i116: Basic of Programming

12. Programming language processor: interpreter

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Interpreter for Minila

Interpreter for Minila

- The three new statements have been introduced:
  - – Empty statement
  - – Conditional (**if**) statement
  - – Loop (**while**) statement
- Then, we need to have method interpret(…) in the class for each of the three statements.

Interpreter for Minila

```
class EmptyParseTree(StmParseTree):
    …
  def interpret(self,env):
      return env
```

EmptyStm .interpret({…})

returns

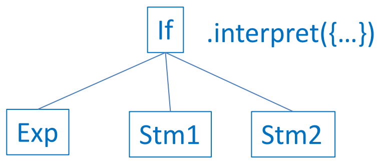
{…}

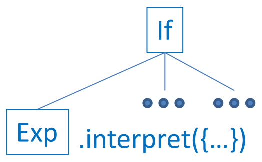
without changing the environment as it is.

Interpreter for Minila

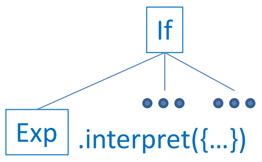
```
class IfParseTree(StmParseTree):
  …
  def interpret(self,env):
    if self.exp.interpret(env) != 0:
      return self.stm1.interpret(env)
    else:
      return self.stm2.interpret(env)
```

Interpreter for Minila



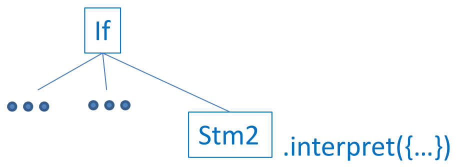


Interpreter for Minila

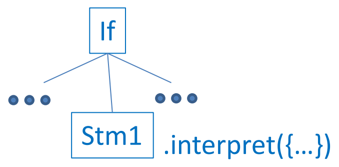


Note that an exception may be raised in Exp.interpret({…}).

If it returns 0,

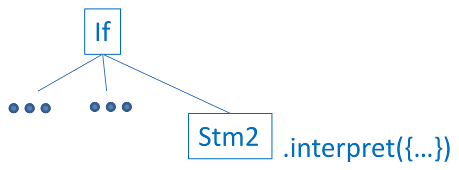


Otherwise,



Interpreter for Minila

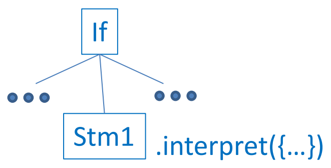
If it returns 0,



which returns the new environment obtained by Stm2.interpret({…}).

Note that an exception may be raised in Stm2.interpret({…}).

Otherwise,



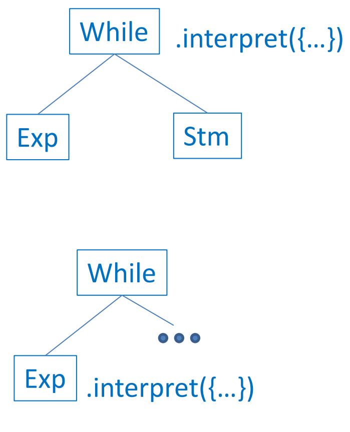
which returns the new environment obtained by Stm1.interpret({…}).

Note that an exception may be raised in Stm1.interpret({…}).

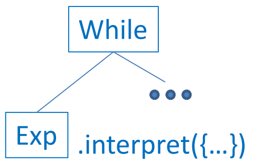
Interpreter for Minila

```
class WhileParseTree(StmParseTree):
    …
    def interpret(self,env):
        if self.exp.interpret(env) == 0:
            return env
        else:
            return self.interpret(self.stm.interpret(env))
```

Interpreter for Minila



Interpreter for Minila

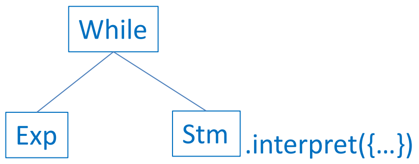


Note that an exception may be raised in Exp.interpret({…}).

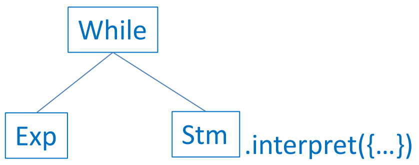
If it returns 0,

{…} is returned as the result of interpreting the While statement.

