i116: Basic of Programming

7. Arithmetic calculator: virtual machine & compiler

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Virtual machine for an arithmetic calculator
- Compiler for an arithmetic calculator

Virtual machine for an arithmetic calculator

- A virtual machine (VM) is a software system that emulates a physical machine like a microprocessor, such as Java virtual machine (JVM).
- The VM developed here uses a stack (of integers) and then is called a stack machine.
- It has a collection of commands (of instructions) like Java bytecode.
- It handles a list of commands with a stack.

Virtual machine for an arithmetic calculator

We first prepare command names.

```
from enum import *
class CName(Enum):
    PUSH = auto()
    MONE = auto()
    MUL = auto()
    QUO = auto()
    REM = auto()
    ADD = auto()
    SUB = auto()
```

minus one

```
    EQ = auto()
    NEQ = auto()
    LT = auto()
    GT = auto()
    AND = auto()
    OR = auto()
    NSC = auto()
```

no such command

Virtual machine for an arithmetic calculator

```
def __str__(self):
    if self == CName.PUSH:
        return 'push'
    elif self == CName.MONE:
        return 'mone'
    elif self == CName.MUL:
        return 'mul'
    elif self == CName.QUO:
        return 'quo'
    elif self == CName.REM:
        return 'rem'
    elif self == CName.ADD:
        return 'add'
    elif self == CName.SUB:
        return 'sub'
    elif self == CName.EQ:
        return 'eq'
    elif self == CName.NEQ:
        return 'neq'
    elif self == CName.LT:
        return 'lt'
    elif self == CName.GT:
        return 'gt'
    elif self == CName.AND:
        return 'and'
    elif self == CName.OR:
        return 'or'
    else:
        return 'noSuchCommand'
```


Virtual machine for an arithmetic calculator

We then prepare the commands used for our VM.

```
class Command(object):
    def __init__(self, cn, x):
        self.cname = cn
        if cn == CName.PUSH:
            self.num = x
    def __str__(self):
        if self.cname == CName.PUSH:
            return str(self.cname) + '(' + str(self.num) + ')'
        else:
            return str(self.cname)
```

push(3)

add

neq

and

Virtual machine for an arithmetic calculator

Command list Stack

[push(*n*), …] 

[mone, …] 

If the stack is empty, then an exception called VMError will be raised.

Virtual machine for an arithmetic calculator

[add, …] 

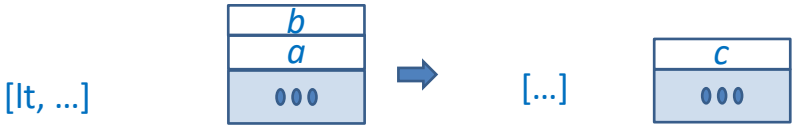
If the stack is empty or only consists of one element, then an exception called VMError will be raised.

[quo, …] 

If the stack is empty or only consists of one element, then an exception called VMError will be raised.

If *b* = 0, then an exception called DivisionByZero will be raised.

Virtual machine for an arithmetic calculator

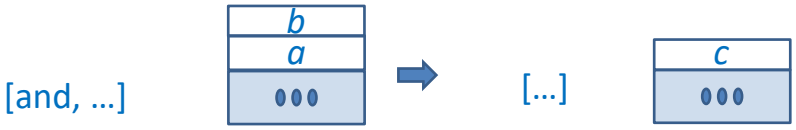


$$c = \begin{cases} 1 & \text{if } a < b \\ 0 & \text{otherwise} \end{cases}$$

If the stack is empty or only consists of one element, then an exception called VMError will be raised.

i116 Basic of Programming - 7. Arithmetic calculator: virtual machine & compiler

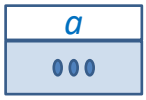
Virtual machine for an arithmetic calculator



$$c = \begin{cases} 0 & \text{if } a = 0 \text{ or } b = 0 \\ 1 & \text{otherwise} \end{cases}$$

If the stack is empty or only consists of one element, then an exception called VMError will be raised.

Virtual machine for an arithmetic calculator

[ ] 

➡ The VM returns *a* as the result.

If the stack is empty, then an exception called VMError will be raised.

