

i116: Basic of Programming

14. Programming language processor: compiler

Kazuhiro Ogata

i116 Basic of Programming - 14. Programming language processor: compiler

Roadmap

- Compiler for Minila

Compiler for Minila

- Minila has three more statements than the assignment calculator:
 - Empty statement
 - Conditional (**if**) statement
 - Loop (**while**) statement
- The compiler for Minila needs to generate command lists for the three statements.

Compiler for Minila

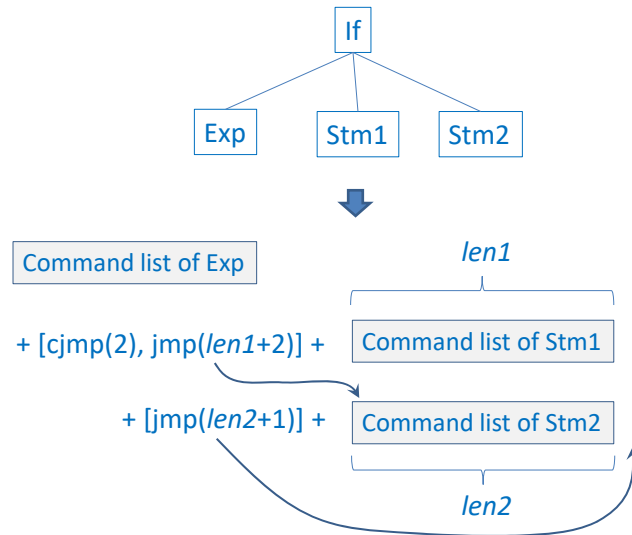
EmptyStm



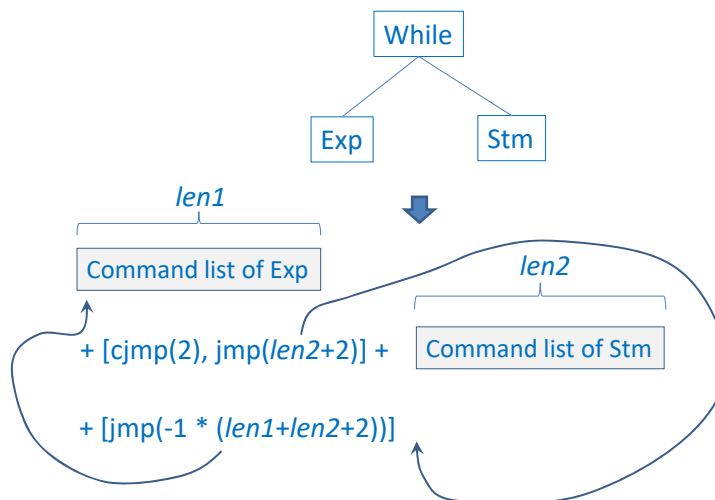
[]

The empty list of commands

Compiler for Minila



Compiler for Minila



Compiler for Minila

```
class EmptyParseTree(StmParseTree):
```

```
    ...
```

```
    def compile(self):
```

```
        return []
```

```
EmptyStm .compile()
```

generates the empty list of commands

```
[]
```

Compiler for Minila

```
class IfParseTree(StmParseTree):
```

```
    ...
```

```
    def compile(self):
```

```
        c1 = self.exp.compile()
```

```
        c2 = self.stm1.compile()
```

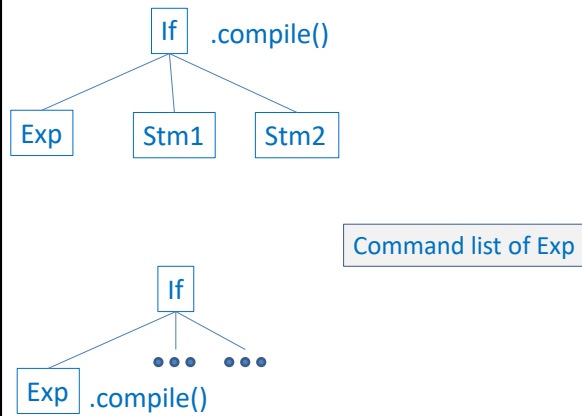
```
        c3 = self.stm2.compile()
```

```
        c4 = [Command(CName.CJMP,2), Command(CName.JMP,len(c2) + 2)]
```

