

Semantic Analysis

TEACHING ASSISTANT: DAVID TRABISH

Semantic Analysis

We need to check the following:

- Type checking
 - $1 + \text{"1"}$
- Scopes
 - Undefined variables
- Other
 - Division by zero
 - Const variables
 - Visibility semantics in classes (public, private, ...)

Symbol Table

- Maintain a stack of scopes
- Each scope maps identifiers to their **type information**
- Identifiers may be:
 - Variable names
 - Function names
 - Method names

Symbol Table

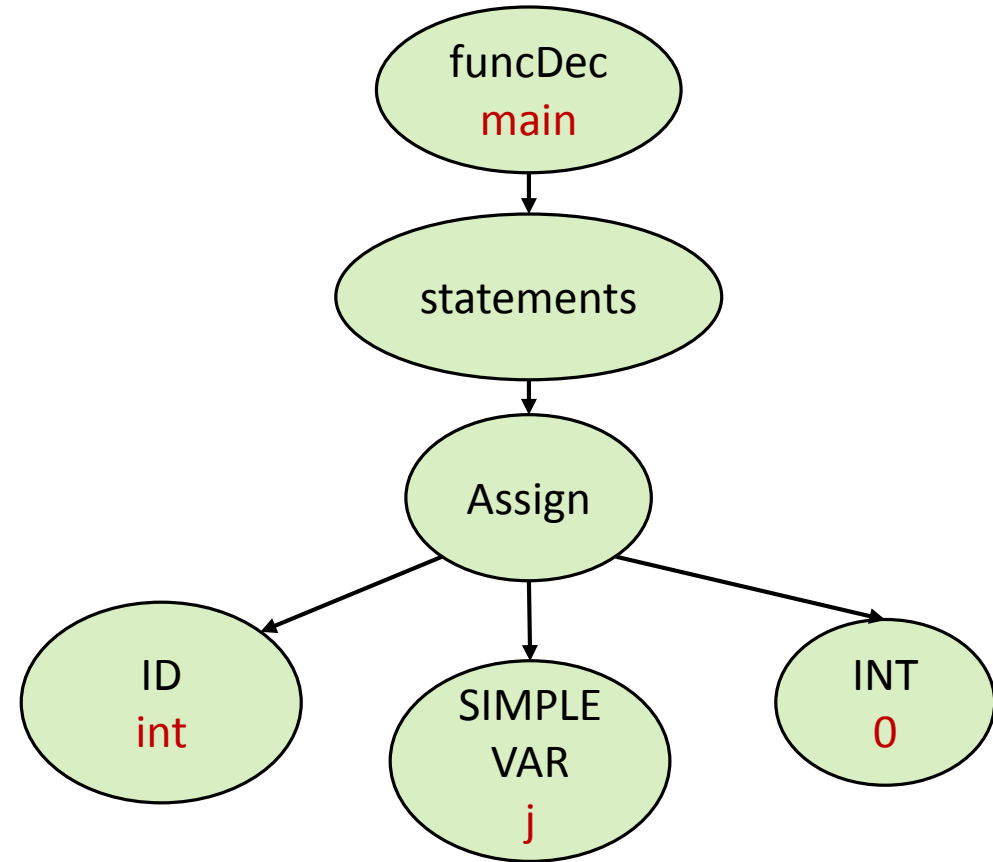
- When we reach a variable/function/... declaration
 - Update the map of the current scope (top of the stack)
- When we reach a new block, **push** a new scope
- When we leave a block, **pop** the top scope
- Begin with the global (initial scope)
 - Functions, global variables, ...

Symbol Table

- When we need to resolve an identifier
 - Scan the scopes (starting from the top)
 - Stop at the first matching scope
 - If no scope was found, we have an error...

Assignments

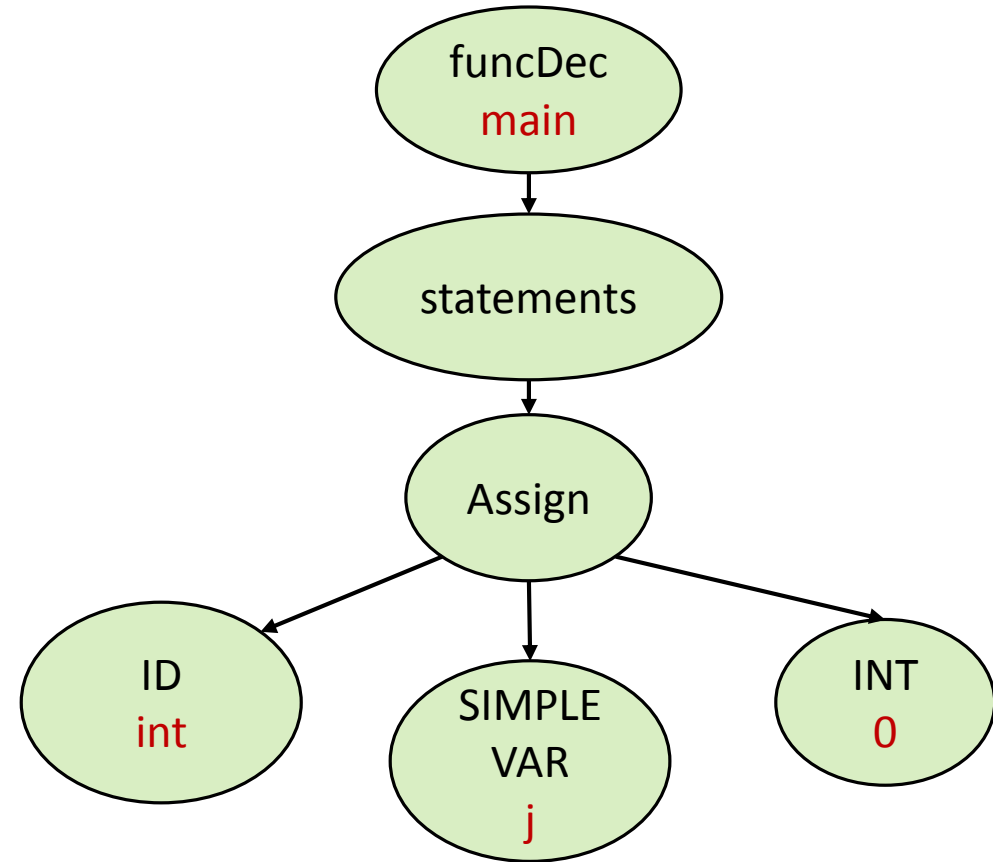
```
void main() {  
    int j = 0;  
}
```



Assignments

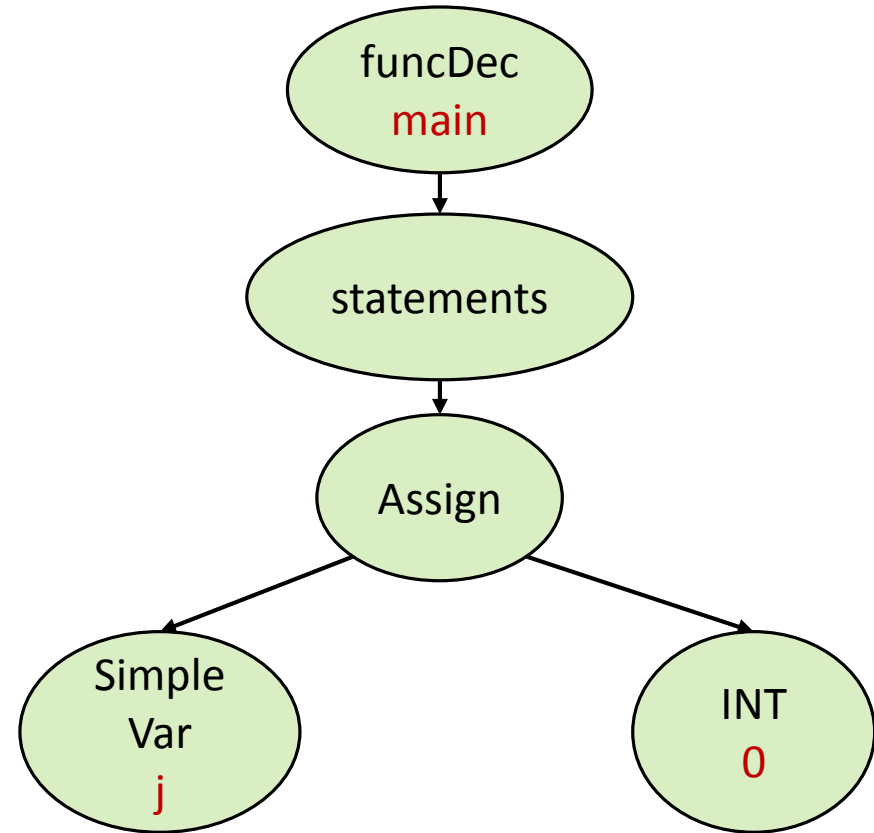
```
void main() {  
    int j = 0;  
}
```

Valid



Assignments

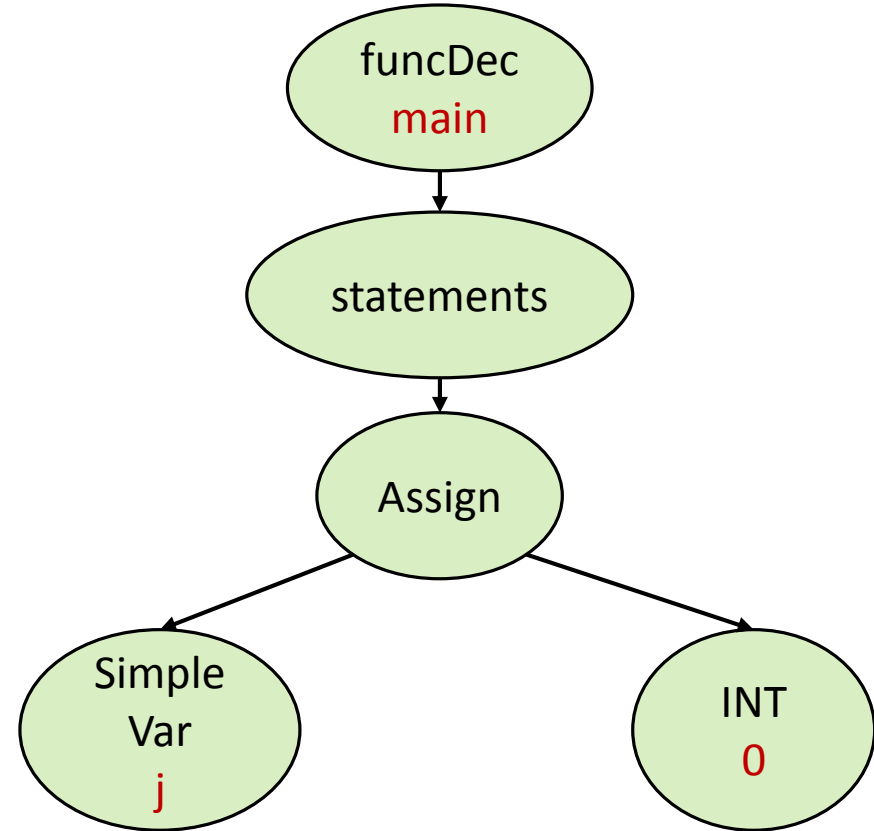
```
void main() {  
    j = 0;  
}
```



Assignments

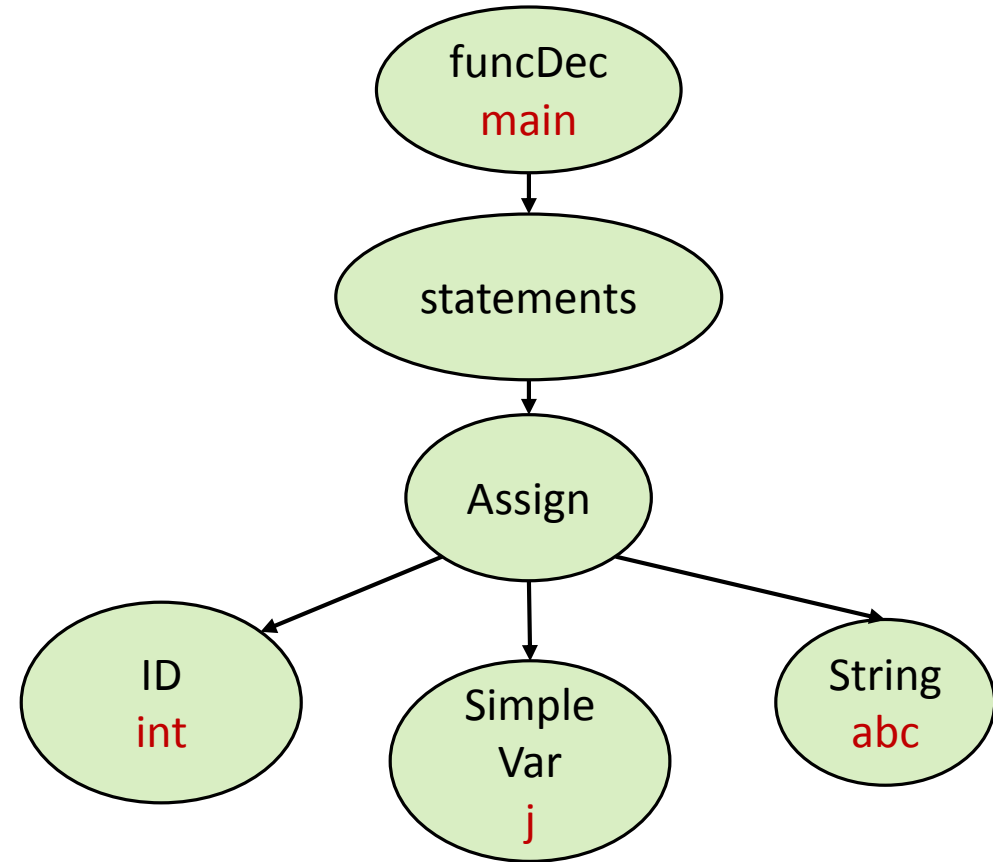
```
void main() {  
    j = 0;  
}
```

Invalid



Assignments

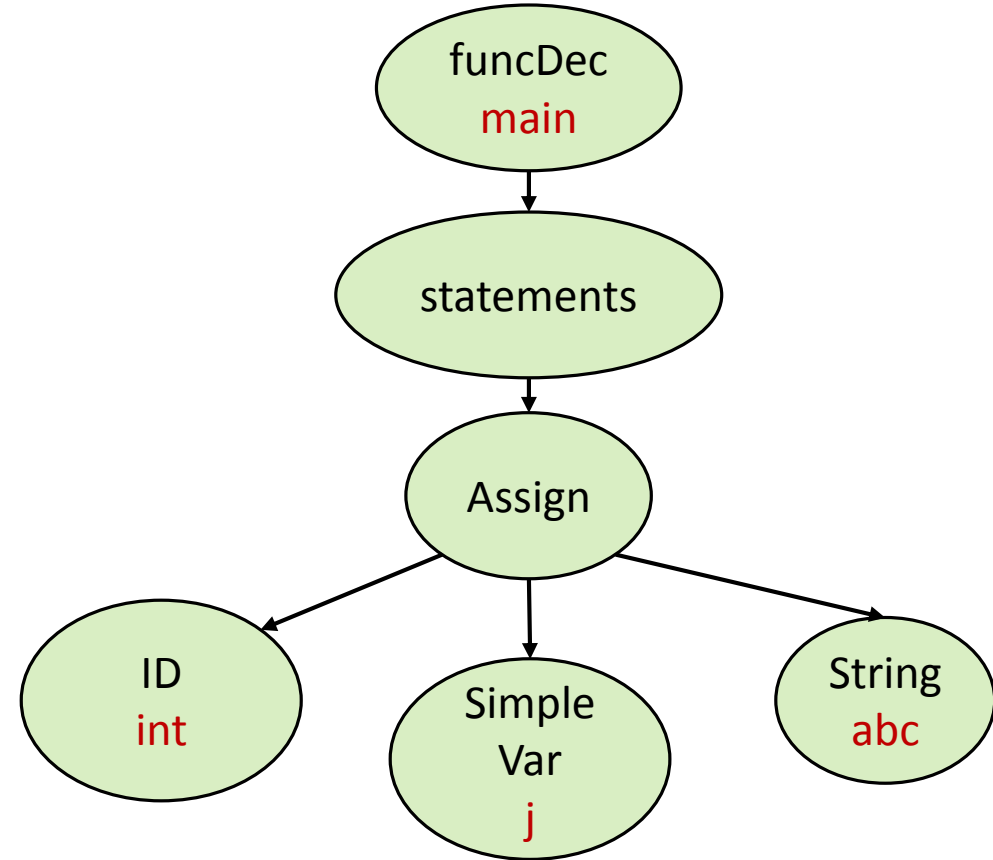
```
void main() {  
    int j = "abc";  
}
```



Assignments

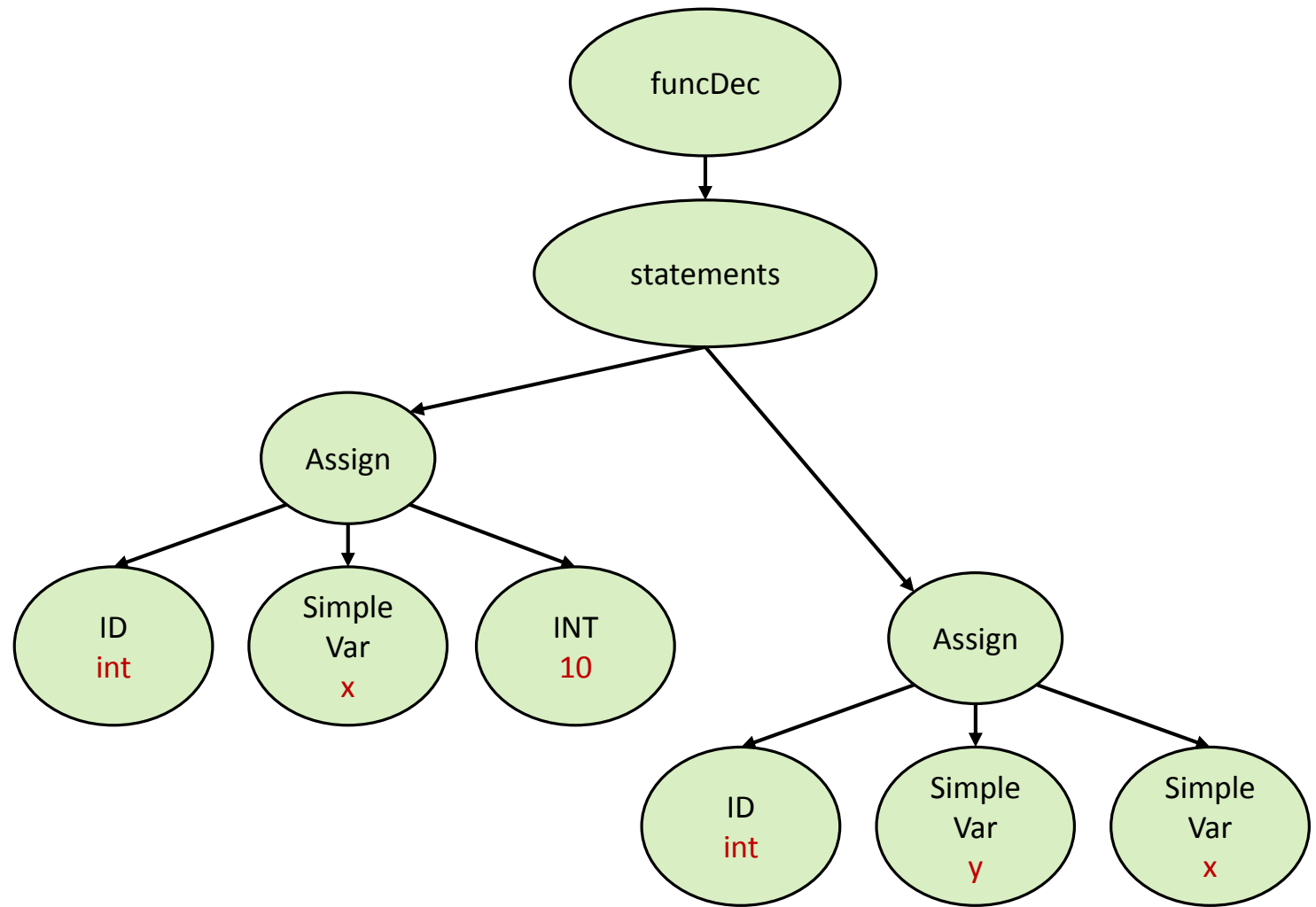
```
void main() {  
    int j = "abc";  
}
```

Invalid



Assignments

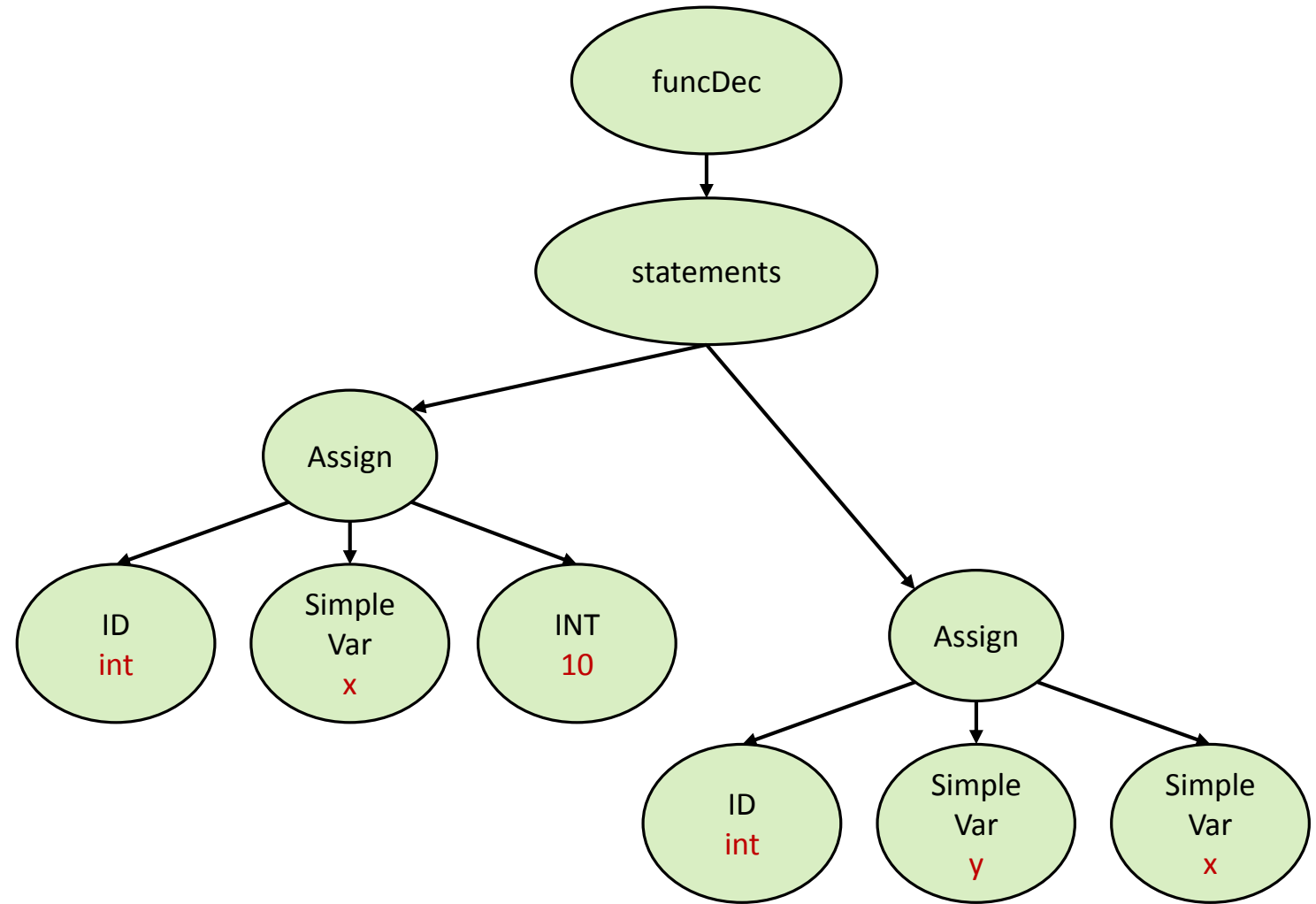
```
void main() {  
    int x = 10;  
    int y = x;  
}
```



Assignments

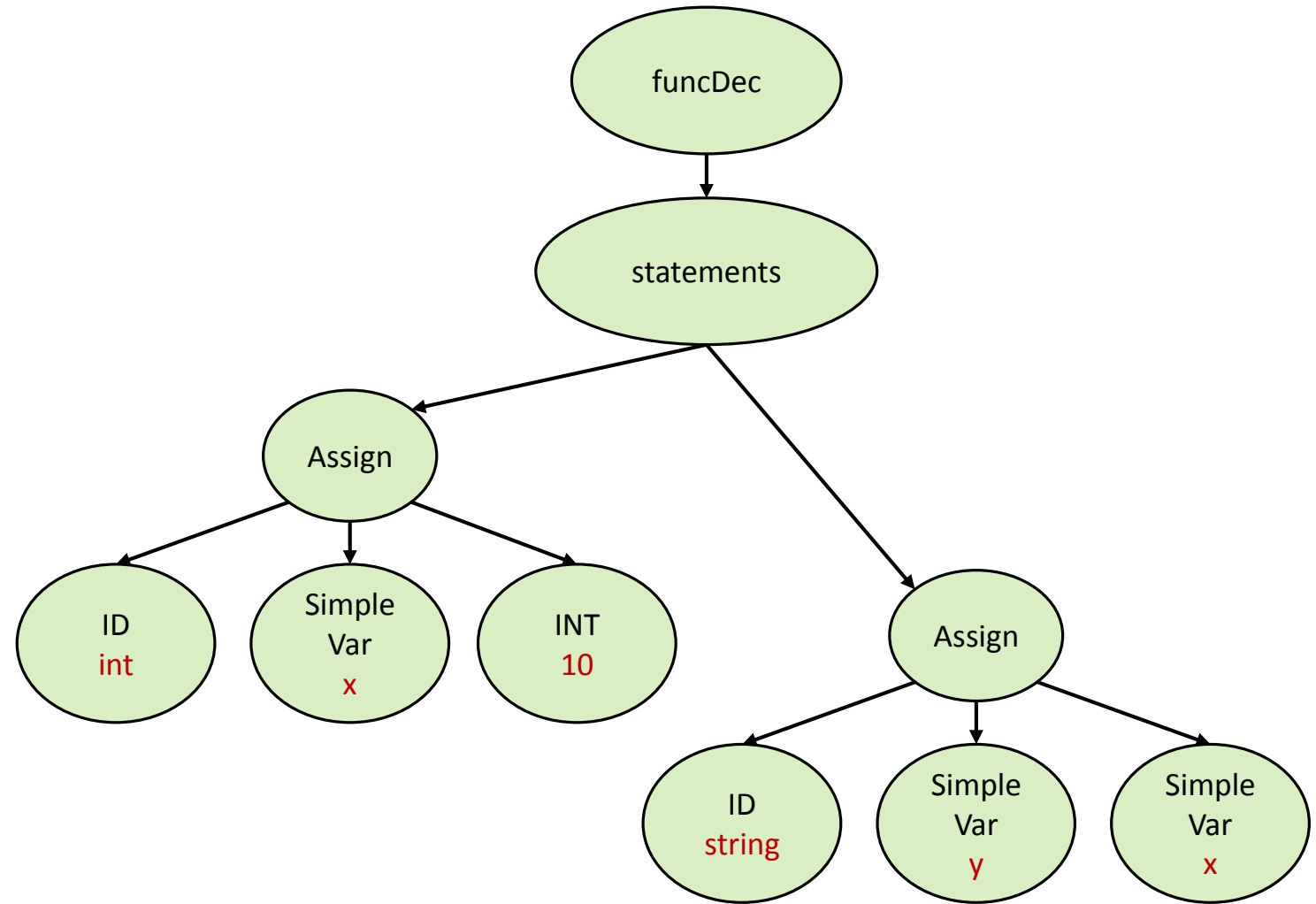
```
void main() {  
    int x = 10;  
    int y = x;  
}
```

Valid



Assignments

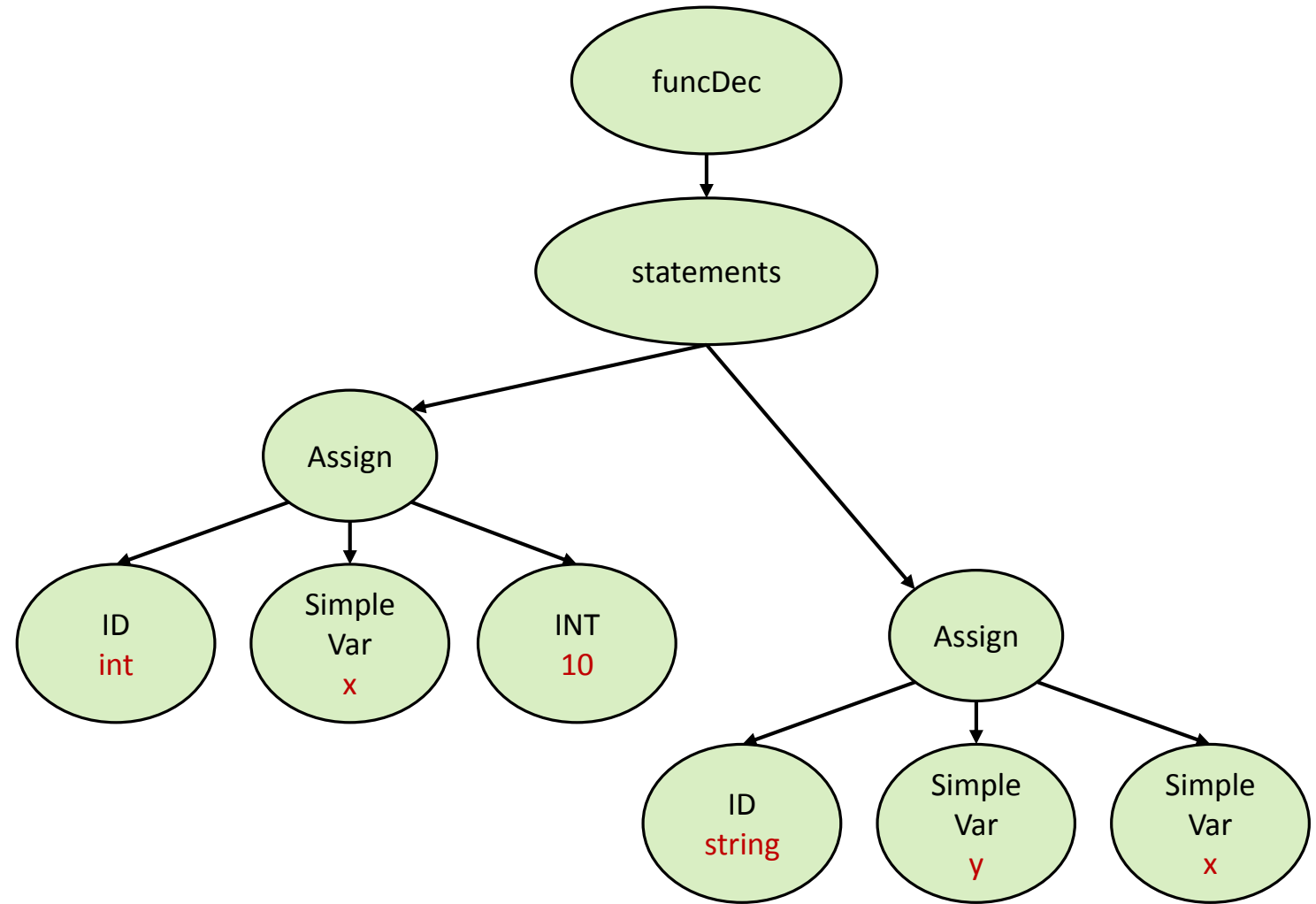
```
void main() {  
  int x = 10;  
  string y = x;  
}
```



Assignments

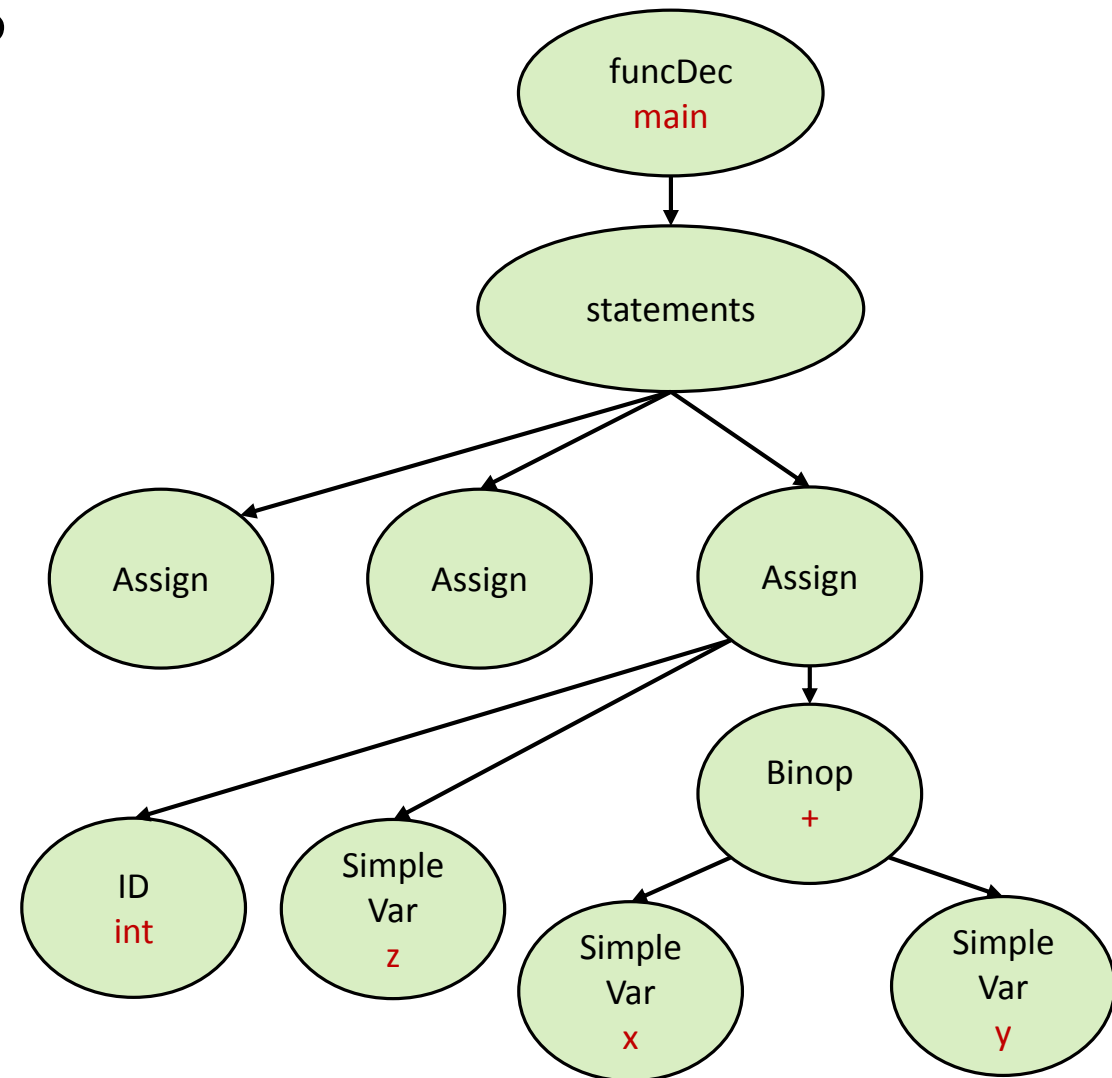
```
void main() {  
  int x = 10;  
  string y = x;  
}
```

