i116: Basic of Programming

6. Arithmetic calculator: parse tree & interpreter

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Arithmetic expressions
- Parse trees for arithmetic expressions
- Interpreter for arithmetic expressions

Arithmetic expressions

3 + 4 * 5

(3 + 4) * 5

3 + -(4 * 5)

3 + (-4 * 5)

((3 + 4) * 5) / 3

((3 - 4) * 5) % 0

3 < 5 || 3 = 5

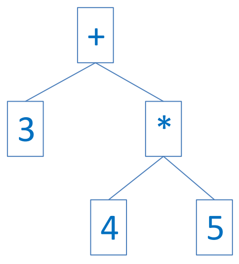
5 < 5 || 5 = 5

4 < 3 || 4 = 4 && 0 > -1 && (3 = 4 || 3 !=4)

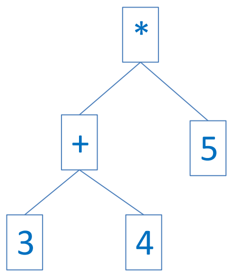
Arithmetic expressions

They can be represented as trees.

3 + 4 * 5

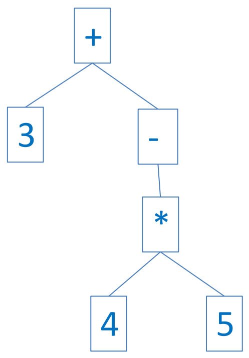


(3 + 4) * 5

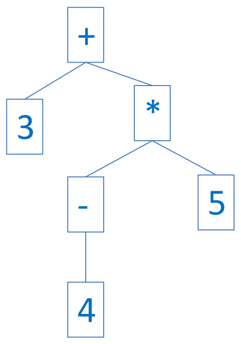


Arithmetic expressions

3 + -(4 * 5)

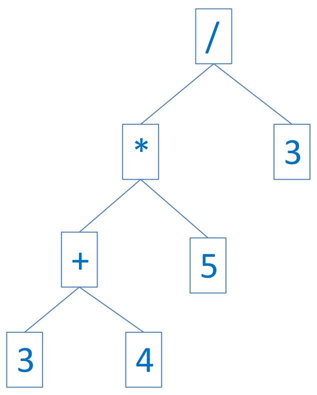


3 + (-4 * 5)

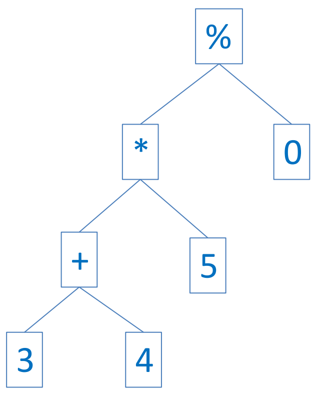


Arithmetic expressions

((3 + 4) * 5) / 3



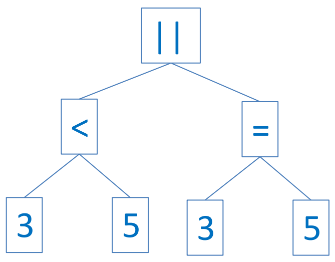
((3 + 4) * 5) % 0

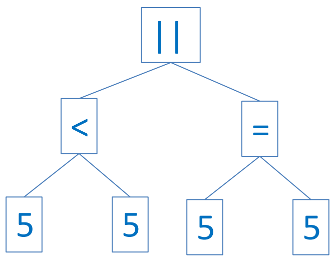


Arithmetic expressions

3 < 5 || 3 = 5

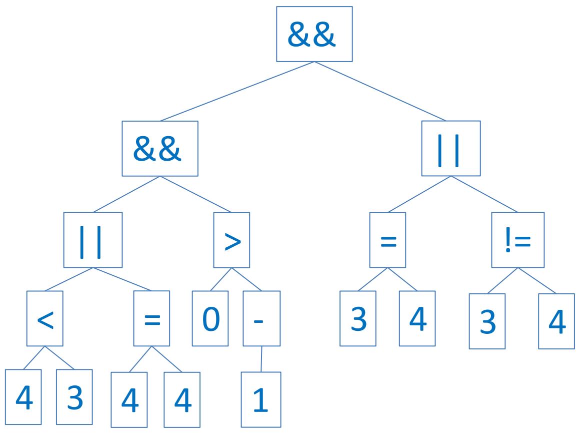
5 < 5 || 5 = 5





Arithmetic expressions

4 < 3 || 4 = 4 && 0 > -1 && (3 = 4 || 3 !=4)