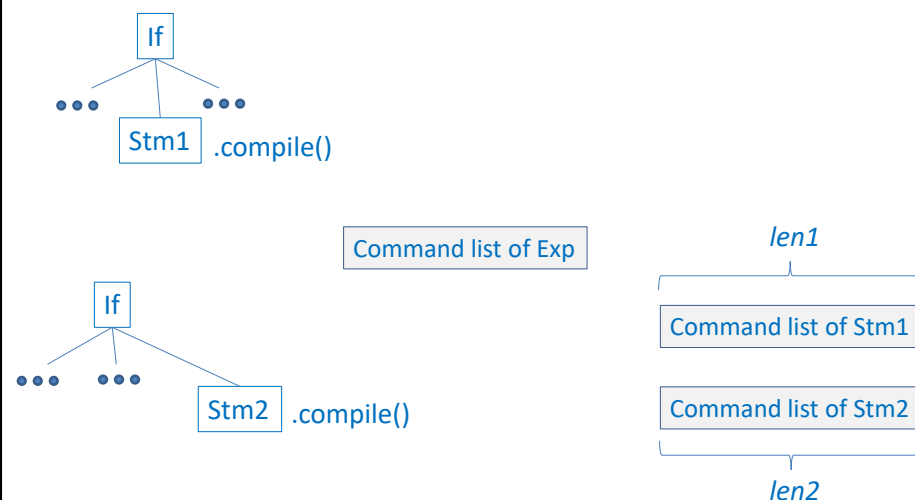
```
        c5 = [Command(CName.JMP,len(c3) + 1)]
```

```
        return c1 + c4 + c2 + c5 + c3
```

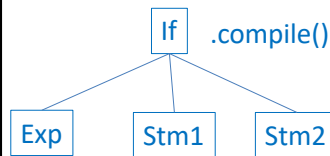
Compiler for Minila



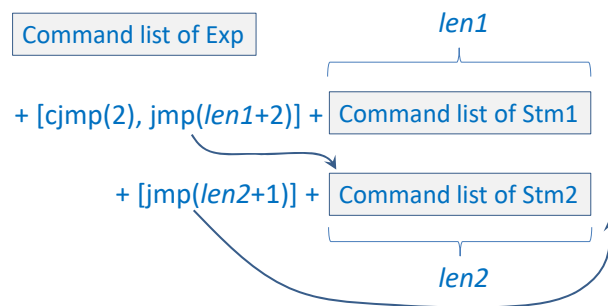
Compiler for Minila



Compiler for Minila



It generates the following:



Compiler for Minila

```
class WhileParseTree(StmParseTree):
```

```
...
```

```
def compile(self):
```

```
    c1 = self.exp.compile()
```

```
    c2 = self.stm.compile()
```

```
    size1 = len(c1)
```

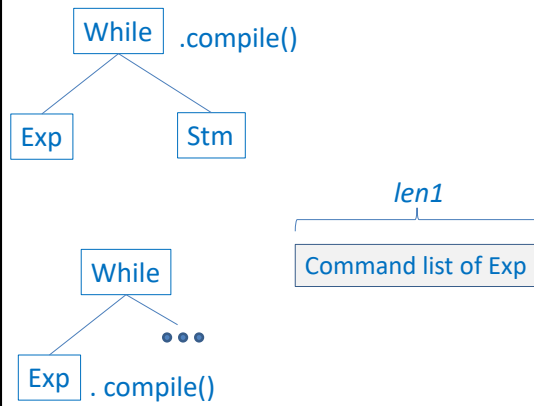
```
    size2 = len(c2)
```

```
    c3 = [Command(CName.CJMP,2), Command(CName.JMP,size2 + 2)]
```