Invalid



Binary Operations

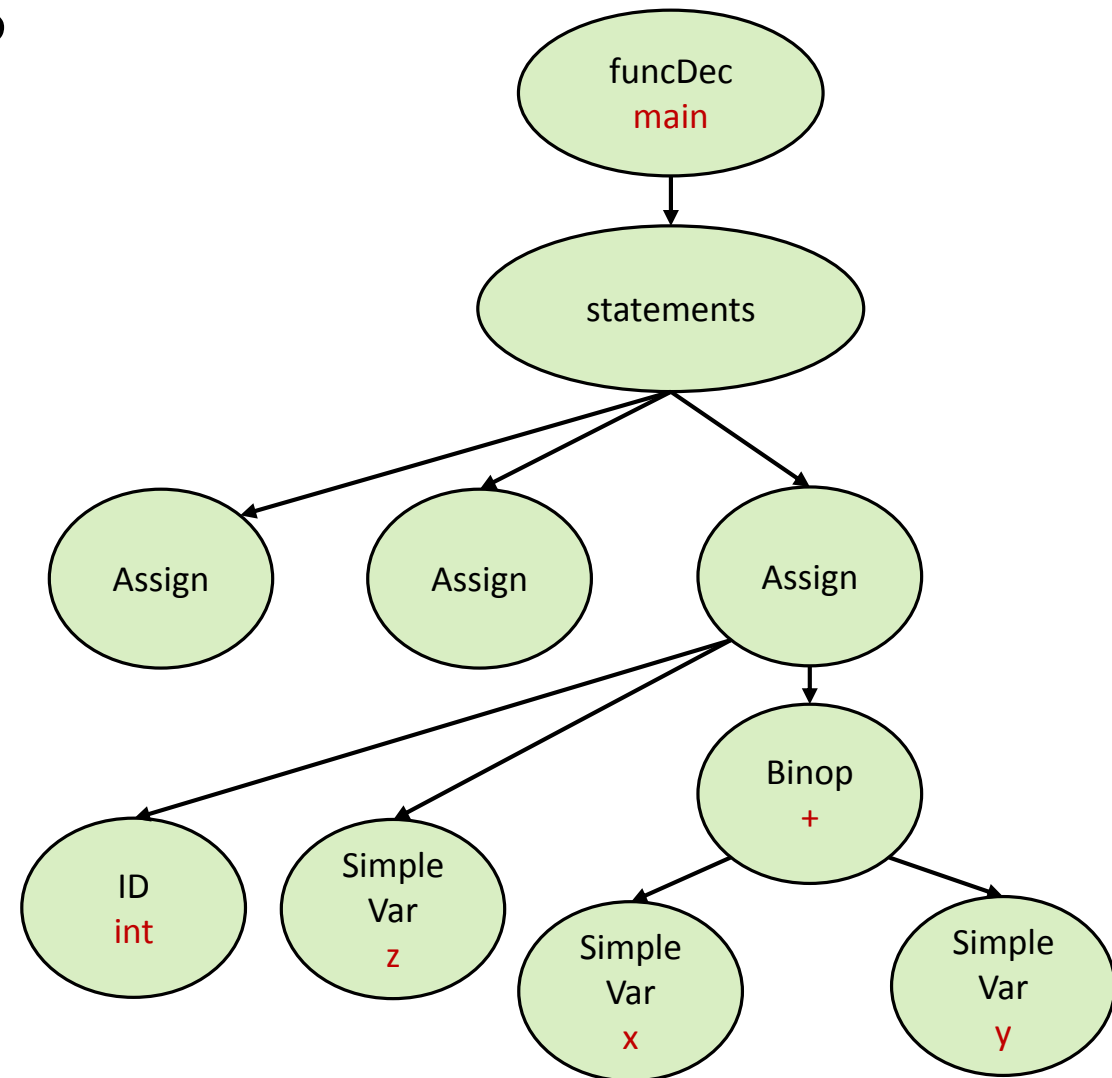
```
void main() {  
    int x = 1;  
    int y = 2;  
    int z = x + y;  
}
```



Binary Operations

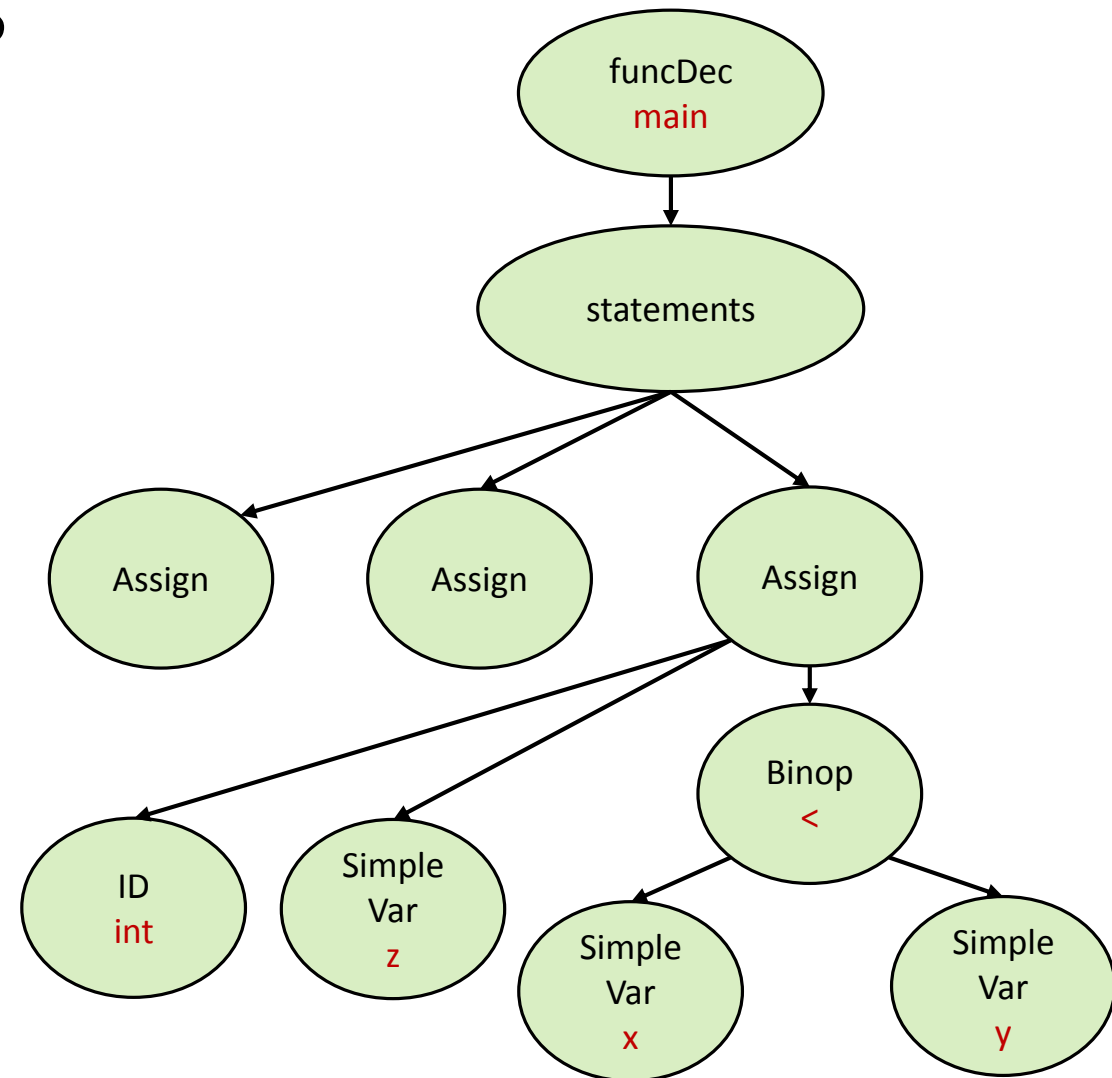
```
void main() {  
    int x = 1;  
    int y = 2;  
    int z = x + y;  
}
```

Valid



Binary Operations

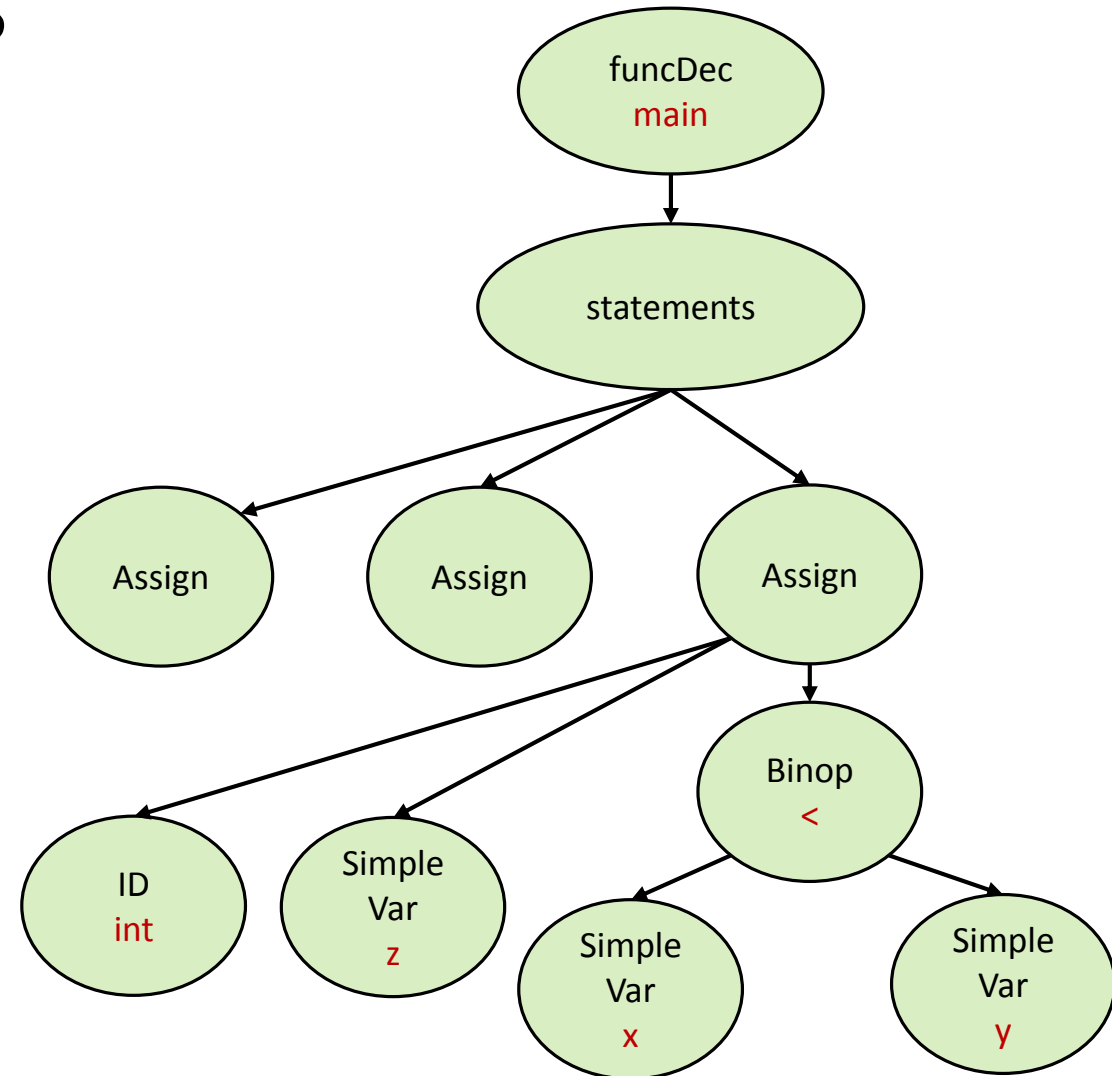
```
void main() {  
    int x = 1;  
    string y = "A";  
    int z = x < y;  
}
```



Binary Operations

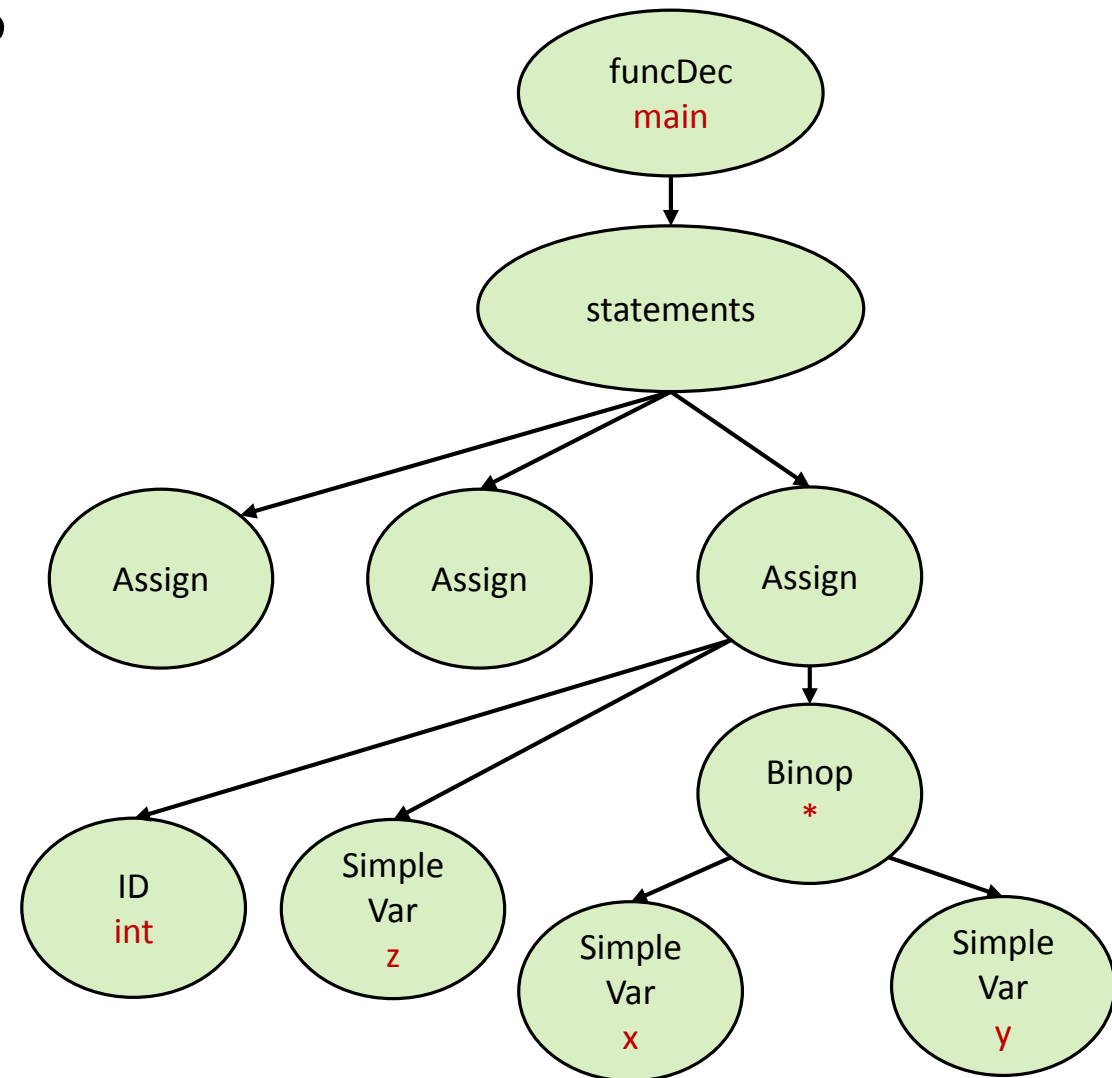
```
void main() {  
  int x = 1;  
  string y = "A";  
  int z = x < y;  
}
```

Invalid



Binary Operations

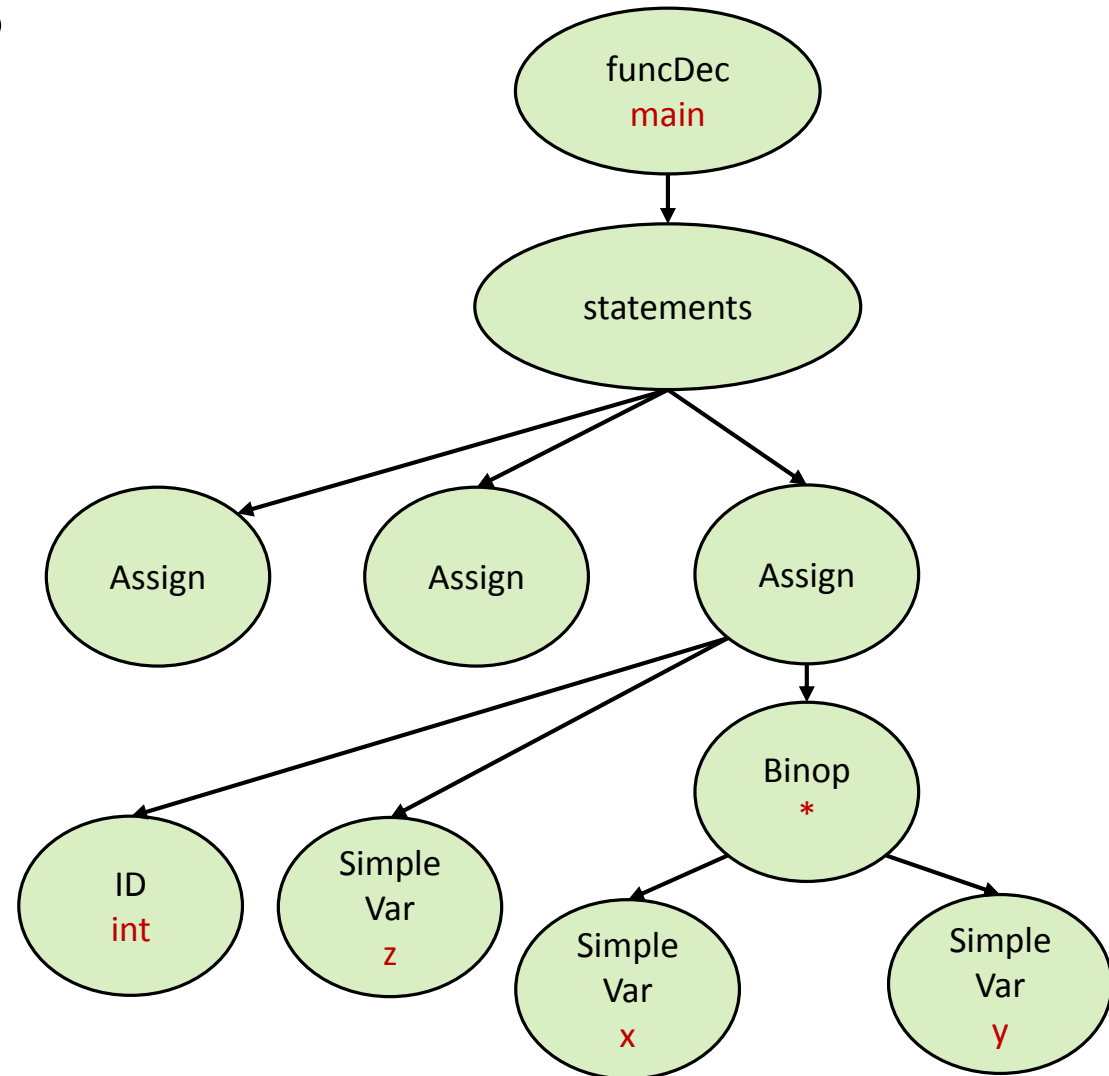
```
void main() {  
    string x = "A";  
    string y = "B";  
    string z = x * y;  
}
```



Binary Operations

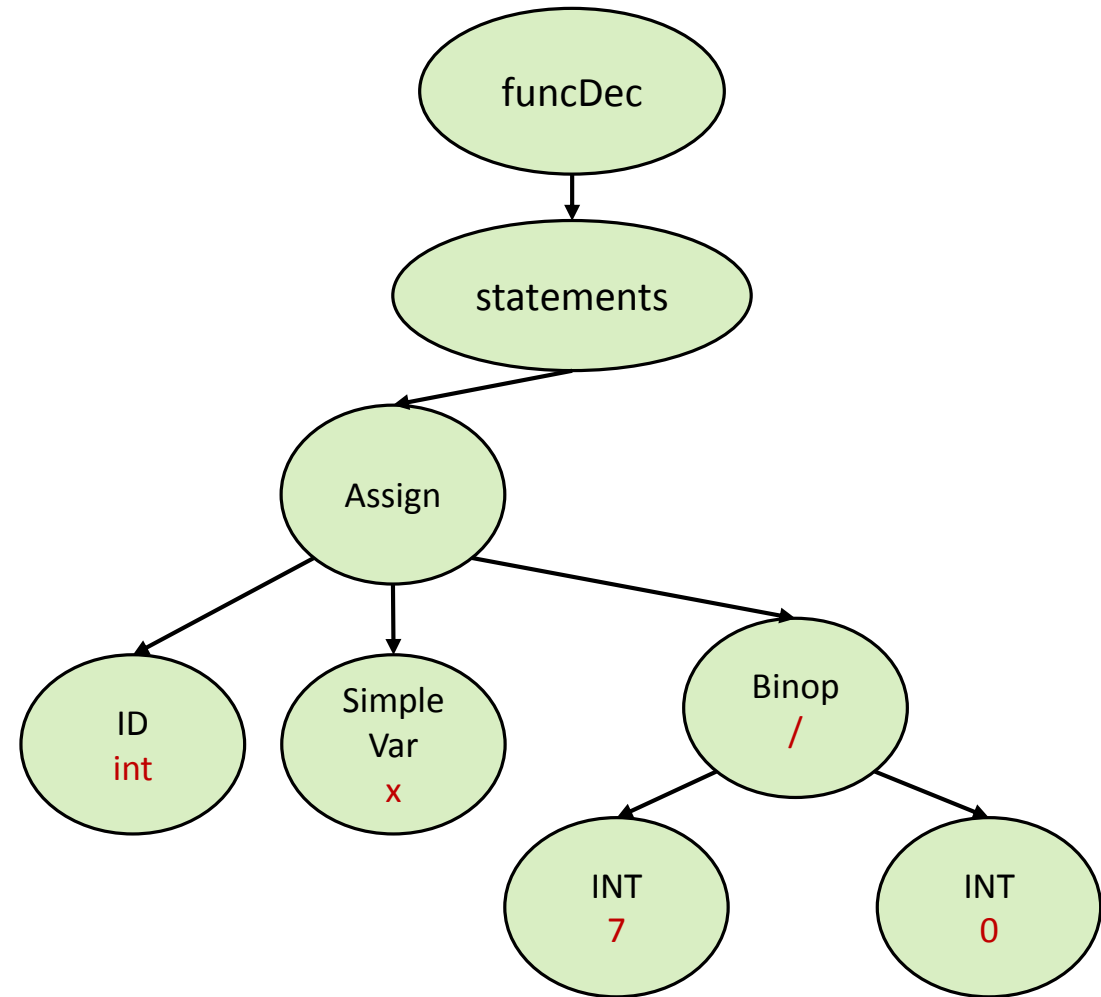
```
void main() {  
    string x = "A";  
    string y = "B";  
    string z = x * y;  
}
```

Invalid



Binary Operations

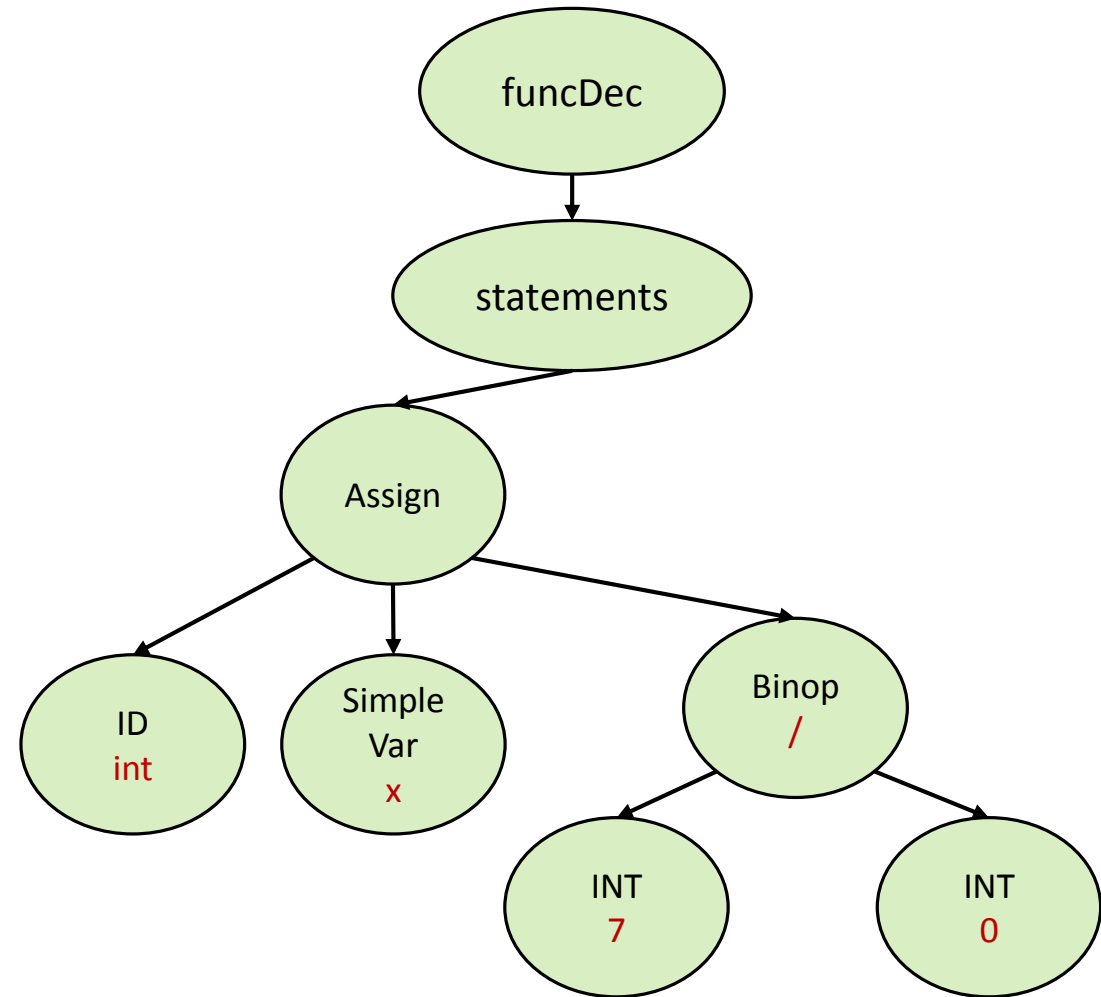
```
void main() {  
    int x = 7 / 0;  
}
```



Binary Operations

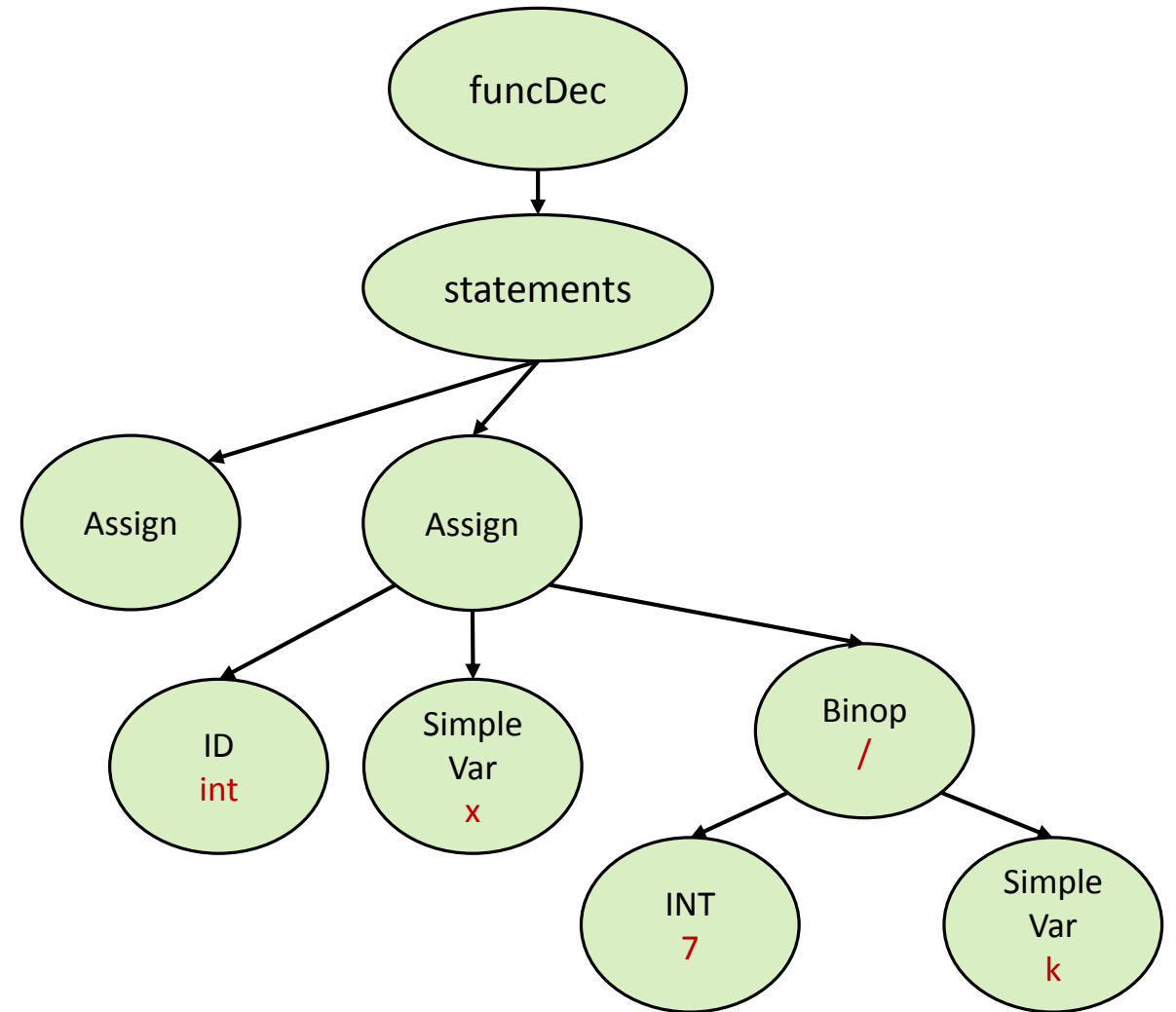
```
void main() {  
    int x = 7 / 0;  
}
```

Invalid



Binary Operations

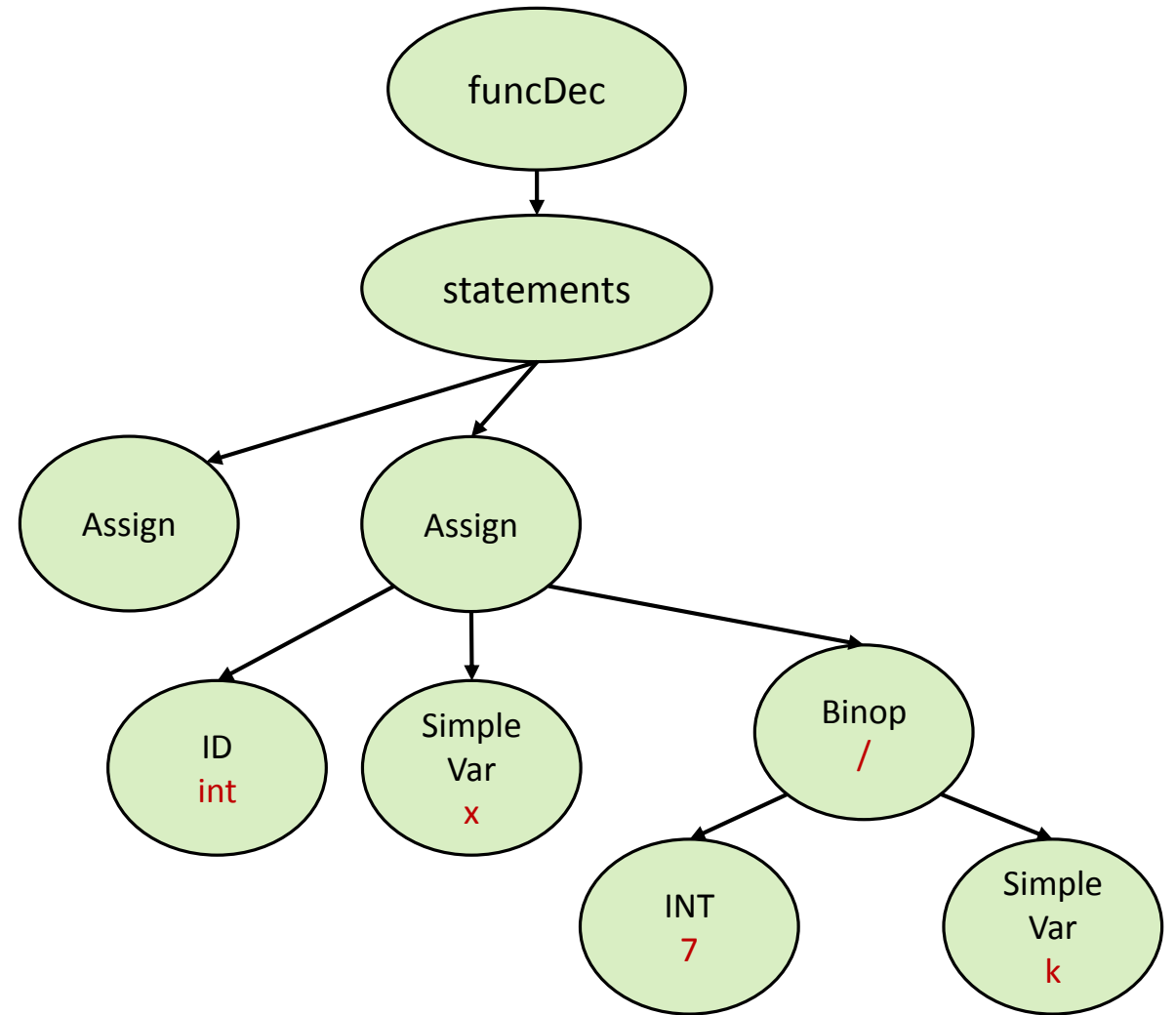
```
void main() {  
    int k = 0;  
    int x = 7 / k;  
}
```