Arithmetic expressions

E ::= *Nat* | (*E*) | -*E* |
E + *E* | *E* * *E* | *E* – *E* | *E* / *E* | *E* % *E* |
E = *E* | *E* != *E* | *E* < *E* | *E* > *E* |
E && *E* | *E* || *E*

Nat ::= [0-9]+

(strong)

- (unary)

*   /   %

+   - (binary)

precedence

=   !=   <   >

&& ||

(weak)

Binary operators are left-associative,
e.g., 3 + 4 + 5 means (3 + 4) + 5.

Parse trees for arithmetic expressions

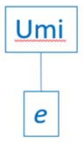
```
class ExpParseTree(object):
  def __str__(self):
    pass
```

```
class NumParseTree(ExpParseTree):
  def __init__(self, n):
    self.num = n
  def __str__(self):
    return str(self.num)
```

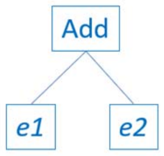
aNum: *n*

Parse trees for arithmetic expressions

```
class UmiParseTree(ExpParseTree):
  def __init__(self, e):
    self.exp = e
  def __str__(self):
    return '(-' + str(self.exp) + ')'
```

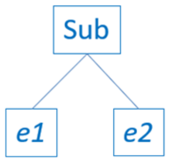


```
class AddParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' + ' + str(self.exp2) + ')'
```

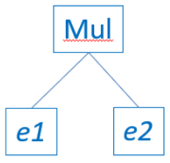


Parse trees for arithmetic expressions

```
class SubParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' - ' + str(self.exp2) + ')'
```

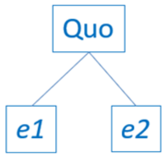


```
class MulParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' * ' + str(self.exp2) + ')'
```

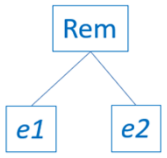


Parse trees for arithmetic expressions

```
class QuoParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' / ' + str(self.exp2) + ')'
```

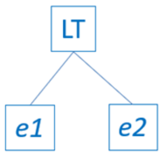


```
class RemParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' % ' + str(self.exp2) + ')'
```

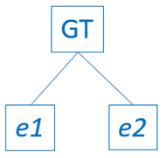


Parse trees for arithmetic expressions

```
class LTParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' < ' + str(self.exp2) + ')'
```

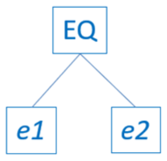


```
class GTParseTree(ExpParseTree):
  def __init__(self, e1, e2):
    self.exp1 = e1
    self.exp2 = e2
  def __str__(self):
    return '(' + str(self.exp1) + ' > ' + str(self.exp2) + ')'
```

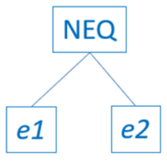


Parse trees for arithmetic expressions

```
class EQParseTree(ExpParseTree):
    def __init__(self, e1, e2):
        self.exp1 = e1
        self.exp2 = e2
    def __str__(self):
        return '(' + str(self.exp1) + ' = ' + str(self.exp2) + ')'
```

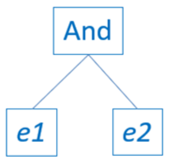


```
class NEQParseTree(ExpParseTree):
    def __init__(self, e1, e2):
        self.exp1 = e1
        self.exp2 = e2
    def __str__(self):
        return '(' + str(self.exp1) + ' != ' + str(self.exp2) + ')'
```