Otherwise,

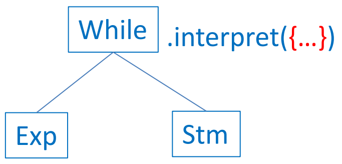


Interpreter for Minila

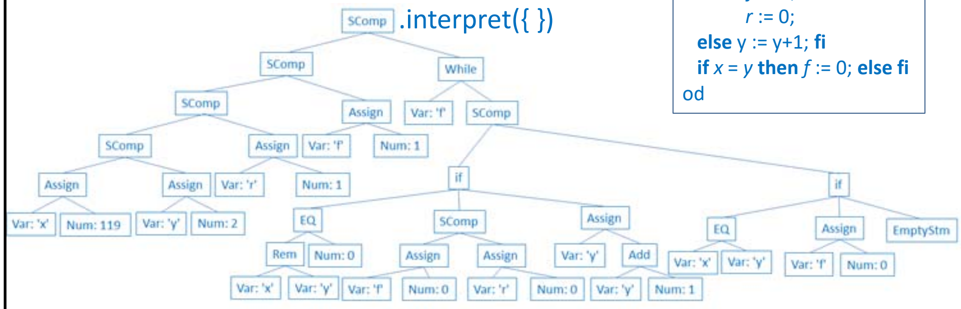


Note that an exception may be raised in Stm.interpret({…}).

If Stm.interpret({…}) returns a new environment {…},

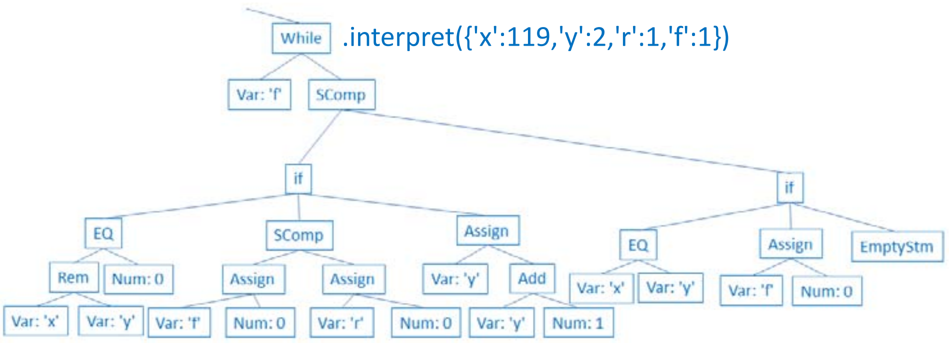


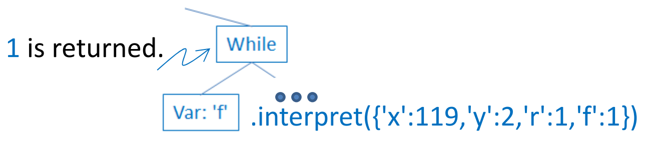
Interpreter for Minila



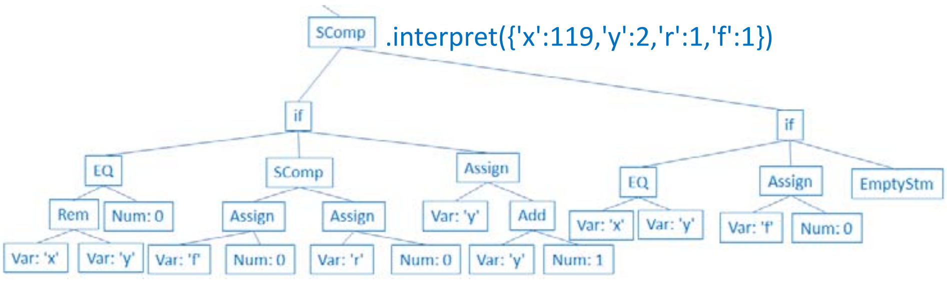
```
x := 119;
y := 2;
r := 1;
f := 1;
while f do
   if x % y = 0
   then f := 0;
           r := 0;
  else y := y+1; fi
  if x = y then f := 0; else fi
od
```