Binary Operations

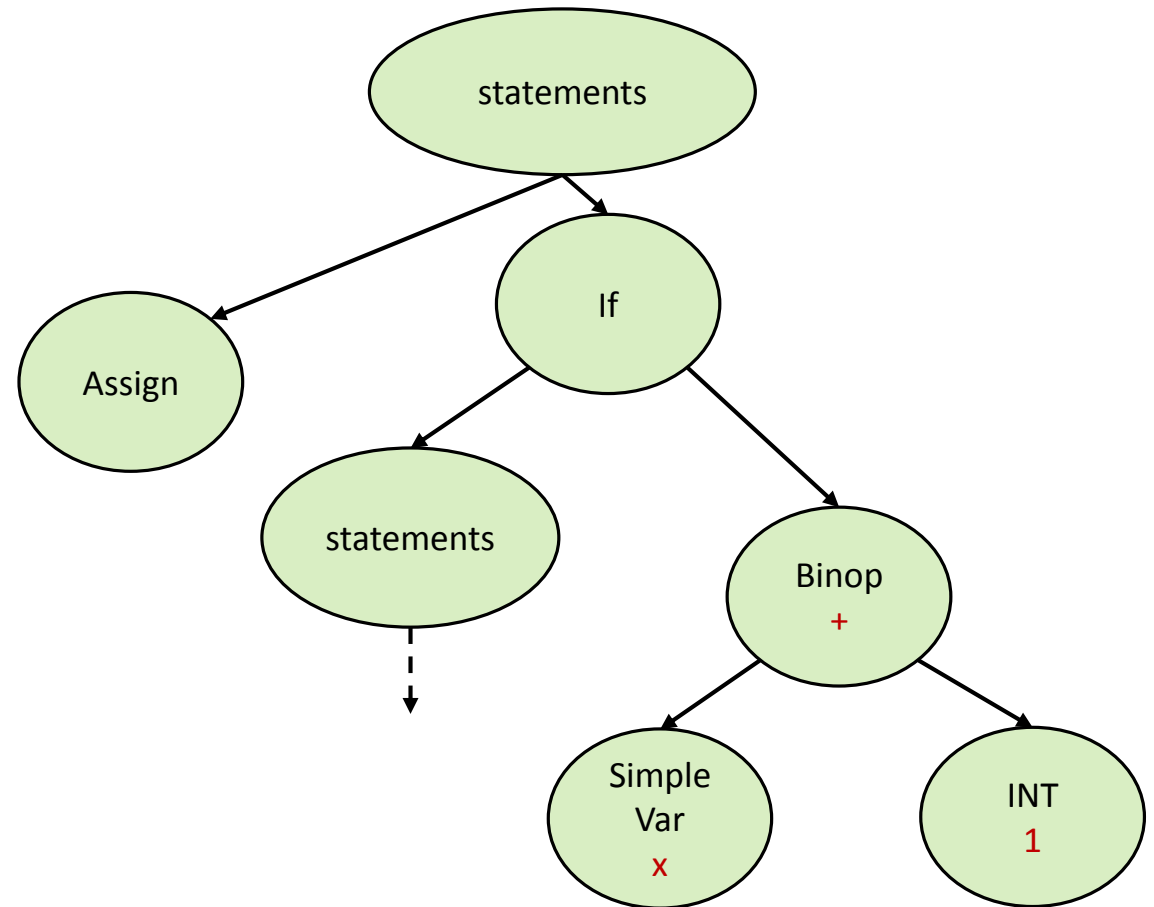
```
void main() {  
    int k = 0;  
    int x = 7 / k;  
}
```

Depends



If, While, ...

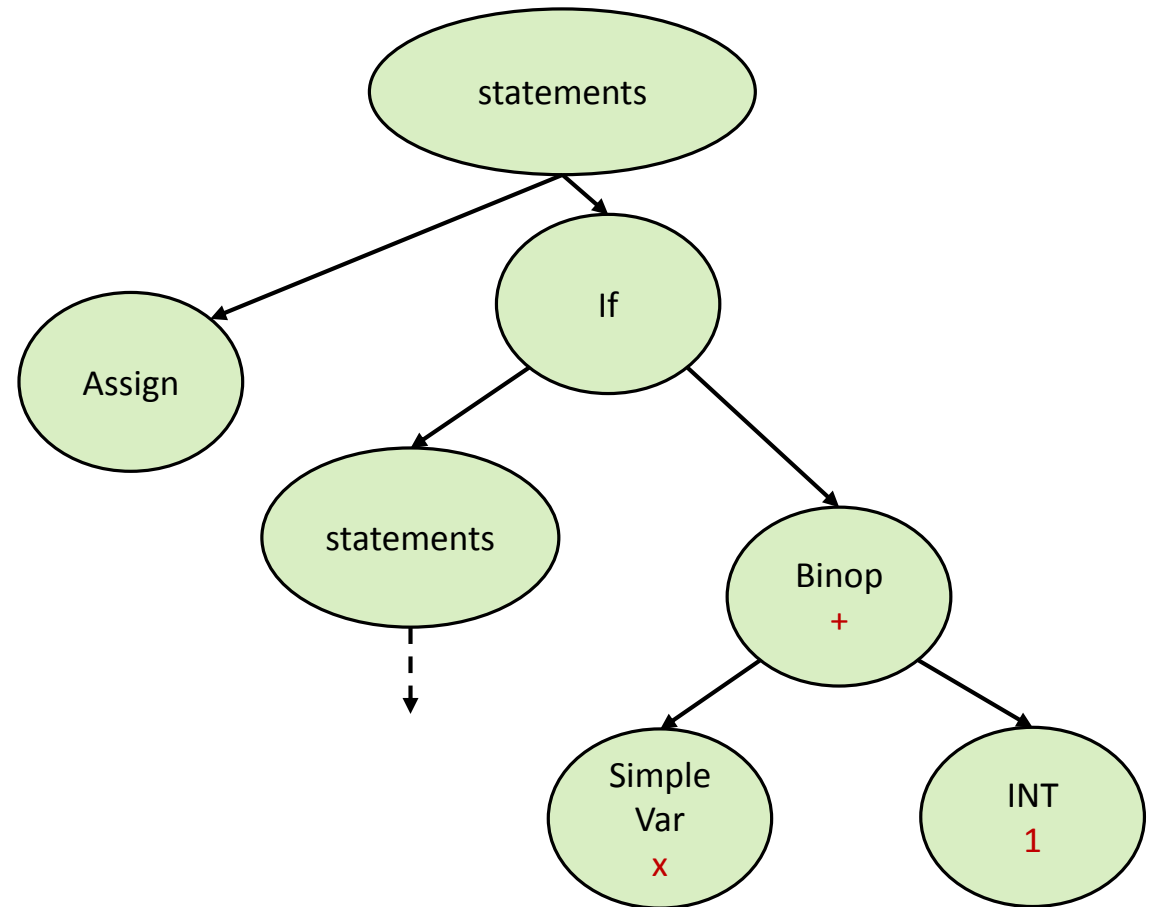
```
void main() {  
    int x = 1;  
    if (x + 1) {  
        int z = 2;  
    }  
}
```



If, While, ...

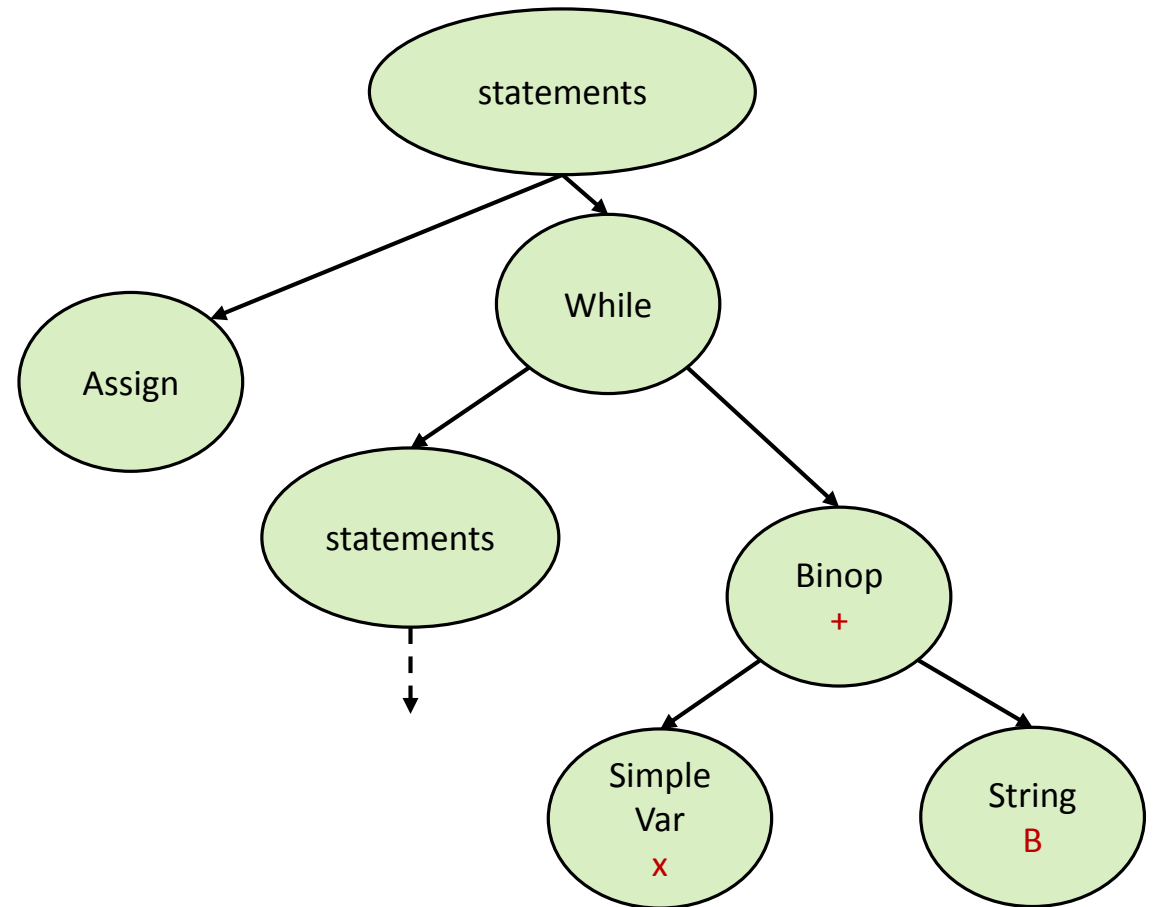
```
void main() {  
    int x = 1;  
    if (x + 1) {  
        int z = 2;  
    }  
}
```

Valid



If, While, ...

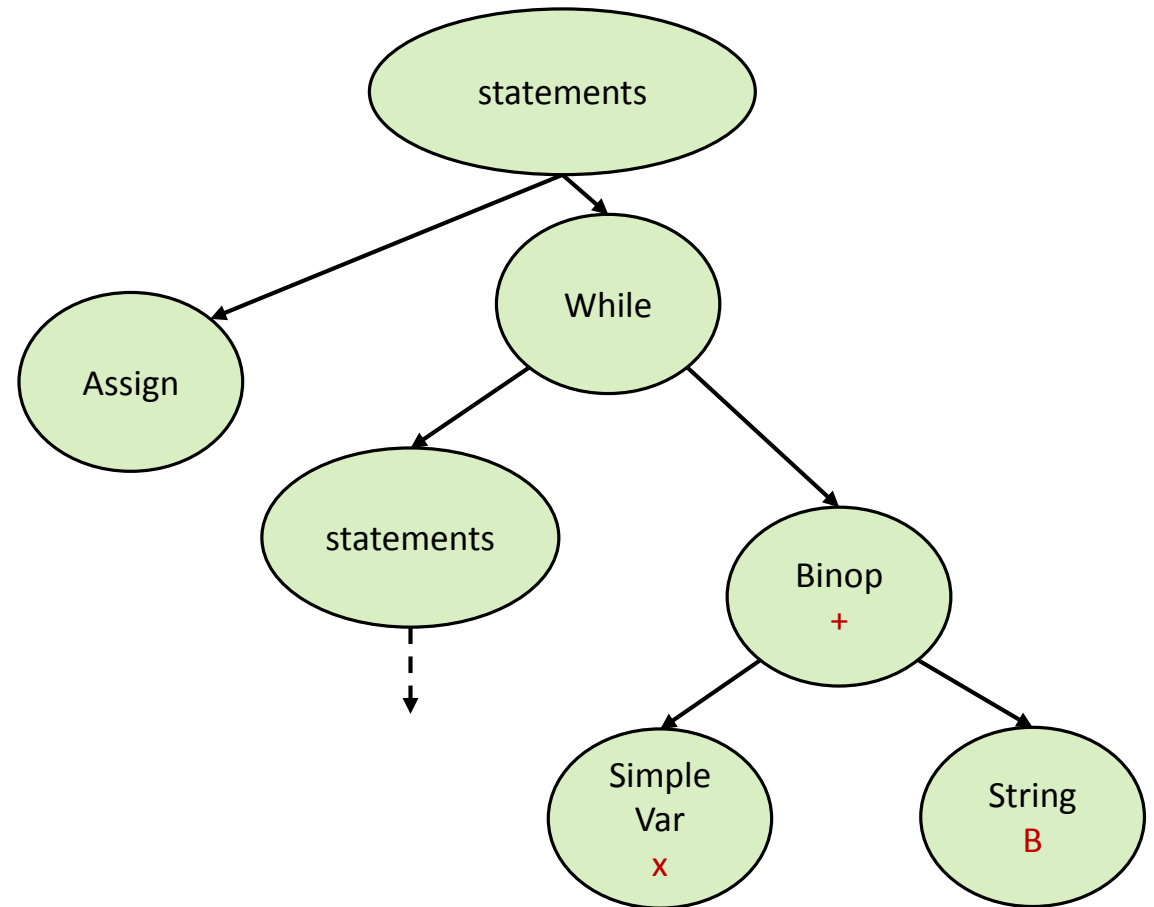
```
void main() {  
    string x = "A";  
    while (x + "B") {  
        int z = 2;  
    }  
}
```



If, While, ...

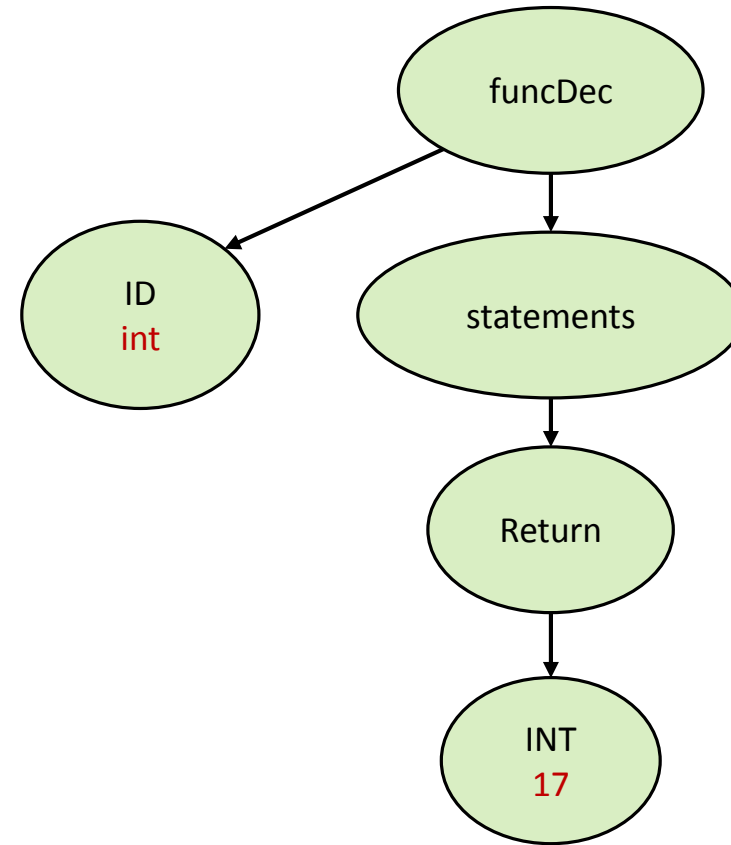
```
void main() {  
    string x = "A";  
    while (x + "B") {  
        int z = 2;  
    }  
}
```

Invalid



Return Statement

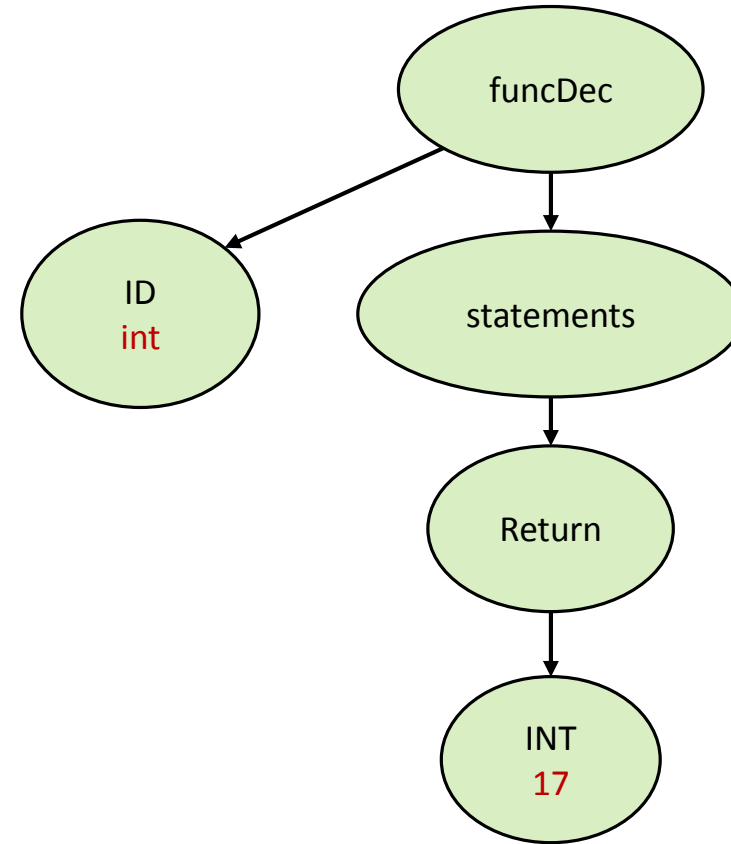
```
int main() {  
    return 17;  
}
```



Return Statement

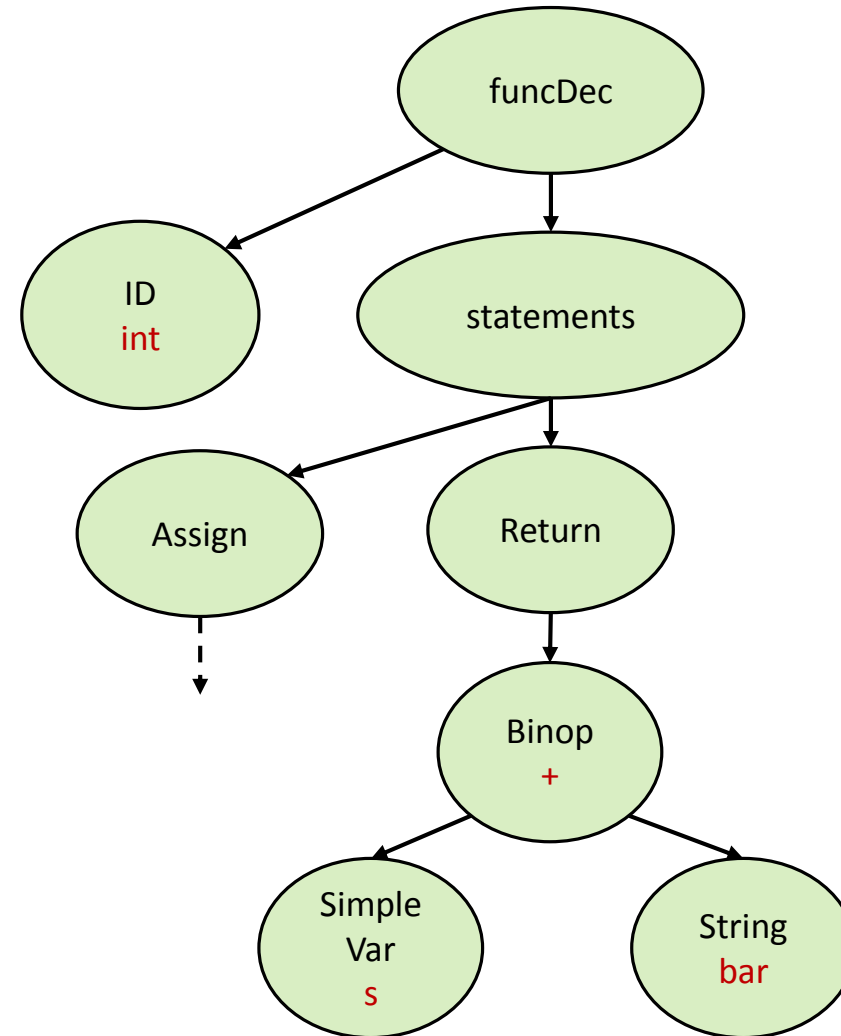
```
int main() {  
    return 17;  
}
```

Valid



Return Statement

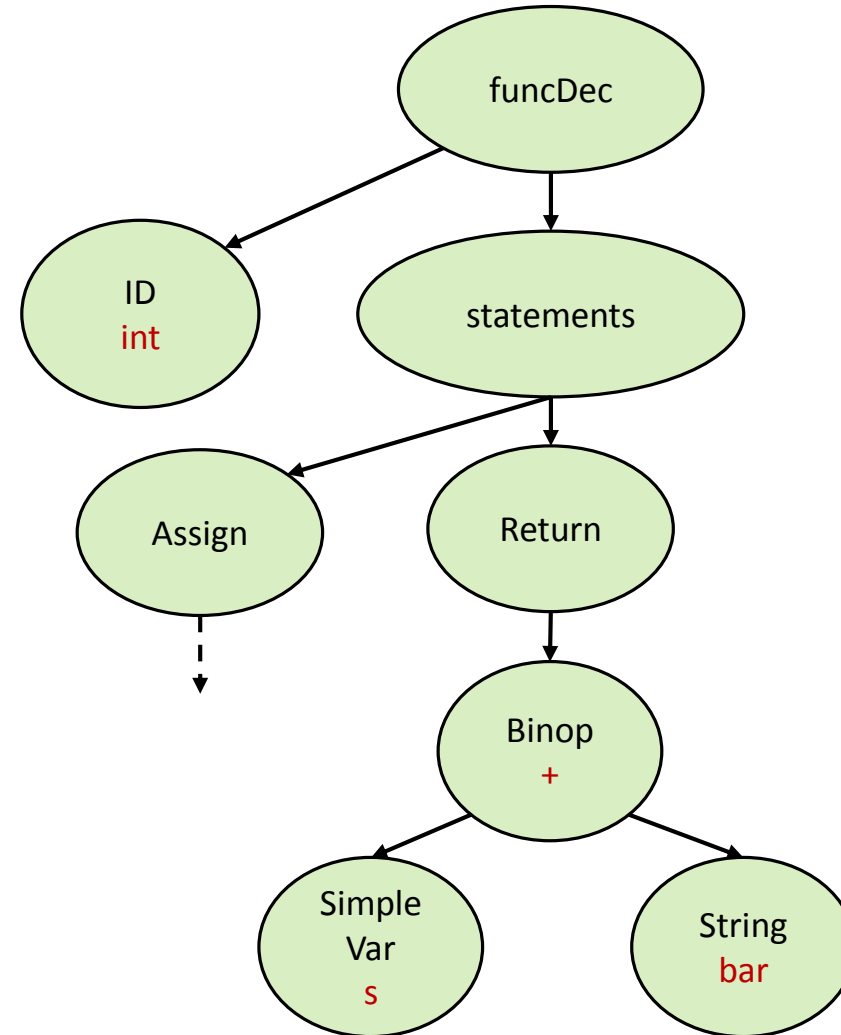
```
int main() {  
    string s = "foo"  
    return s + "bar";  
}
```



Return Statement

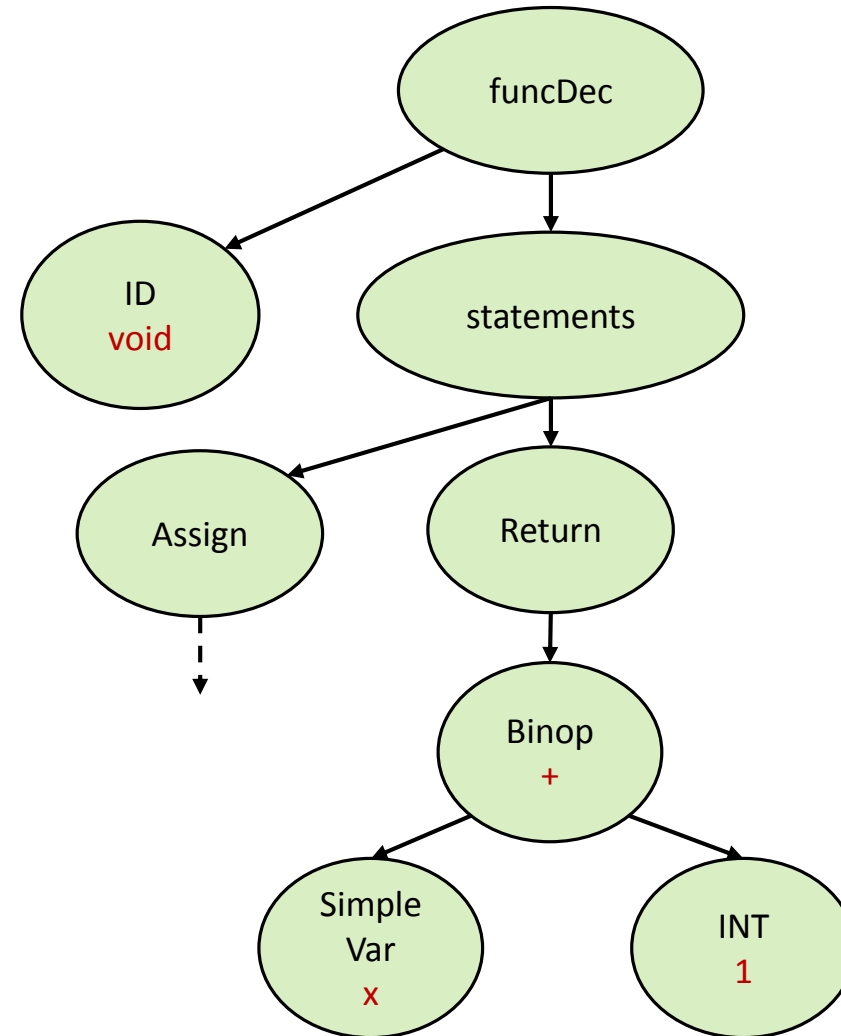
```
int main() {  
    string s = "foo"  
    return s + "bar";  
}
```

Invalid



Return Statement

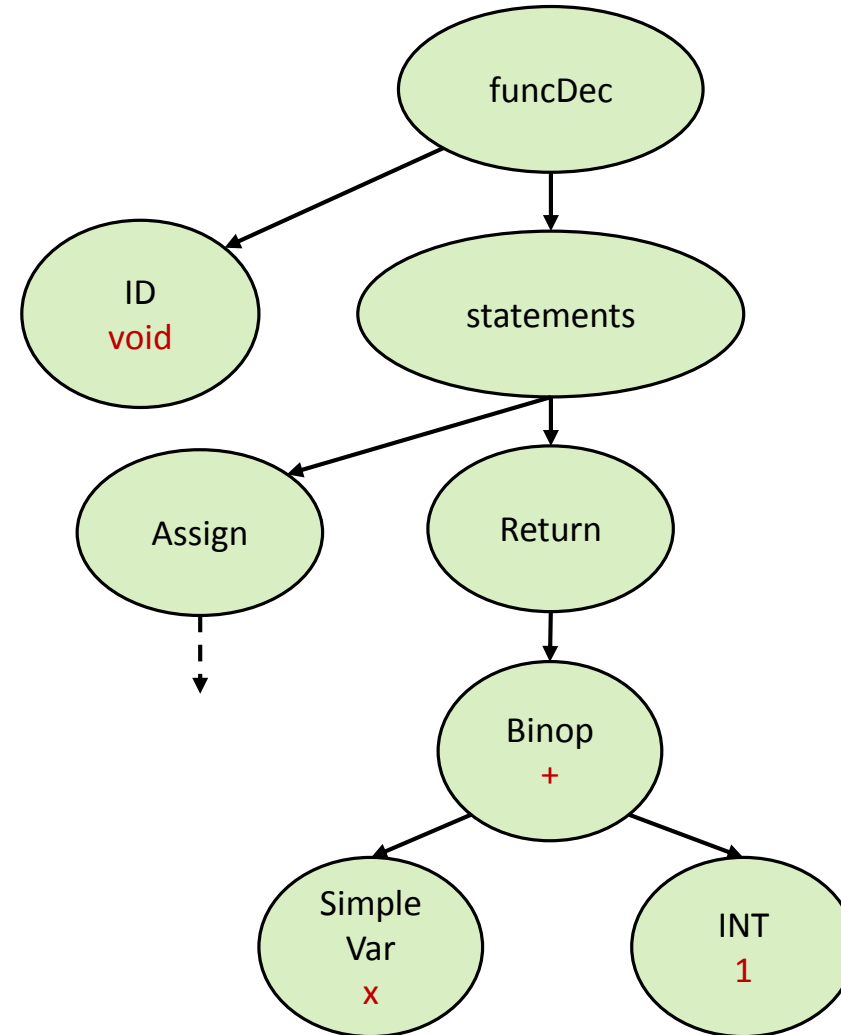
```
void main() {  
    int x = 1;  
    return x + 1;  
}
```



Return Statement

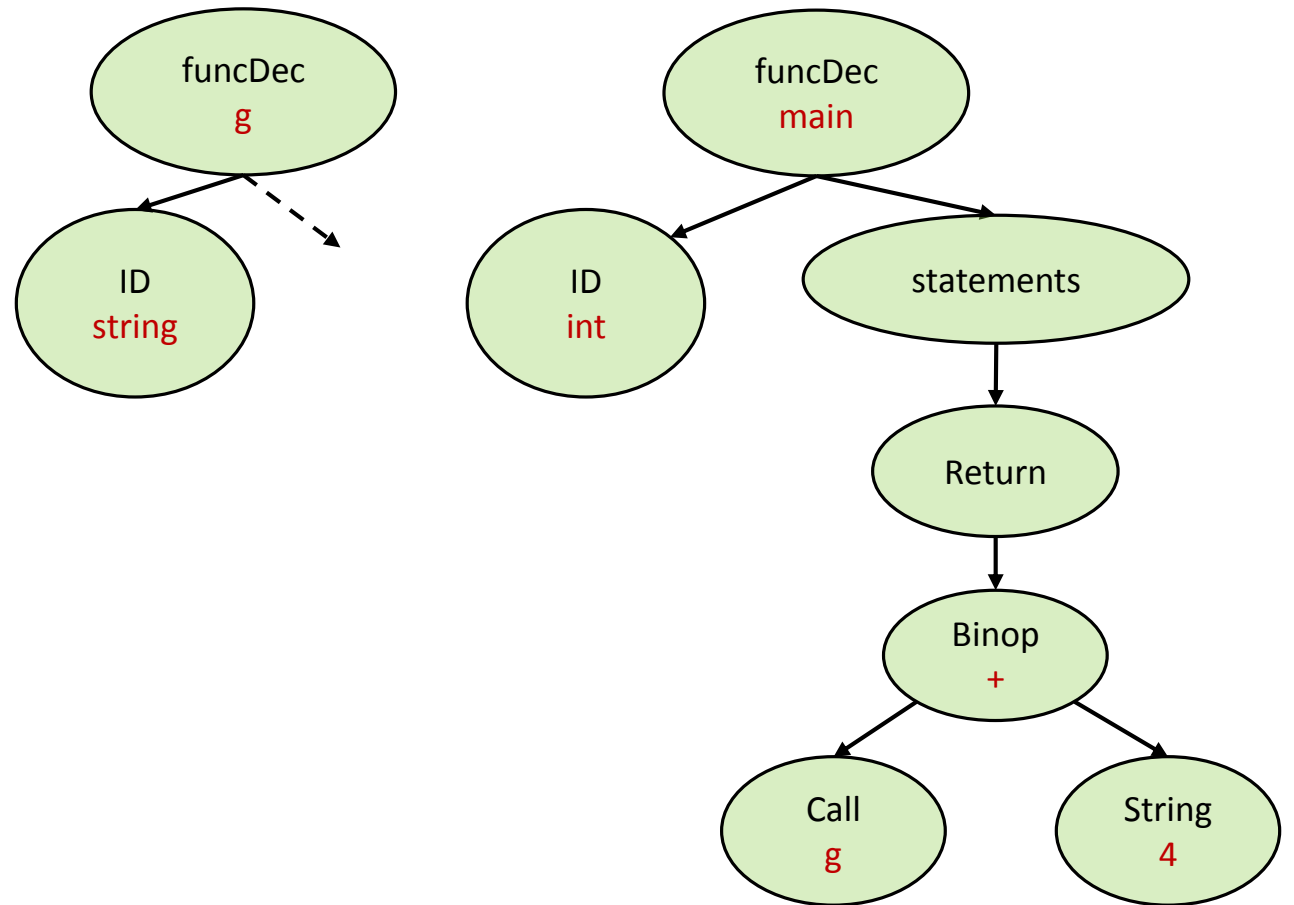
```
void main() {  
    int x = 1;  
    return x + 1;  
}
```

