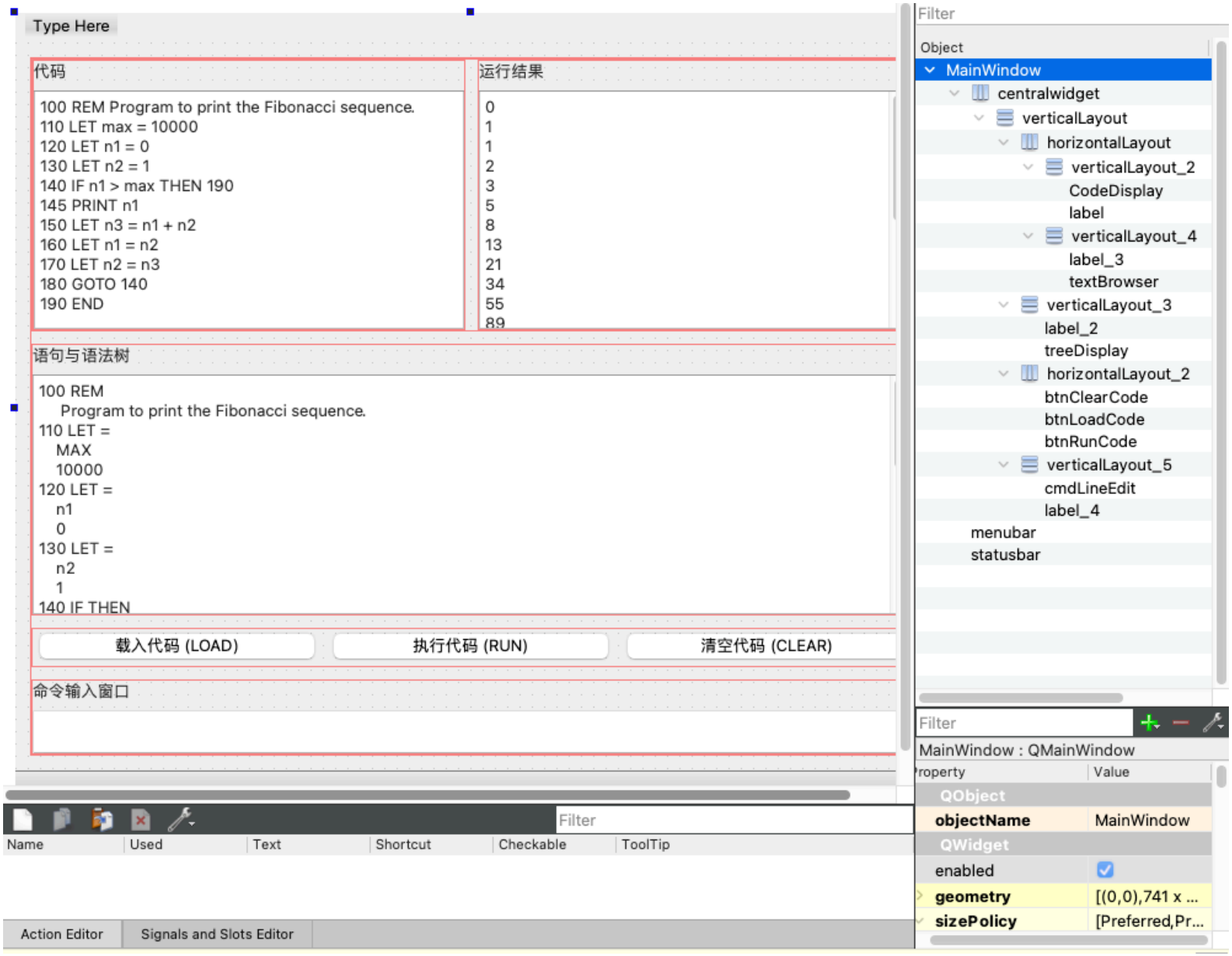


Recommended Files

`main.cpp`	Main file of the project
`mainwindow.h/cpp/ui`	Main Window (and UI)
`program.h/cpp`	Declare/implement a stored program
`statement.h/cpp`	Declare/implement statements
`exp.h/cpp`	Declare/implement expressions
`parser.h/cpp`	Parse a given expression
`tokenizer.h/cpp`	Convert strings to a list of tokens
`evalstate.h/cpp`	A space storing all variables during evaluation


GuiBasic
代码
100 REM Program to print the Fibonacci sequence.
110 LET max = 10000
120 LET n1 = 0
130 LET n2 = 1
140 IF n1 > max THEN 190
145 PRINT n1
150 LET n3 = n1 + n2
160 LET n1 = n2
170 LET n2 = n3
180 GOTO 140
190 END
运行结果
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
语句与语法树
100 REM
Program to print the Fibonacci sequence.
110 LET =
MAX
10000
120 LET =
n1
0
130 LET =
n2
1
140 IF THEN
>
n1
max
190
150 LET =
n3
+
n1
n2
160 LET =
n1
n2
载入代码 (LOAD)
执行代码 (RUN)
清空代码 (CLEAR)
命令输入窗口

Recommended UI Layout



多阶段提交

1. 画 GUI 界面，实现交互（如获取用户输入）
2. 实现程序保存、加载、编辑、显示等
3. 实现 Let、Print 等语句
4. 实现 Expression 的 parsing 和 evaluate（此时应该有完整的程序结构，链表+语法树）
5. 项目的其他部分（最终提交）

迟交/未提交 每次 -2