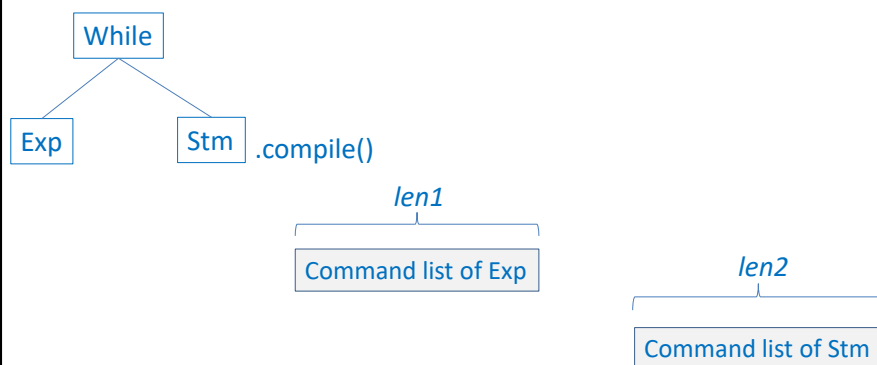
```
    c4 = [Command(CName.JMP,-1 * (size1 + size2 + 2))]
```

```
    return c1 + c3 + c2 + c4
```

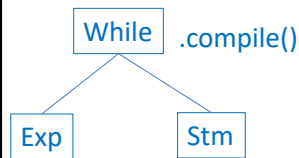
Compiler for Minila



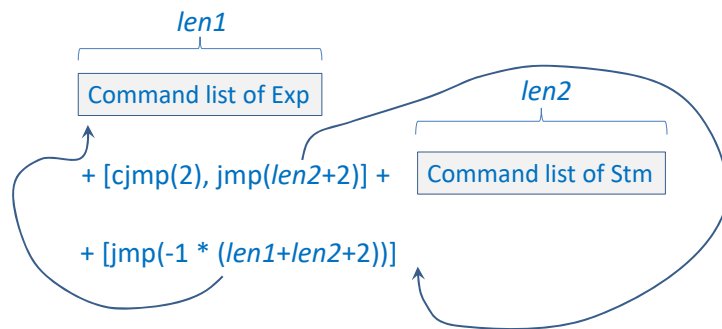
Compiler for Minila



Compiler for Minila



It generates the following:



Compiler for Minila

- Let *pgm* be a program in Minila.
- After generating the command list from *pgm*, the command **quit** is finally generated.

Command list of *pgm* + [quit]

- To this end, we add the method `genCode()` to the class `StmParseTree`.

Compiler for Minila

```
class StmParseTree(object):
```

...

```
def genCode(self):
```

```
return self.compile() + [Command(CName.QUIT,None)]
```

```
pgm.genCode()
```

It invokes the method `genCode()` in `StmParseTree` because `genCode()` is not defined in each of the five statement classes.

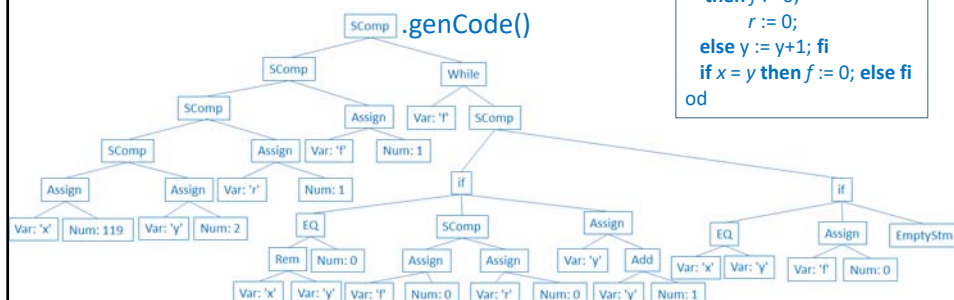
Command list of *pgm* + [quit]

