

# Intermediate Representation

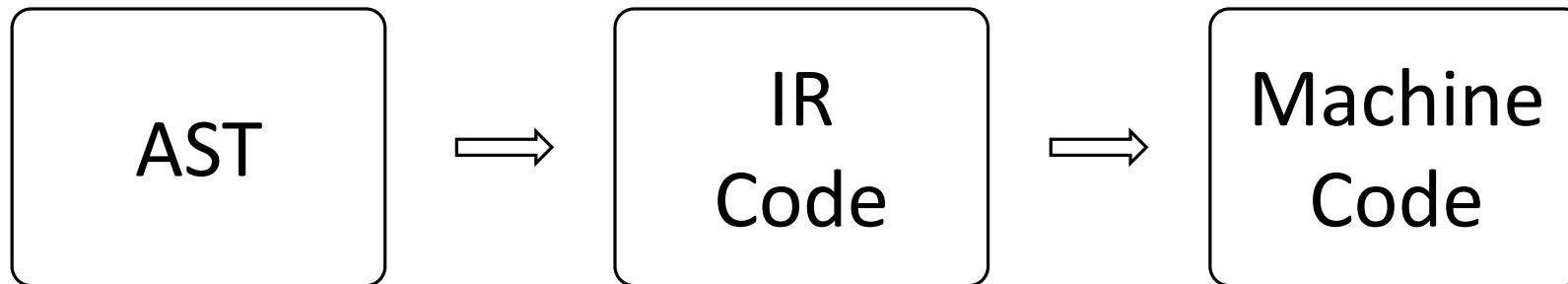
---

TEACHING ASSISTANT: DAVID TRABISH

A solid orange horizontal bar spanning the width of the slide at the bottom.

# Intermediate Representation

- Generic representation of instructions
  - Allows **language** and **machine** independent optimizations
  - Not executable



# IR Language

- Temporary variables (IR registers)
  - t1, t2, ... (unlimited)
- Instructions
  - assignments, add, sub, call, return, ...
- Labels
  - label\_1:

# IR Instructions

Constant assignment:

- **<register> = <constant>**
  - $t1 = 7$

# IR Instructions

Read from memory:

- **<register> = <variable\_name>**
  - $t1 = x$

Write to memory:

- **<variable\_name> = <register>**
  - $y = t2$

# IR Instructions

Arithmetic operations:

- **<register> = op <register> <register> ...**
  - t4 = add t1 t2
  - t0 = sub t0 t1

# IR Instructions

Branches:

- **br <label>**
  - br some\_label
- **beq <register> [<constant> | <register>] <label>**
  - beq t1 0 label\_1
  - beq t2 t3 label\_7

# IR Instructions

Functions:

- **call** <function\_name> <args>
  - call bar
  - call foo t1 t2
- **<register> = call** <function\_name> <args>
  - t8 = call foo t7
- **return** <register>
  - return t3



# IR Instructions

Arrays:

- **<register> = new\_array <register>**
  - t0 = new\_array t1
- **<register> = array\_access <register> <register>**
  - t0 = array\_access t1 t2
- **array\_set <register> <register> <register>**
  - array\_set t0 t1 t2

# IR Instructions

Classes:

- **<register> = new\_class <type\_name>**
  - t0 = new\_class Base
- **<register> = field\_access <register> <field\_name>**
  - t0 = field\_access t1 name
- **field\_set <register> <field\_name> <register>**
  - field\_set t0 name t2
- **virtual\_call <register> <method\_name> <args>**
  - virtual\_call t1 foo
- **<register> = virtual\_call <register> <method\_name> <args>**
  - t0 = virtual\_call t1 foo t20, t21

# IR Example

```
int foo(int x, int y) {  
    int z = x + y;  
    int w = z + 1;  
    return w;  
}
```

```
{  
    t1 = x  
    t2 = y  
    t3 = add t1, t2  
    z = t3  
    t4 = z  
    t5 = 1  
    t6 = add t4, t5  
    w = t6  
    t7 = w  
    return t7  
}
```

# Translating AST to IR

- Input: **AST**
- Output: **List of IR instructions**
- Done using **AST visitor**

# Translating Expressions

Basic algorithm:

*visit(node):*

$instlist_1, t_1 = visit(node.child_1)$

...

$instlist_n, t_n = visit(node.child_n)$

$instlist, t_{new} = assemble(instlist_1, \dots, instlist_n, t_1, \dots, t_n)$

*return instlist, t<sub>new</sub>*

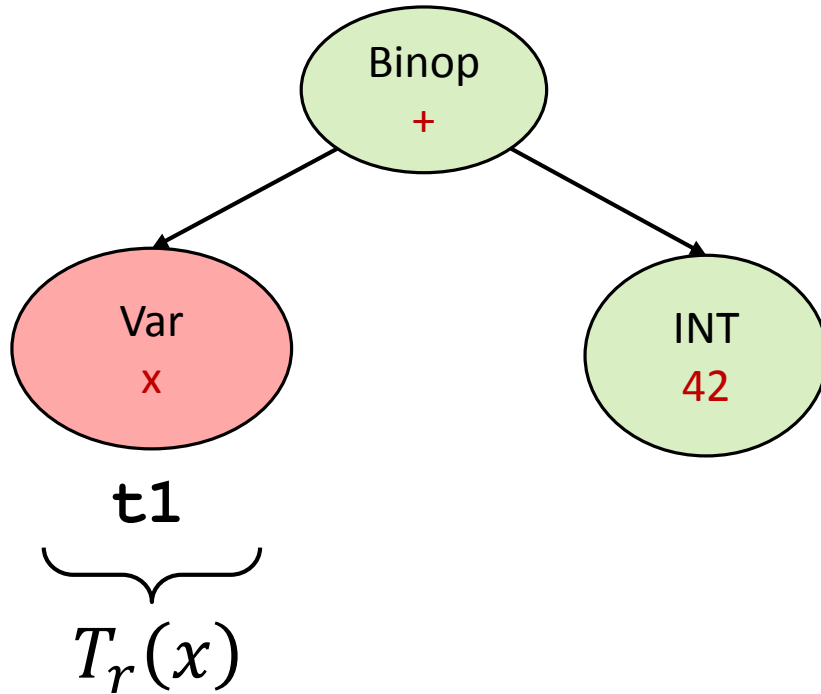
# Translating Expressions

For an AST node  $e$  we define:

- $T_c(e)$ 
  - The generated instructions (code)
- $T_r(e)$ 
  - The register holding the result of the computation

# Translating Expressions

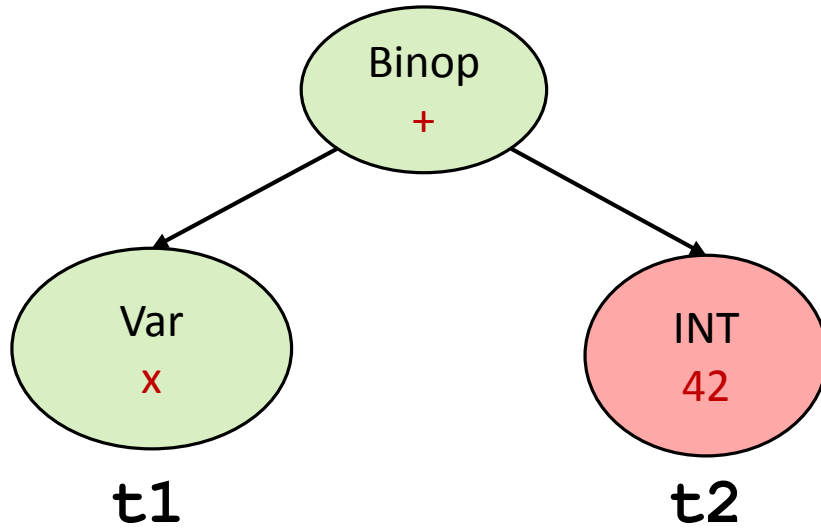
$x + 42$ :



$$\underbrace{\mathbf{t1} = \mathbf{x}}_{T_c(x)}$$

# Translating Expressions

$x + 42$ :

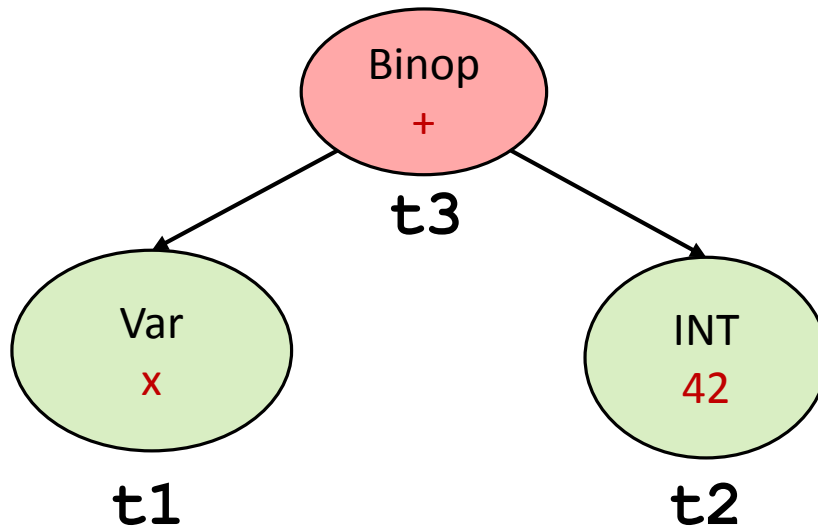


**t1 = x**  
**t2 = 42**



# Translating Expressions

$x + 42$ :