Invalid



Return Statement

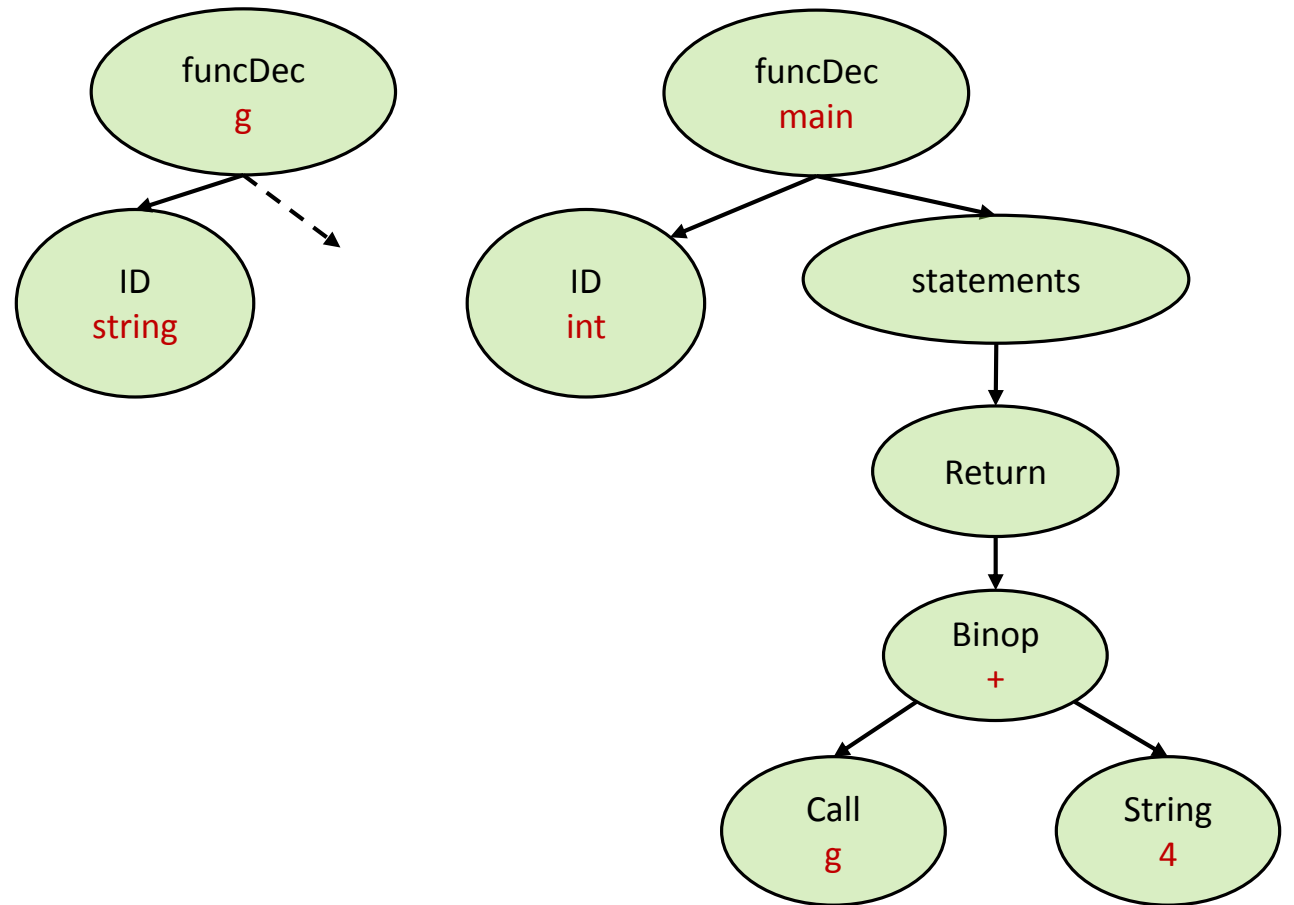
```
string g() {  
    return "123";  
}  
int main() {  
    return g() + "4";  
}
```



Return Statement

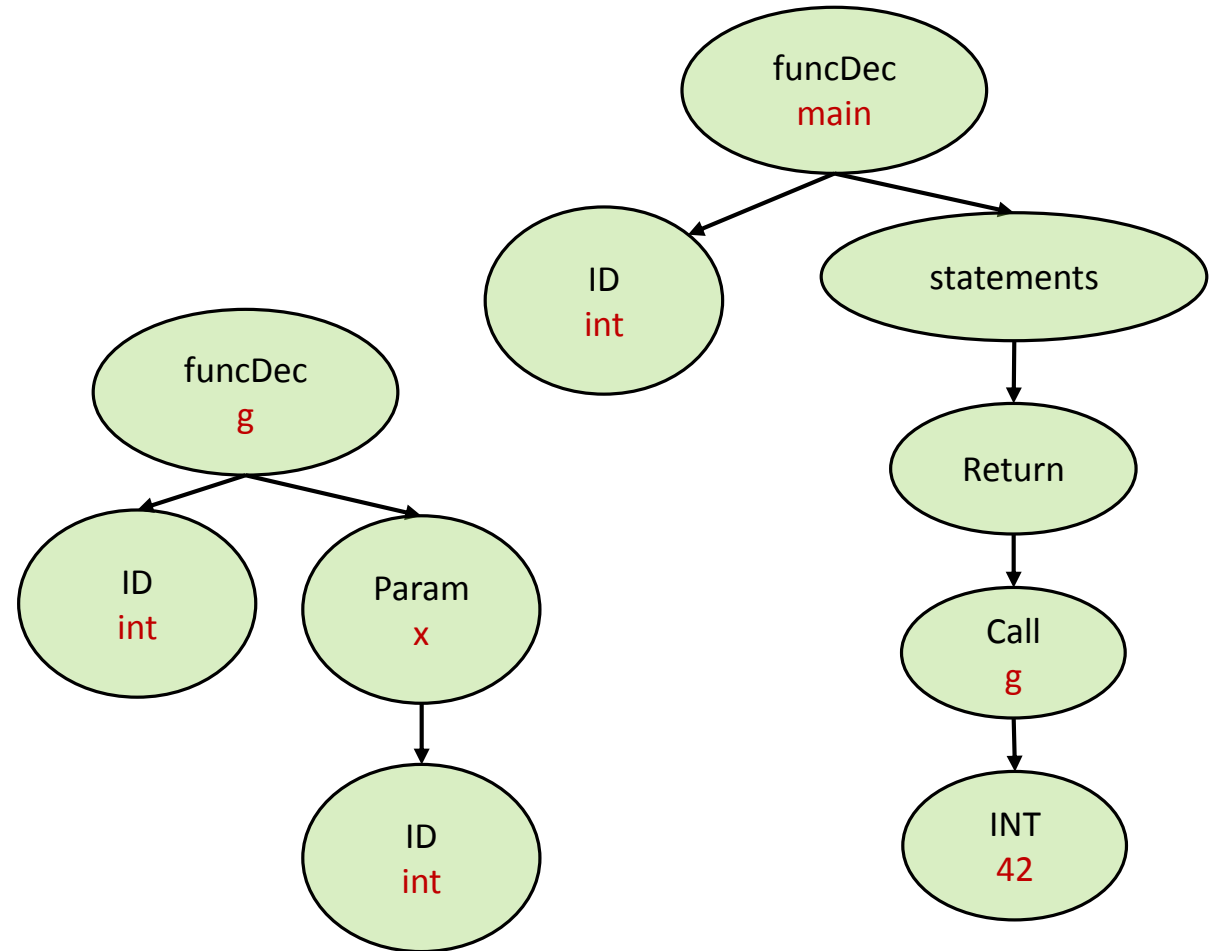
```
string g() {  
    return "123";  
}  
int main() {  
    return g() + "4";  
}
```

Invalid



Function Calls

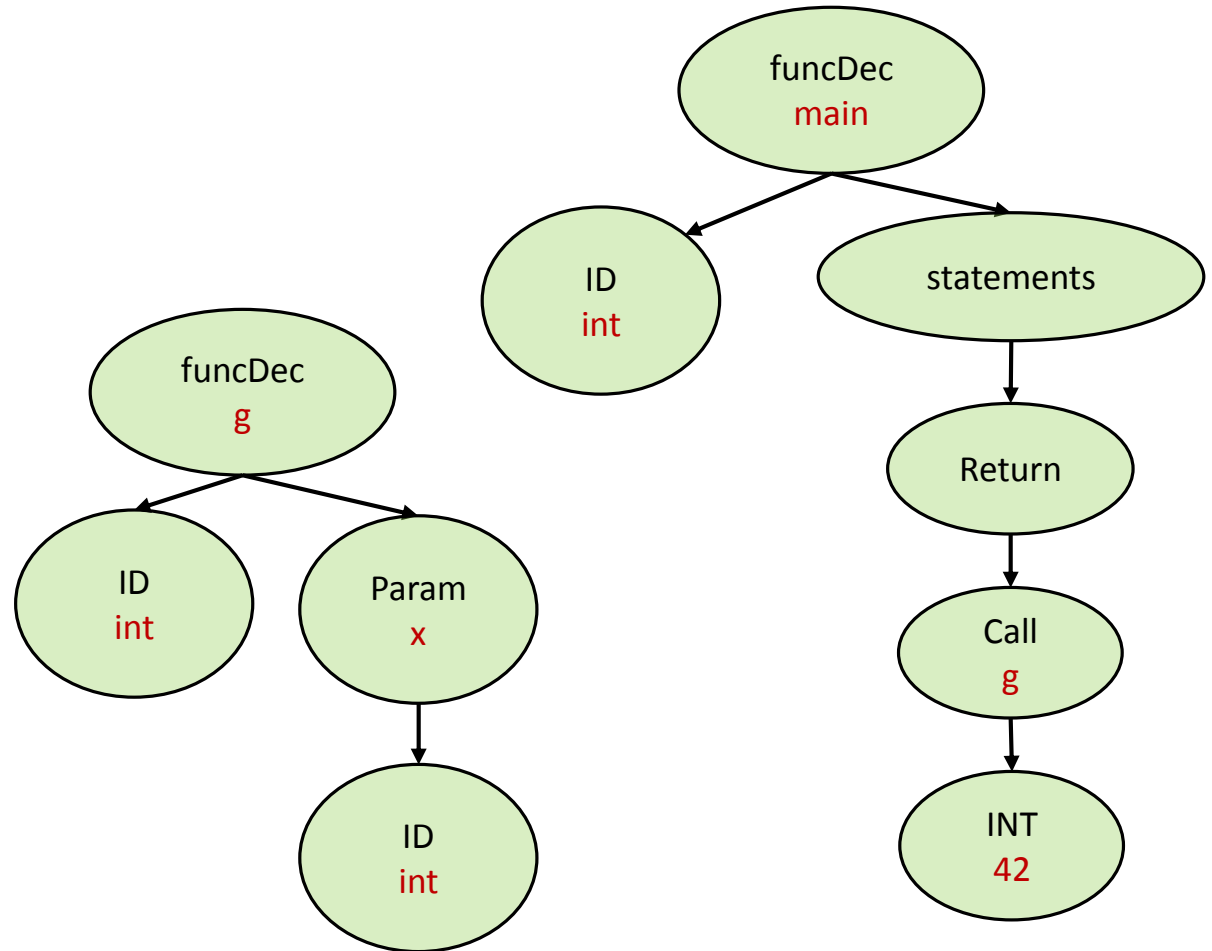
```
int g(int x) {  
    return x + 1;  
}  
int main() {  
    return g(42);  
}
```



Function Calls

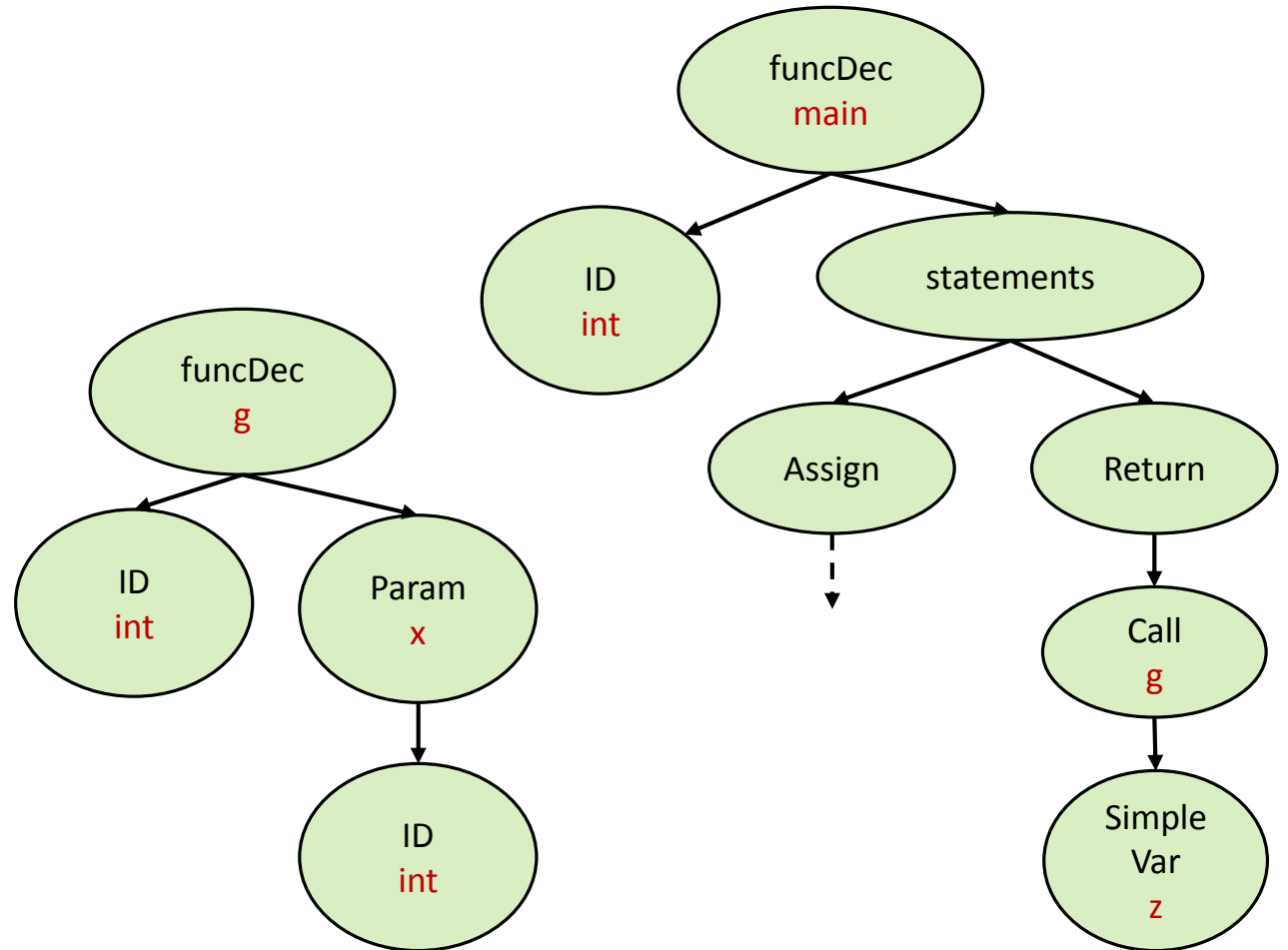
```
int g(int x) {  
    return x + 1;  
}  
int main() {  
    return g(42);  
}
```

Valid



Function Calls

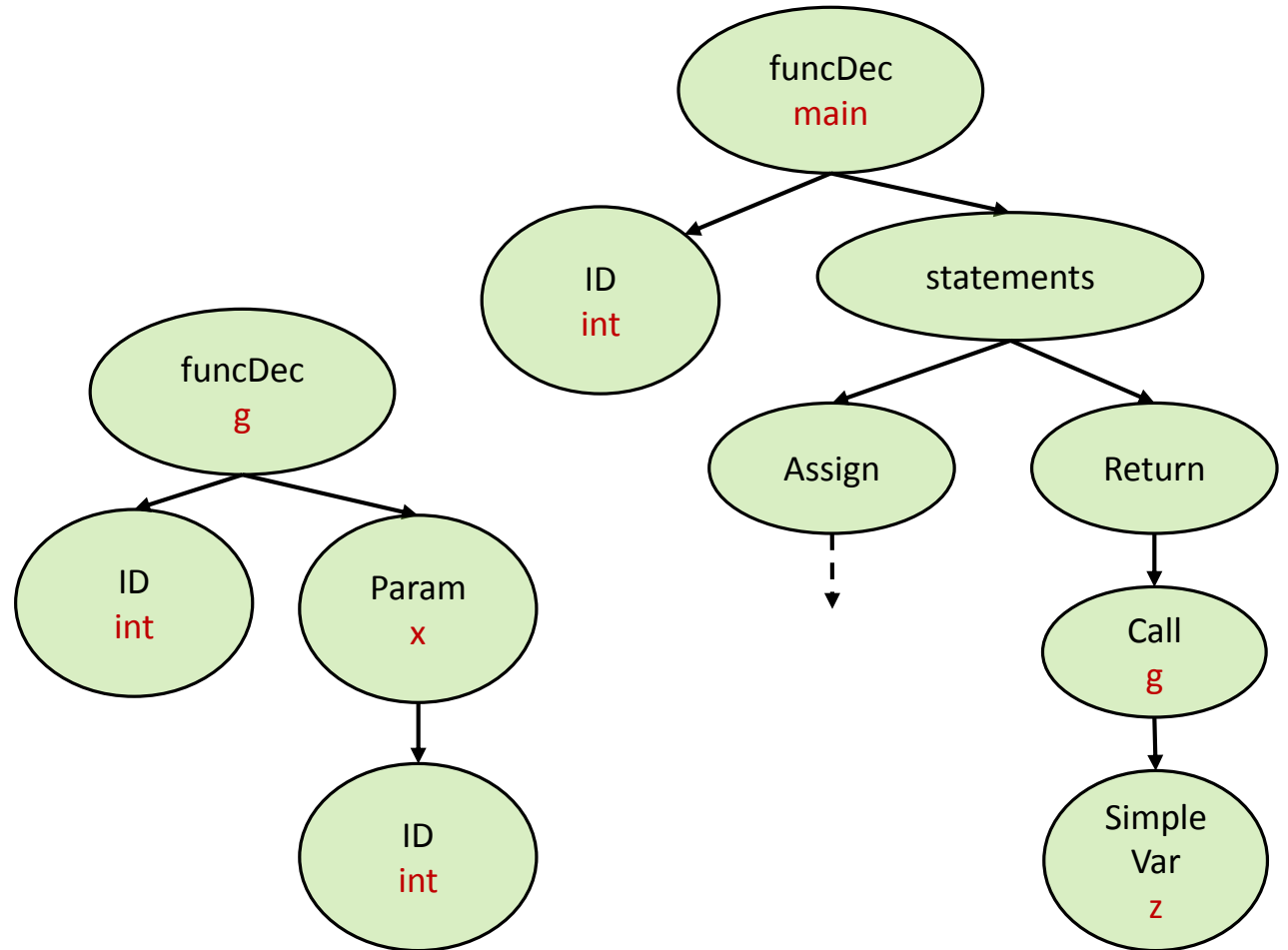
```
int g(int x) {  
    return x + 1;  
}  
int main() {  
    string z = "..."  
    return g(z);  
}
```



Function Calls

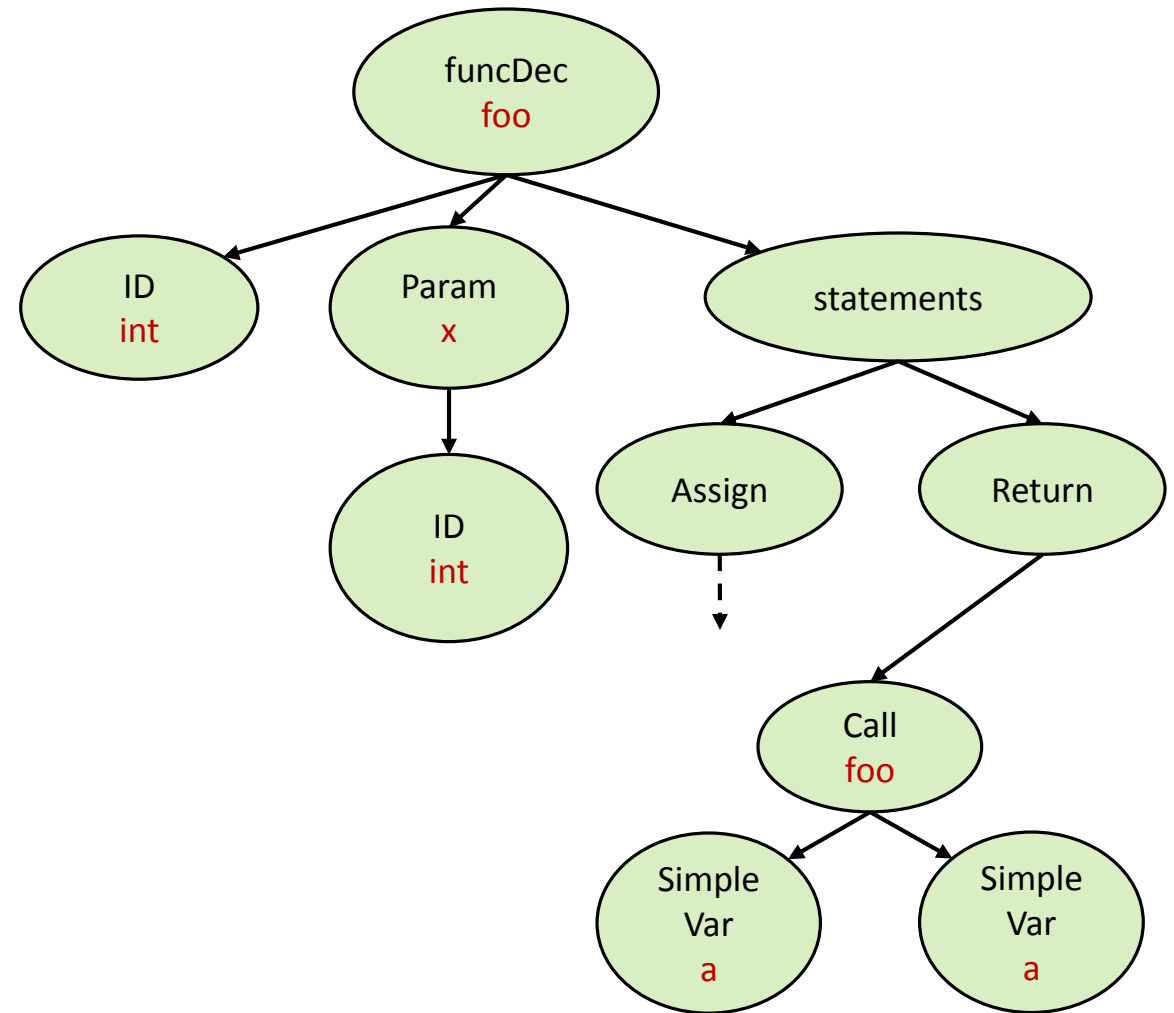
```
int g(int x) {  
    return x + 1;  
}  
int main() {  
    string z = "..."  
    return g(z);  
}
```

Invalid



Function Calls

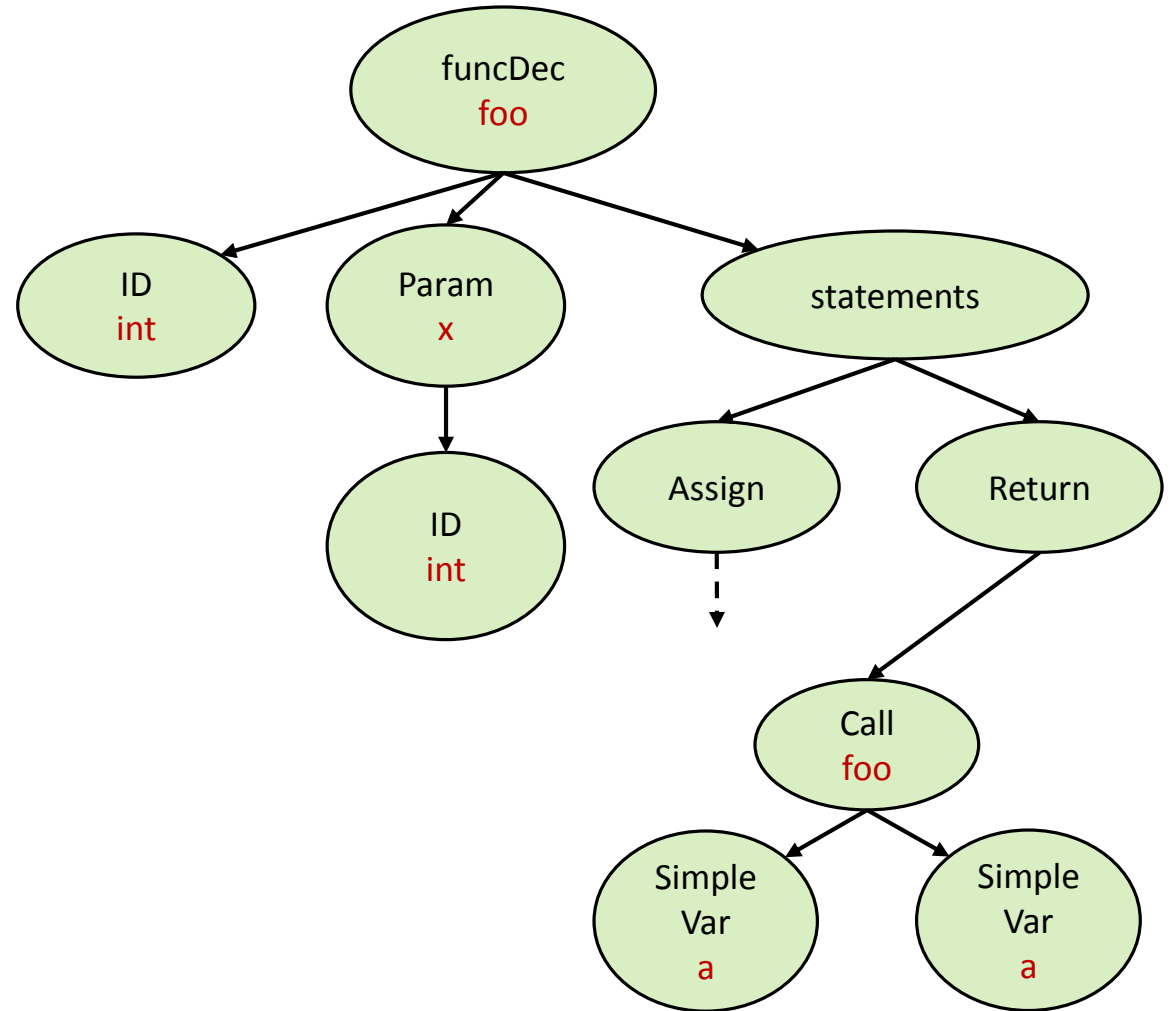
```
int foo(int k) {  
    int a = k * 10;  
    return foo(a, a);  
}
```



Function Calls

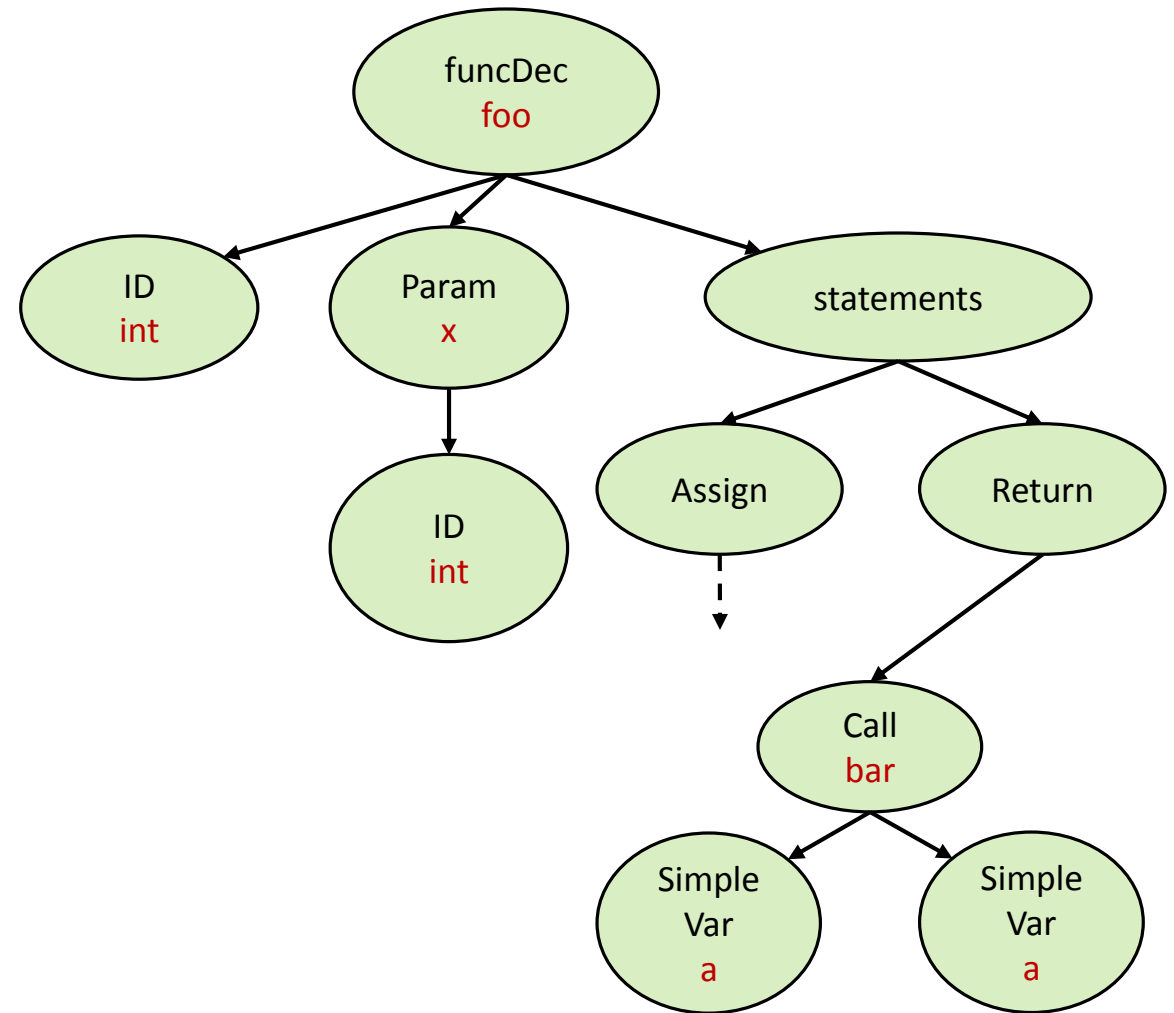
```
int foo(int k) {  
    int a = k * 10;  
    return foo(a, a);  
}
```

Invalid



Function Calls

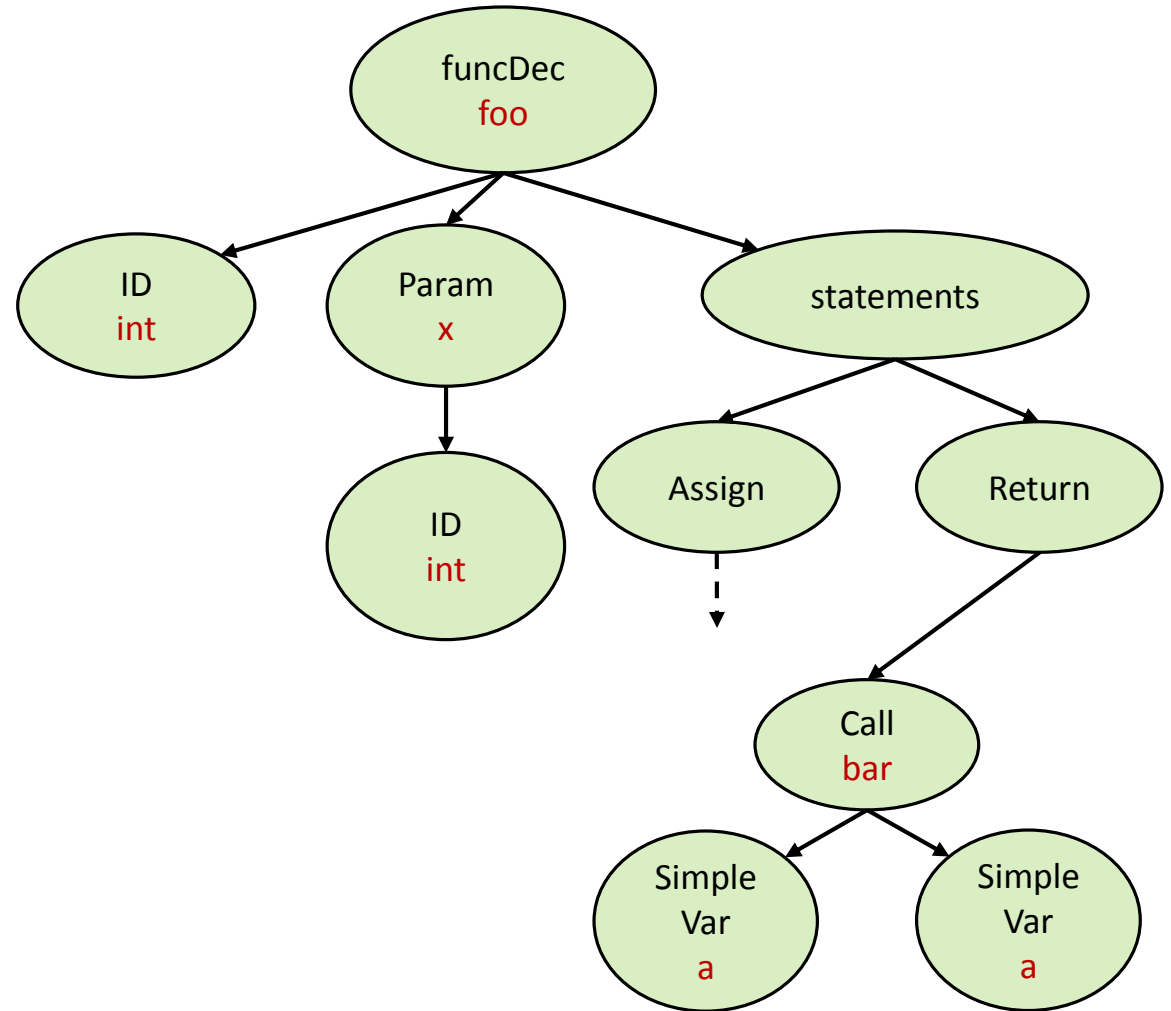
```
int foo(int k) {  
    int a = k * 10;  
    return bar(a, a);  
}
```