generated by

```
pgm.compile()
```

Compiler 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
```



```

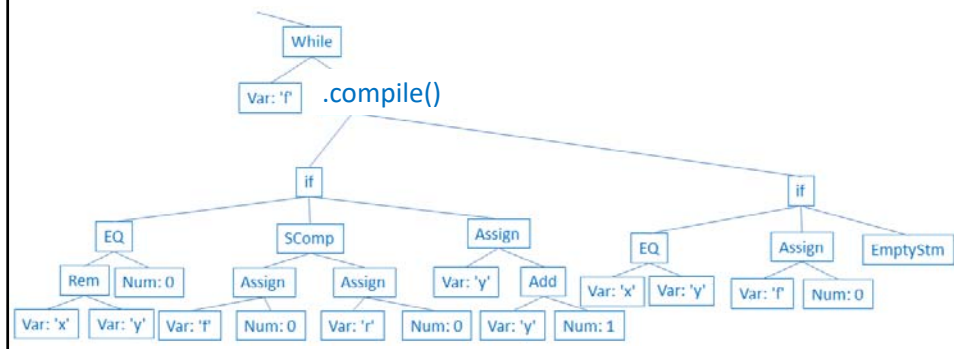
graph TD
    Root[SComp.compile()] --> SComp1[SComp]
    Root --> While[While]
    SComp1 --> SComp2[SComp]
    SComp1 --> Assign1[Assign]
    SComp2 --> SComp3[SComp]
    SComp2 --> Assign2[Assign]
    SComp3 --> SComp4[SComp]
    SComp3 --> Assign3[Assign]
    SComp4 --> VarX[Var: 'x']
    SComp4 --> Num119[Num: 119]
    Assign3 --> VarY[Var: 'y']
    Assign3 --> Num2[Num: 2]
    Assign1 --> VarP[Var: 'f']
    Assign1 --> Num1[Num: 1]
    While --> If1[if]
    While --> SComp5[SComp]
    While --> If2[if]
    While --> EmptyStm[EmptyStm]
    If1 --> EQ1[EQ]
    EQ1 --> Rem[Rem]
    EQ1 --> Num0[Num: 0]
    SComp5 --> Assign4[Assign]
    SComp5 --> Assign5[Assign]
    Assign4 --> VarX2[Var: 'x']
    Assign4 --> VarY2[Var: 'y']
    Assign4 --> VarP2[Var: 'f']
    Assign4 --> Num02[Num: 0]
    Assign5 --> VarY3[Var: 'y']
    Assign5 --> Num03[Num: 0]
    If2 --> EQ2[EQ]
    EQ2 --> VarX3[Var: 'x']
    EQ2 --> VarY3[Var: 'y']
    If2 --> Assign6[Assign]
    Assign6 --> VarP3[Var: 'f']
    Assign6 --> Num04[Num: 0]
  
```