Virtual machine for an arithmetic calculator

```
class DivisionByZero(Exception):
    pass

class VMError(Exception):
    pass
```

```
class Stack(object):
    ...
    def isEmpty(self):
        pass
    def isEmpOrOne(self):
        pass
```

```
class EmptyStack(Stack):
    ...
    def isEmpty(self):
        return True
    def isEmpOrOne(self):
        return True
```

```
class NeStack(Stack):
    ...
    def isEmpty(self):
        return False
    def isEmpOrOne(self):
        return self.botom.isEmpty()
```

Virtual machine for an arithmetic calculator

The procedure l2s converts a list of objects to a string such that the elements are displayed so that humans can read.

```
def l2s(lst):
    s = '['
    l = len(lst)
    for e in lst:
        s = s + str(e)
        l = l - 1
        if l > 0:
            s = s + ', '
    s = s + ']'
    return s
```


Virtual machine for an arithmetic calculator

The VM is defined as follows:

```
class VM(object):
    def __init__(self, cl):
        self.clist = cl
        self.stk = EmptyStack()

    def str(self):
        return 'stack: ' + str(self.stk) + ', command list: ' + l2s(self.clist)
```

Virtual machine for an arithmetic calculator

```
def run(self):
    for com in self.clist:
        if com.cname == CName.PUSH:
            self.stk = self.stk.push(com.num)
        elif com.cname == CName.MONE:
            if self.stk.isEmpty():
                raise VMError('stk is empty for mone')
            x = self.stk.top()
            self.stk = self.stk.pop().push(-1 * x)
        elif com.cname == CName.MUL:
            if self.stk.isEmpOrOne():
                raise VMError('stk consists of 1 or 0 element for mul')
            y = self.stk.top()
            x = self.stk.pop().top()
            self.stk = self.stk.pop().pop().push(x * y)
```


Virtual machine for an arithmetic calculator

```
        elif com.cname == CName.QUO:
            if self.stk.isEmpOrOne():
                raise VMError('stk consists of 1 or 0 element for quo')
            y = self.stk.top()
            x = self.stk.pop().top()
            if y == 0:
                raise DivisionByZero('division by zero in VM')
            self.stk = self.stk.pop().pop().push(x // y)
        elif com.cname == CName.REM:
            if self.stk.isEmpOrOne():
                raise VMError('stk consists of 1 or 0 element for rem')
            y = self.stk.top()
            x = self.stk.pop().top()
            if y == 0:
                raise DivisionByZero('division by zero in VM')
            self.stk = self.stk.pop().pop().push(x % y)
```

Virtual machine for an arithmetic calculator

```
elif com.cname == CName.ADD:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for add')
    y = self.stk.top()
    x = self.stk.pop().top()
    self.stk = self.stk.pop().pop().push(x + y)
elif com.cname == CName.SUB:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for sub')
    y = self.stk.top()
    x = self.stk.pop().top()
    self.stk = self.stk.pop().pop().push(x - y)
```


Virtual machine for an arithmetic calculator

```
elif com.cname == CName.EQ:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for eq')
    y = self.stk.top()
    x = self.stk.pop().top()
    if x == y:
        self.stk = self.stk.pop().pop().push(1)
    else:
        self.stk = self.stk.pop().pop().push(0)
elif com.cname == CName.NEQ:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for neq')
    y = self.stk.top()
    x = self.stk.pop().top()
    if x == y:
        self.stk = self.stk.pop().pop().push(0)
    else:
        self.stk = self.stk.pop().pop().push(1)
```

Virtual machine for an arithmetic calculator

```
elif com.cname == CName.LT:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for lt')
    y = self.stk.top()
    x = self.stk.pop().top()
    if x < y:
        self.stk = self.stk.pop().pop().push(1)
    else:
        self.stk = self.stk.pop().pop().push(0)
elif com.cname == CName.GT:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for gt')
    y = self.stk.top()
    x = self.stk.pop().top()
    if x > y:
        self.stk = self.stk.pop().pop().push(1)
    else:
        self.stk = self.stk.pop().pop().push(0)
```