Function Calls

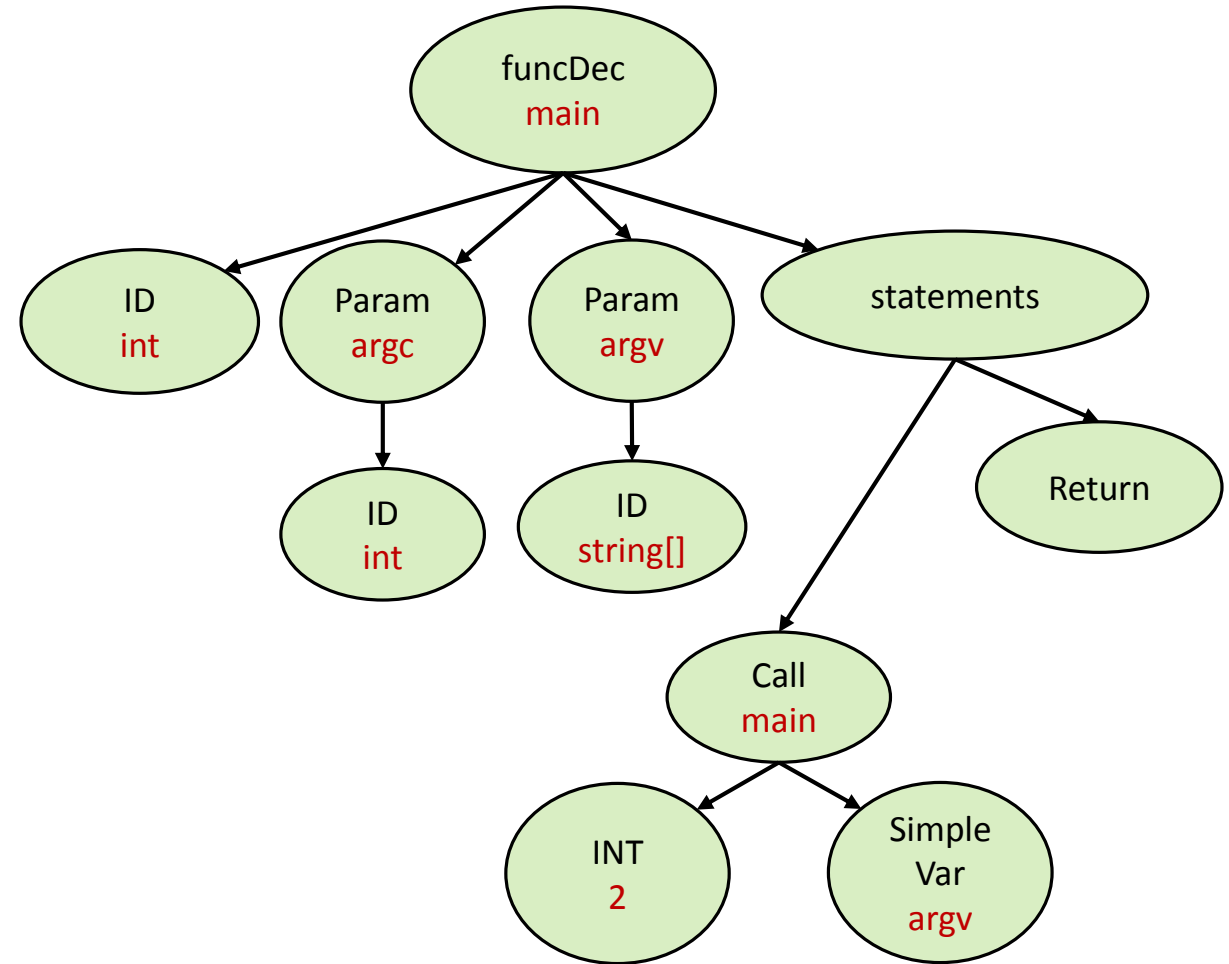
```
int foo(int k) {  
    int a = k * 10;  
    return bar(a, a);  
}
```

Invalid



Function Calls

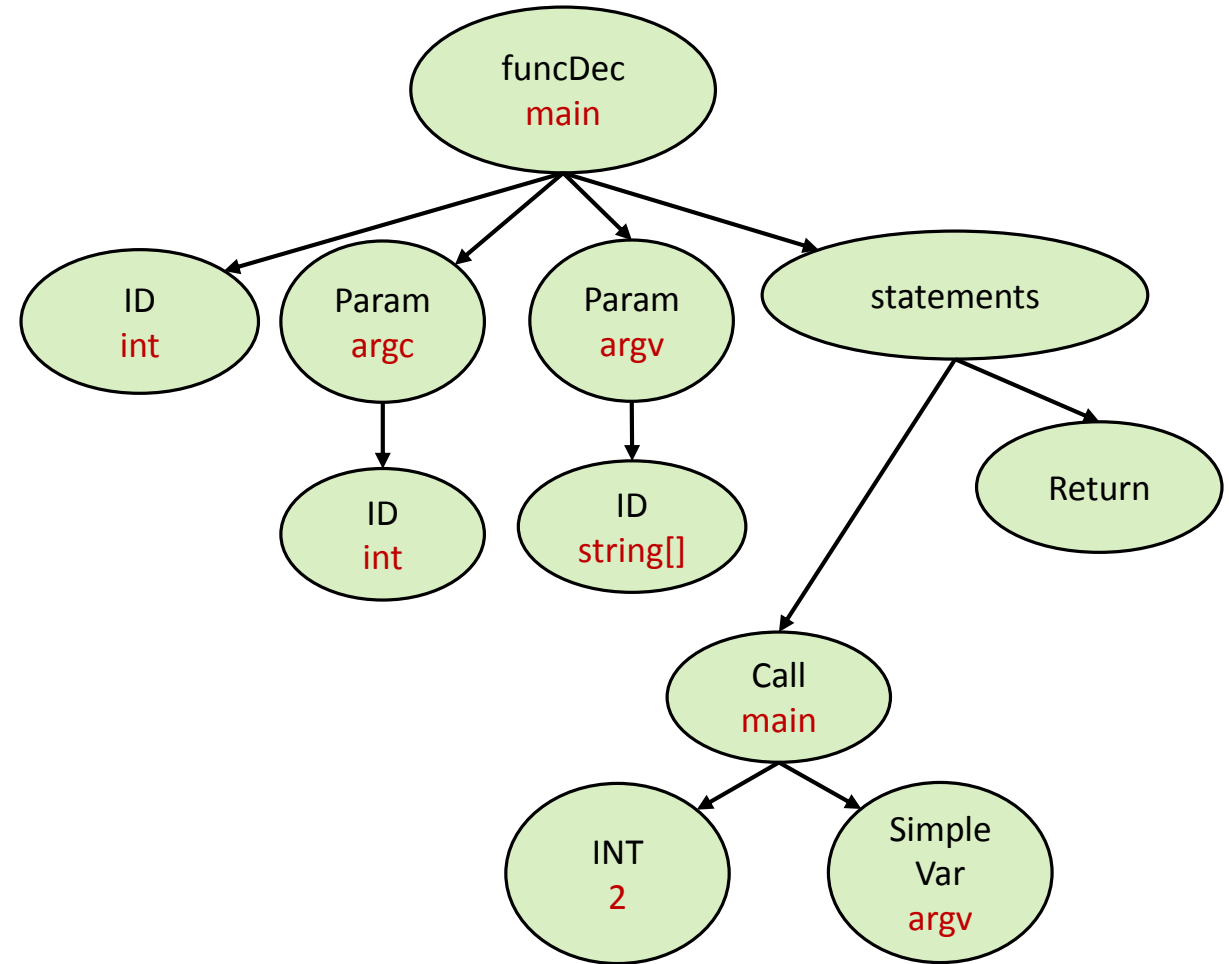
```
int main(int argc,  
          string argv[]){  
    main(2, argv);  
    return 0;  
}
```



Function Calls

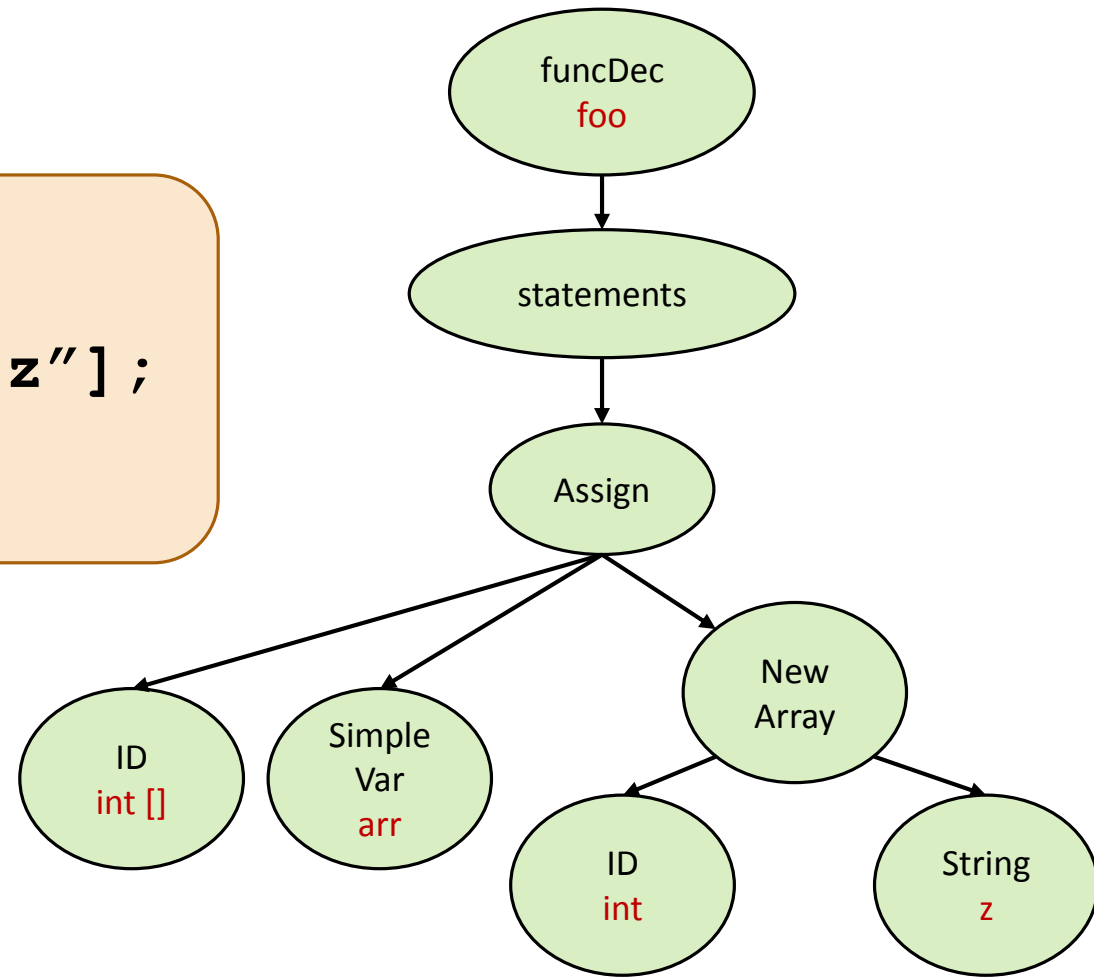
```
int main(int argc,  
          string argv[]){  
    main(2, argv);  
    return 0;  
}
```

Valid



Arrays

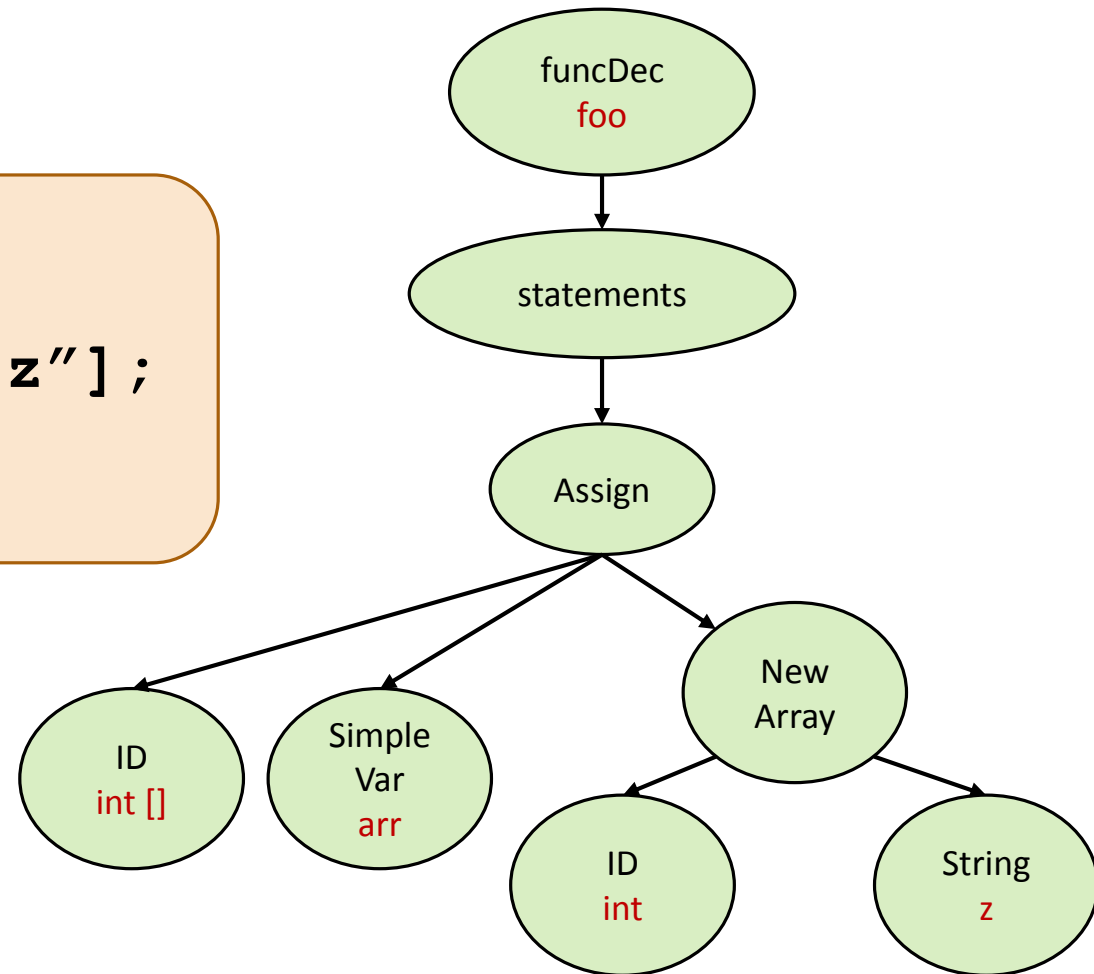
```
void foo(void) {  
    int[] arr = new int["z"];  
}
```



Arrays

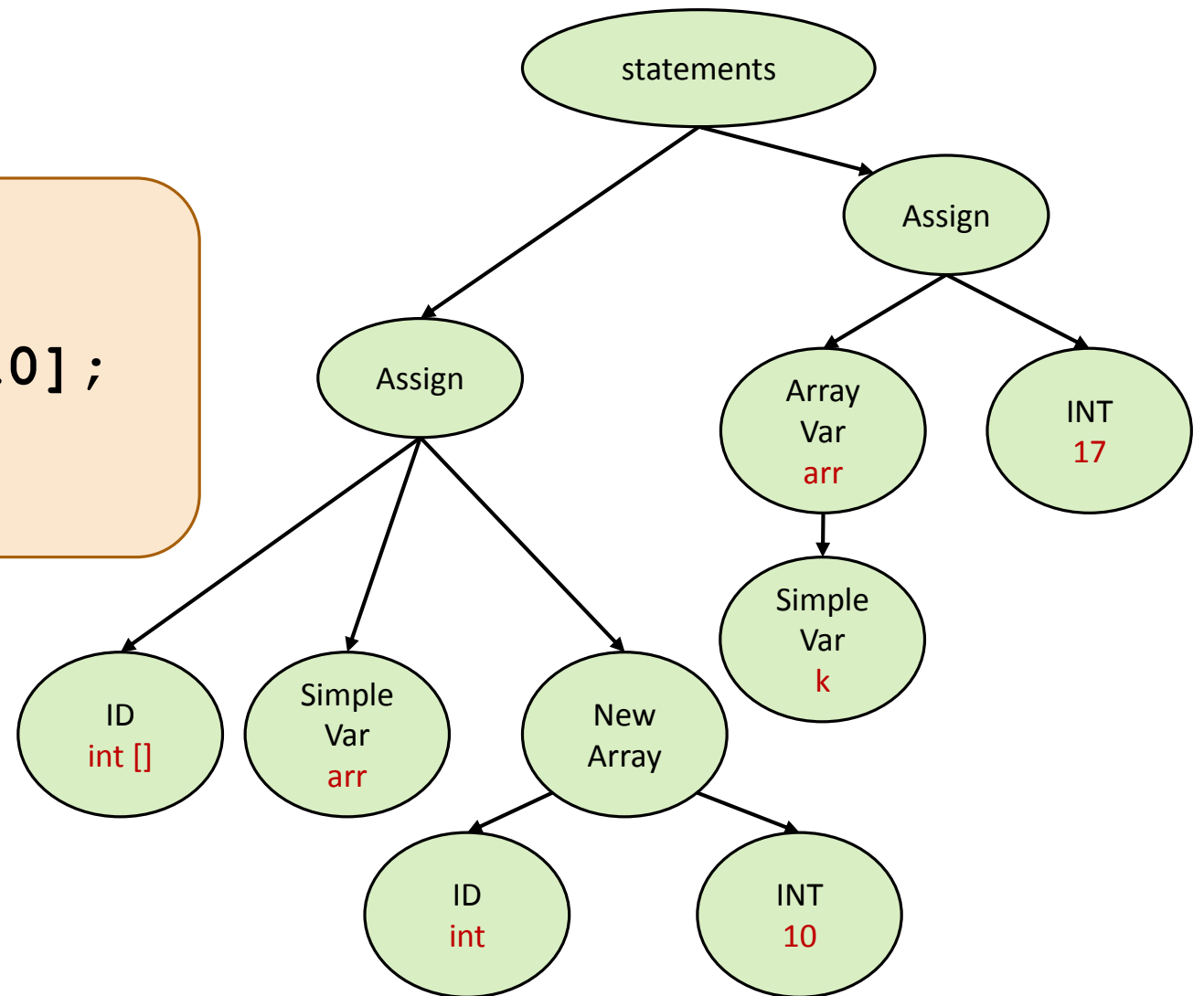
```
void foo(void) {  
    int[] arr = new int["z"];  
}
```

Invalid



Arrays

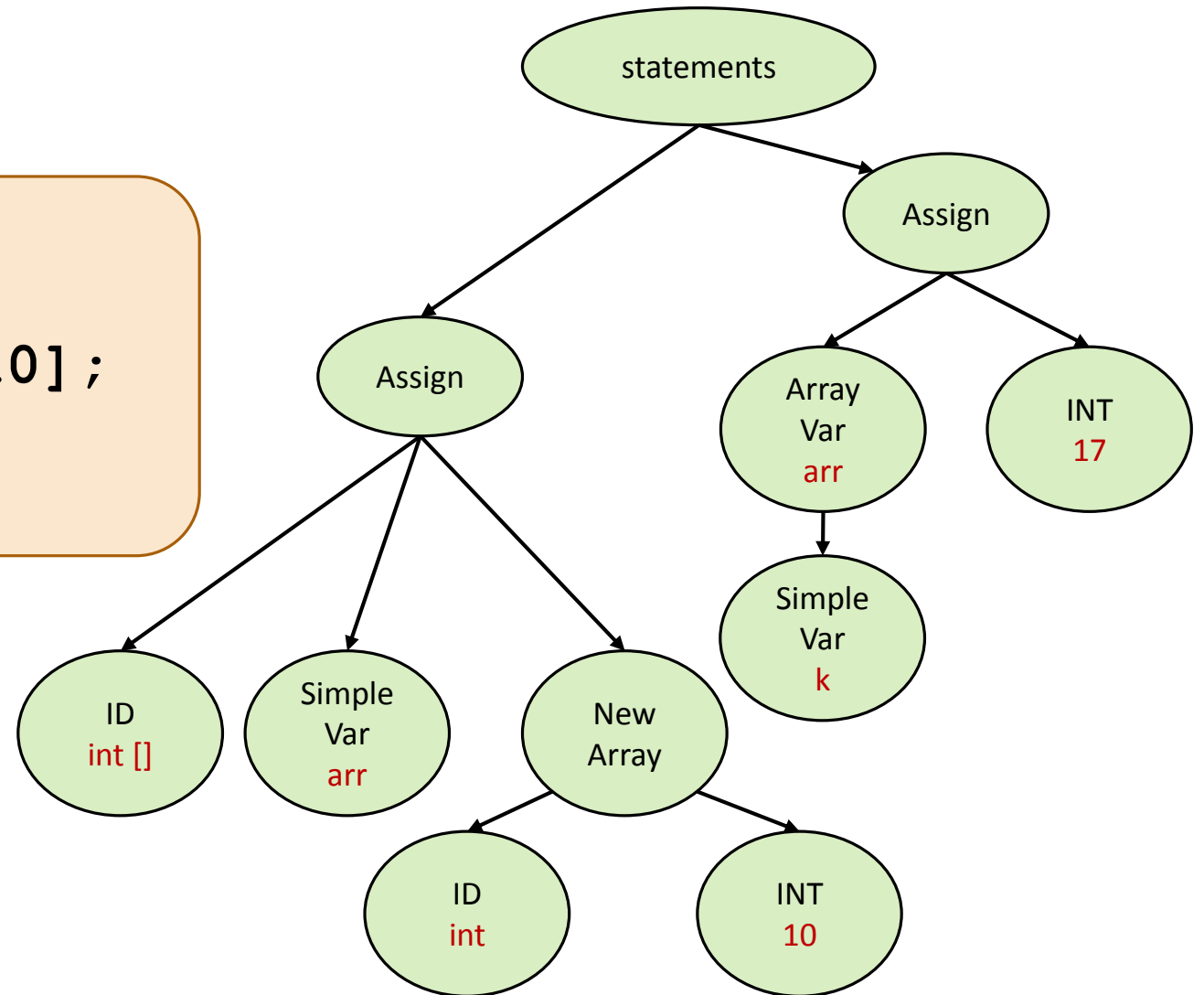
```
void foo(int d) {  
    int k = 3;  
    int[] arr = new int[10];  
    arr[k] = 17;  
}
```



Arrays

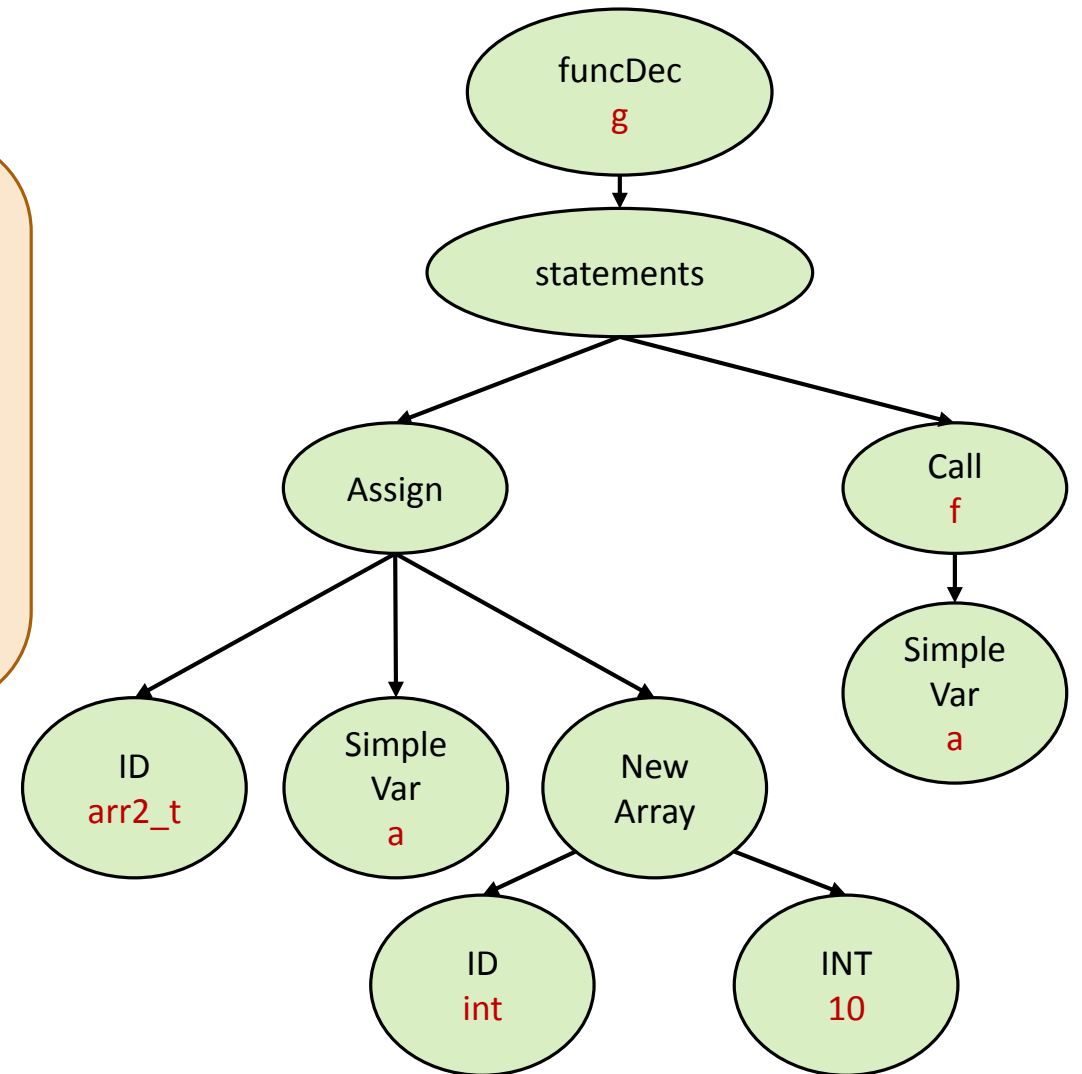
```
void foo(int d) {  
    int k = 3;  
    int[] arr = new int[10];  
    arr[k] = 17;  
}
```

Valid



Arrays

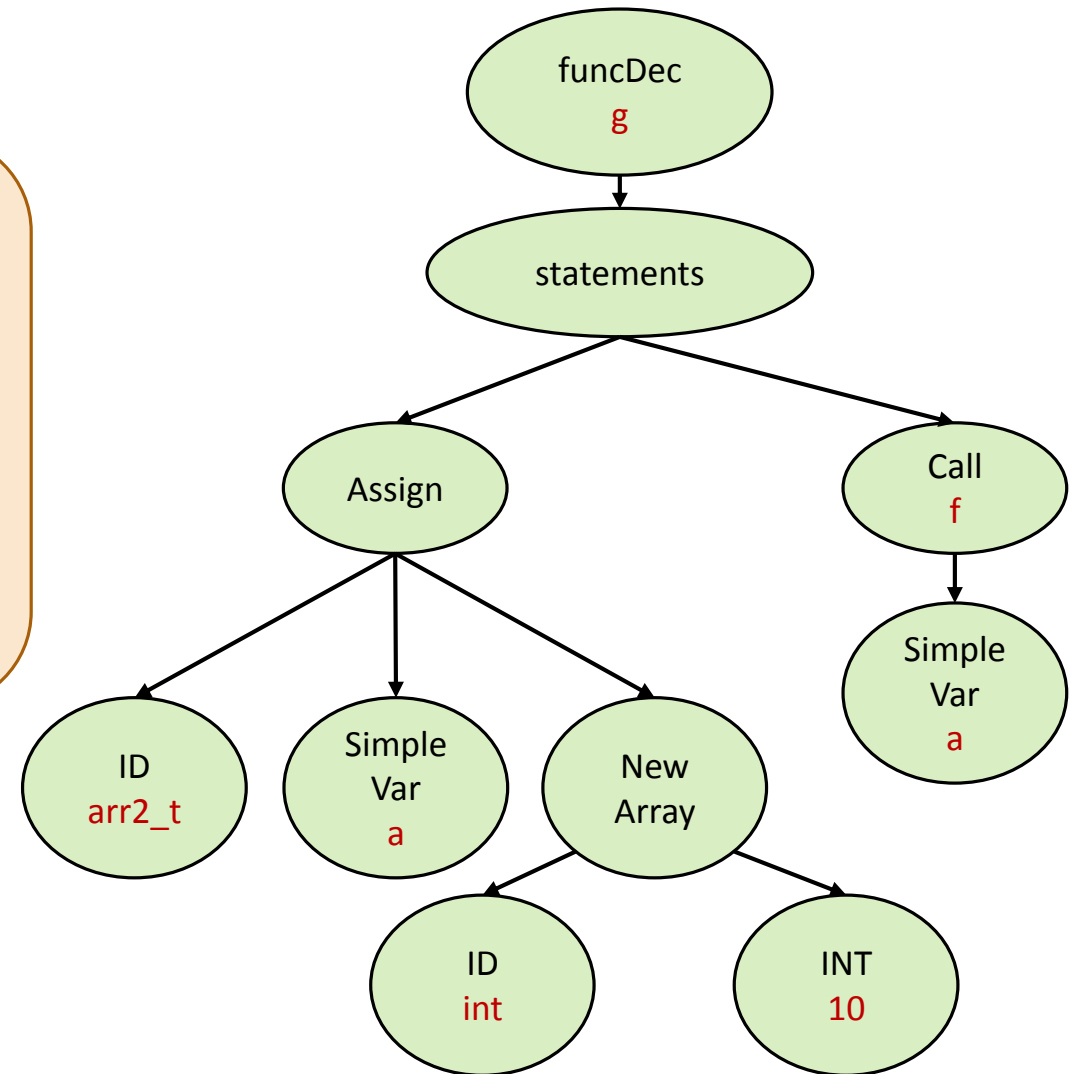
```
typedef int arr1_t[];  
typedef int arr2_t[];  
void f(arr1_t a) { }  
void g() {  
    arr2_t a = new int[10];  
    f(a);  
}
```



Arrays

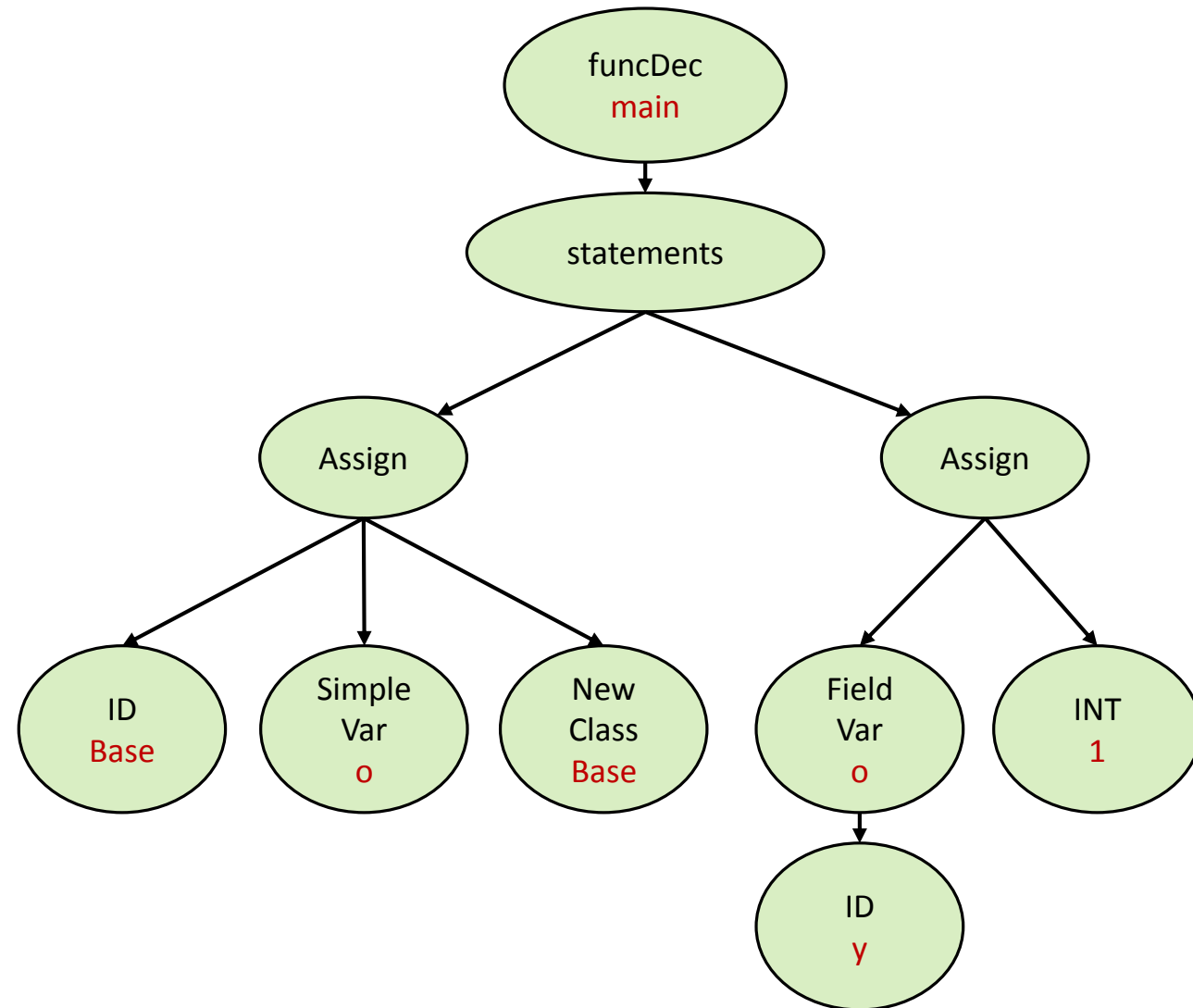
```
typedef int arr1_t[];  
typedef int arr2_t[];  
void f(arr1_t a) { }  
void g() {  
    arr2_t a = new int[10];  
    f(a);  
}
```