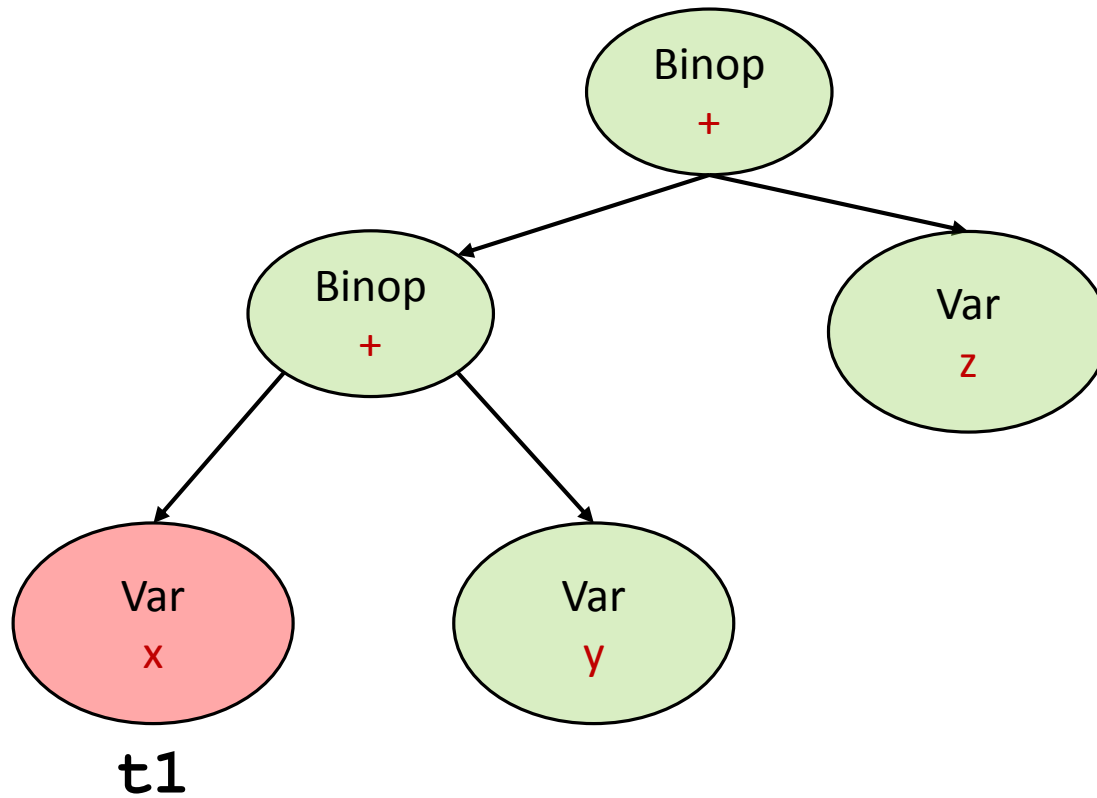
**t1 = x**

**t2 = 42**

**t3 = add t1, t2**

# Translating Expressions

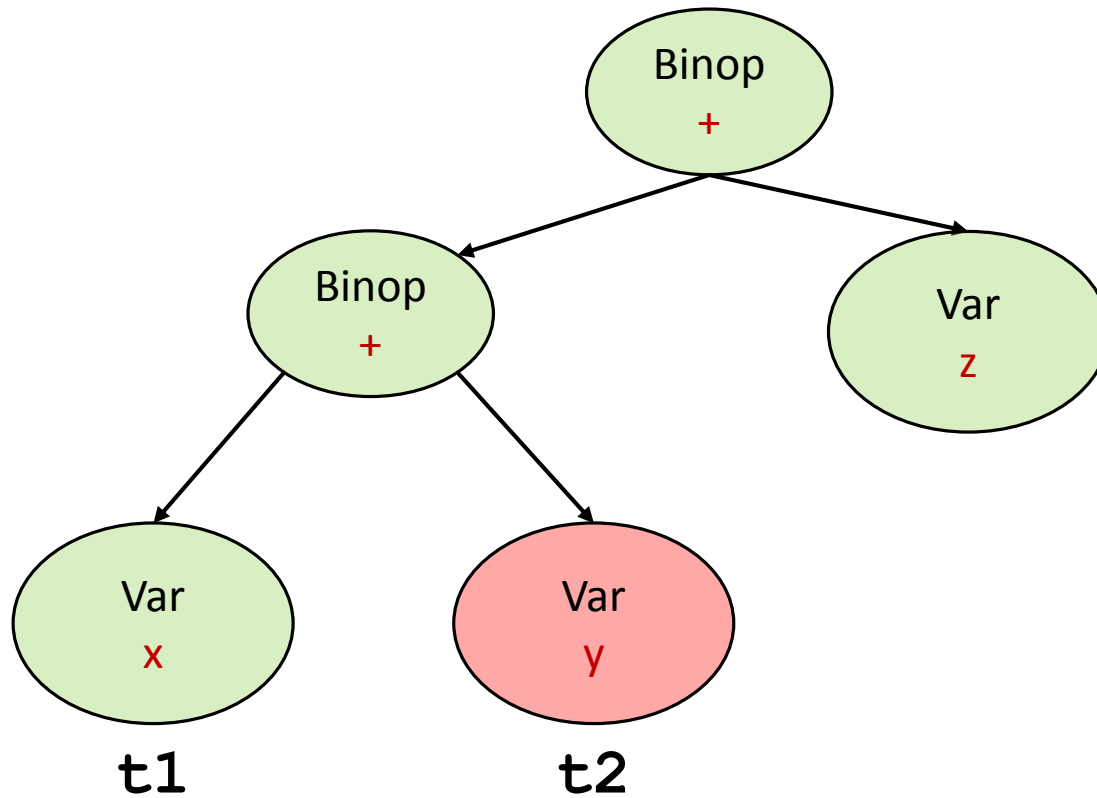
$x + y + z$ :



**t1 = x**

# Translating Expressions

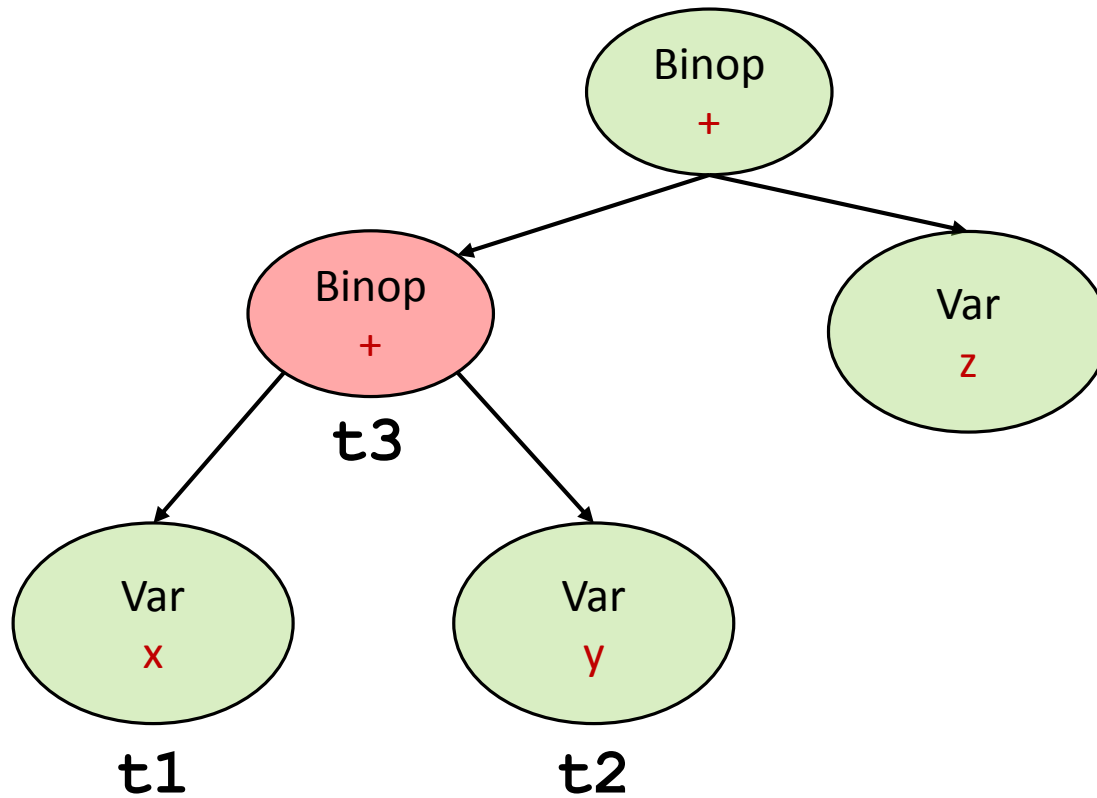
$x + y + z$ :



**t1 = x**  
**t2 = y**

# Translating Expressions

$x + y + z$ :



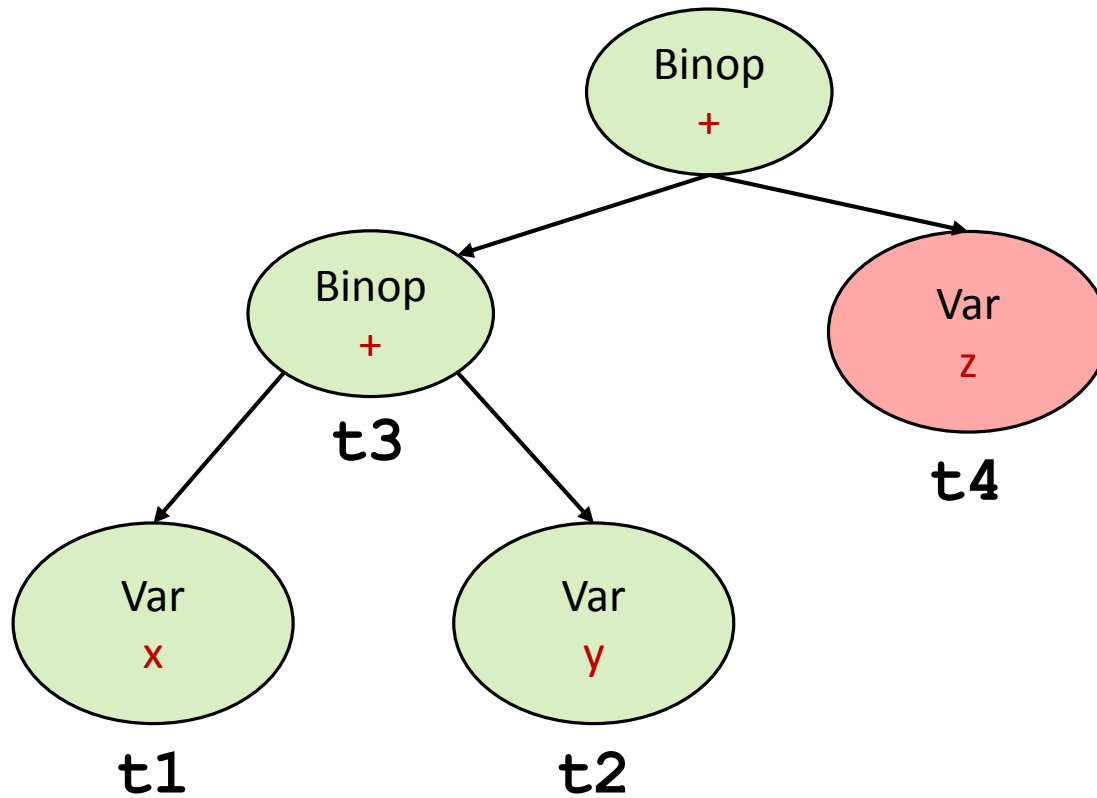
**t1** = **x**

**t2** = **y**

**t3** = add **t1**, **t2**

# Translating Expressions

$x + y + z$ :



**t1 = x**

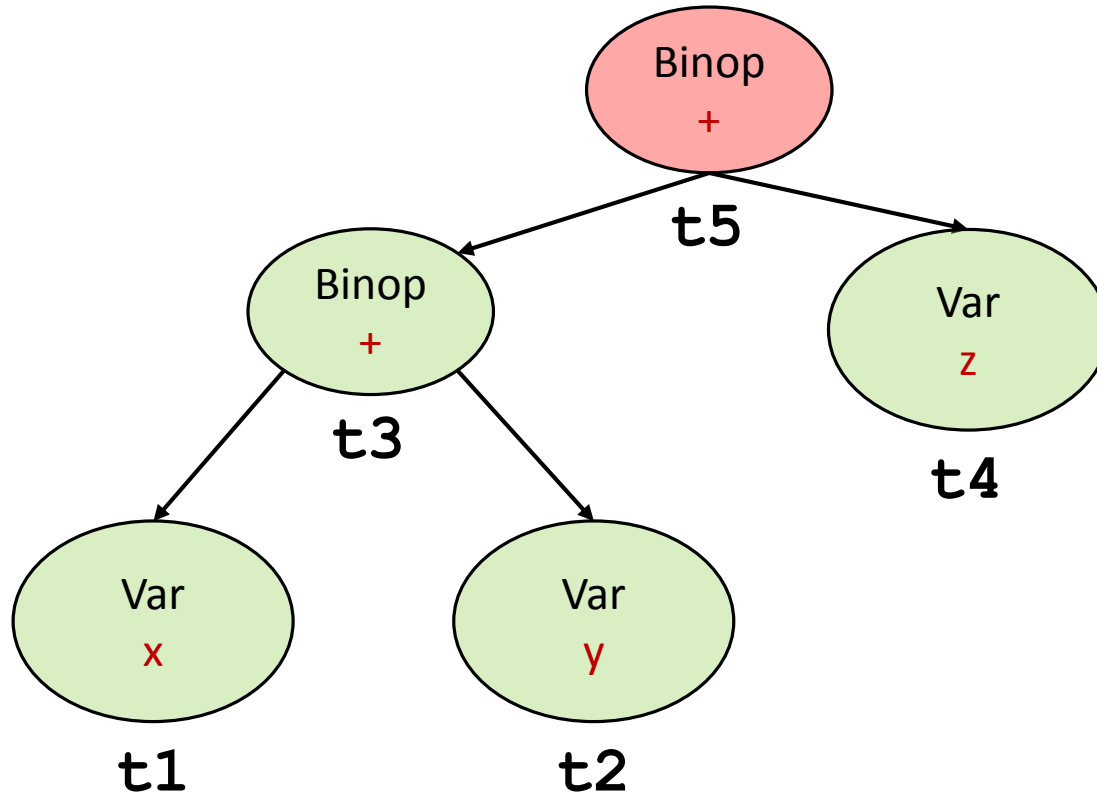
**t2 = y**

**t3 = add t1, t2**

**t4 = z**

# Translating Expressions

$x + y + z$ :



t1 = x

t2 = y

t3 = add t1, t2

t4 = z

t5 = add t3, t4

# Translating Expressions

$e_1 == e_2$ :

$T_c(e_1) \{ \dots$   
t1 = ...