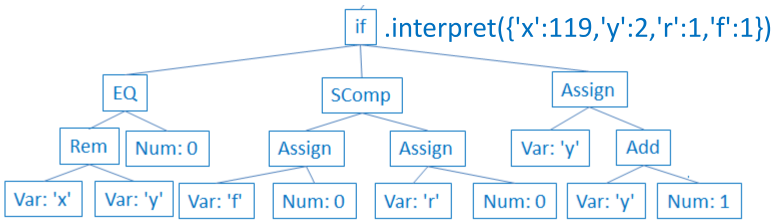
Interpreter for Minila



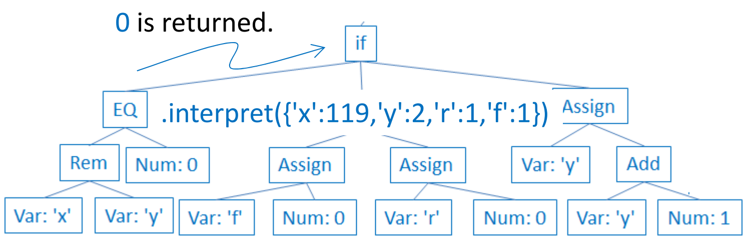


Interpreter for Minila

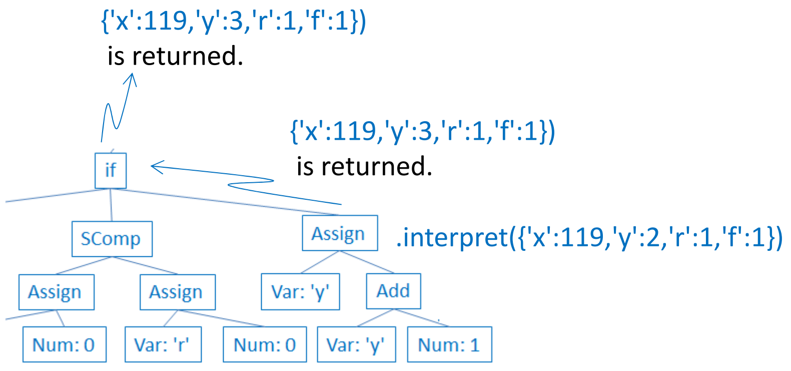




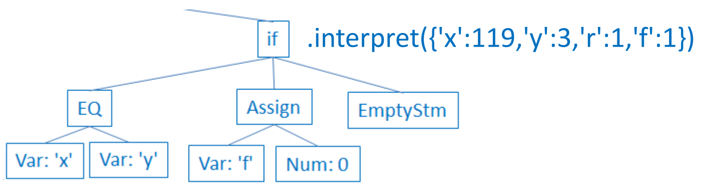
Interpreter for Minila

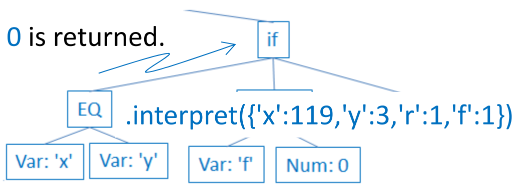


Interpreter for Minila

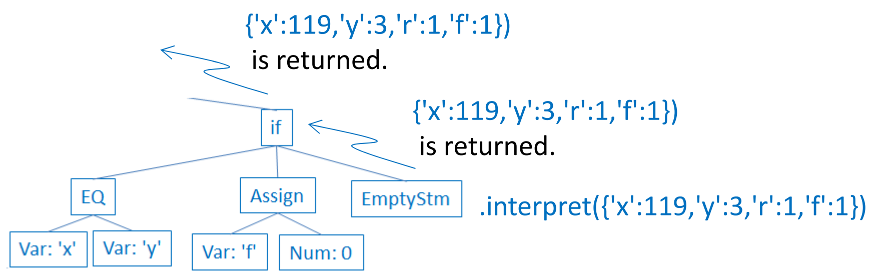

{'x':119,'y':3,'r':1,'f':1})
is returned.
{'x':119,'y':3,'r':1,'f':1})
is returned.
if
SComp
Assign
.interpret({'x':119,'y':2,'r':1,'f':1})
Assign
Assign
Var: 'y'
Add
Num: 0
Var: 'r'
Num: 0
Var: 'y'
Num: 1