Invalid



Classes

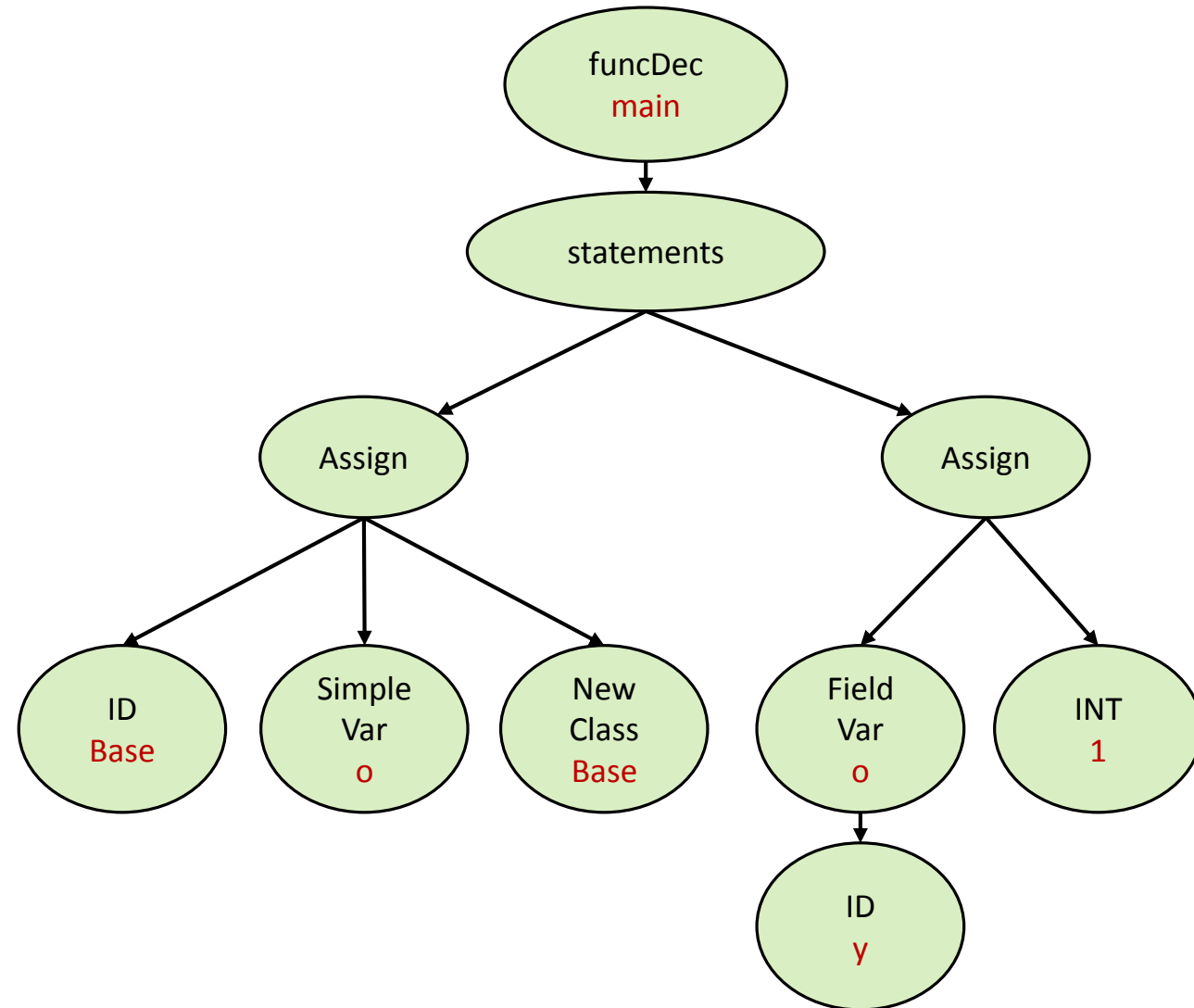
```
class Base {  
    int x;  
}  
void main() {  
    Base o = new Base;  
    o.y = 1;  
}
```



Classes

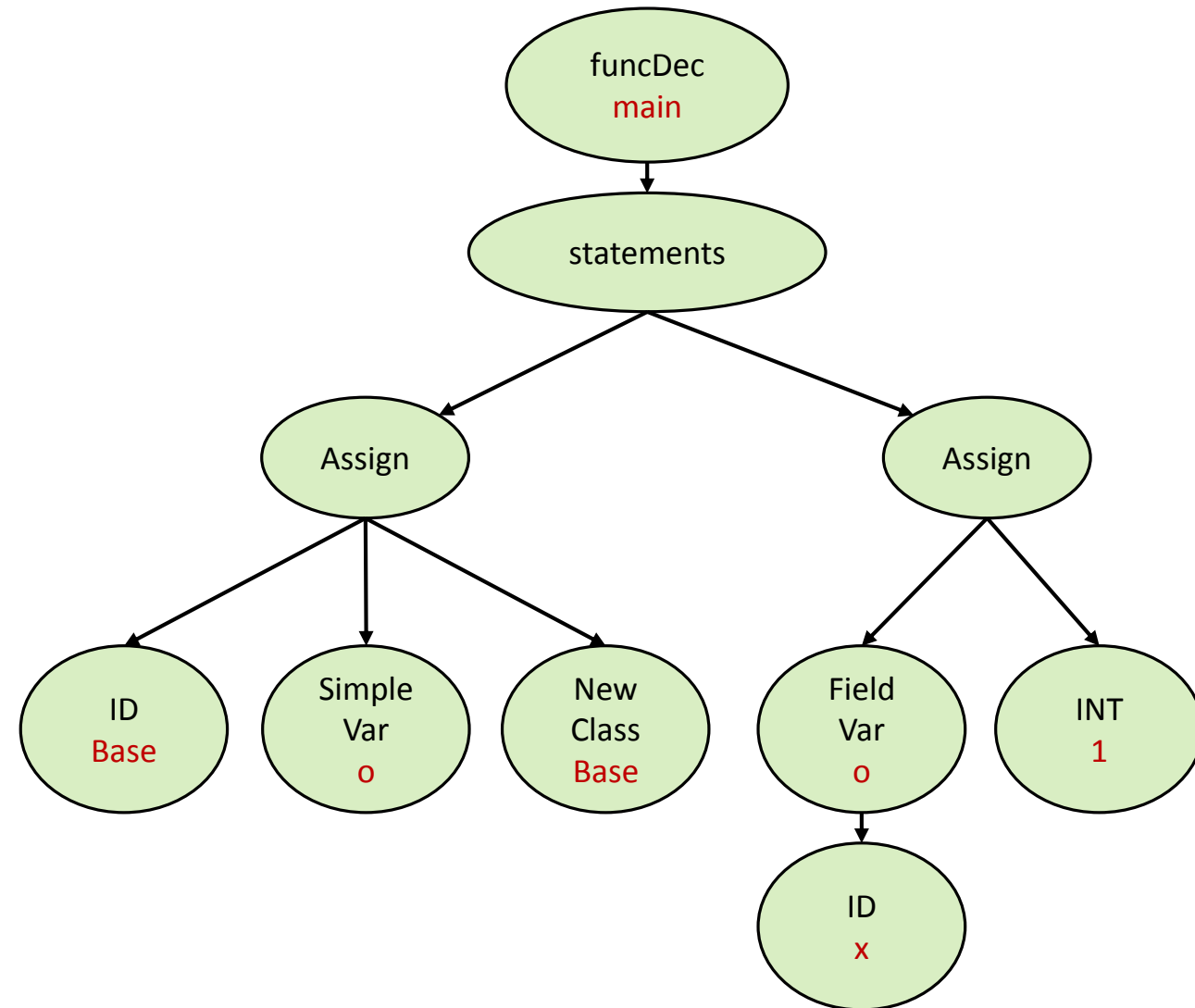
```
class Base {  
    int x;  
}  
void main() {  
    Base o = new Base;  
    o.y = 1;  
}
```

Invalid



Classes

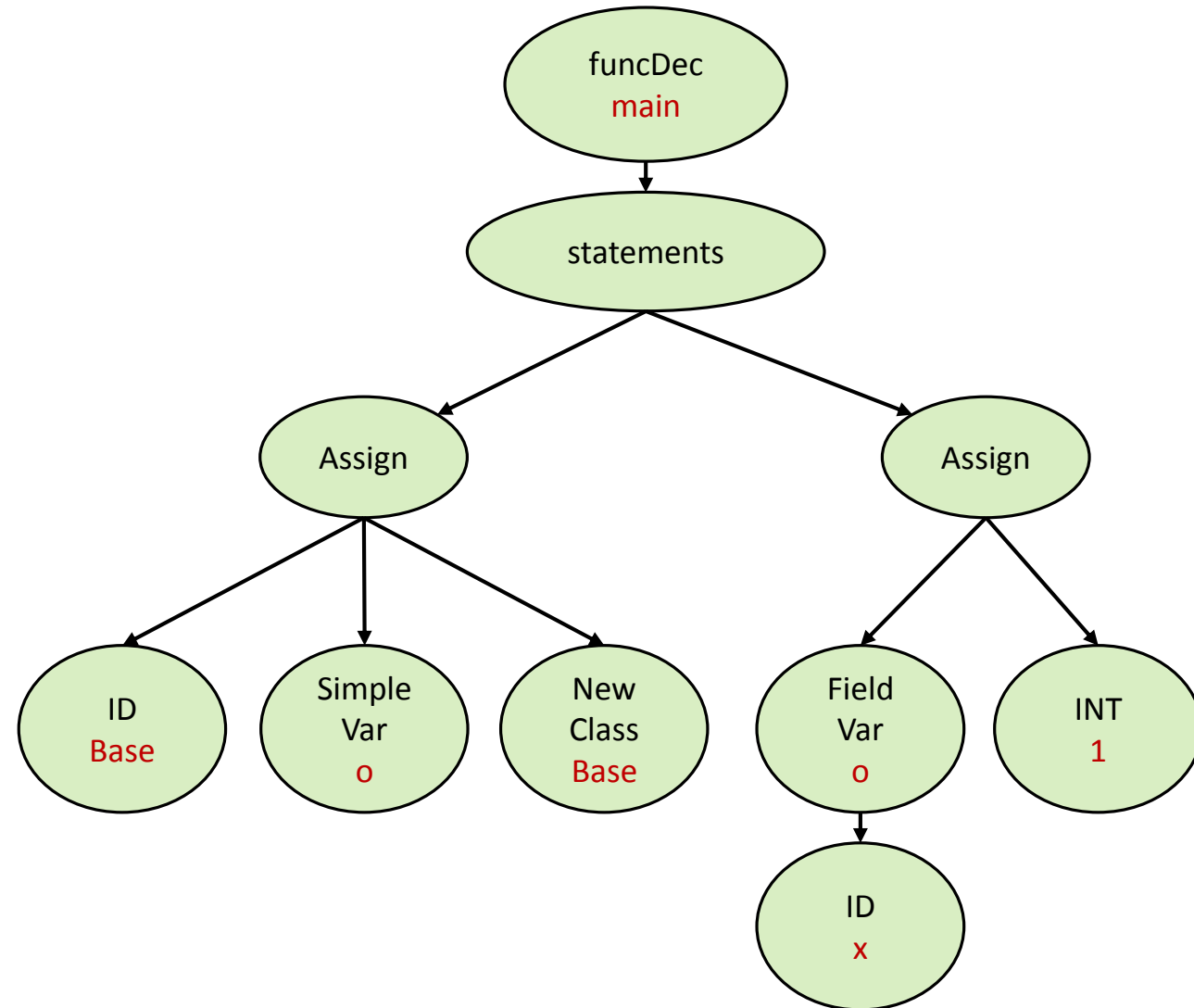
```
class Base {  
    int x;  
}  
void main() {  
    Base o = new Base;  
    o.x = 1;  
}
```



Classes

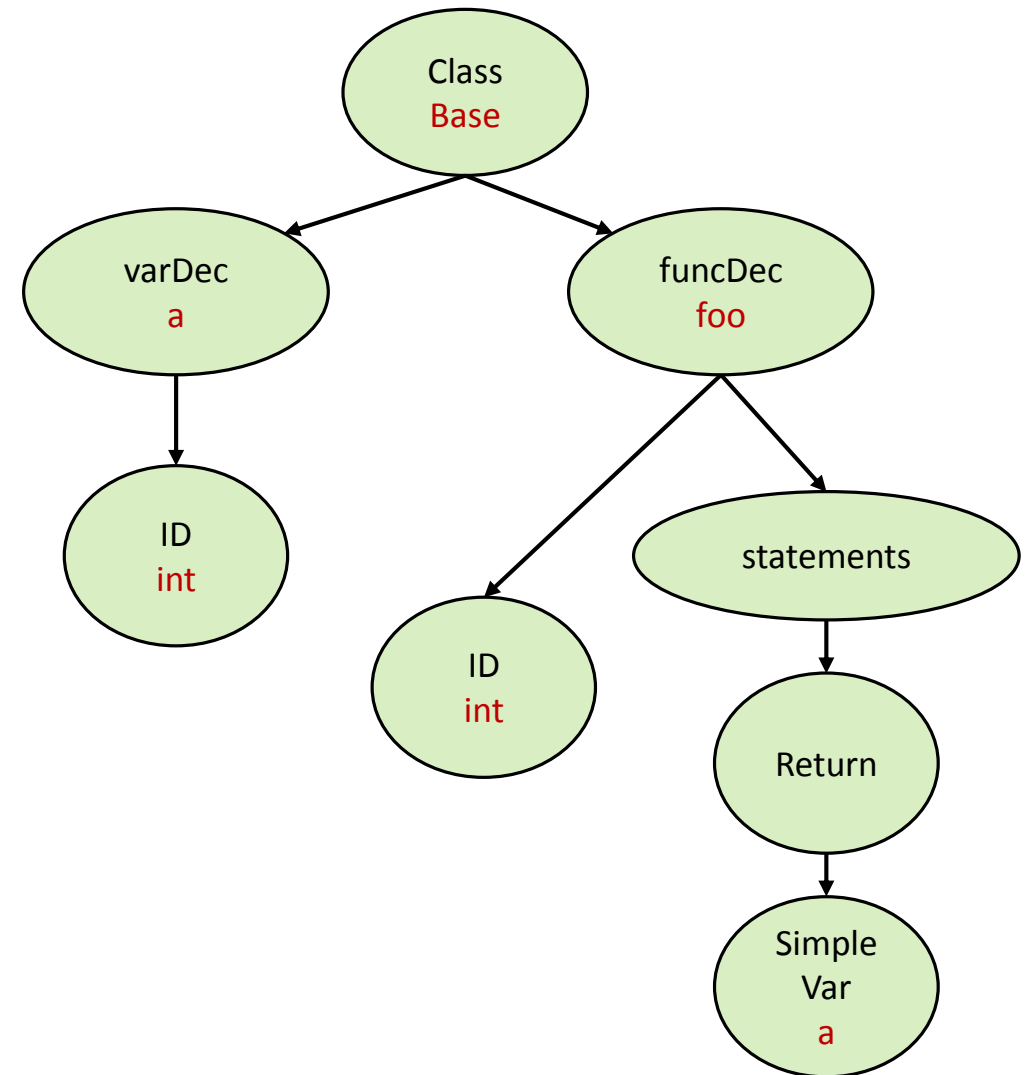
```
class Base {  
    int x;  
}  
void main() {  
    Base o = new Base;  
    o.x = 1;  
}
```

Valid



Classes

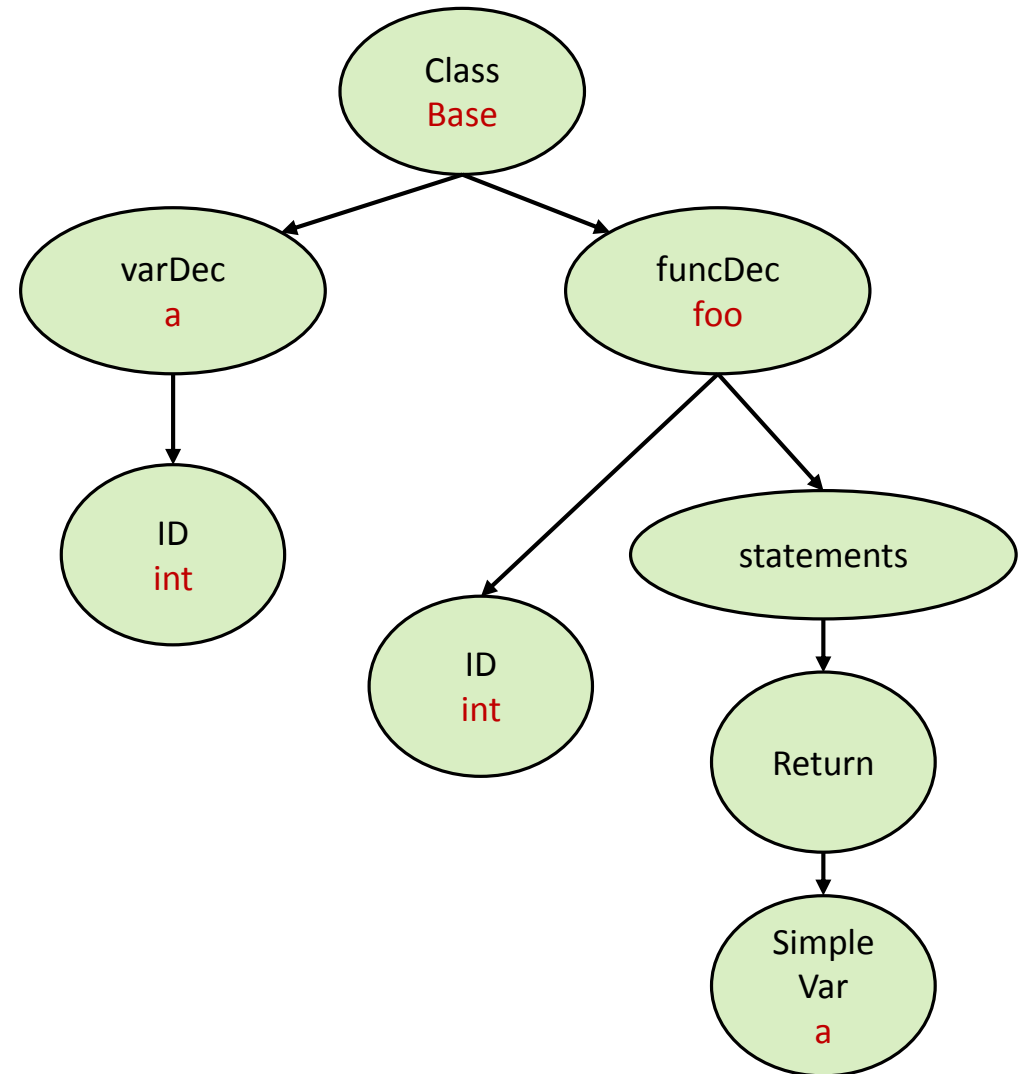
```
class Base {  
  int a;  
  int foo() {  
    return a;  
  }  
}
```



Classes

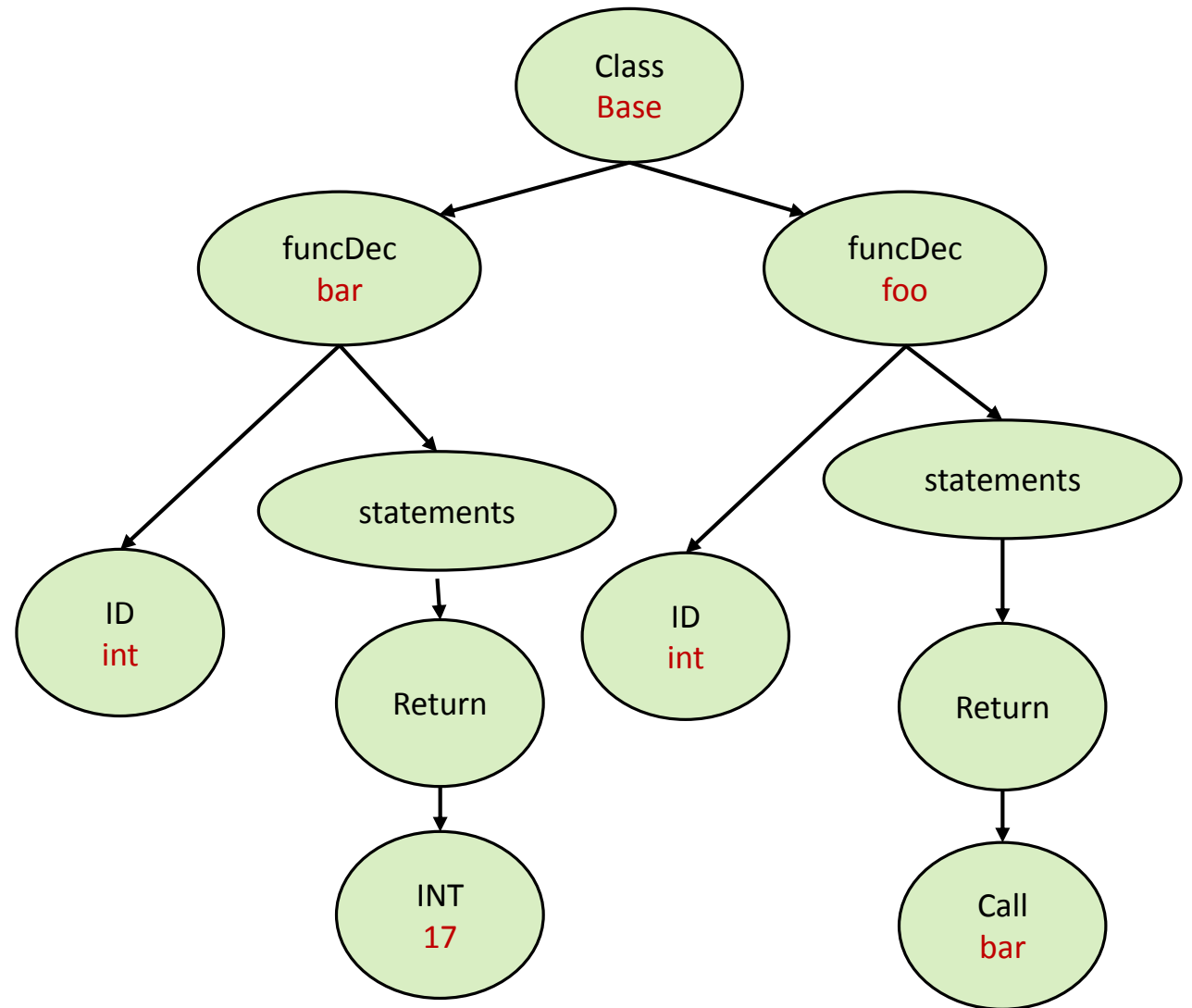
```
class Base {  
  int a;  
  int foo() {  
    return a;  
  }  
}
```

Valid



Classes

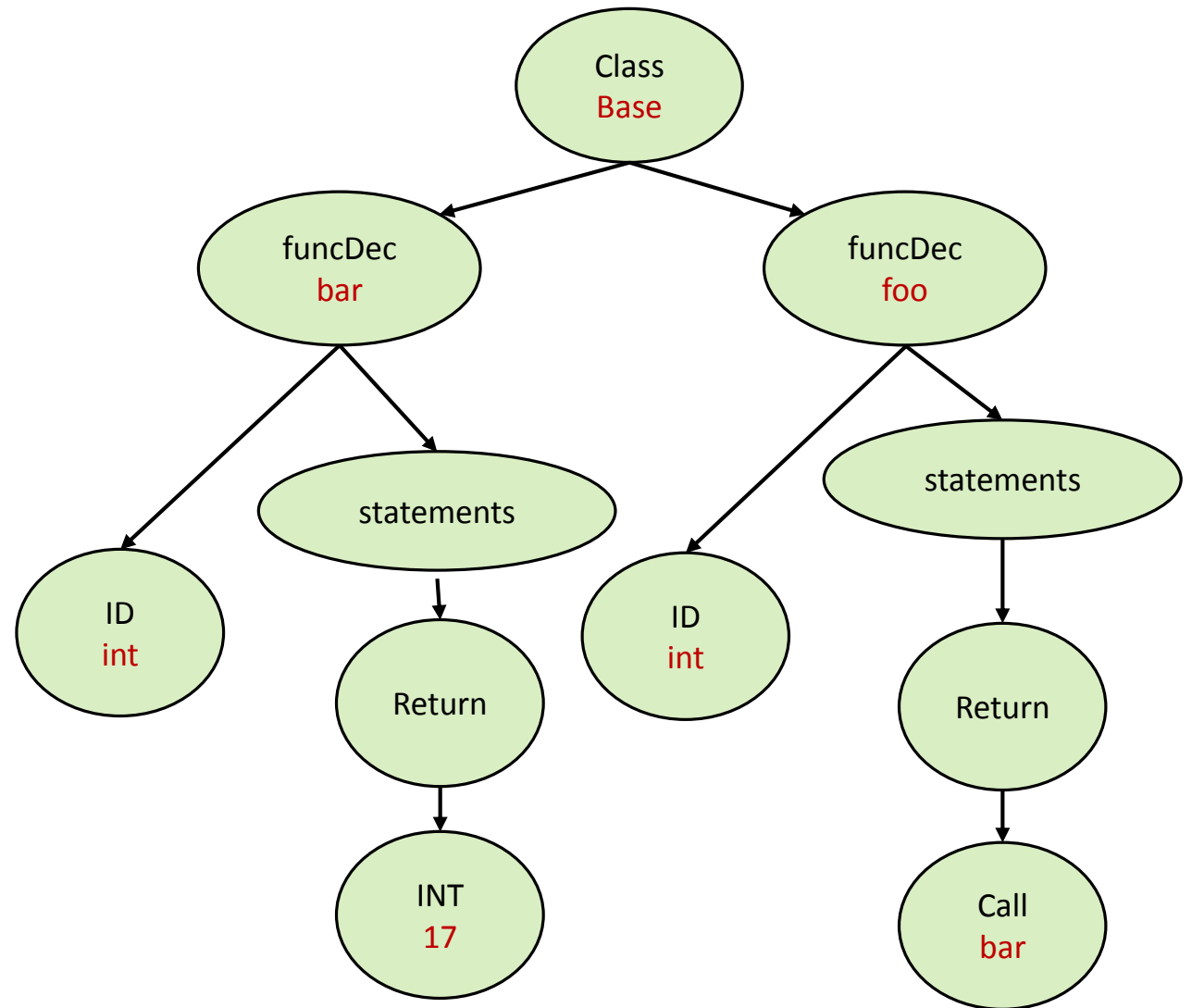
```
class Base {  
  int bar() {  
    return 17;  
  }  
  int foo() {  
    return bar();  
  }  
}
```



Classes

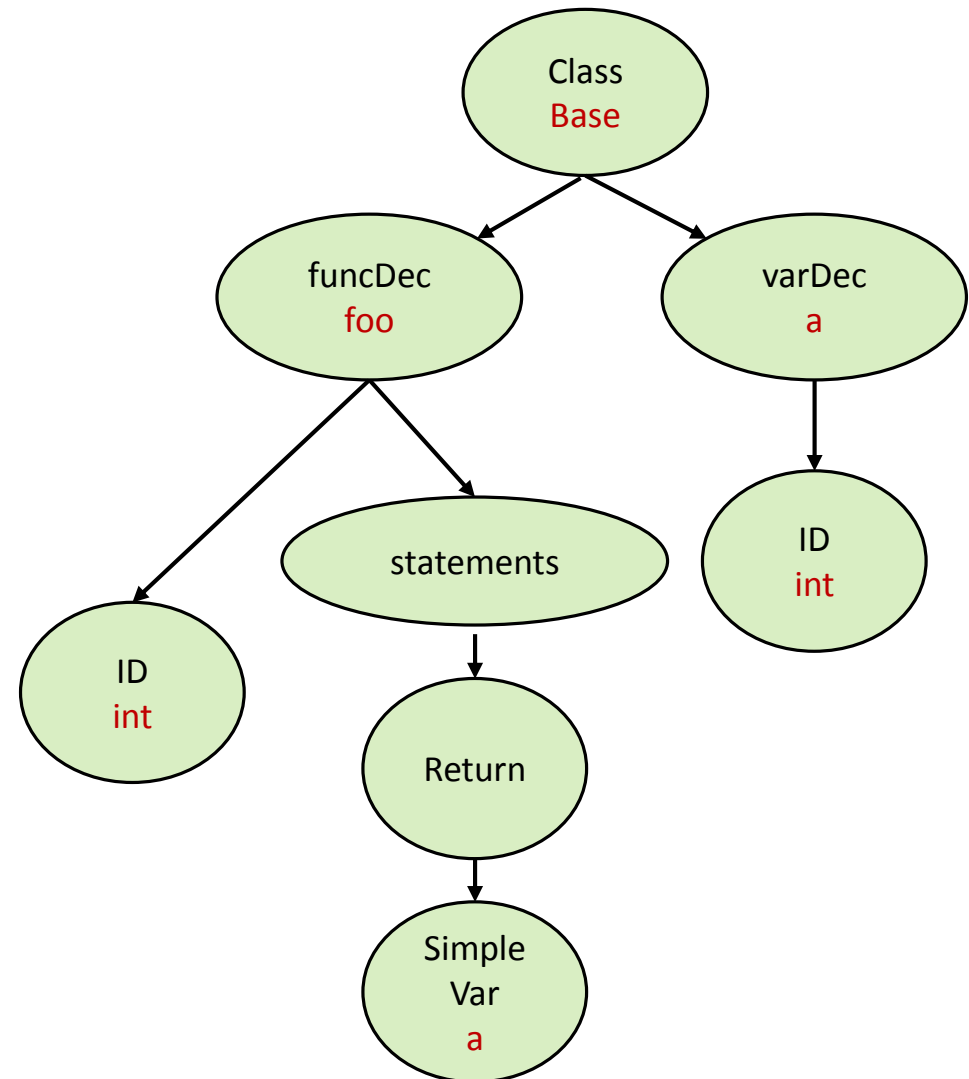
```
class Base {  
  int bar() {  
    return 17;  
  }  
  int foo() {  
    return bar();  
  }  
}
```

Valid



Classes

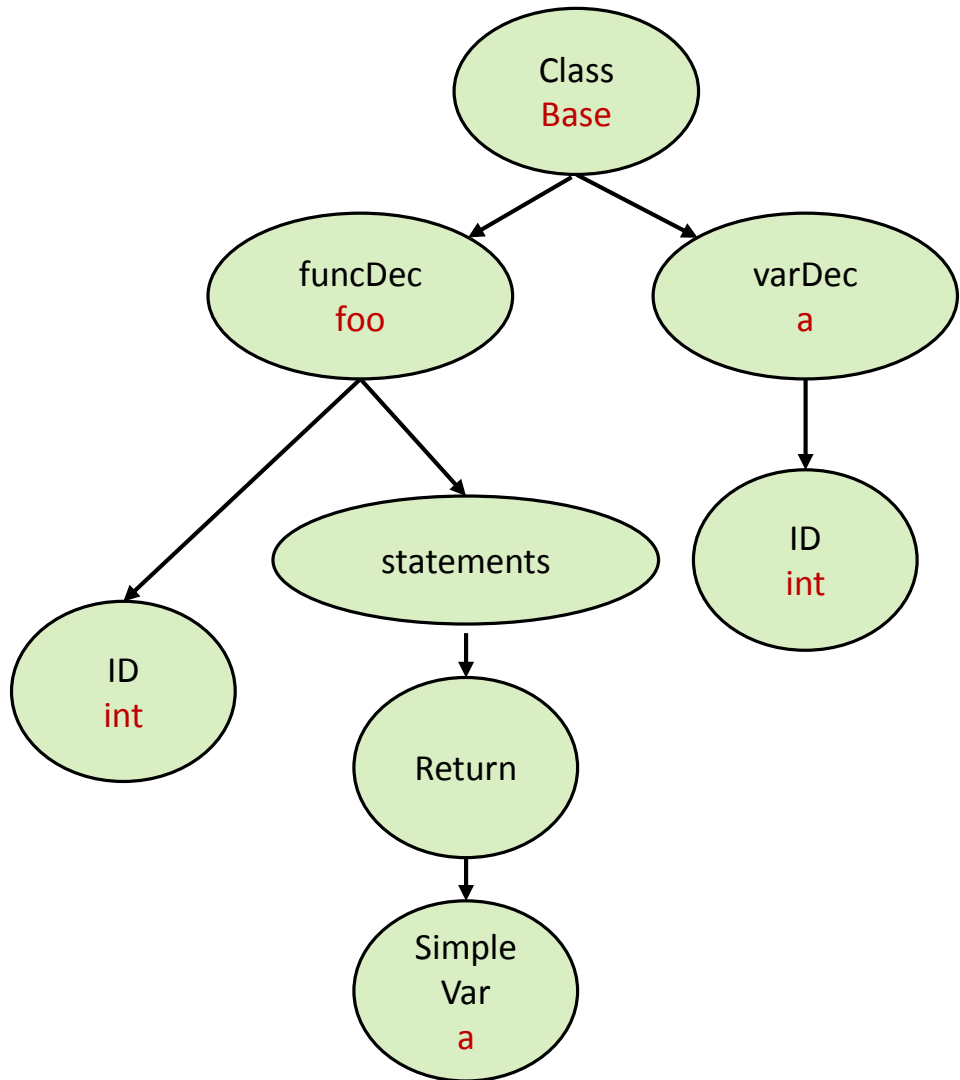
```
class Base {  
  int foo() {  
    return a;  
  }  
  int a;  
}
```