$T_c(e_2) \{ \dots$   
t2 = ...

unique register → t3 = 1  
beq t1, t2, end\_label  
t3 = 0  
end\_label:  
unique label

# Translating Expressions

`a == b + 1:`

```
t1 = a
t2 = b
t3 = 1
t4 = add t2, t3
t5 = 1
beq t1, t4, end_label
t5 = 0
end_label:
```



# Translating Expressions

$e_1$  and  $e_2$ :

$T_c(e_1)$  {  $\dots$   
t1 =  $\dots$   
t3 = 0  
 $T_r(e_1)$  beq t1, 0, end\_label

$T_c(e_2)$  {  $\dots$   
t2 =  $\dots$   
t3 = and t1, t2  
 $T_r(e_2)$  end\_label:

# Translating Expressions

$e_1$  or  $e_2$ :

$T_c(e_1)$  {  $\dots$   
           $t1 = \dots$   
           $t3 = 1$   
 $T_r(e_1)$      $\text{beq } t1, 1, \text{end\_label}$

$T_c(e_2)$  {  $\dots$   
           $t2 = \dots$   
           $t3 = \text{or } t1, t2$   
 $T_r(e_2)$      $\text{end\_label:}$

# Translating Statements

$f(e_1, e_2, \dots)$ : (as rvalue)

$T_c(e_1) \{ \begin{array}{l} \dots \\ t1 = \dots \end{array}$

$T_c(e_2) \{ \begin{array}{l} \dots \\ t2 = \dots \end{array}$

$\begin{array}{l} \dots \\ t3 = \text{call } f, t1, t2, \dots \end{array}$

# Translating Statements

`func(2, x + 1):`

`t1 = 2`

`t2 = x`

`t3 = 1`

`t4 = add t2, t3`

`t5 = call func, t1, t4`

# Translating Expressions

*new type*[*e*]:

$T_c(e)$   $\{ \overset{\dots}{t1} = \dots$   
 $T_r(e_1)$   $\swarrow$   
 $t2 = \text{new\_array } t1$

# Translating Expressions

`new int[k+1]:`

`t1 = k`

`t2 = 1`

`t3 = add t1, t2`

`t4 = new_array t3`

# Translating Expressions

$e_1[e_2]$ :

$T_c(e_1) \{ \overset{\cdot \cdot \cdot}{t1} = \dots$

$T_r(e_1) \rightarrow$

$T_c(e_2) \{ \overset{\cdot \cdot \cdot}{t2} = \dots$

$T_r(e_2) \rightarrow t3 = \text{array\_access } t1, t2$

# Translating Expressions

`x[z+1]:`

`t1 = x`

`t2 = z`

`t3 = 1`

`t4 = add t2, t3`

`t5 = array_access t1, t4`



# Translating Expressions

*new type:*

```
t1 = new_class type
```

# Translating Expressions

new Point:

```
t1 = new_class Point
```

# Translating Expressions

$e.f$  :

$$\begin{array}{l} T_c(e) \quad \{ \begin{array}{l} \dots \\ t1 = \dots \end{array} \\ T_r(e) \quad \swarrow \quad t2 = \text{field\_access } t1, f \end{array}$$

# Translating Expressions

`x[3].foo`:

```
t1 = x
t2 = 3
t3 = array_access t1, t2
t4 = field_access t3, foo
```

# Translating Basic Block

$s_1; s_2; \dots:$

$T_c(s_1)$

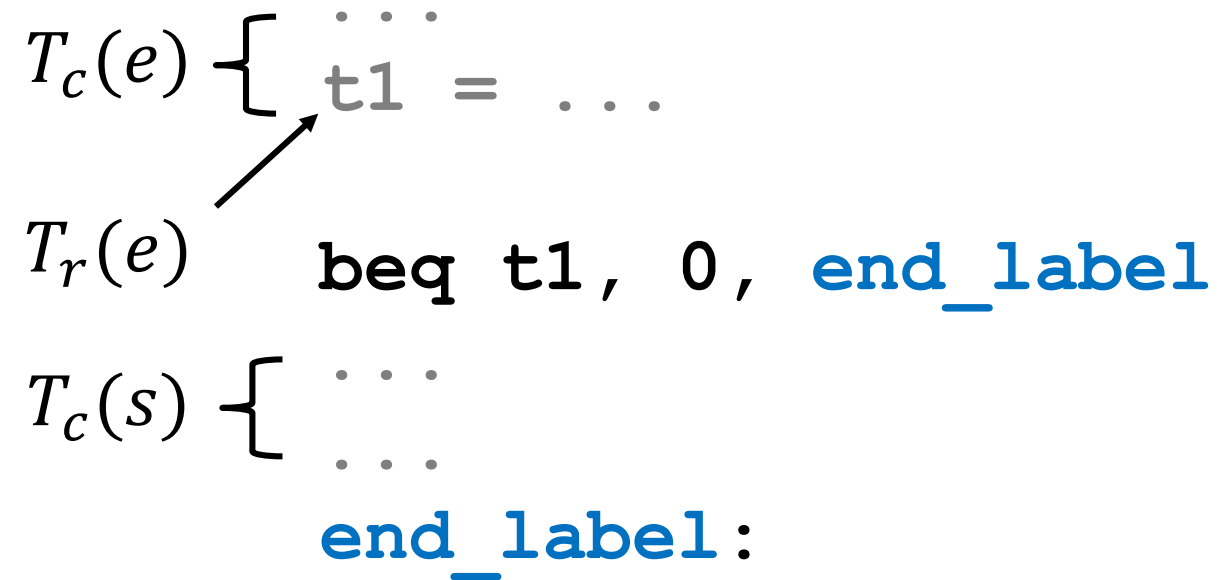
$T_c(s_2)$

...

# Translating Statements

*if* (*e*) {*s*}:

$T_c(e) \{ \begin{array}{l} \dots \\ t1 = \dots \end{array}$   
 $T_r(e) \quad \text{beq } t1, 0, \text{end\_label}$   
 $T_c(s) \{ \begin{array}{l} \dots \\ \dots \end{array}$   
 $\text{end\_label:}$



# Translating Statements

if (x \* y) { z = 0; }:

```
t1 = x
t2 = y
t3 = mul t1, t2
beq t3, 0, end_label
t4 = 0
z = t4
end_label:
```

# Translating Statements

*if* (*e*) {*s*<sub>1</sub>} *else* {*s*<sub>2</sub>}:

$T_c(e)$  {  
    ...  
    t1 = ...  
    beq t1, 0, false\_label  
}

$T_r(e_1)$  ↗

$T_c(s_1)$  {  
    ...  
    ...  
    br end\_label  
    false\_label:  
}

$T_c(s_2)$  {  
    ...  
    ...  
    end\_label:  
}



# Translating Statements

if (w) { z = 0; } else { z = 100; }:

```
t1 = w
beq t1, 0, false_label
t2 = 0
z = t2
br end_label
false_label:
t3 = 100
z = t3
end_label:
```

# Translating Statements

*while* (*e*) {*s*} :

$T_c(e)$  {  
     $\dots$   
     $t1 = \dots$   
     $\text{beq } t1, 0, \text{end\_label}$   
 $T_r(e)$  ↗  
 $T_c(s)$  {  
     $\dots$   
     $\dots$   
     $\text{br cond\_label}$   
     $\text{end\_label:}$

# Translating Statements

while (z / x) { }:

```
cond_label:  
t1 = z  
t2 = x  
t3 = div t1, t2  
beq t3, 0, end_label  
br cond_label  
end_label:
```

# Translating Statements

$f(e_1, e_2, \dots) :$

$T_c(e_1) \{ \begin{array}{l} \dots \\ \text{t1} = \dots \end{array}$

$T_c(e_2) \{ \begin{array}{l} \dots \\ \text{t2} = \dots \end{array}$

$\dots$   
**call**  $f, \text{t1}, \text{t2}, \dots$

# Translating Statements

`func(2, x + 1):`

`t1 = 2`

`t2 = x`

`t3 = 1`

`t4 = add t2, t3`

`call func, t1, t4`

# Translating Statements

*return e:*

$$T_c(e) \{ \begin{array}{l} \dots \\ t1 = \dots \\ \text{return } t1 \end{array}$$

# Translating Statements

return w \* 3:

```
t1 = w
t2 = 3
t3 = mul t1, t2
return t3
```

# Translating Statements

$e_1[e_2] = e_3$ :

$T_c(e_1) \{ \dots$   
t1

$T_c(e_2) \{ \dots$   
t2 = ...

$T_c(e_3) \{ \dots$   
t3 = ...

**array\_set** t1, t2, t3



# Translating Statements

`arr[0] = x+1:`

```
t1 = arr
```

```
t2 = 0
```

```
t3 = x
```

```
t4 = 1
```

```
t5 = add t3, t4
```

```
array_set t1, t2, t5
```

# Translating Statements

$o.f = e$ :

$T_c(o) \{ \dots$   
t1

$T_c(e) \{ \dots$   
t2 = ...  
**field\_set** t1, f, t2

# Translating Statements

obj.flag = 7:

```
t1 = obj  
t2 = 7  
field_set t1, flag, t2
```

# Translating Statements

$o.f(e_1, e_2, \dots) :$

$T_c(o) \{ \dots$   
t1

$T_c(e_1) \{ \dots$   
t2 = ...

$T_c(e_2) \{ \dots$   
t3 = ...

...  
**virtual\_call** t1 f t2, t3, ...