Virtual machine for an arithmetic calculator

```
elif com.cname == CName.AND:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for and')
    y = self.stk.top()
    x = self.stk.pop().top()
    if x == 0 or y == 0:
        self.stk = self.stk.pop().pop().push(0)
    else:
        self.stk = self.stk.pop().pop().push(1)
elif com.cname == CName.OR:
    if self.stk.isEmpOrOne():
        raise VMError('stk consists of 1 or 0 element for or')
    y = self.stk.top()
    x = self.stk.pop().top()
    if x == 0 and y == 0:
        self.stk = self.stk.pop().pop().push(0)
    else:
        self.stk = self.stk.pop().pop().push(1)
```

Virtual machine for an arithmetic calculator

```
        else:
            raise VMError("An invalid command was met!")
    if self.stk.isEmpty():
        raise VMError('stk is empty when vm terminates!')
    return self.stk.top()
```


Virtual machine for an arithmetic calculator

```
push3 = Command(CName.PUSH,3)
push4 = Command(CName.PUSH,4)
push5 = Command(CName.PUSH,5)
add = Command(CName.ADD,None)
mul = Command(CName.MUL,None)
cl = [push3, push4, push5, mul, add]
print(l2s(cl))
vm = VM(cl)
print('VM result: ', vm.run())
```

```
push3 = Command(CName.PUSH,3)
add = Command(CName.ADD,None)
cl = [push3, add]
print(l2s(cl))
vm = VM(cl)
try:
    print('VM result: ', vm.run())
except VMError as em:
    print(em)
```

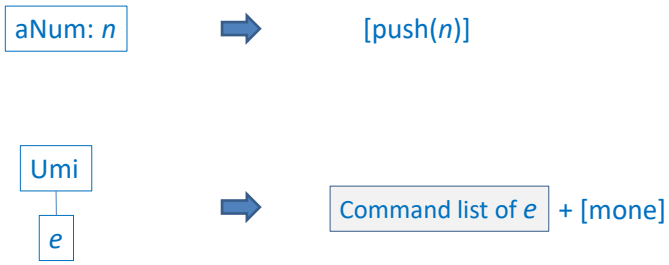
Virtual machine for an arithmetic calculator

```
push3 = Command(CName.PUSH,3)
push4 = Command(CName.PUSH,4)
push2 = Command(CName.PUSH,2)
add = Command(CName.ADD,None)
quo = Command(CName.QUO,None)
cl = [push3, push4, add, push2, quo]
print(l2s(cl))
vm = VM(cl)
print('VM result: ', vm.run())
```

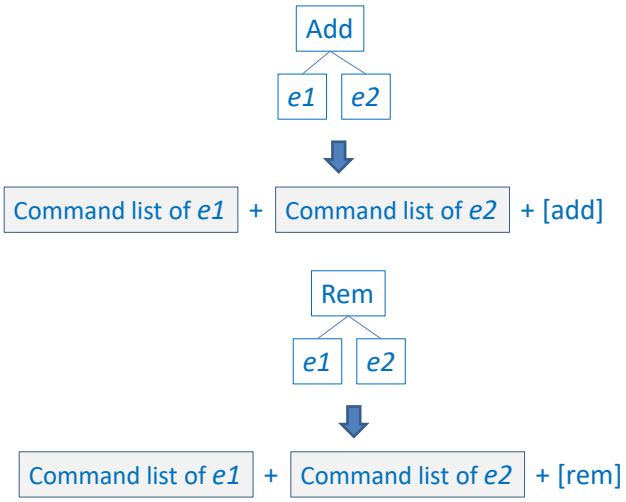
```
push3 = Command(CName.PUSH,3)
push4 = Command(CName.PUSH,4)
push0 = Command(CName.PUSH,0)
add = Command(CName.ADD,None)
rem = Command(CName.QUO,None)
cl = [push3, push4, add, push0, rem]
print(l2s(cl))
vm = VM(cl)
try:
   print('VM result: ', vm.run())
except DivisionByZero as em:
   print(em)
```