```

graph TD
    While[While .compile()] --> VarP[Var: 'P']
    While --> SComp1[SComp]
    SComp1 --> if1[if]
    SComp1 --> if2[if]
    if1 --> EQ1[EQ]
    if1 --> SComp2[SComp]
    if1 --> Assign1[Assign]
    EQ1 --> Rem[Rem]
    EQ1 --> Num0_1[Num: 0]
    Rem --> VarX_1[Var: 'x']
    Rem --> VarY_1[Var: 'y']
    SComp2 --> Assign2[Assign]
    SComp2 --> Assign3[Assign]
    Assign2 --> VarP_2[Var: 'P']
    Assign2 --> Num0_2[Num: 0]
    Assign3 --> VarR_1[Var: 'r']
    Assign3 --> Num0_3[Num: 0]
    Assign1 --> VarY_2[Var: 'y']
    Assign1 --> Add[Add]
    Add --> VarY_3[Var: 'y']
    Add --> Num1[Num: 1]
    if2 --> EQ2[EQ]
    if2 --> Assign4[Assign]
    if2 --> EmptyStm[EmptyStm]
    EQ2 --> VarX_2[Var: 'x']
    EQ2 --> VarY_2_2[Var: 'y']
    Assign4 --> VarP_3[Var: 'P']
    Assign4 --> Num0_4[Num: 0]
  
```

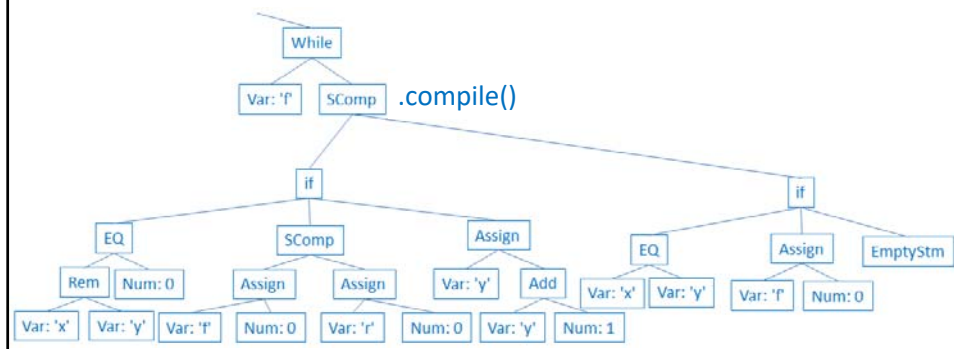
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(??),



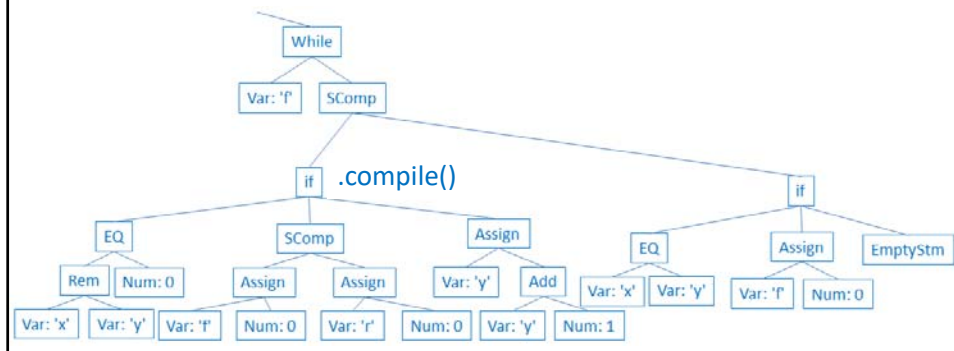
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(??),



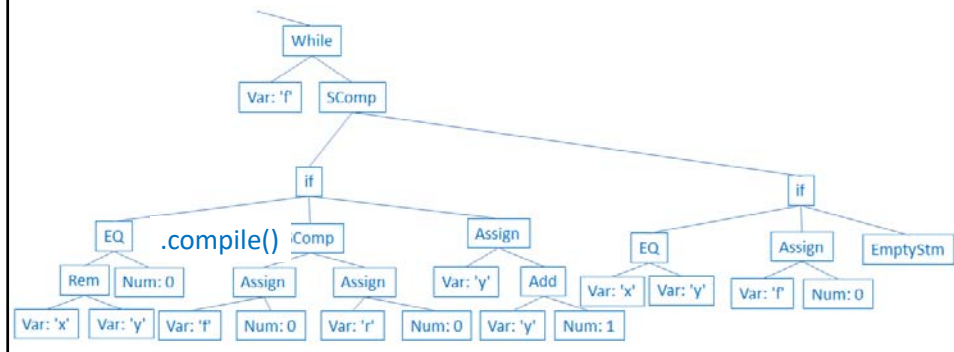
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(??),



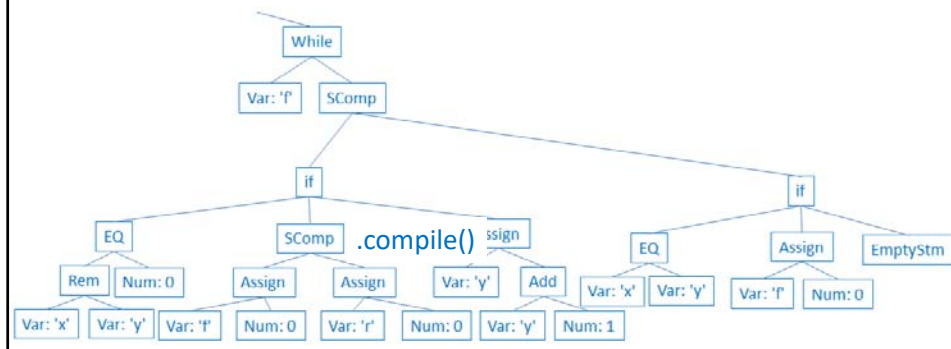
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(??),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(??),