# Translating Statements

`obj.bar(2, x + 1):`

```
t1 = obj
```

```
t2 = 2
```

```
t3 = x
```

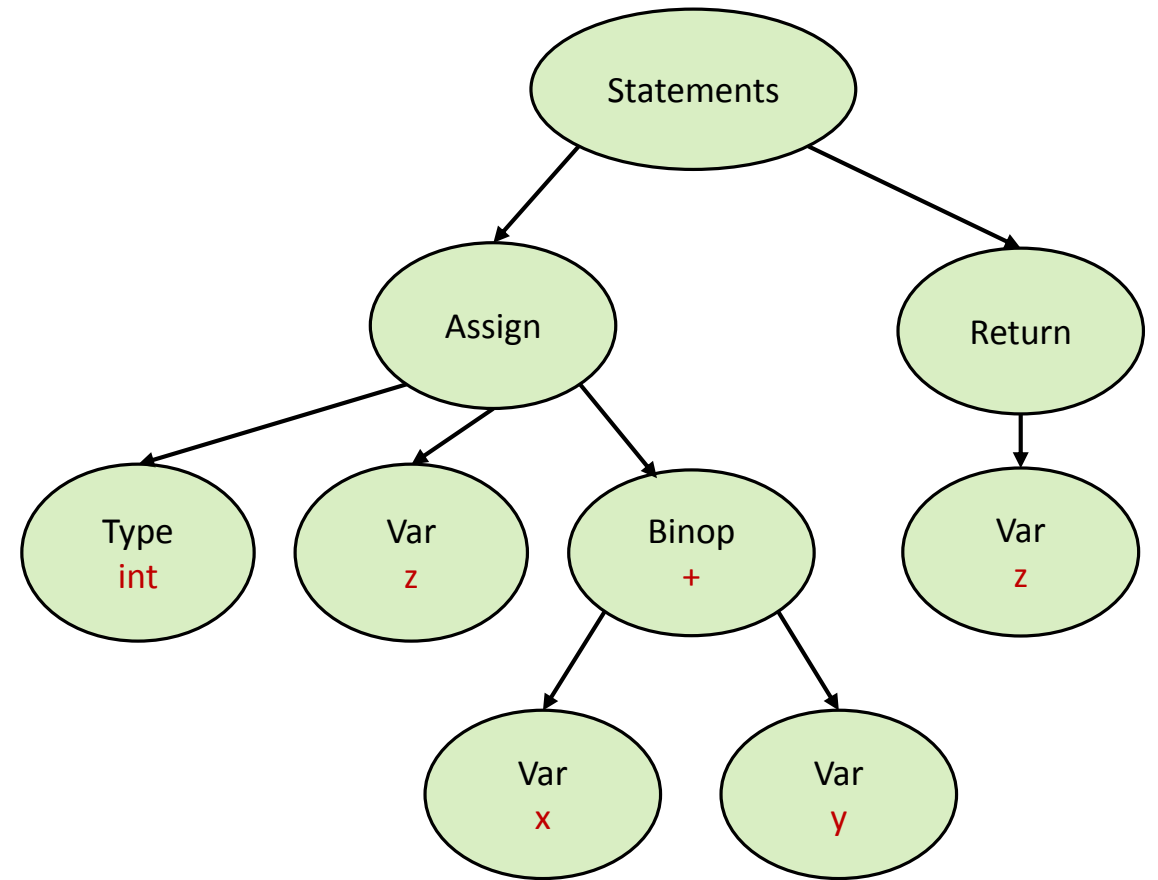
```
t4 = 1
```

```
t5 = add t3, t4
```

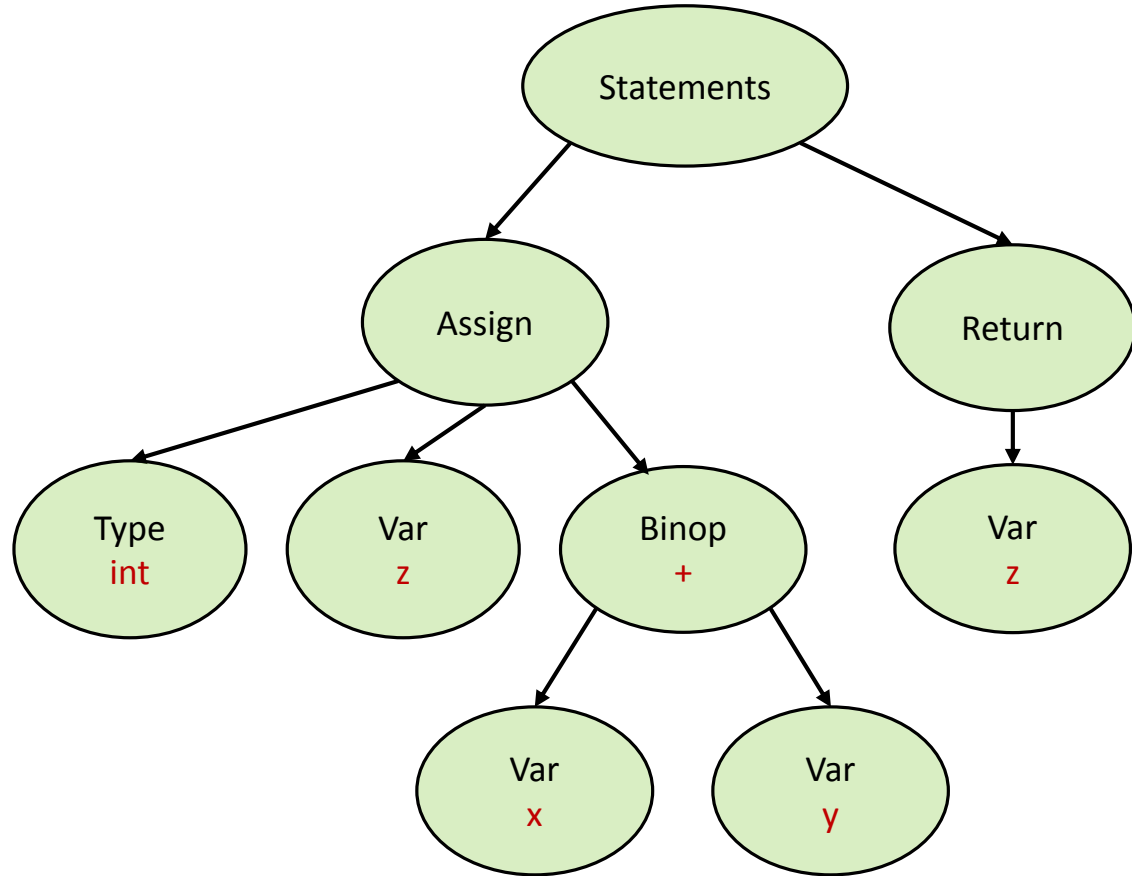
```
virtual_call t1, bar, t2, t5
```

# Example

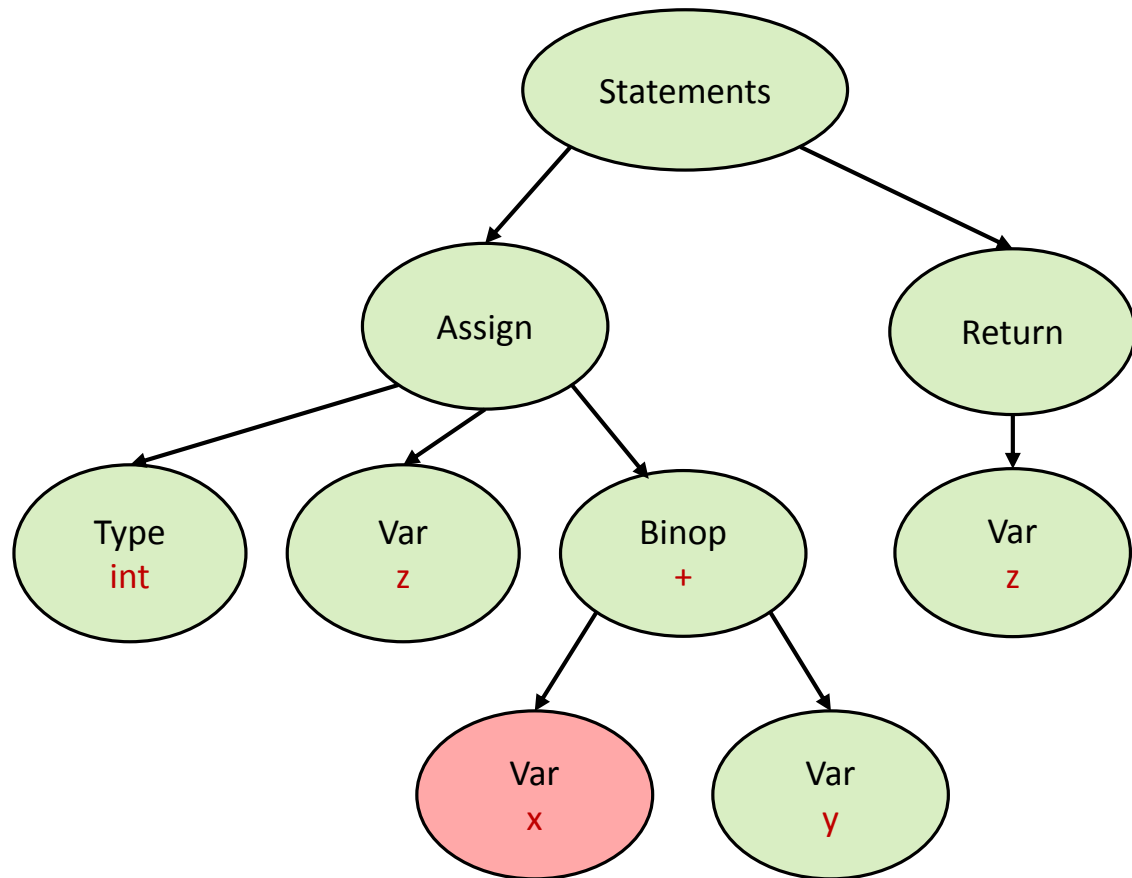
```
int f(int x, int y) {  
    int z = x + y;  
    return z;  
}
```