Interpreter for Minila

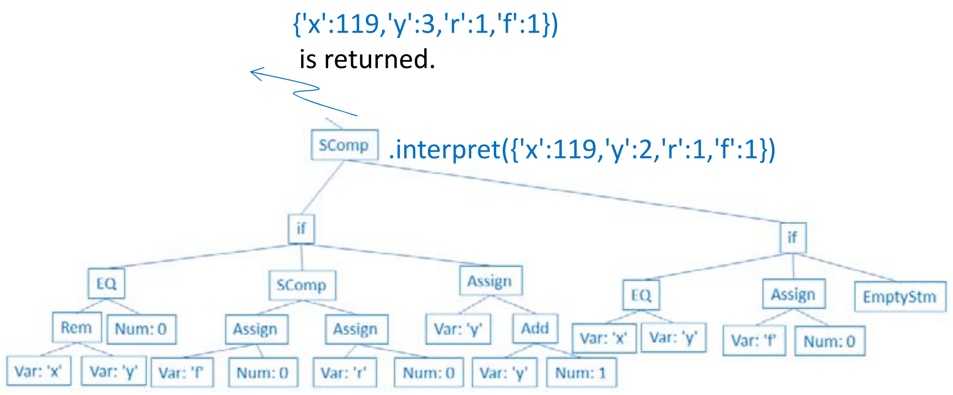




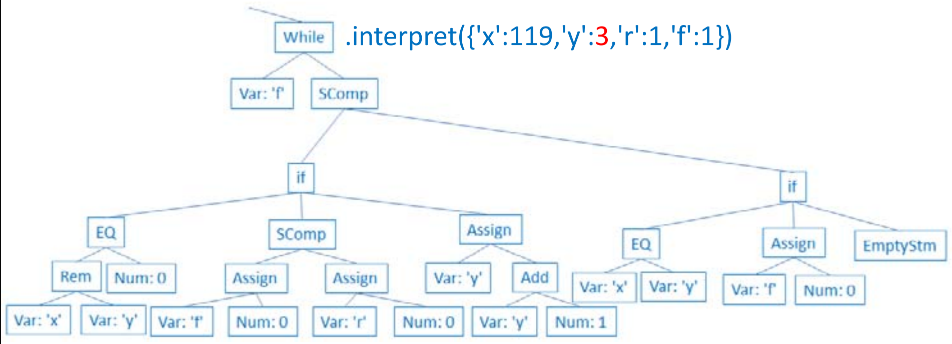
Interpreter for Minila


{'x':119,'y':3,'r':1,'f':1})
is returned.
{'x':119,'y':3,'r':1,'f':1})
is returned.
if
EQ
Assign
EmptyStm
.interpret({'x':119,'y':3,'r':1,'f':1})
Var: 'x'
Var: 'y'
Var: 'f'
Num: 0

Interpreter for Minila

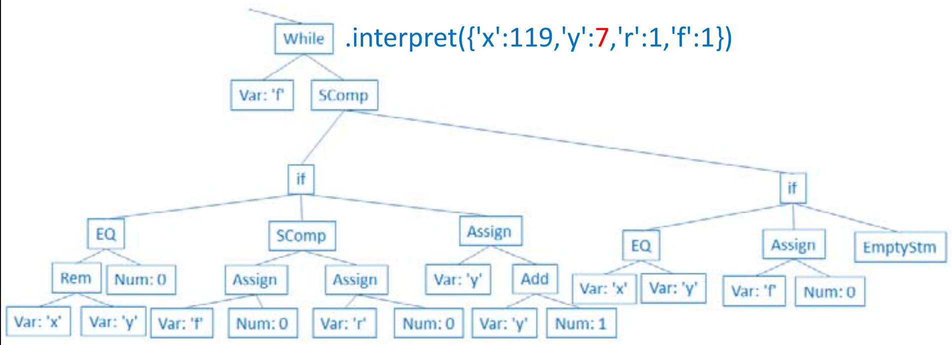


Interpreter for Minila



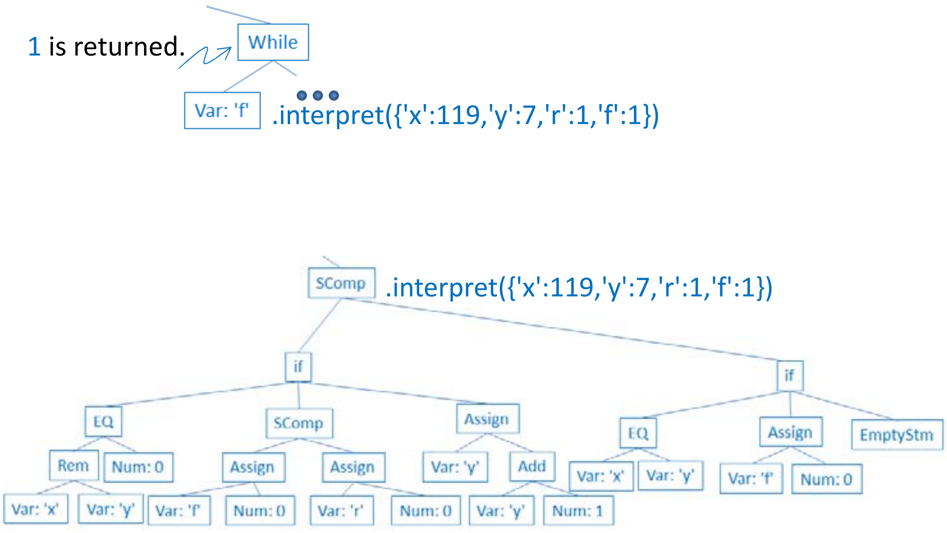
.interpret({'x':119,'y':3,'r':1,'f':1})