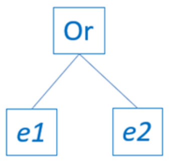


Parse trees for arithmetic expressions

```
class AndParseTree(ExpParseTree):
    def __init__(self, e1, e2):
        self.exp1 = e1
        self.exp2 = e2
    def __str__(self):
        return '(' + str(self.exp1) + ' && ' + str(self.exp2) + ')'
```

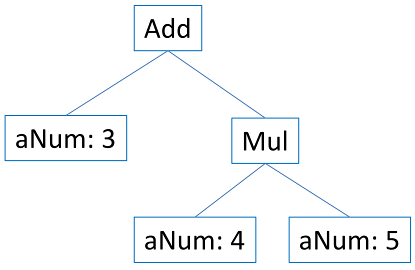


```
class OrParseTree(ExpParseTree):
    def __init__(self, e1, e2):
        self.exp1 = e1
        self.exp2 = e2
    def __str__(self):
        return '(' + str(self.exp1) + ' || ' + str(self.exp2) + ')'
```



Parse trees for arithmetic expressions

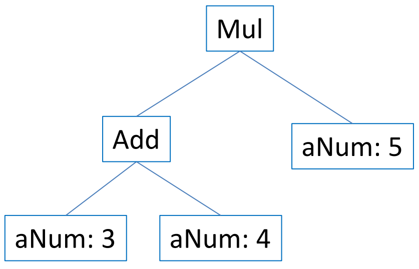
3 + 4 * 5



```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = MulParseTree(four,five)
e2 = AddParseTree(three,e1)
print(three)
print(four)
print(five)
print(e1)
print(e2)
```

Parse trees for arithmetic expressions

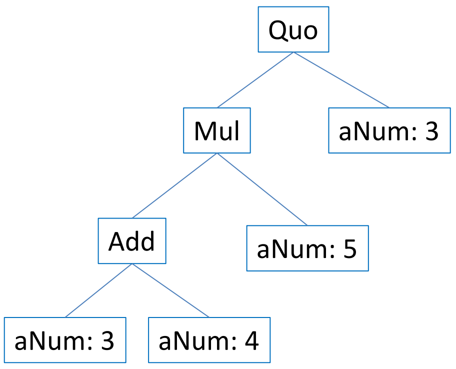
(3 + 4) * 5



```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
print(three)
print(four)
print(five)
print(e1)
print(e2)
```

Parse trees for arithmetic expressions

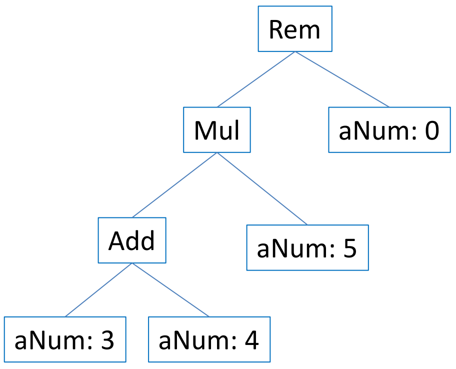
((3 + 4) * 5) / 3



```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
e3 = QuoParseTree(e2,three)
print(e3)
```

Parse trees for arithmetic expressions

((3 + 4) * 5) % 0



```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
zero = NumParseTree(0)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
e3 = RemParseTree(e2,zero)
print(e3)
```

Interpreter for arithmetic expressions

- Develop an arithmetic calculator as an interpreter.
- The parse tree of an arithmetic expression is interpreted to calculate the expression.

Interpreter for arithmetic expressions

```
class ExpParseTree(object):
  …
  def interpret(self):
    pass
```

```
class NumParseTree(ExpParseTree):
  …
  def interpret(self):
    return self.num
```

$\boxed{\text{aNum: } n}$ .interpret()