# Example



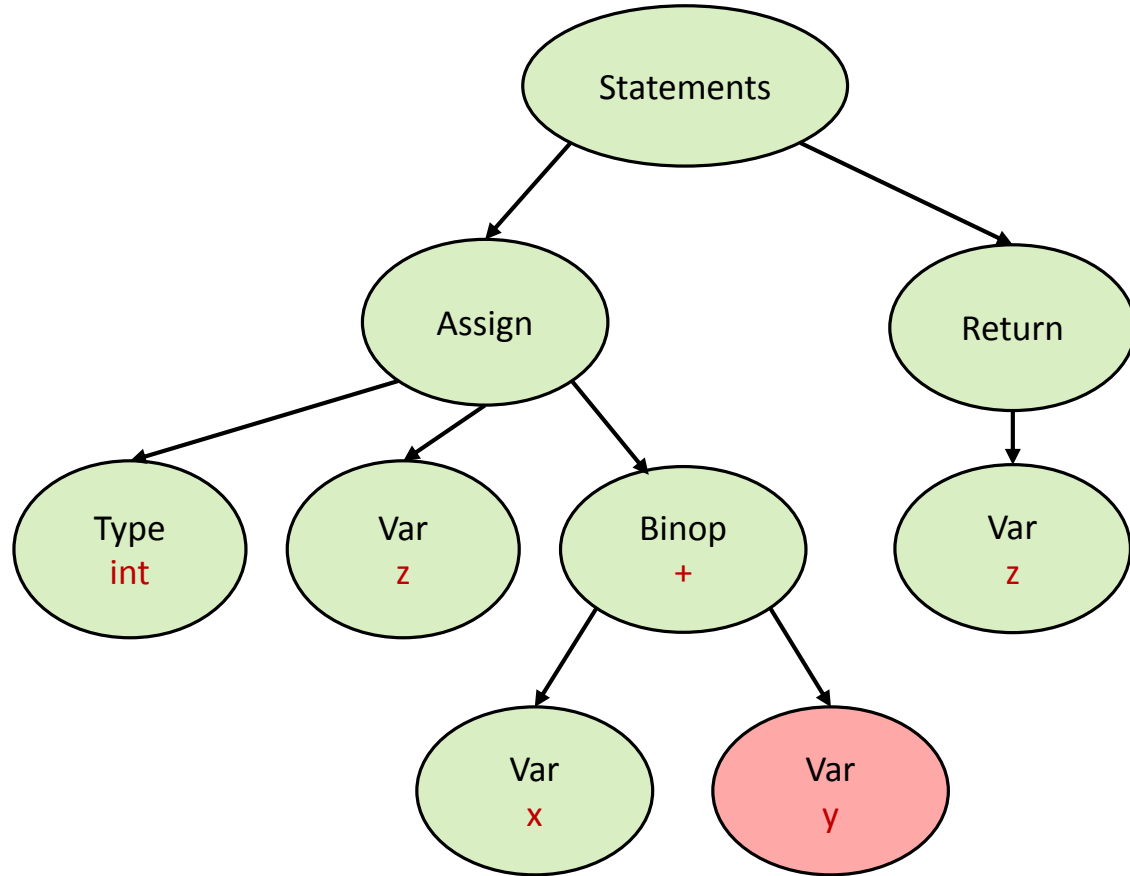
# Example



`t1 = x`



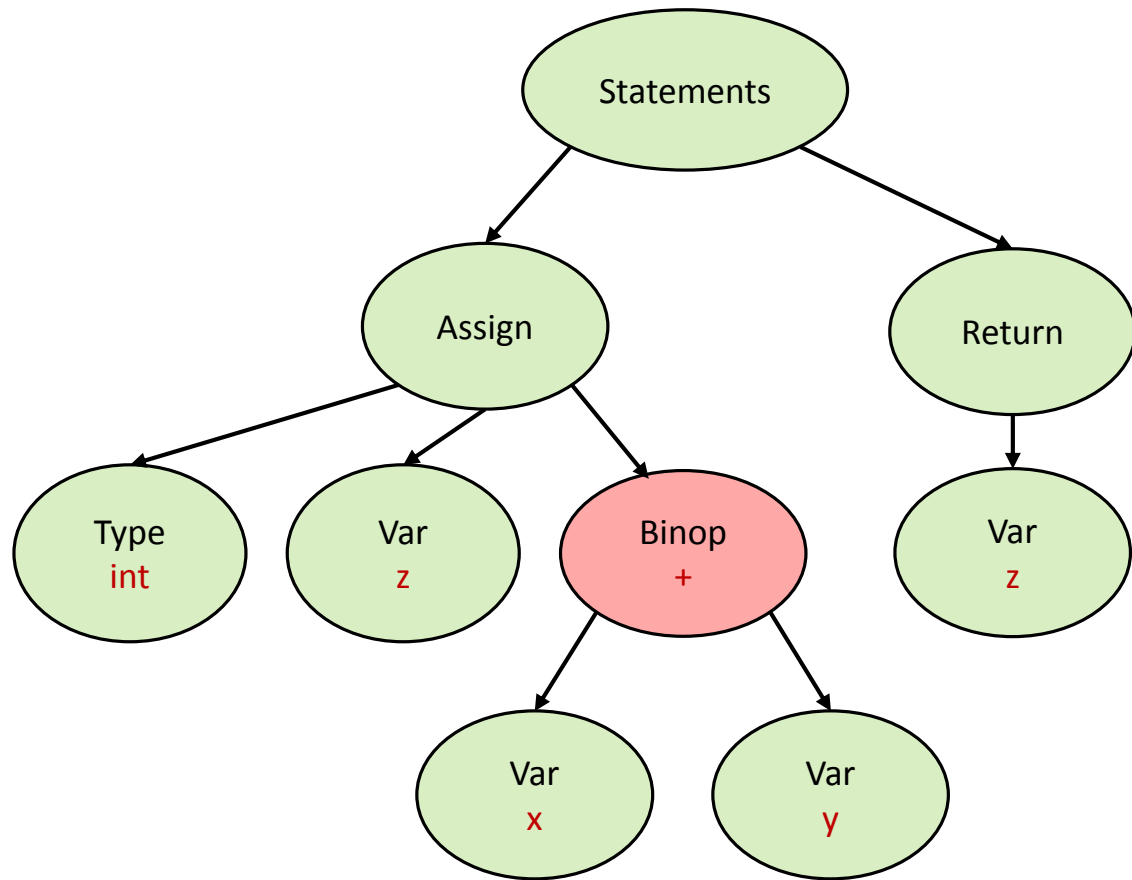
# Example



`t1 = x`

`t2 = y`

# Example

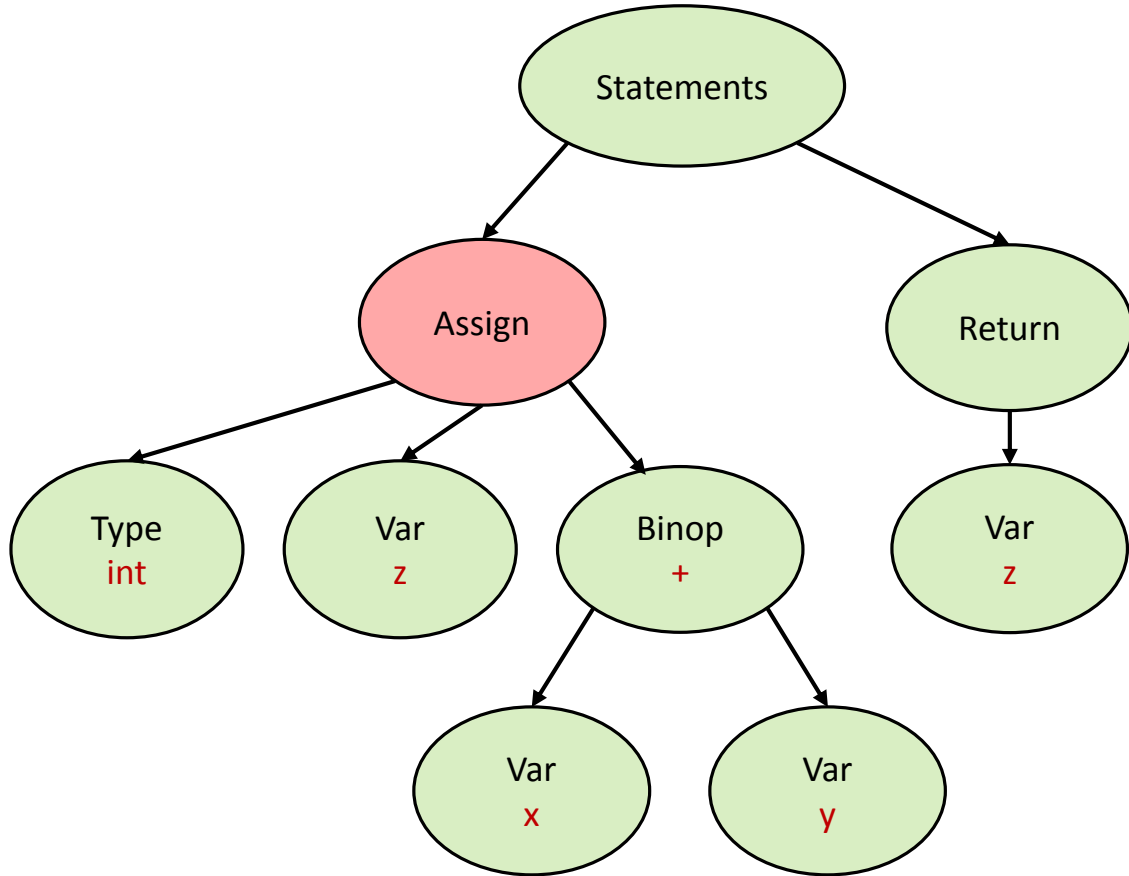


`t1 = x`

`t2 = y`

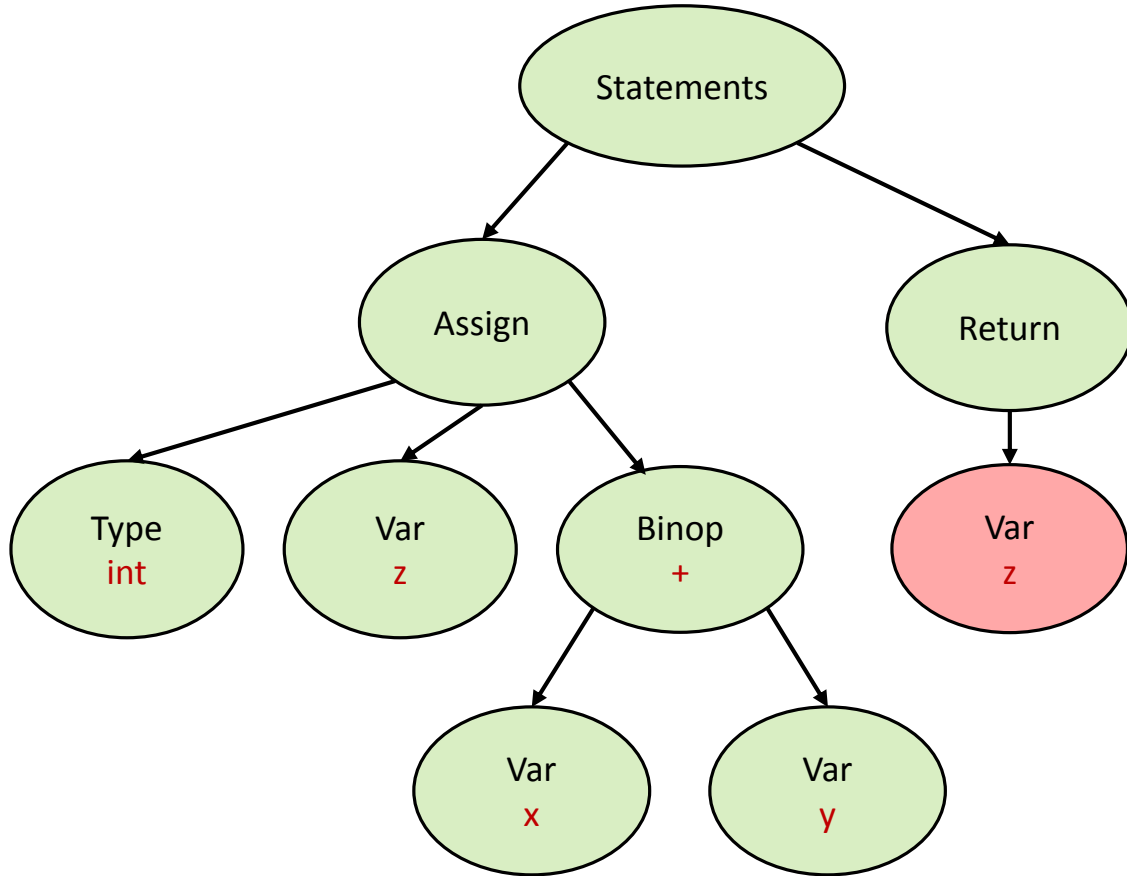
`t3 = add t1, t2`

# Example



```
t1 = x
t2 = y
t3 = add t1, t2
z = t3
```

# Example



t1 = x

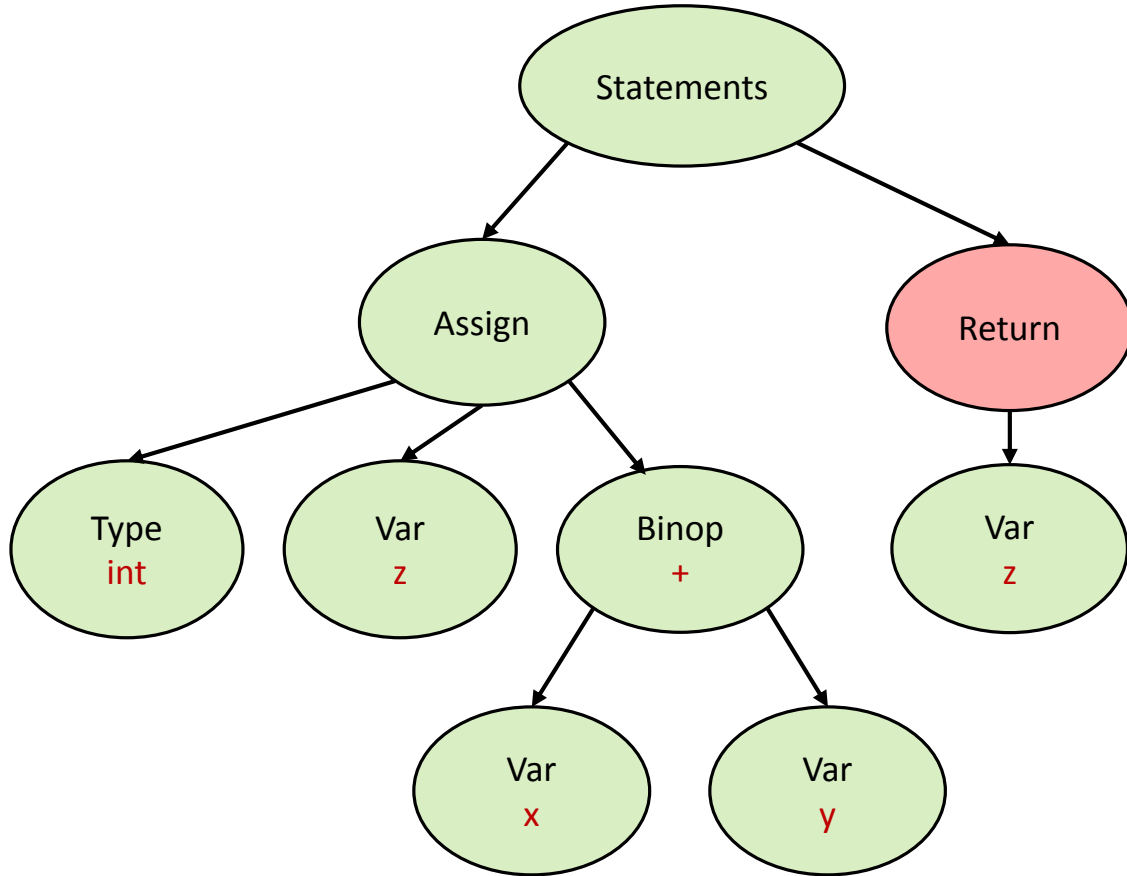
t2 = y

t3 = add t1, t2

z = t3

t4 = z

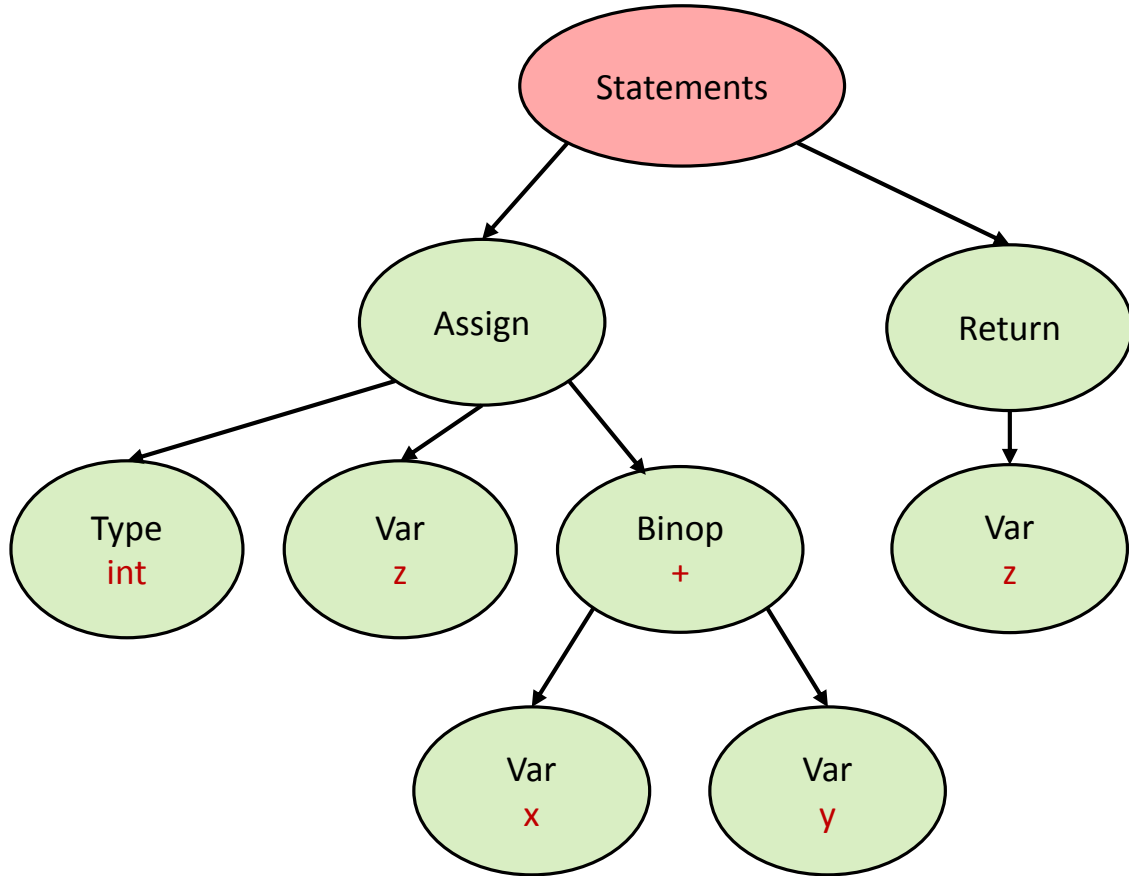
# Example



```
t1 = x
t2 = y
t3 = add t1, t2
z = t3
```

```
t4 = z
return t4
```