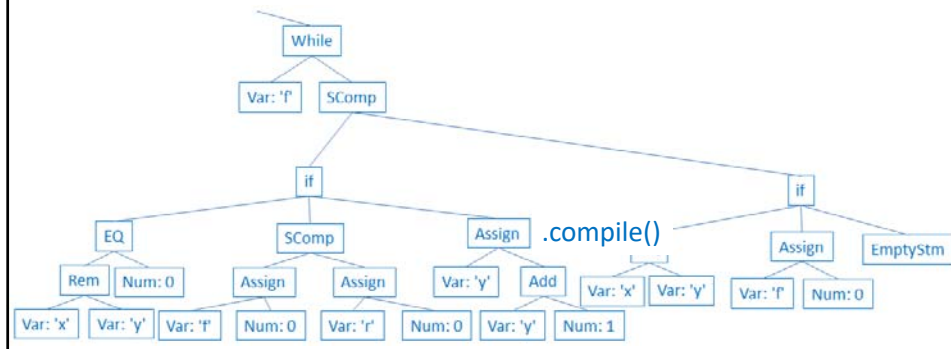
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(???),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(???),
push(0), store(f), push(0), store(r), jmp(???),



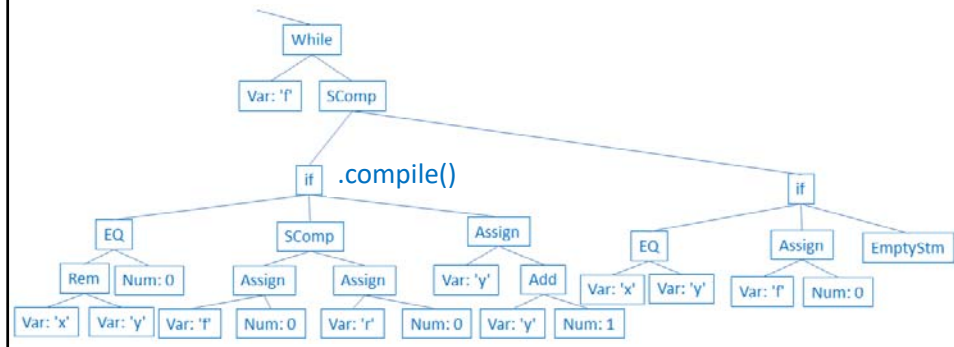
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(???),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(???),
push(0), store(f), push(0), store(r), jmp(???),
load(y), push(1), add, store(y),



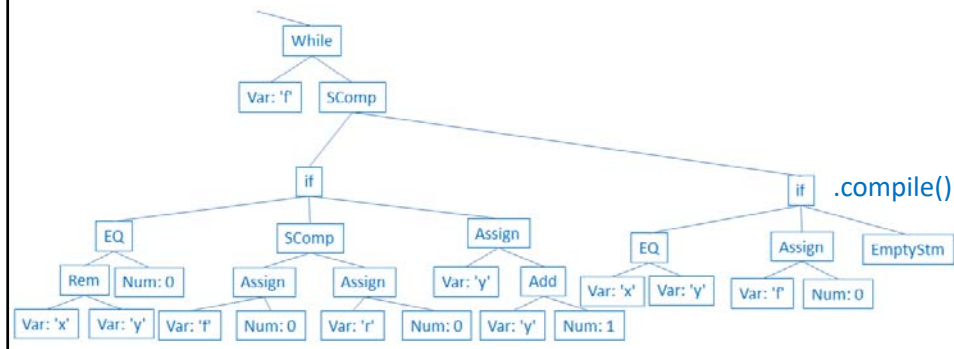
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(??),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(6),
push(0), store(f), push(0), store(r), jmp(5),
load(y), push(1), add, store(y),



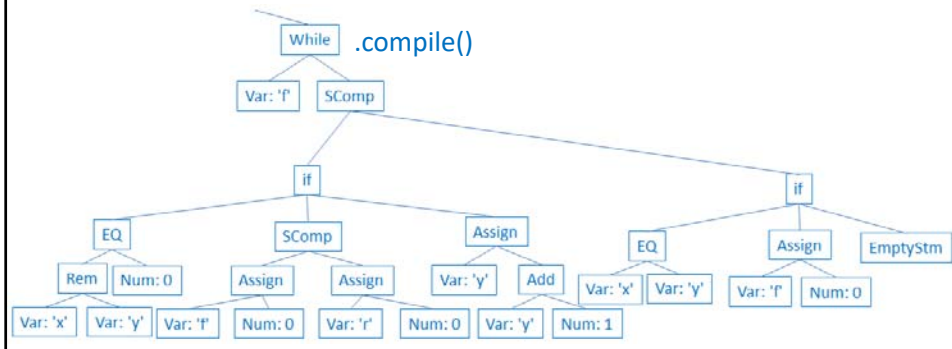
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(??),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(6),
push(0), store(f), push(0), store(r), jmp(5),
load(y), push(1), add, store(y),
load(x), load(y), eq, cjmp(2), jmp(4), push(0), store(f), jmp(1),



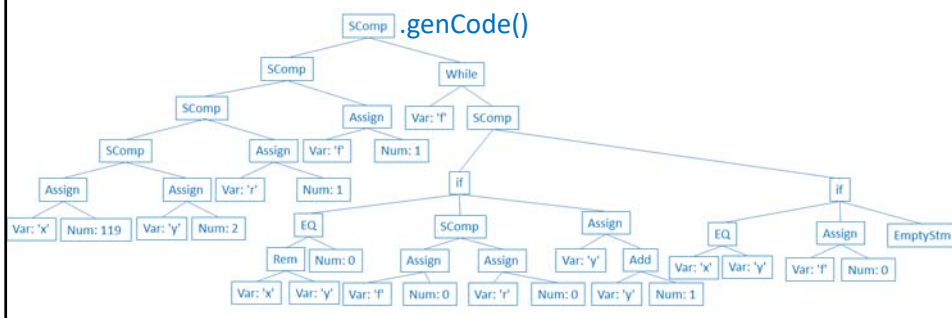
Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(26),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(6),
push(0), store(f), push(0), store(r), jmp(5),
load(y), push(1), add, store(y),
load(x), load(y), eq, cjmp(2), jmp(4), push(0), store(f), jmp(1), jmp(-27),
quit]