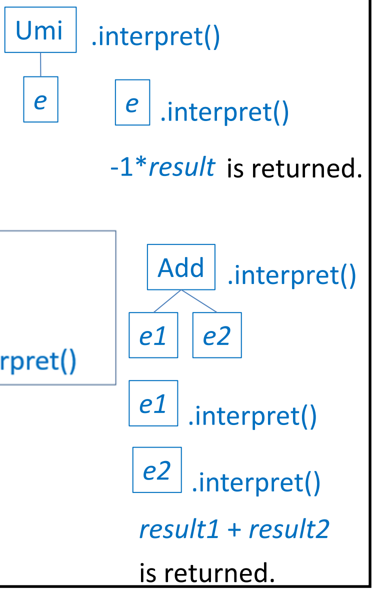
returns

n

Interpreter for arithmetic expressions

```
class UmiParseTree(ExpParseTree):
  ...
  def interpret(self):
    return -1 * self.exp.interpret()
```

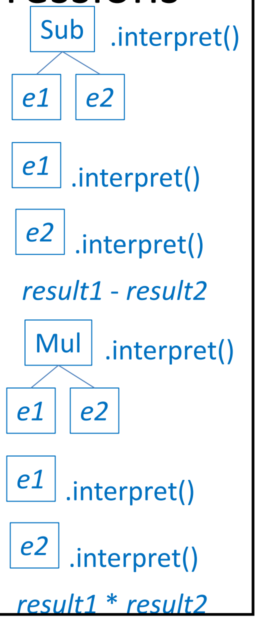
```
class AddParseTree(ExpParseTree):
  ...
  def interpret(self):
    return self.exp1.interpret() + self.exp2.interpret()
```



Interpreter for arithmetic expressions

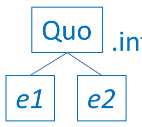
```
class SubParseTree(ExpParseTree):
  ...
  def interpret(self):
    return self.exp1.interpret() - self.exp2.interpret()
```

```
class MulParseTree(ExpParseTree):
  ...
  def interpret(self):
    return self.exp1.interpret() * self.exp2.interpret()
```



Interpreter for arithmetic expressions

```
class QuoParseTree(ExpParseTree):
  ...
  def interpret(self):
    if self.exp2.interpret() == 0:
      raise DivisionByZero('division by zero')
    else:
      return self.exp1.interpret() // self.exp2.interpret()
```



.interpret()

If $\boxed{e2}$ .interpret() returns 0,

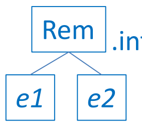
then an exception is raised.

Otherwise,

$\boxed{e1}$ .interpret() // $\boxed{e2}$ .interpret() is returned.

Interpreter for arithmetic expressions

```
class RemParseTree(ExpParseTree):
  ...
  def interpret(self):
    if self.exp2.interpret() == 0:
      raise DivisionByZero('division by zero')
    else:
      return self.exp1.interpret() % self.exp2.interpret()
```



.interpret()

If $\boxed{e2}$ .interpret() returns 0,

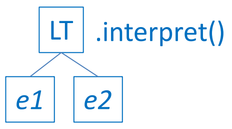
then an exception is raised.

Otherwise,

$\boxed{e1}$ .interpret() % $\boxed{e2}$ .interpret() is returned.

Interpreter for arithmetic expressions

```
class LTParseTree(ExpParseTree):
  …
  def interpret(self):
    if self.exp1.interpret() < self.exp2.interpret():
      return 1
    else:
      return 0
```



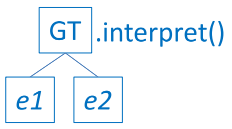
If $\boxed{e1}$ .interpret() < $\boxed{e2}$ .interpret()

then 1 is returned.

Otherwise, 0 is returned.

Interpreter for arithmetic expressions

```
class GTParseTree(ExpParseTree):
  …
  def interpret(self):
    if self.exp1.interpret() > self.exp2.interpret():
      return 1
    else:
      return 0
```



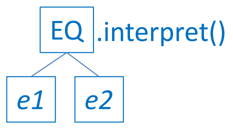
If $\boxed{e1}$ .interpret() > $\boxed{e2}$ .interpret()

then 1 is returned.

Otherwise, 0 is returned.