Compiler for an arithmetic calculator

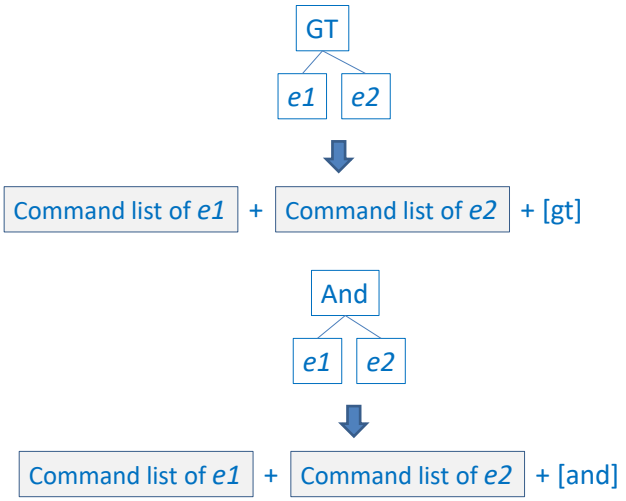
- We will make a compiler that generates a list of commands from an arithmetic expression parse tree.



Compiler for an arithmetic calculator



Compiler for an arithmetic calculator



Compiler for an arithmetic calculator

```
class ExpParseTree(object):
    ...
    def compile(self):
        pass
```

```
class NumParseTree(ExpParseTree):
    ...
    def compile(self):
        return [Command(CName.PUSH,self.num)]
```


Compiler for an arithmetic calculator

```
class UmiParseTree(ExpParseTree):
    ...
    def compile(self):
        cl1 = self.exp.compile()
        return cl1 + [Command(CName.MONE,None)]
```

```
class AddParseTree(ExpParseTree):
    ...
    def compile(self):
        cl1 = self.exp1.compile()
        cl2 = cl1 + self.exp2.compile()
        return cl2 + [Command(CName.ADD,None)]
```

Compiler for an arithmetic calculator

```
class SubParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.SUB,None)]
```

```
class MulParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.MUL,None)]
```


Compiler for an arithmetic calculator

```
class QuoParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.QUO,None)]
```

```
class RemParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.REM,None)]
```

Compiler for an arithmetic calculator

```
class LTParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.LT,None)]
```

```
class GTParseTree(ExpParseTree):
  exp1 = ExpParseTree()
  exp2 = ExpParseTree()
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.GT,None)]
```

i116 Basic of Programming - 7. Arithmetic calculator: virtual machine & compiler

Compiler for an arithmetic calculator

```
class EQParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.EQ,None)]
```

```
class NEQParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.NEQ,None)]
```

Compiler for an arithmetic calculator

```
class AndParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.AND,None)]
```

```
class OrParseTree(ExpParseTree):
  …
   def compile(self):
     cl1 = self.exp1.compile()
     cl2 = cl1 + self.exp2.compile()
     return cl2 + [Command(CName.OR,None)]
```


Compiler for an arithmetic calculator

3 + 4 * 5

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = MulParseTree(four,five)
e2 = AddParseTree(three,e1)
print(e2)
print(l2s(e2.compile()))
print(VM(e2.compile()).run())
```

(3 + 4) * 5

```
three = NumParseTree(3)
four = NumParseTree(4)
five = NumParseTree(5)
e1 = AddParseTree(three,four)
e2 = MulParseTree(e1,five)
print(e2)
print(l2s(e2.compile()))
print(VM(e2.compile()).run())
```

Compiler for an arithmetic calculator

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(l2s(e3.compile()))
print(VM(e3.compile()).run())
```

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(l2s(e3.compile()))
print(VM(e3.compile()).run())
```

i116 Basic of Programming - 7. Arithmetic calculator: virtual machine & compiler

Compiler for an arithmetic calculator

((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(l2s(e3.compile()))
print(VM(e3.compile()).run())
```

((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(l2s(e3.compile()))
try:
   print(VM(e3.compile()).run())
except DivisionByZero as em:
   print(em)
```

Compiler for an arithmetic calculator

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(l2s(e3.compile()))
print(VM(e3.compile()).run())
```

5 < 5 || 5 = 5

```
five = NumParseTree(5)
e1 = LTParseTree(five,five)
e2 = EQParseTree(five,five)
e3 = OrParseTree(e1,e2)
print(e3)
print(l2s(e3.compile()))
print(VM(e3.compile()).run())
```


Compiler for an arithmetic calculator

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(l2s(e10.compile()))
print(VM(e10.compile()).run())
```