Interpreter for Minila

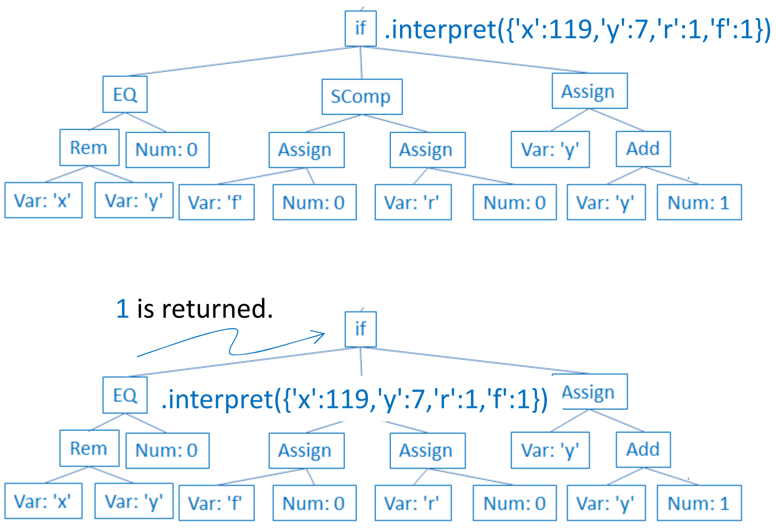


.interpret({'x':119,'y':7,'r':1,'f':1})