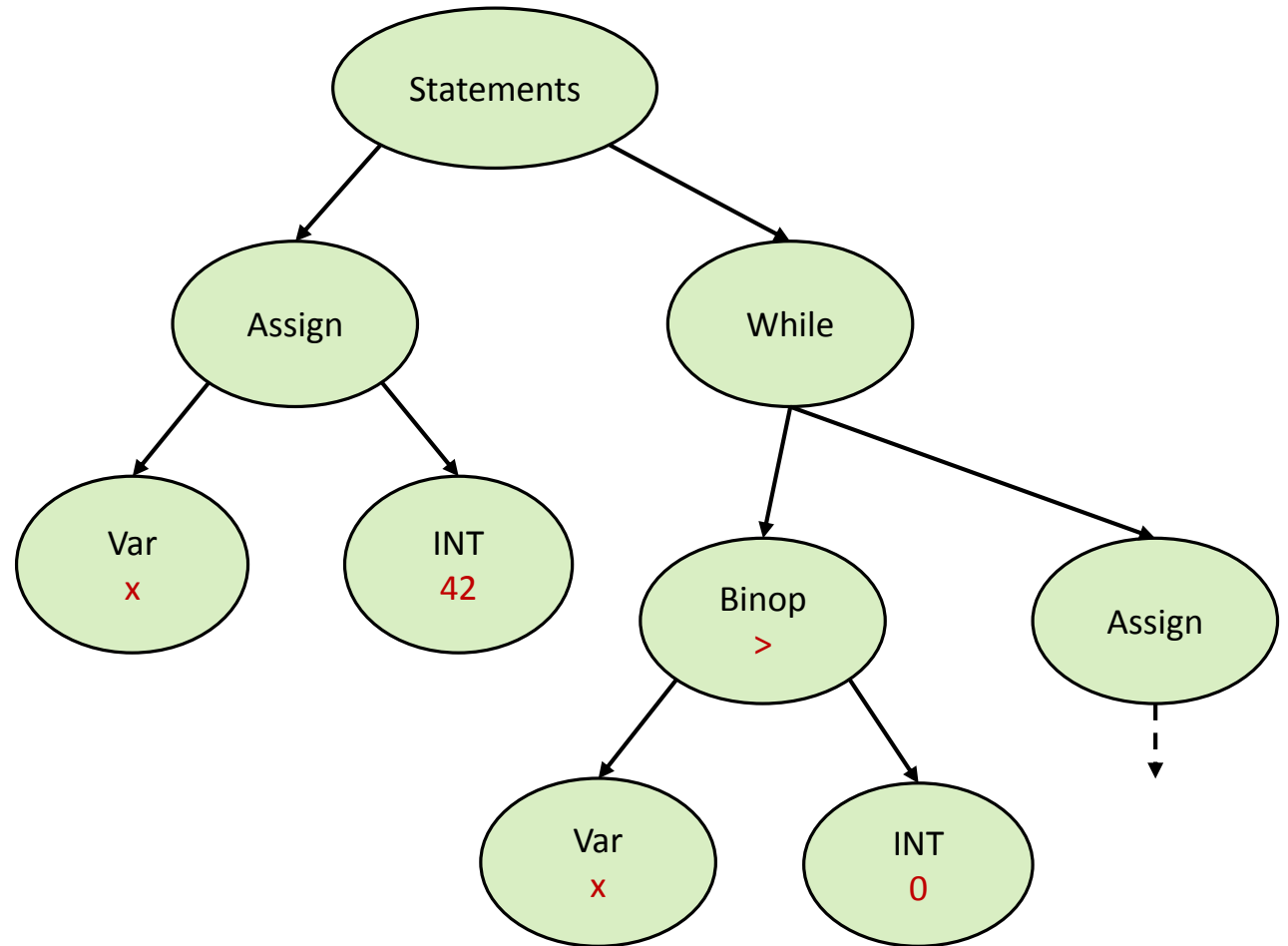
# Example



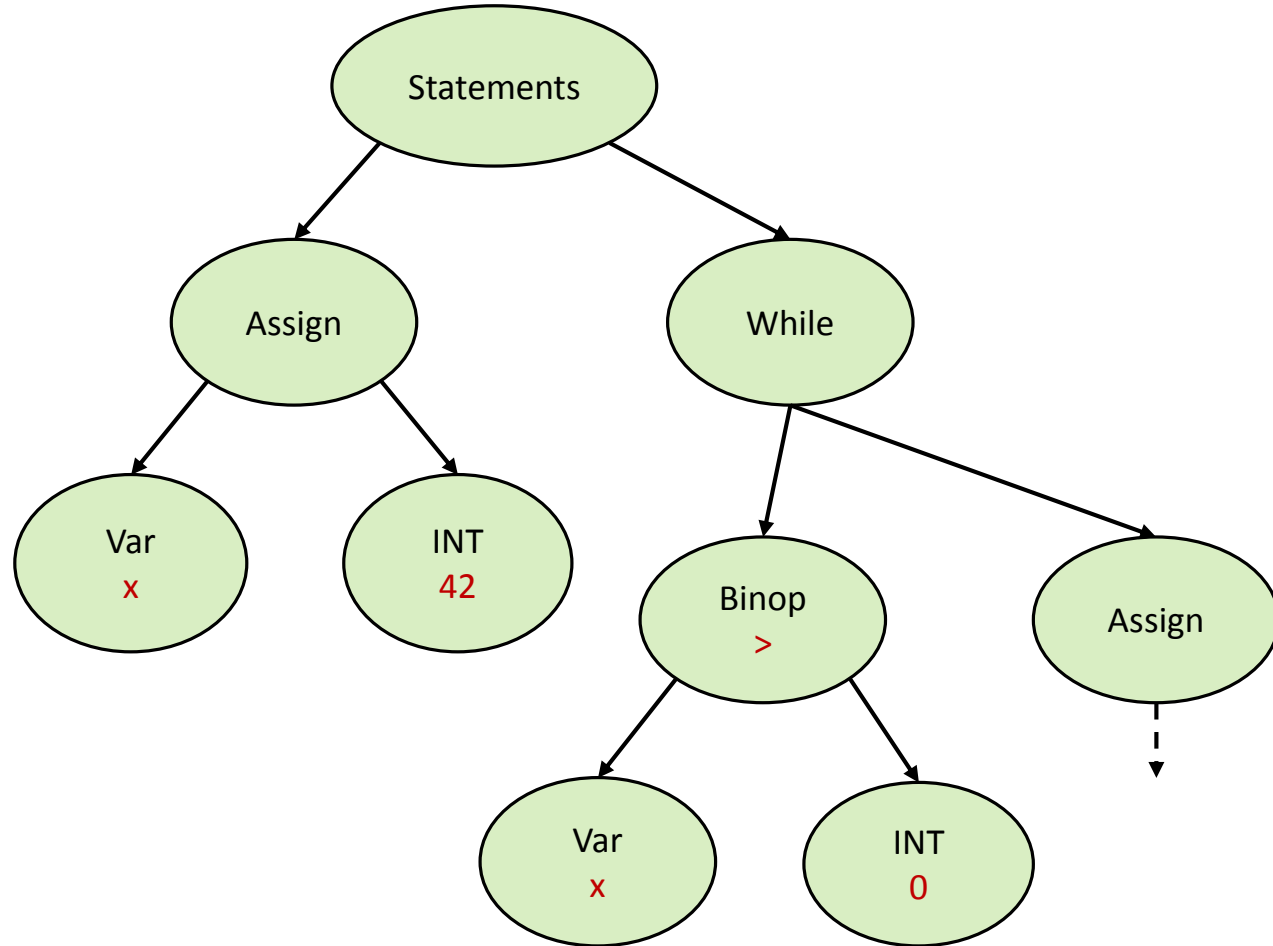
```
t1 = x
t2 = y
t3 = add t1, t2
z = t3
t4 = z
return t4
```