Interpreter for arithmetic expressions

```
class EQParseTree(ExpParseTree):
  ...
  def interpret(self):
    if self.exp1.interpret() == self.exp2.interpret():
      return 1
    else:
      return 0
```



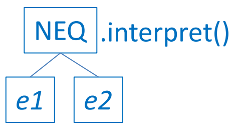
If $\boxed{e1}$ .interpret() == $\boxed{e2}$ .interpret()

then 1 is returned.

Otherwise, 0 is returned.

Interpreter for arithmetic expressions

```
class NEQParseTree(ExpParseTree):
  ...
  def interpret(self):
    if self.exp1.interpret() == self.exp2.interpret():
      return 0
    else:
      return 1
```



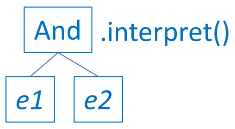
If $\boxed{e1}$ .interpret() == $\boxed{e2}$ .interpret()

then 0 is returned.

Otherwise, 1 is returned.

Interpreter for arithmetic expressions

```
class AndParseTree(ExpParseTree):
  …
  def interpret(self):
    if self.exp1.interpret() == 0 or self.exp2.interpret() == 0:
      return 0
    else:
      return 1
```



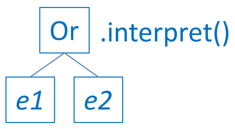
If $\boxed{e1}$ .interpret() == 0 or $\boxed{e2}$ .interpret() == 0

then 0 is returned.

Otherwise, 1 is returned.

Interpreter for arithmetic expressions

```
class OrParseTree(ExpParseTree):
  …
  def interpret(self):
    if self.exp1.interpret() == 0 and self.exp2.interpret() == 0:
      return 0
    else:
      return 1
```



If $\boxed{e1}$ .interpret() == 0 and $\boxed{e2}$ .interpret() == 0

then 0 is returned.

Otherwise, 1 is returned.

Interpreter for arithmetic expressions

3 + 4 * 5

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = MulParseTree(four,five)
e2 = AddParseTree(three,e1)
print(e2)
print(e2.interpret())
```

(3 + 4) * 5

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
print(e2)
print(e2.interpret())
```

Interpreter for arithmetic expressions

3 + -(4 * 5)

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = MulParseTree(four,five)
e2 = UmiParseTree(e1)
e3 = AddParseTree(three,e2)
print(e3)
print(e3.interpret())
```

3 + (-4 * 5)

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = UmiParseTree(four)
e2 = MulParseTree(e1,five)
e3 = AddParseTree(three,e2)
print(e3)
print(e3.interpret())
```

Interpreter for arithmetic expressions

((3 + 4) * 5) / 3

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
e3 = QuoParseTree(e2,three)
print(e3)
print(e3.interpret())
```

((3 + 4) * 5) % 0

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
zero = NumParseTree(0)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
e3 = RemParseTree(e2,zero)
print(e3)
try:
   print(e3.interpret())
except DivisionByZero as em:
   print(em)
```

Interpreter for arithmetic expressions

3 < 5 || 3 = 5

```
three = NumParseTree(3)
five = NumParseTree(5)
e1 = LTParseTree(three,five)
e2 = EQParseTree(three,five)
e3 = OrParseTree(e1,e2)
print(e3)
print(e3.interpret())
```

5 < 5 || 5 = 5

```
five = NumParseTree(5)
e1 = LTParseTree(five,five)
e2 = EQParseTree(five,five)
e3 = OrParseTree(e1,e2)
print(e3)
print(e3.interpret())
```

Interpreter for arithmetic expressions

4 < 3 | | 4 = 4 && 0 > -1 && (3 = 4 | | 3 !=4)

```
zero = NumParseTree(0)
one = NumParseTree(1)
three = NumParseTree(3)
four = NumParseTree(4)
e1 = LTParseTree(four,three)
e2 = EQParseTree(four,four)
e3 = OrParseTree(e1,e2)
e4 = UmiParseTree(one)
e5 = GTParseTree(zero,e4)
e6 = AndParseTree(e3,e5)
e7 = EQParseTree(three,four)
e8 = NEQParseTree(three,four)
e9 = OrParseTree(e7,e8)
e10 = AndParseTree(e6,e9)
print(e10)
print(e10.interpret())
```