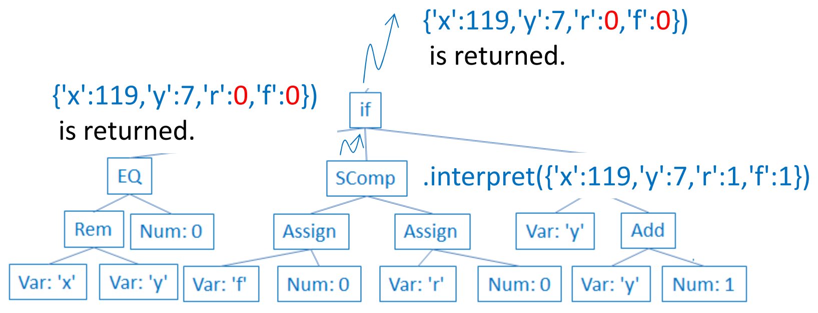
Interpreter for Minila



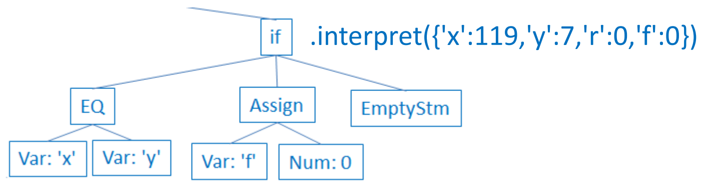
Interpreter for Minila

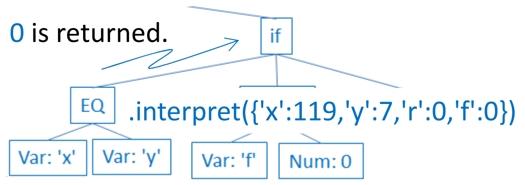


Interpreter for Minila

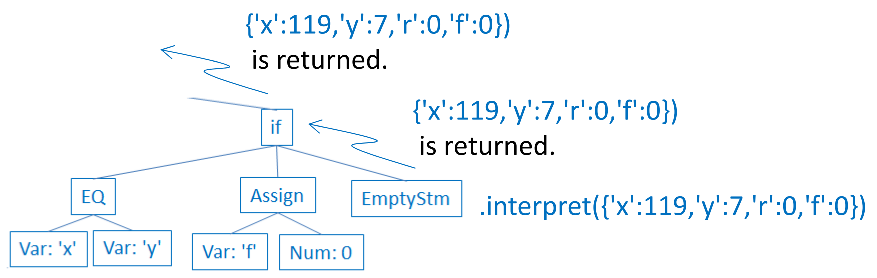


Interpreter for Minila

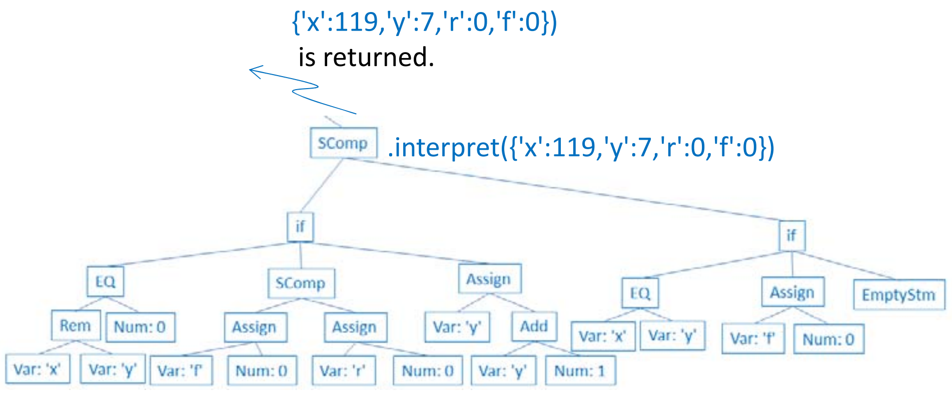




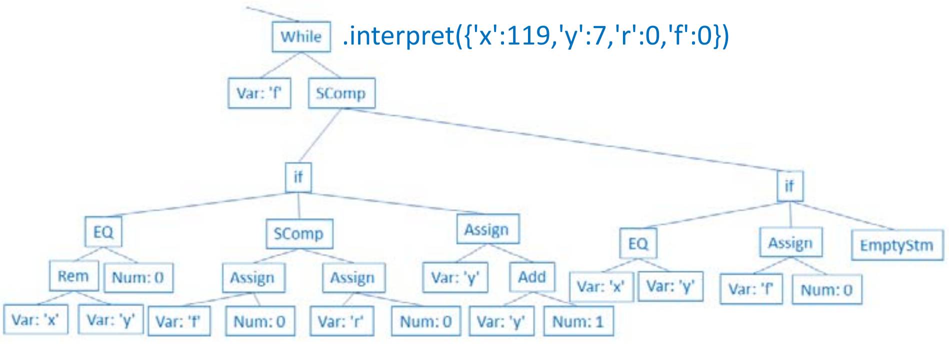
Interpreter for Minila

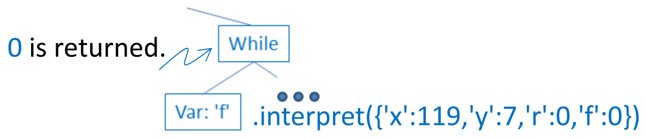

{'x':119,'y':7,'r':0,'f':0})
is returned.
{'x':119,'y':7,'r':0,'f':0})
is returned.
if
EQ
Assign
EmptyStm
.interpret({'x':119,'y':7,'r':0,'f':0})
Var: 'x'
Var: 'y'
Var: 'f'
Num: 0