# Example

```
x = 42;  
while (x > 0) {  
    x = x - 1;  
}
```

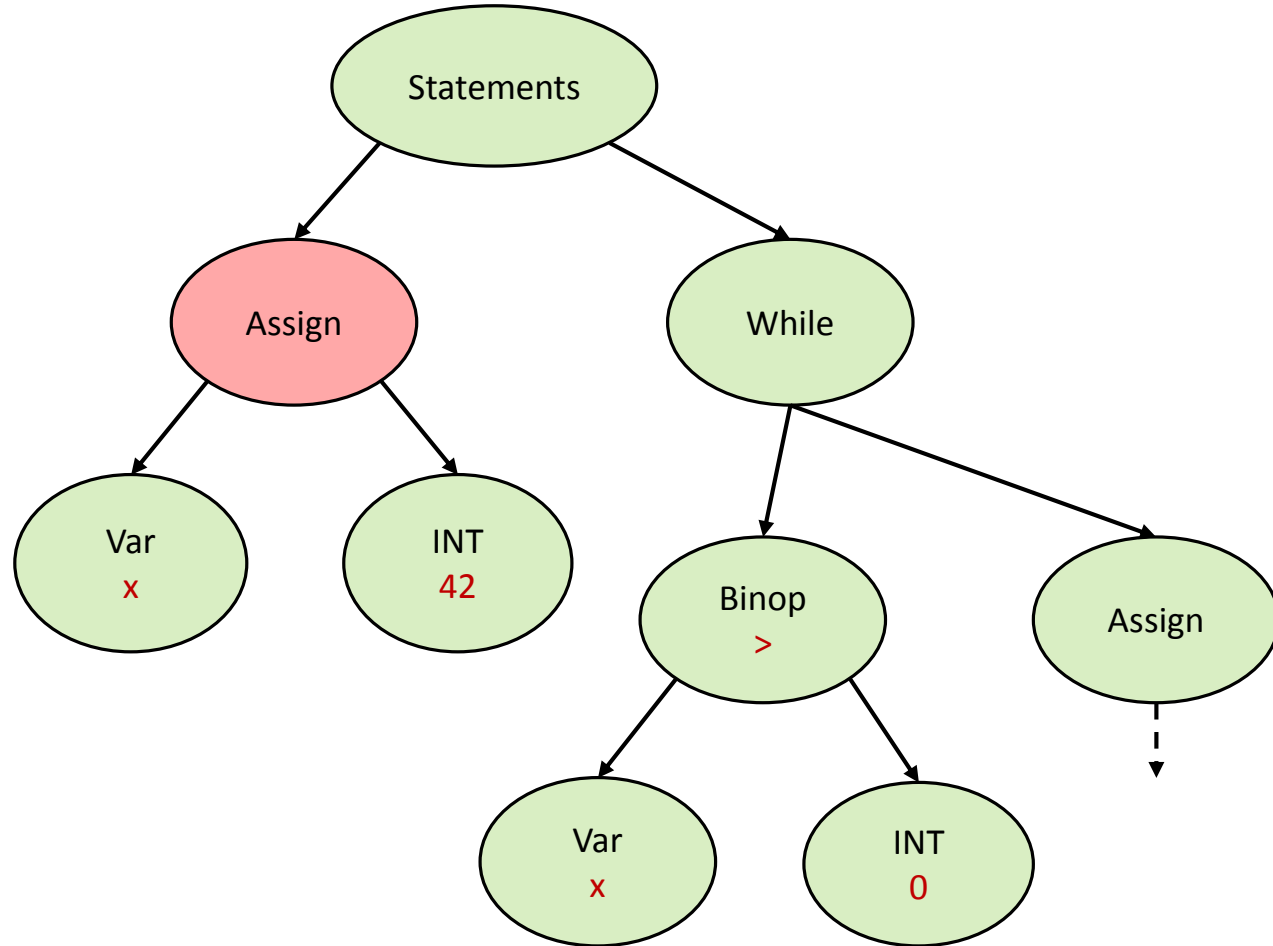


# Example





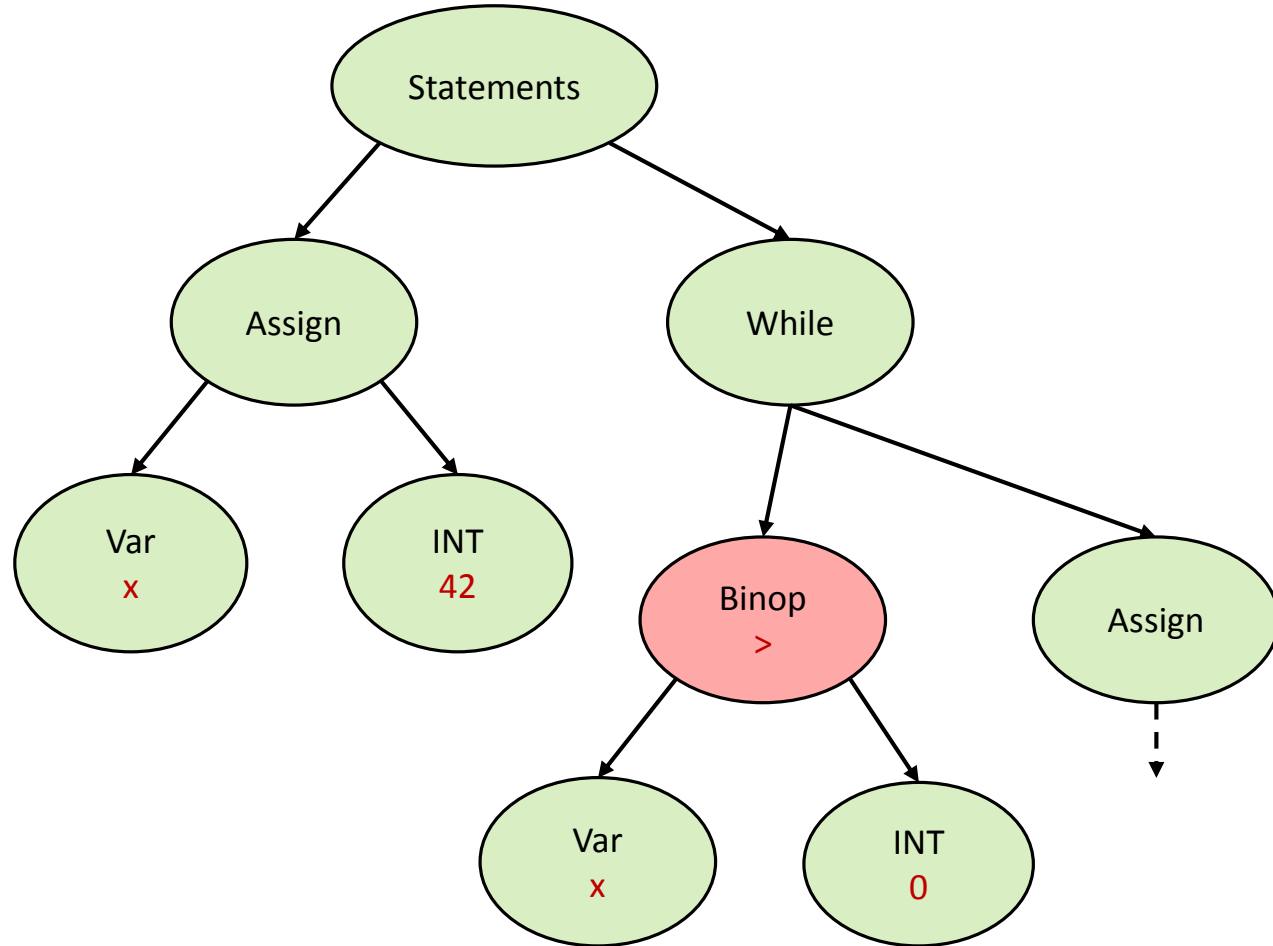
# Example



`t1 = 42`

`x = t1`

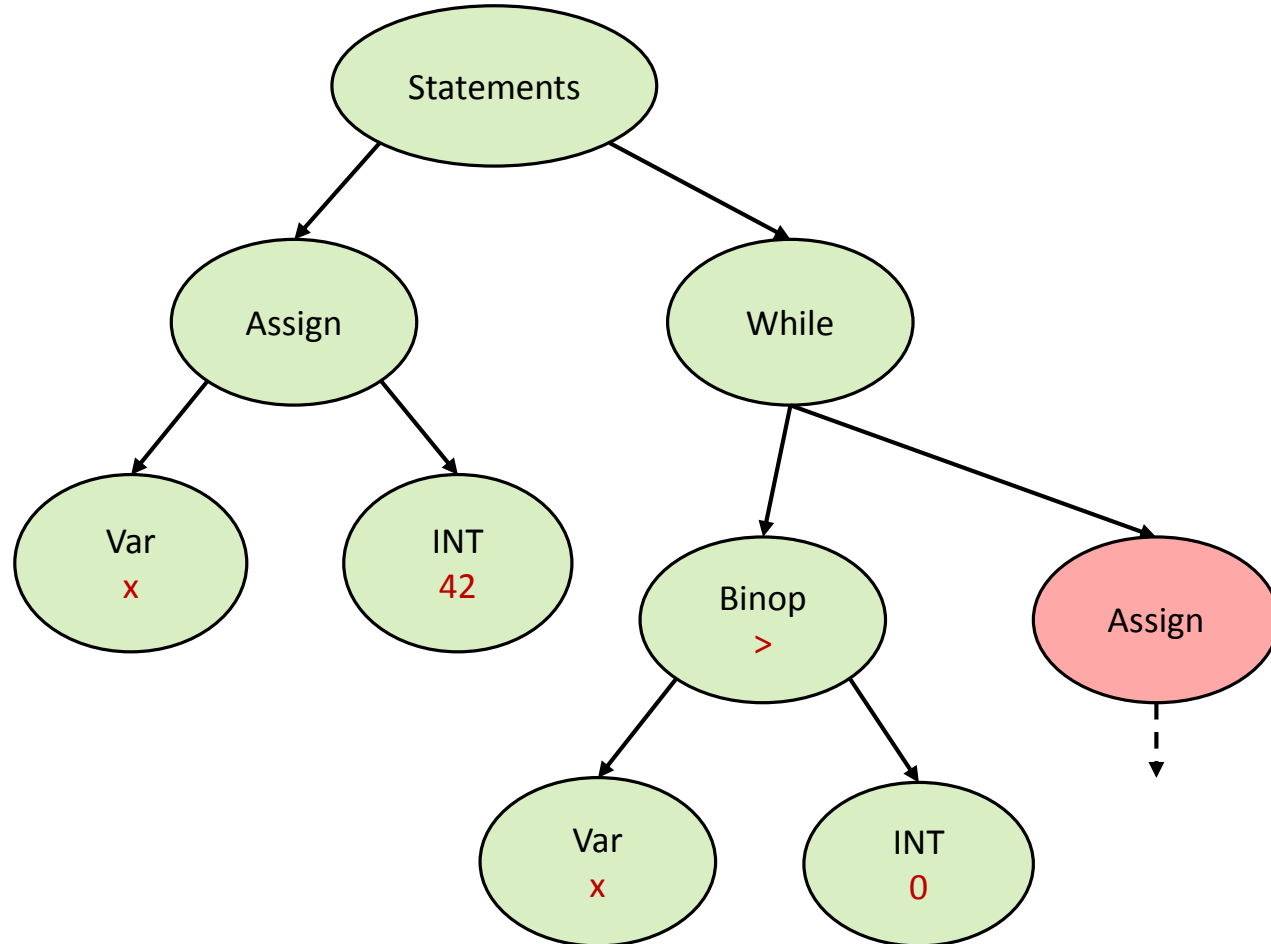
# Example



```
t1 = 42  
x = t1
```

```
t2 = x  
t3 = 0  
t4 = 1  
bgt t2, t3, cmp_label:  
t4 = 0  
cmp_label:
```

# Example

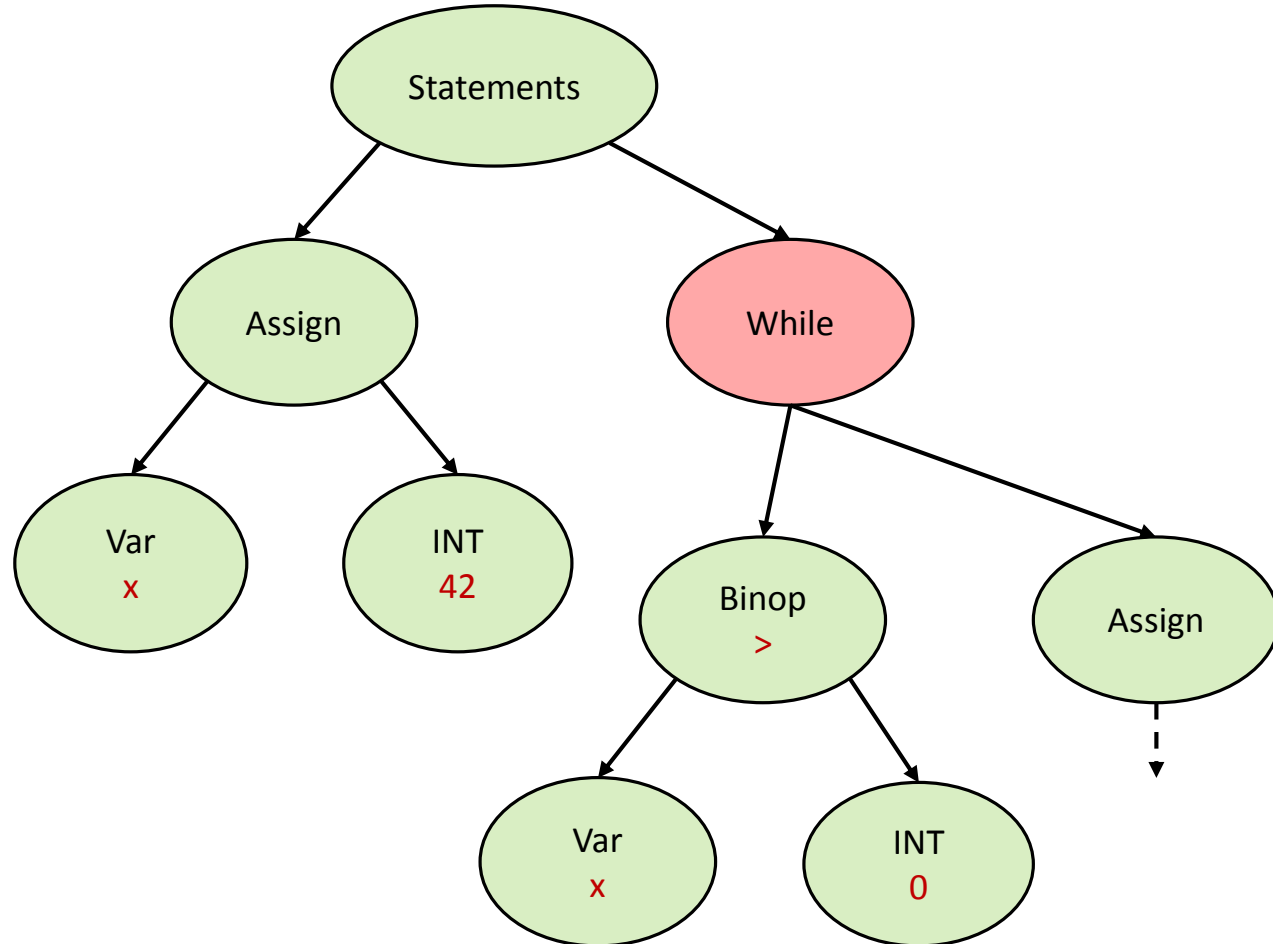


```
t1 = 42
x = t1
```

```
t2 = x
t3 = 0
t4 = 1
bgt t2, t3, cmp_label:
t4 = 0
cmp_label:
```