Classes

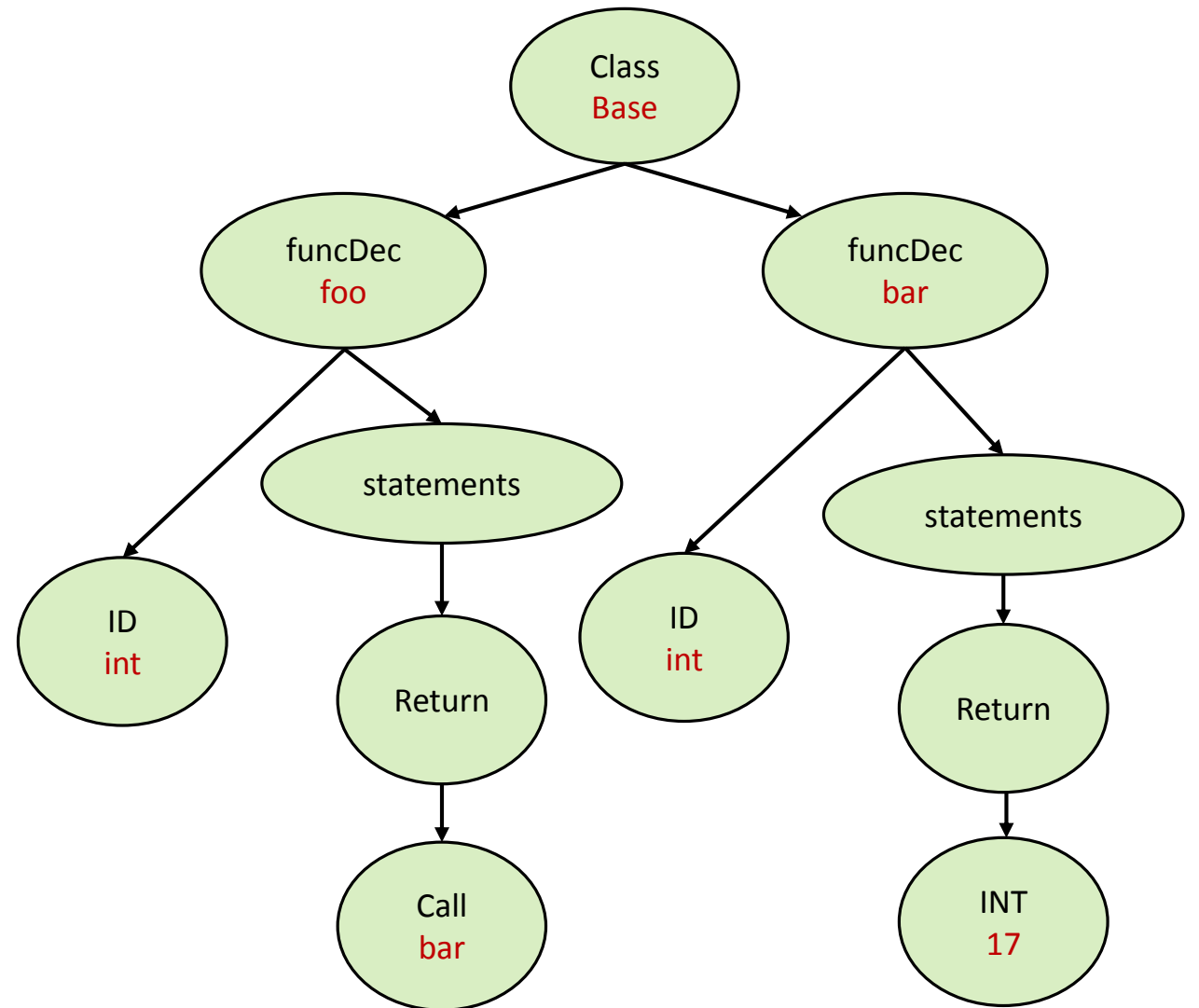
```
class Base {  
    int foo() {  
        return a;  
    }  
    int a;  
}
```

Invalid



Classes

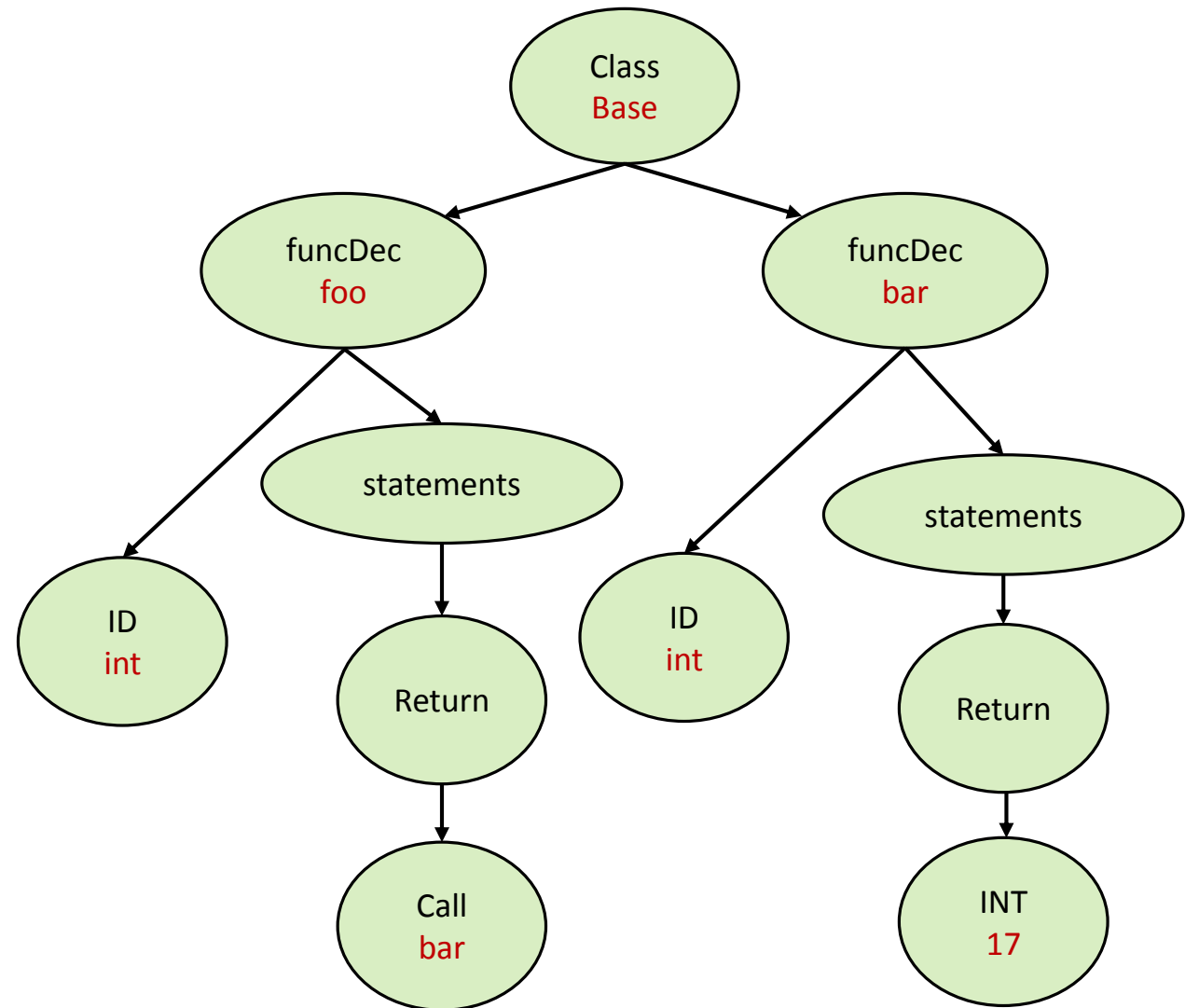
```
class Base {  
  int foo() {  
    return bar();  
  }  
  int bar() {  
    return 17;  
  }  
}
```



Classes

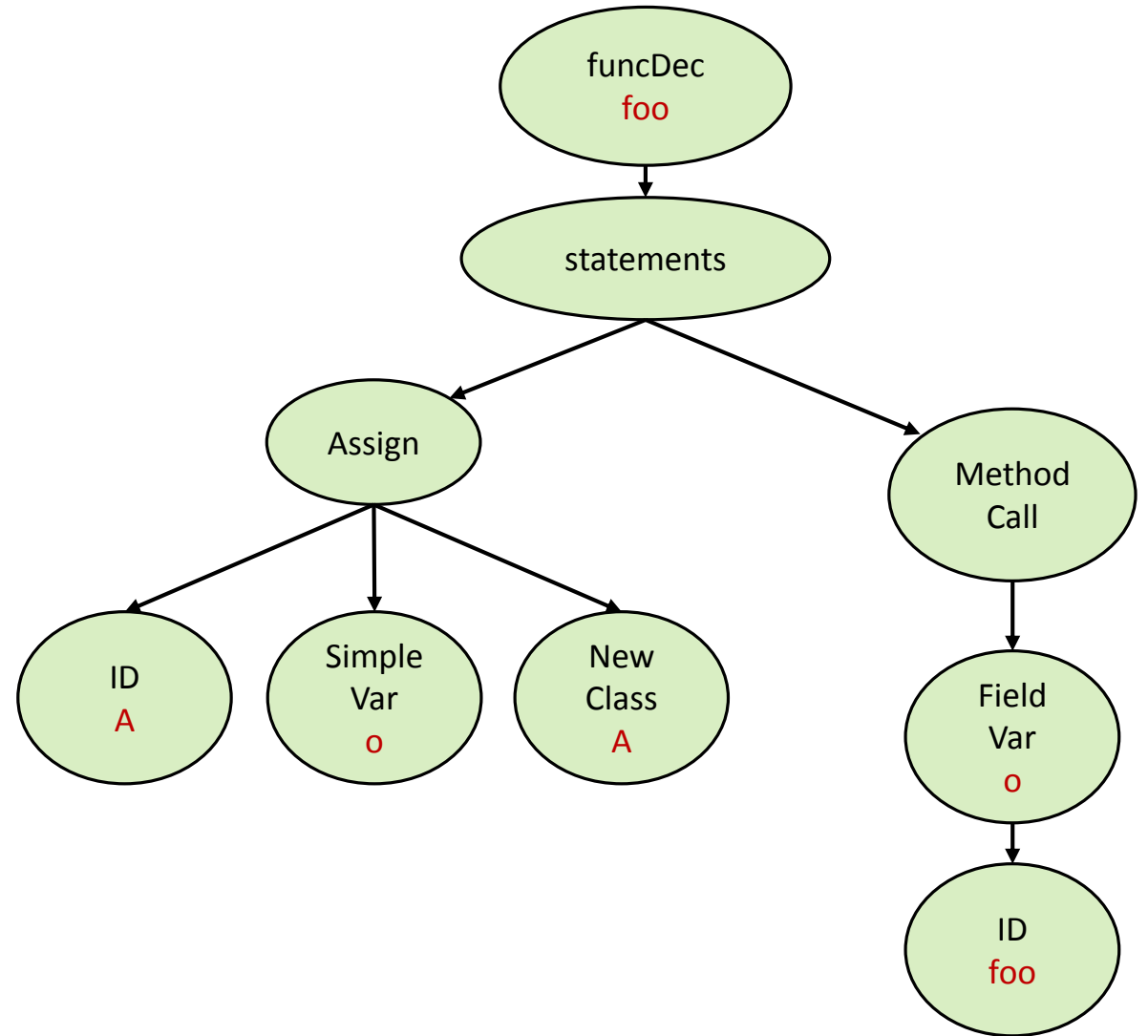
```
class Base {  
  int foo() {  
    return bar();  
  }  
  int bar() {  
    return 17;  
  }  
}
```

Invalid



Classes

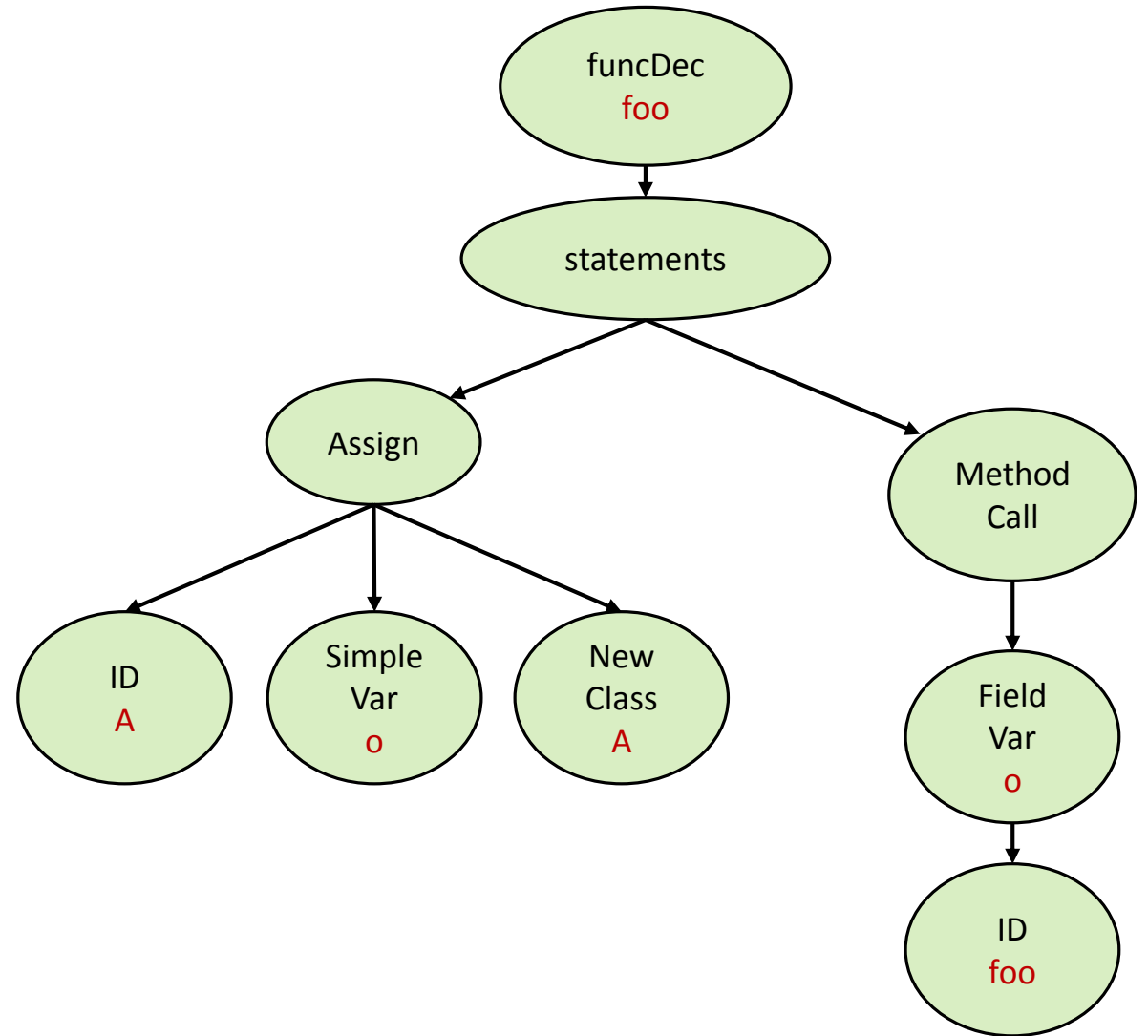
```
class A {  
  void foo() {  
    A o = new A;  
    o.foo();  
  }  
}
```



Classes

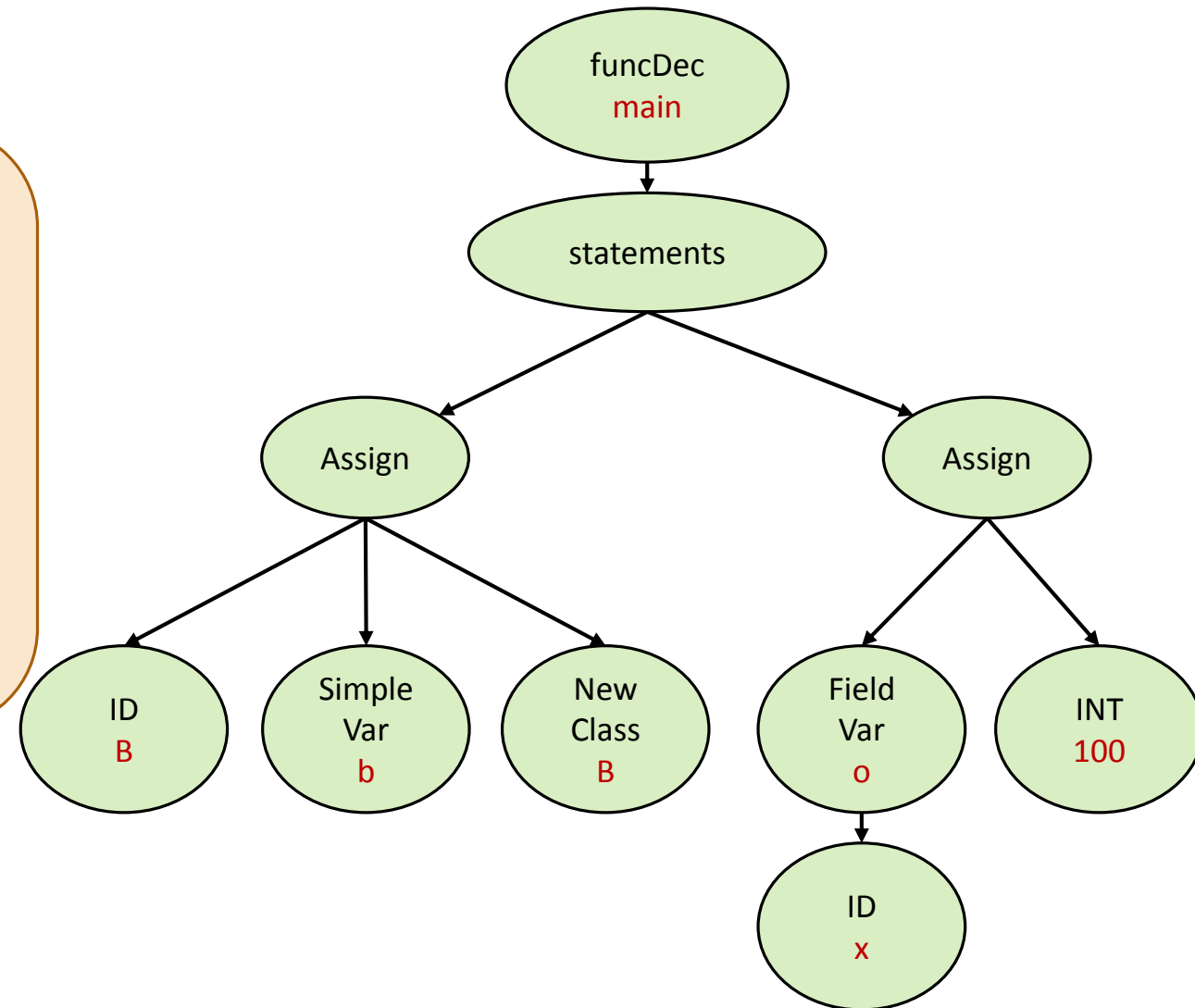
```
class A {  
  void foo() {  
    A o = new A;  
    o.foo();  
  }  
}
```

Valid



Inheritance

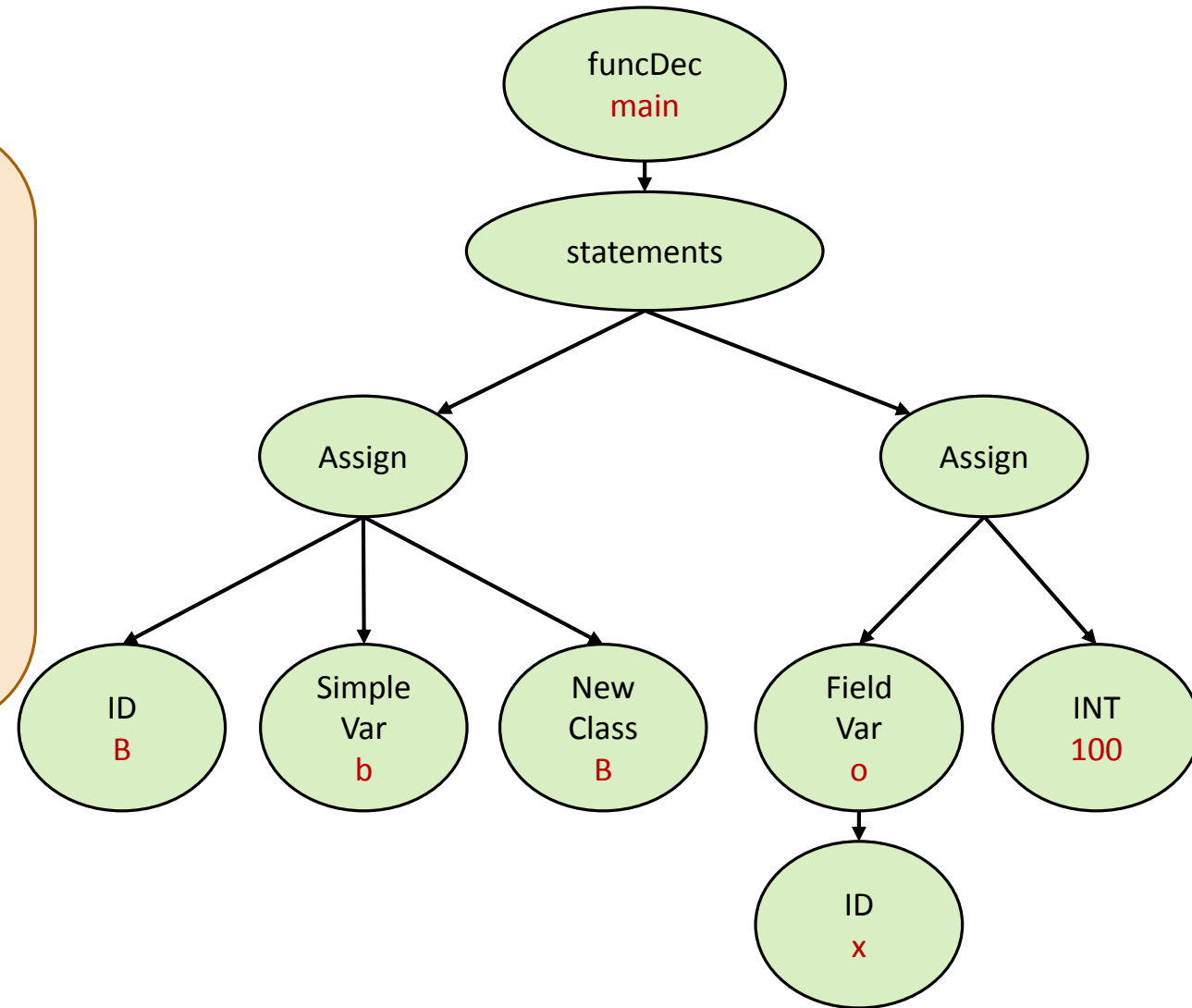
```
class A {  
    int x;  
}  
class B extends A {  
void main() {  
    B o = new B;  
    o.x = 100;  
}
```



Inheritance

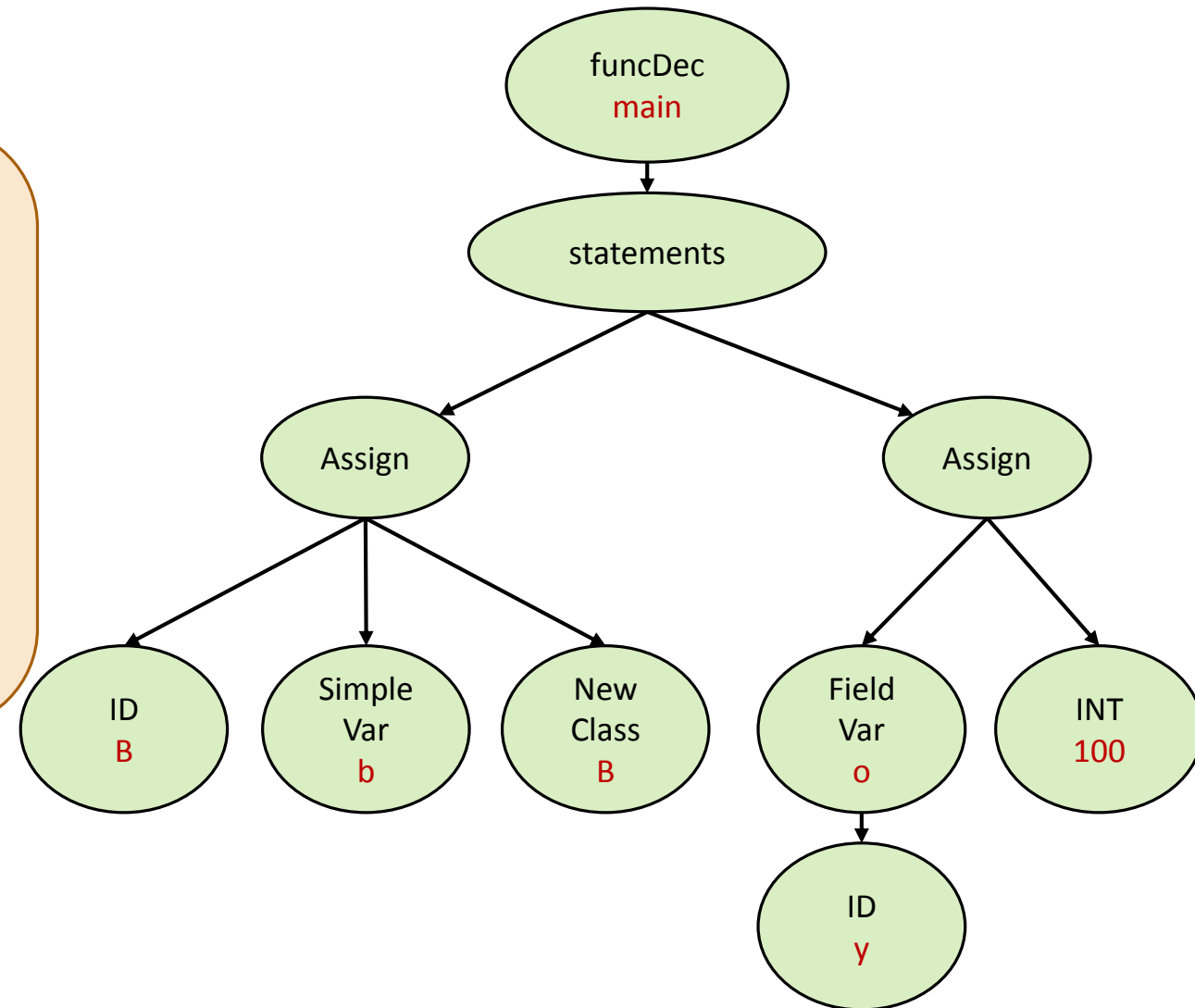
```
class A {  
    int x;  
}  
class B extends A {  
void main() {  
    B o = new B;  
    o.x = 100;  
}
```

Valid



Inheritance

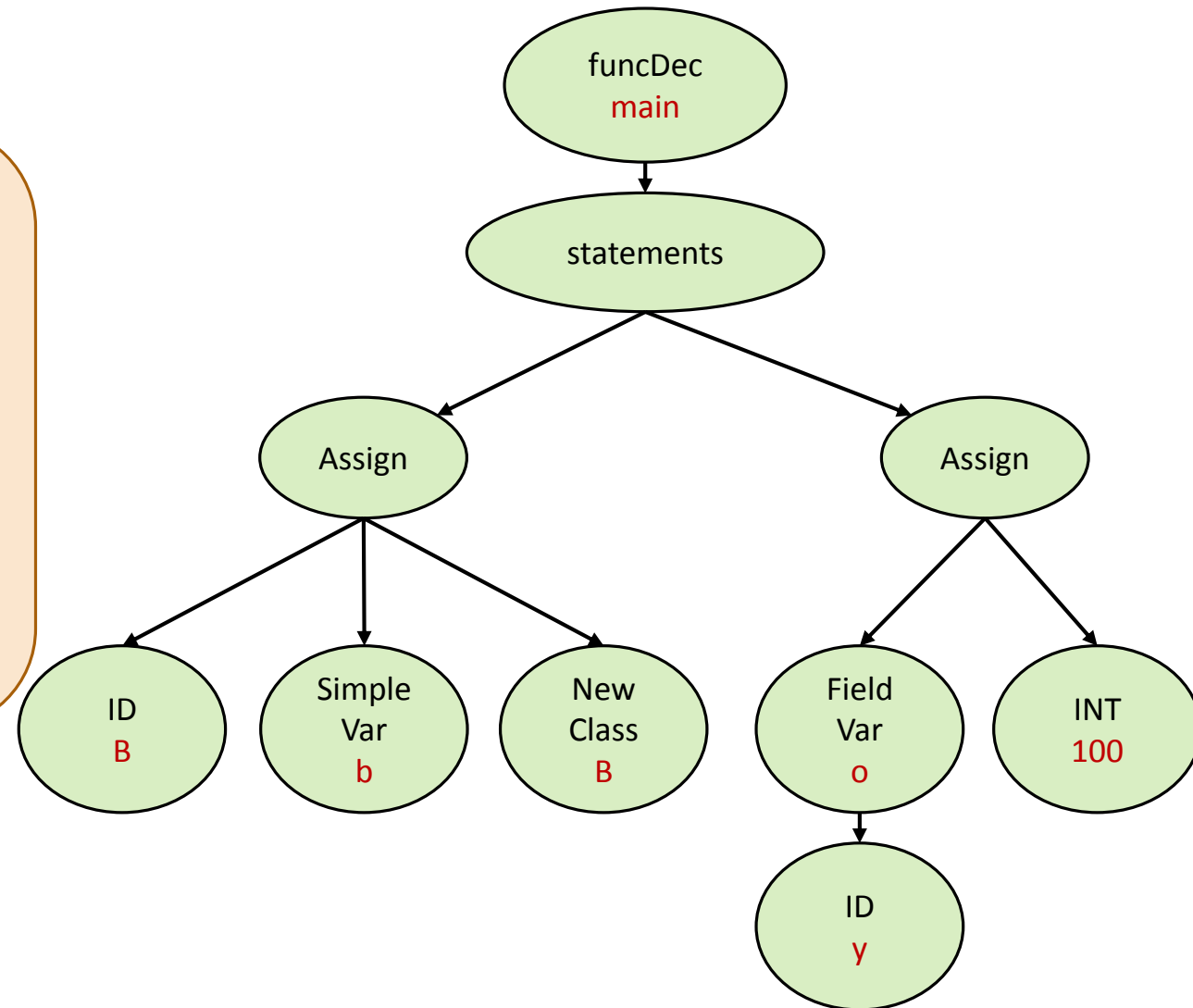
```
class A {  
    int x;  
}  
class B extends A {  
void main() {  
    B o = new B;  
    o.y = 100;  
}
```



Inheritance

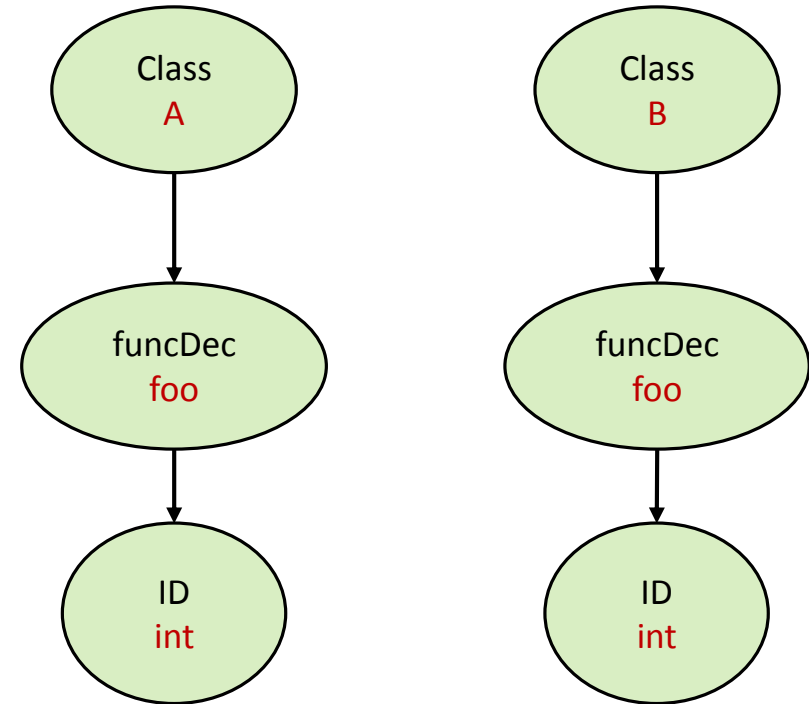
```
class A {  
    int x;  
}  
class B extends A {  
void main() {  
    B o = new B;  
    o.y = 100;  
}
```