Interpreter for Minila

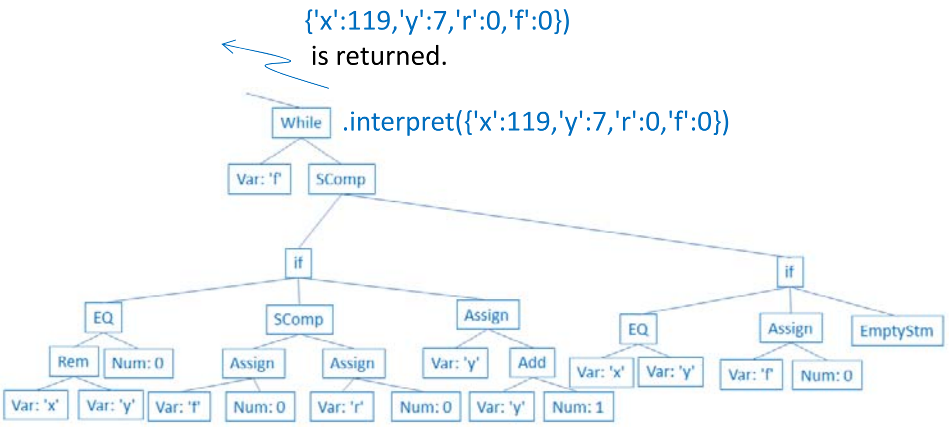


Interpreter for Minila



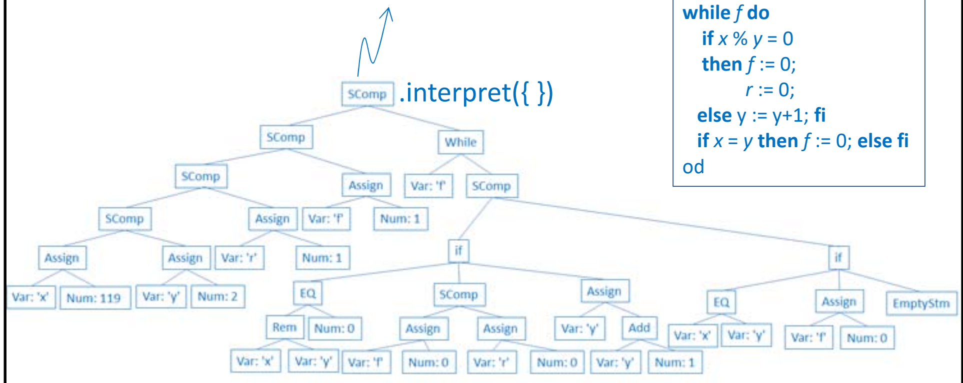


Interpreter for Minila



Interpreter for Minila

{'x':119,'y':7,'r':0,'f':0})
is returned.



```
x := 119;
y := 2;
r := 1;
f := 1;
while f do
   if x % y = 0
   then f := 0;
           r := 0;
  else y := y+1; fi
  if x = y then f := 0; else fi
od
```

Interpreter for Minila

```
from scan import *
from parse import *
fact = ' '\
'   x := 1;'\
'   y := 1;'\
'   while y < 5'\
'   do'\
'      x := x * y;'\
'      y := y + 1;'\
'   od'
tl = scan(fact)
tlo = TokenList(tl)
pt = tlo.parse()
print(pt.interpret({ }))
```

```
from scan import *
from parse import *
fact = ' '\
'   x := 1;'\
'   y := 1;'\
'   while y < 10 || y = 10'\
'   do'\
'      x := x * y;'\
'      y := y + 1;'\
'   od'
tl = scan(fact)
tlo = TokenList(tl)
pt = tlo.parse()
print(pt.interpret({ }))
```

Interpreter for Minila

```
from scan import *
from parse import *
gcd = ' '\
'   x := 19110; '\
'   y := 17850; '\
'   while y != 0 do '\
'     tmp := x%y; '\
'     x := y; '\
'     y := tmp; '\
'   od '
tl = scan(gcd)
tlo = TokenList(tl)
pt = tlo.parse()
print(pt.interpret({ }))
```

Interpreter for Minila

```
from scan import *
from parse import *
isPrime = ' '\
'   x := 119; '\
'   y := 2; '\
'   r := 1; '\
'   f := 1; '\
'   while f do '\
'       if x % y = 0 '\
'       then f := 0; '\
'              r := 0; '\
'      else y := y+1; fi '\
'      if x = y then f := 0; else fi '\
'   od '
```

```
tl = scan(isPrime)
tlo = TokenList(tl)
pt = tlo.parse()
print(pt.interpret({ }))
```

Interpreter for Minila

```
from scan import *
from parse import *
sr = ' '\
'   v0 := 200000000000000000; '\
'   v1 := 0; '\
'   v2 := v0; '\
'   while v1 != v2 do '\
'     if (v2-v1)%2 = 0 '\
'       then v3 := v1+(v2-v1)/2; '\
'       else v3 := v1+(v2-v1)/2+1; '\
'     fi '\
'     if v3*v3 > v0 '\
'       then v2 := v3-1; '\
'       else v1 := v3; '\
'     fi '\
'   od '
```

```
tl = scan(sr)
tlo = TokenList(tl)
pt = tlo.parse()
print(pt.interpret({ }))
```