Interpreter for Minila

[push(119), store(x), push(2), store(y), push(1), store(r), push(1), store(f),
load(f), cjmp(2), jmp(26),
load(x), load(y), rem, push(0), eq, cjmp(2), jmp(6),
push(0), store(f), push(0), store(r), jmp(5),
load(y), push(1), add, store(y),
load(x), load(y), eq, cjmp(2), jmp(4), push(0), store(f), jmp(1), jmp(-27),
quit]



Interpreter for Minila

```

from scan import *
from parse import *
from vm 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()
cl = pt.genCode()
print(l2s(cl))
print(VM(cl).run())

```

```

from scan import *
from parse import *
from vm 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()
cl = pt.genCode()
print(l2s(cl))
print(VM(cl).run())

```

Interpreter for Minila

```

from scan import *
from parse import *
from vm 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()
cl = pt.genCode()
print(l2s(cl))
print(VM(cl).run())

```


Interpreter for Minila

```

from scan import *
from parse import *
from vm 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()
cl = pt.genCode()
print(l2s(cl))
print(VM(cl).run())

```

Interpreter for Minila

```

from scan import *
from parse import *
from vm import *
sr = ' \
' v0 := 20000000000000000; '\
' 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()
cl = pt.genCode()
print(l2s(cl))
print(VM(cl).run())

```