```
t5 = x
t6 = 1
t7 = sub t5, t6
x = t7
```

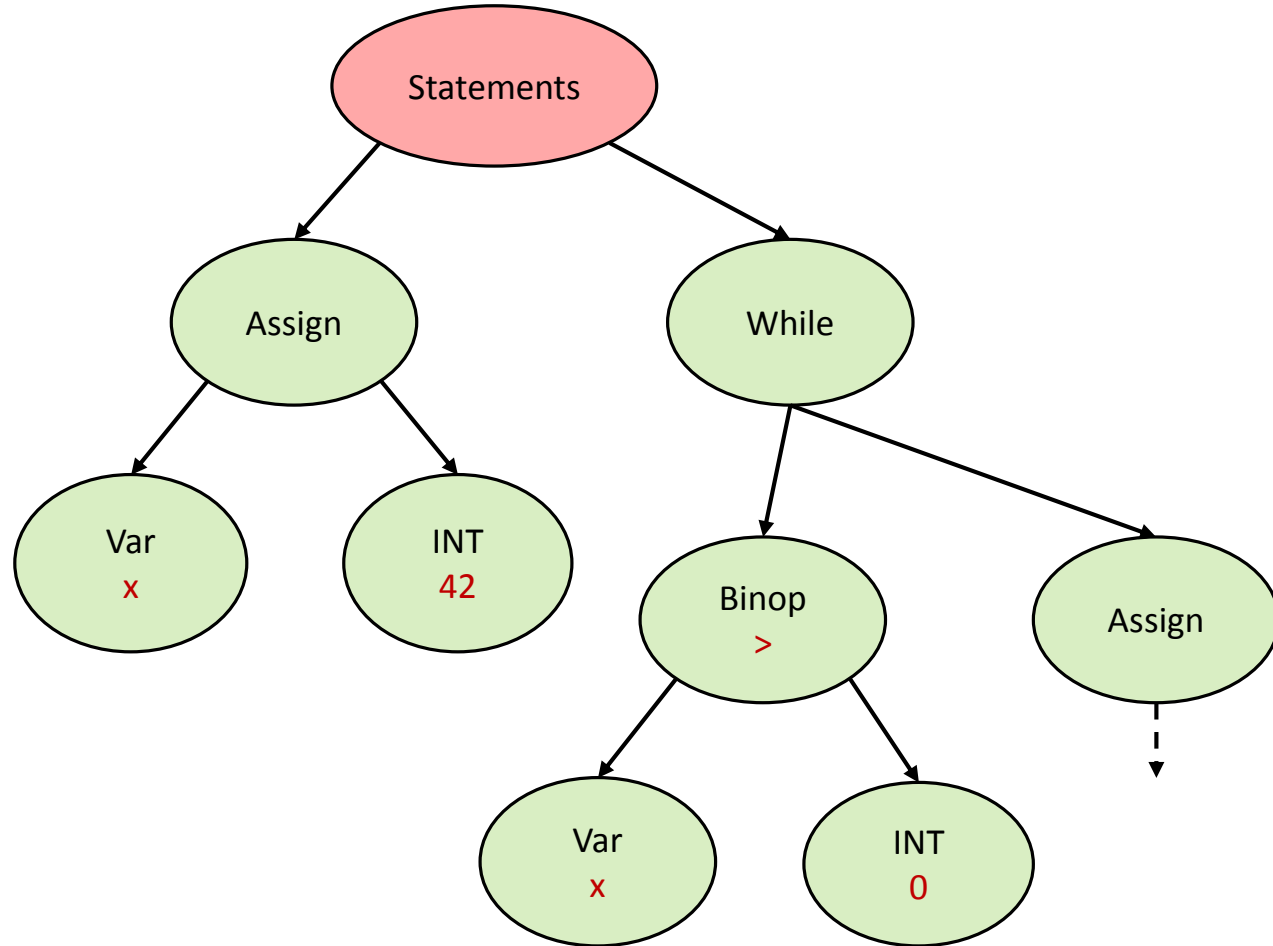
# Example



```
t1 = 42  
x = t1
```

```
cond_label:  
t2 = x  
t3 = 0  
t4 = 1  
bgt t2, t3, cmp_label:  
t4 = 0  
cmp_label:  
beq t4, 0, end_label  
t5 = x  
t6 = 1  
t7 = sub t5, t6  
x = t7  
br cond_label  
end_label:
```

# Example



```
t1 = 42
x = t1
cond_label:
t2 = x
t3 = 0
t4 = 1
bgt t2, t3, cmp_label:
t4 = 0
cmp_label:
beq t4, 0, end_label
t5 = x
t6 = 1
t7 = sub t5, t6
x = t7
br cond_label
end_label:
```