Invalid



Inheritance

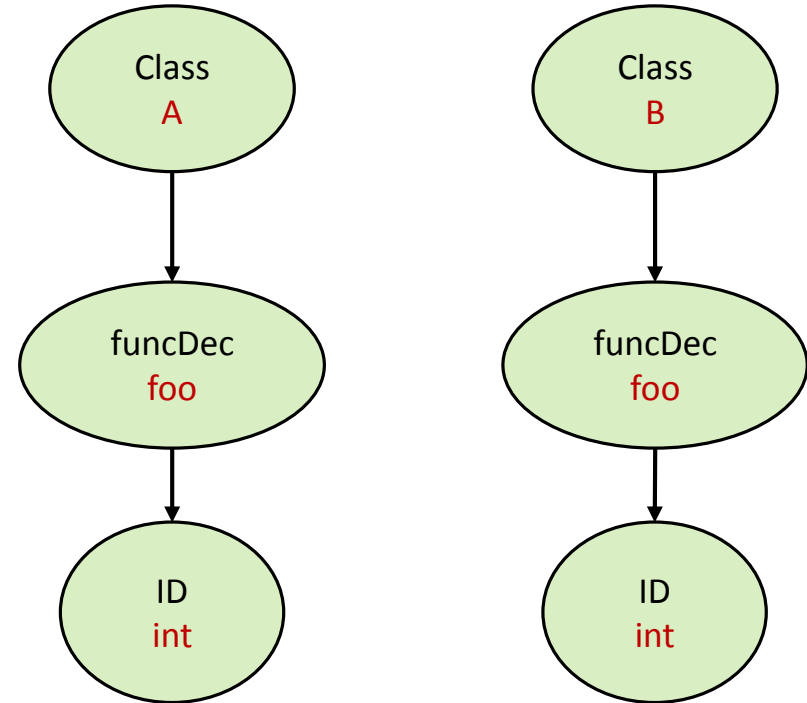
```
class A {  
    int foo() {  
        return 17;  
    }  
}  
class B extends A {  
    int foo() {  
        return 18;  
    }  
}
```



Inheritance

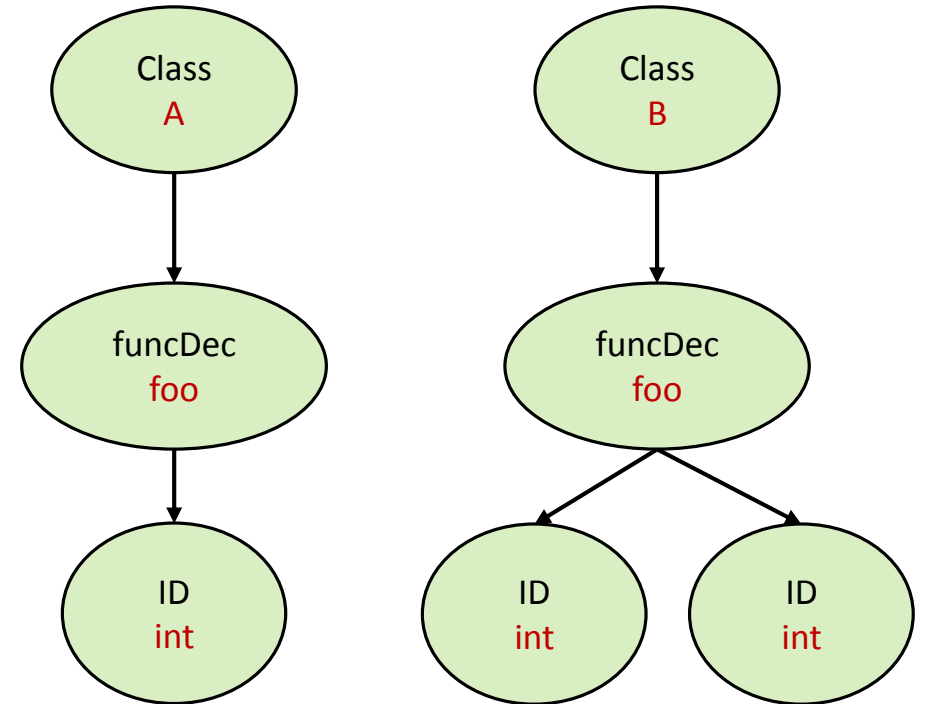
```
class A {  
    int foo() {  
        return 17;  
    }  
}  
class B extends A {  
    int foo() {  
        return 18;  
    }  
}
```

Valid



Inheritance

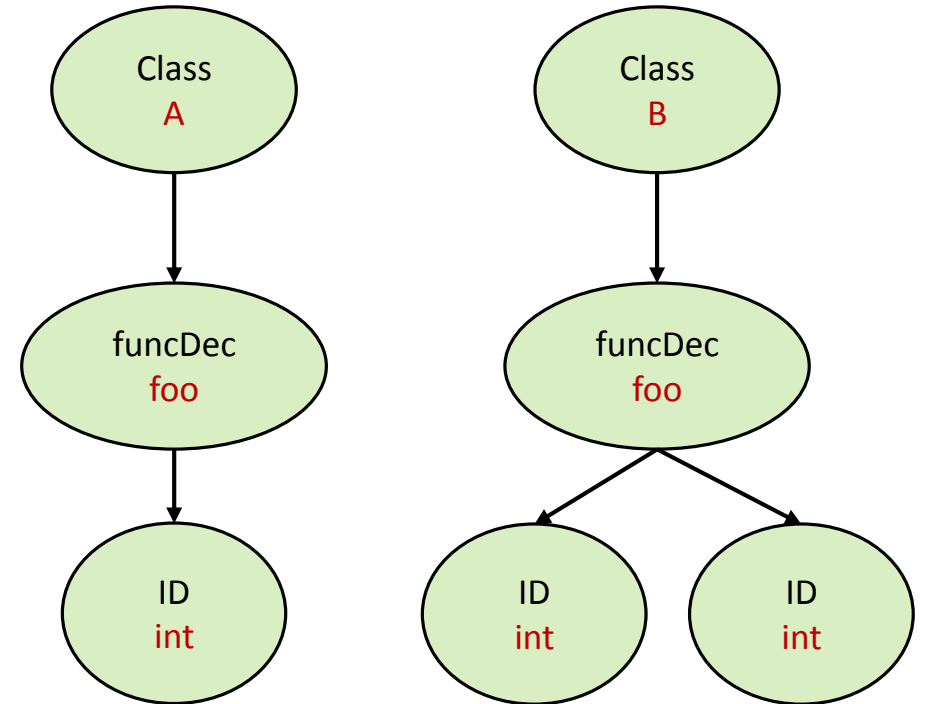
```
class A {  
    int foo() {  
        return 17;  
    }  
}  
class B extends A {  
    int foo(int x) {  
        return x + 1;  
    }  
}
```



Inheritance

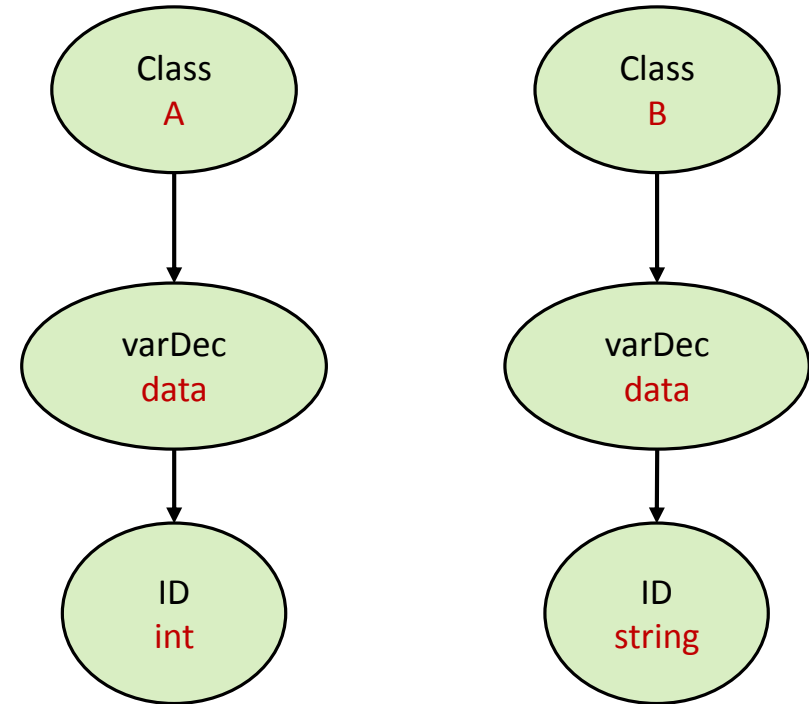
```
class A {  
    int foo() {  
        return 17;  
    }  
}  
class B extends A {  
    int foo(int x) {  
        return x + 1;  
    }  
}
```

Invalid



Inheritance

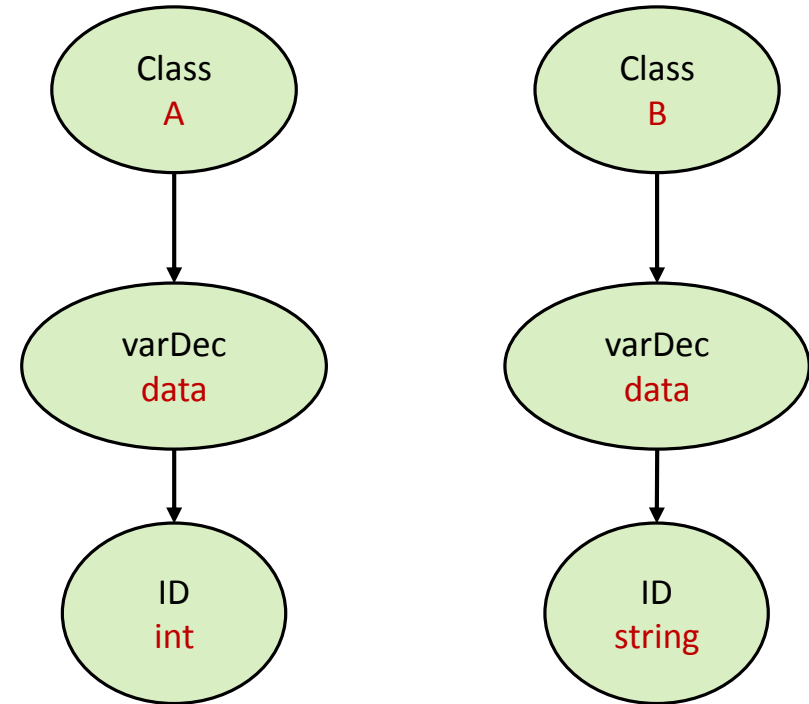
```
class A {  
    int data;  
}  
class B extends A {  
    string data;  
}
```



Inheritance

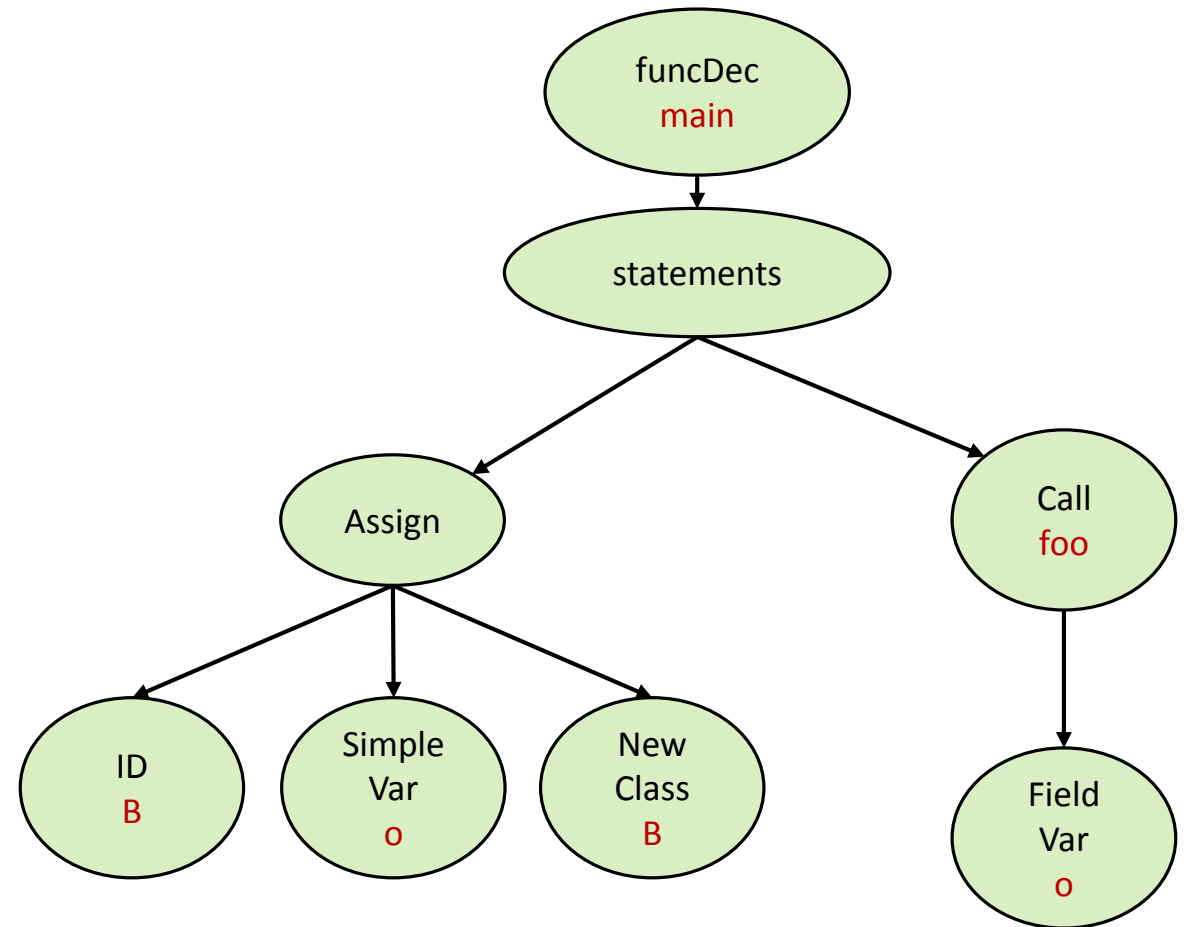
```
class A {  
    int data;  
}  
class B extends A {  
    string data;  
}
```

Invalid



Inheritance

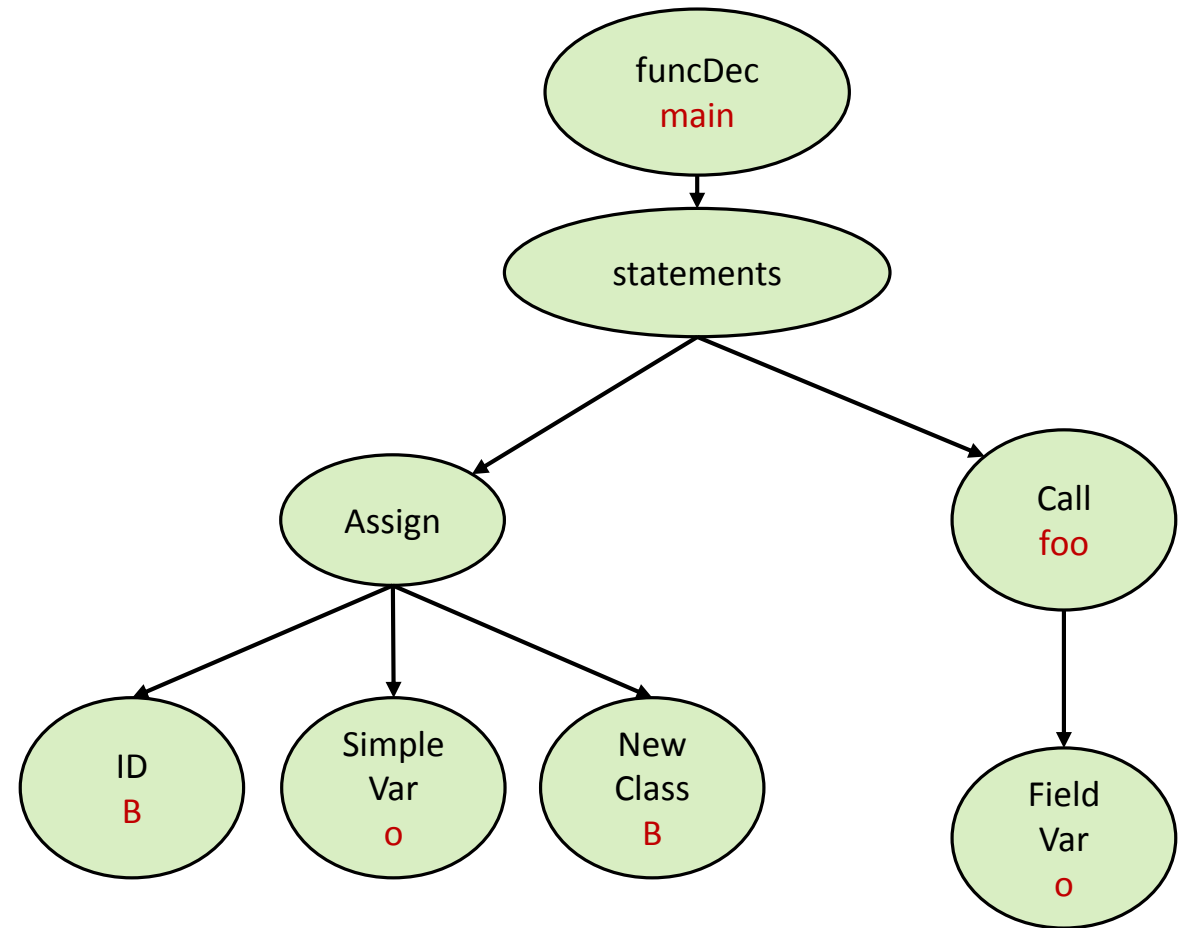
```
class A { }  
class B extends A { }  
void foo(A a) { }  
void main() {  
    B o = new B;  
    foo(o);  
}
```



Inheritance

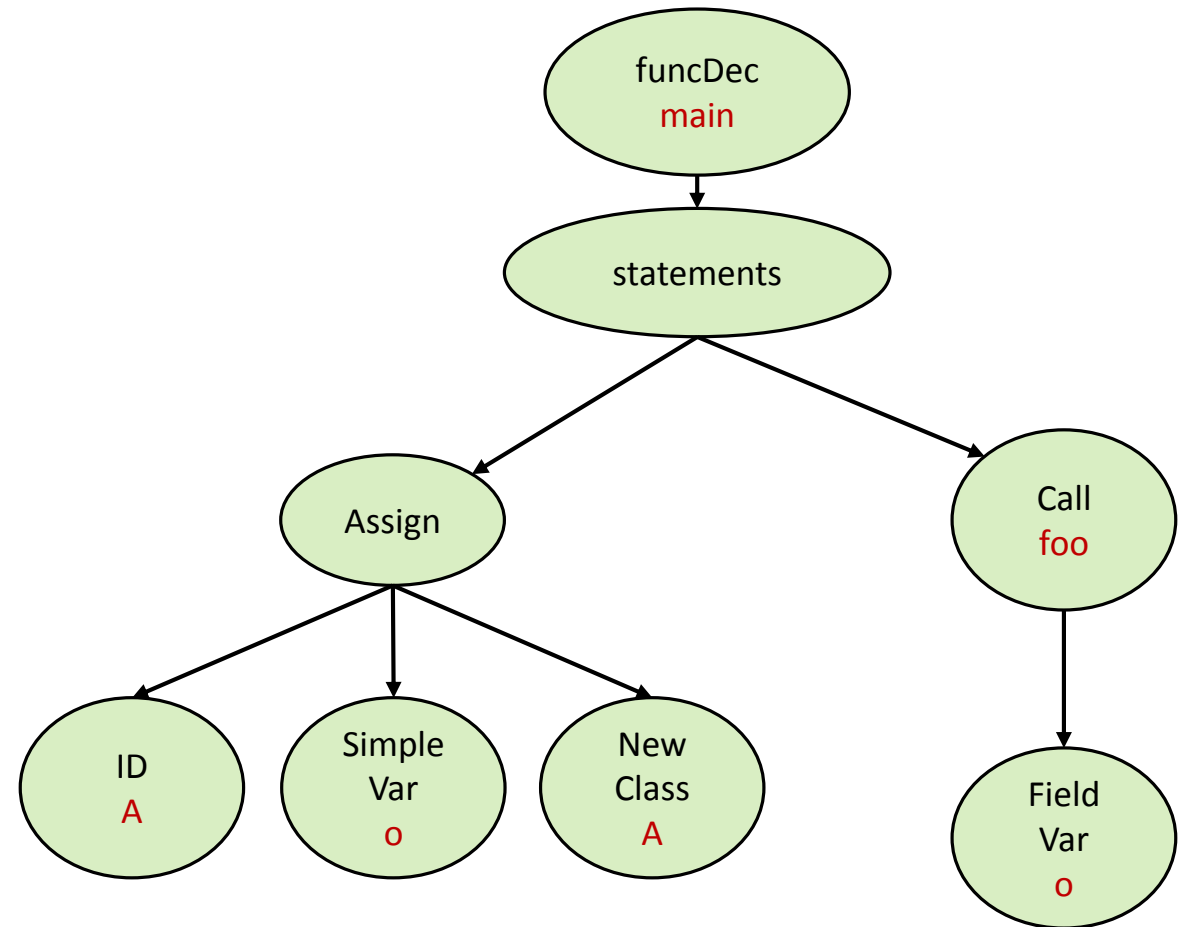
```
class A { }  
class B extends A { }  
void foo(A a) { }  
void main() {  
    B o = new B;  
    foo(o);  
}
```

Valid



Inheritance

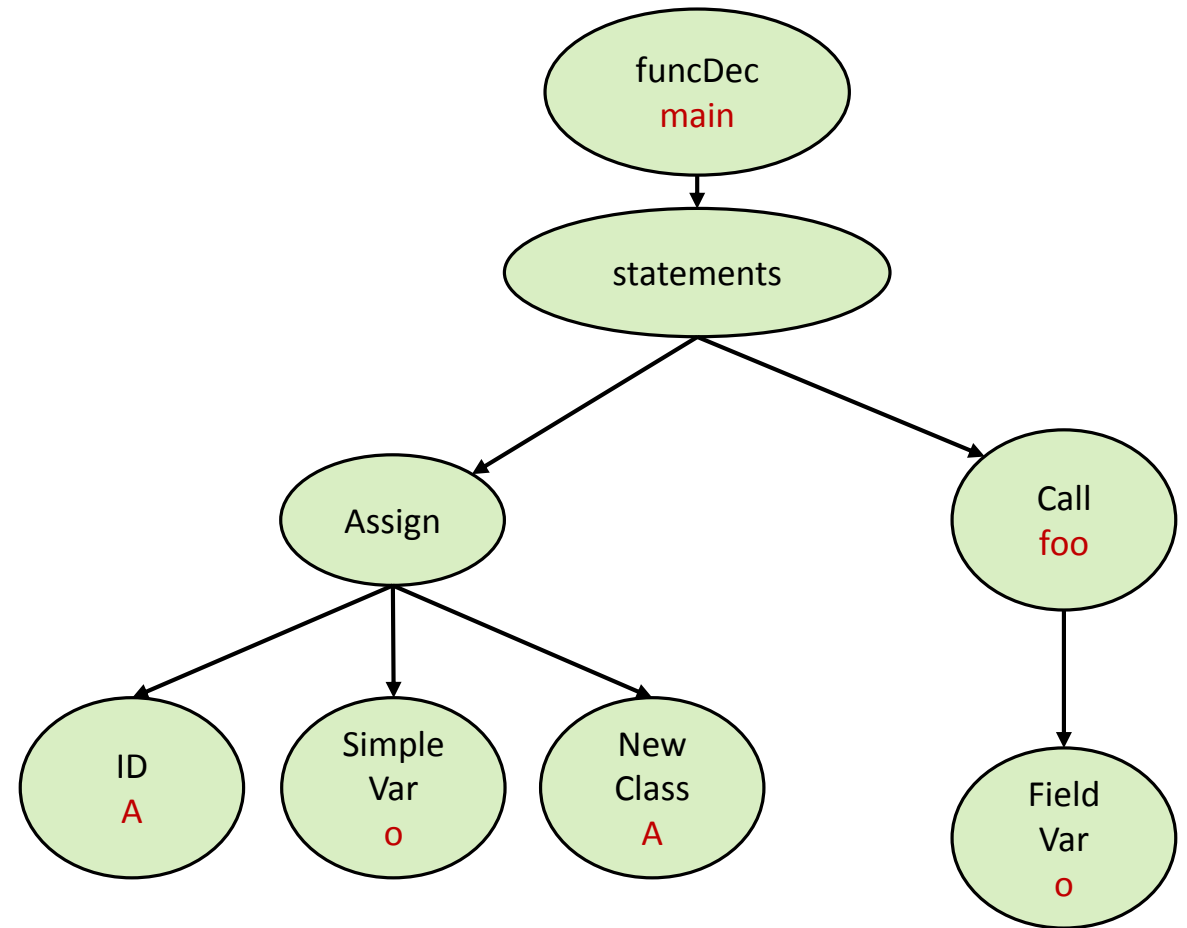
```
class A { }  
class B extends A { }  
void foo(B b) { }  
void main() {  
    A o = new A;  
    foo(o);  
}
```



Inheritance

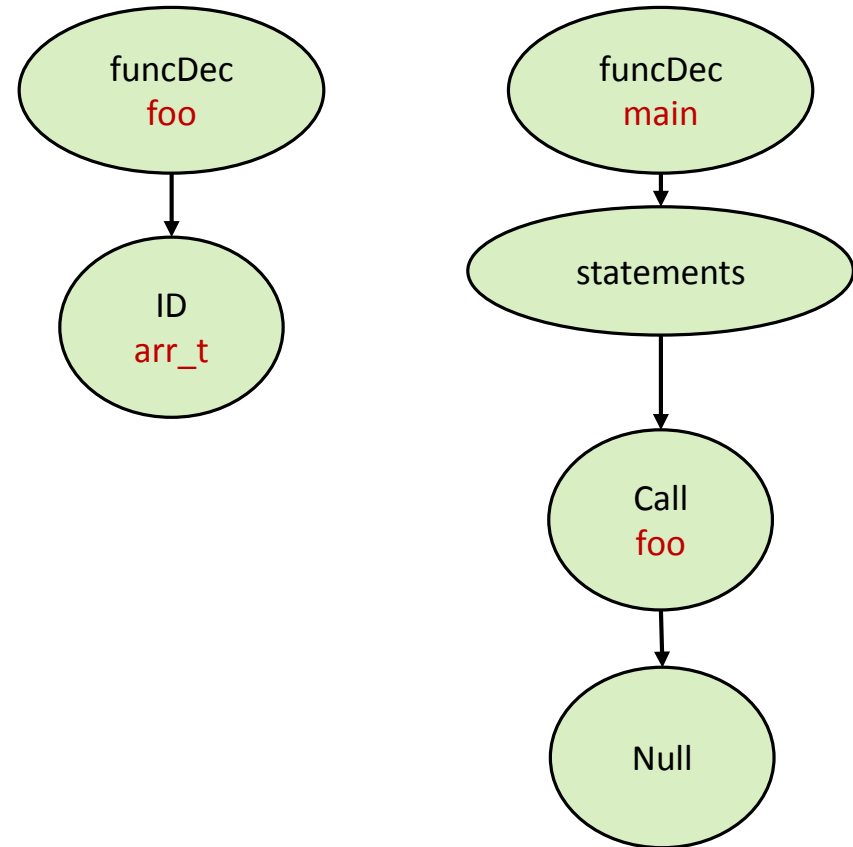
```
class A { }  
class B extends A { }  
void foo(B b) { }  
void main() {  
    A o = new A;  
    foo(o);  
}
```

Invalid



Null

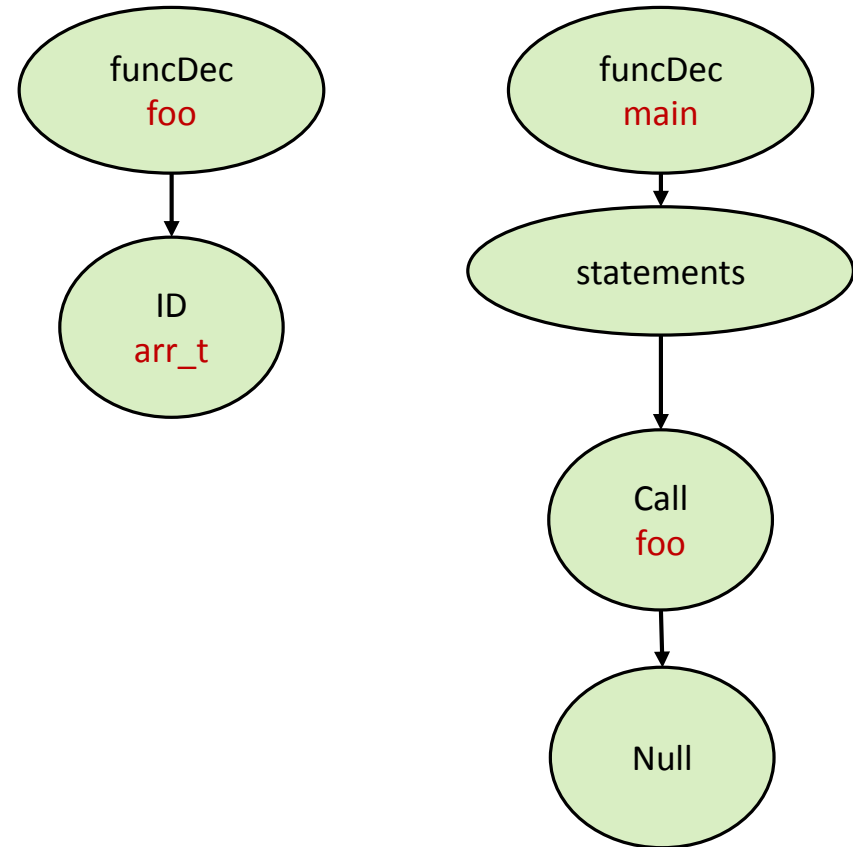
```
typedef int arr_t[];  
void foo(arr_t a) { }  
void main() {  
    foo(null);  
}
```



Null

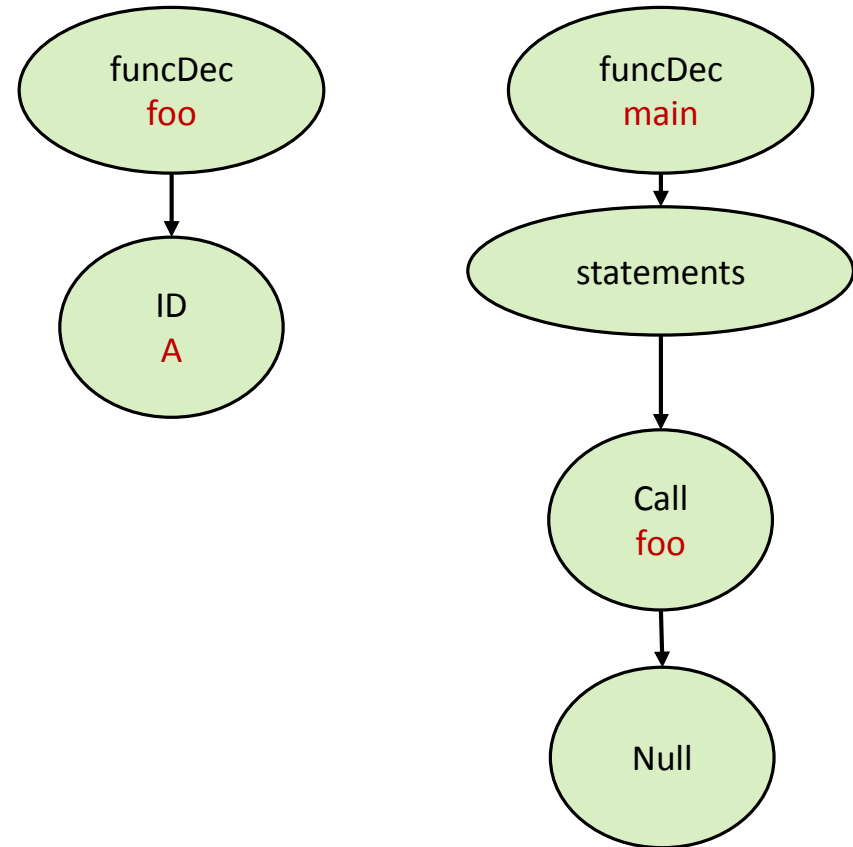
```
typedef int arr_t[];  
void foo(arr_t a) { }  
void main() {  
    foo(null);  
}
```

Valid



Null

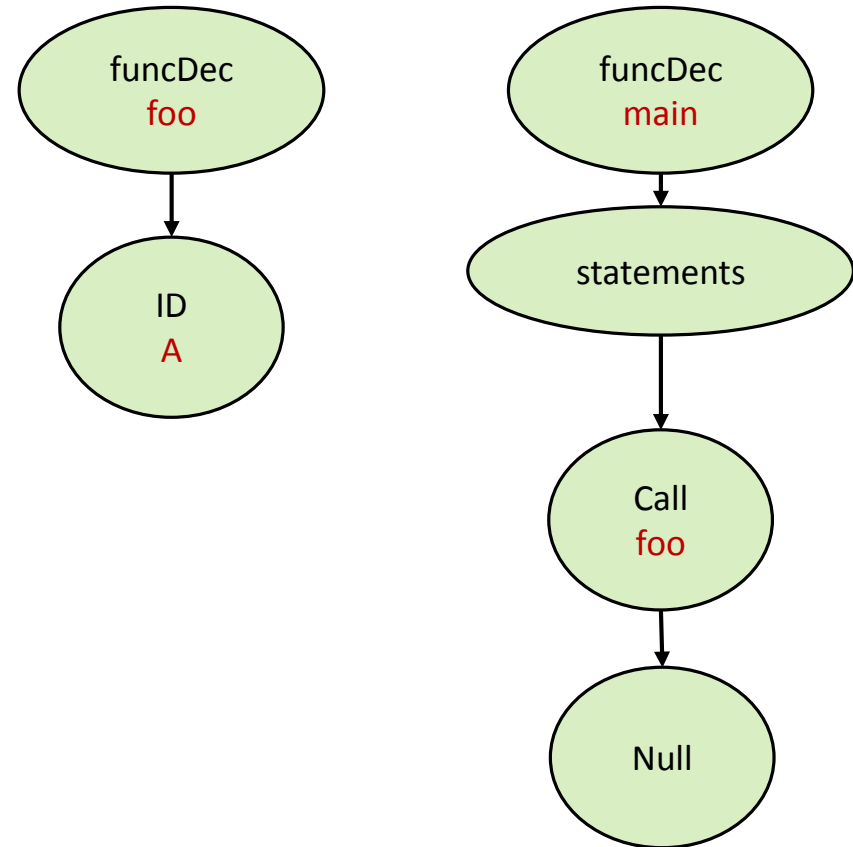
```
class A { };  
void foo(A a) { }  
void main() {  
    foo(null);  
}
```



Null

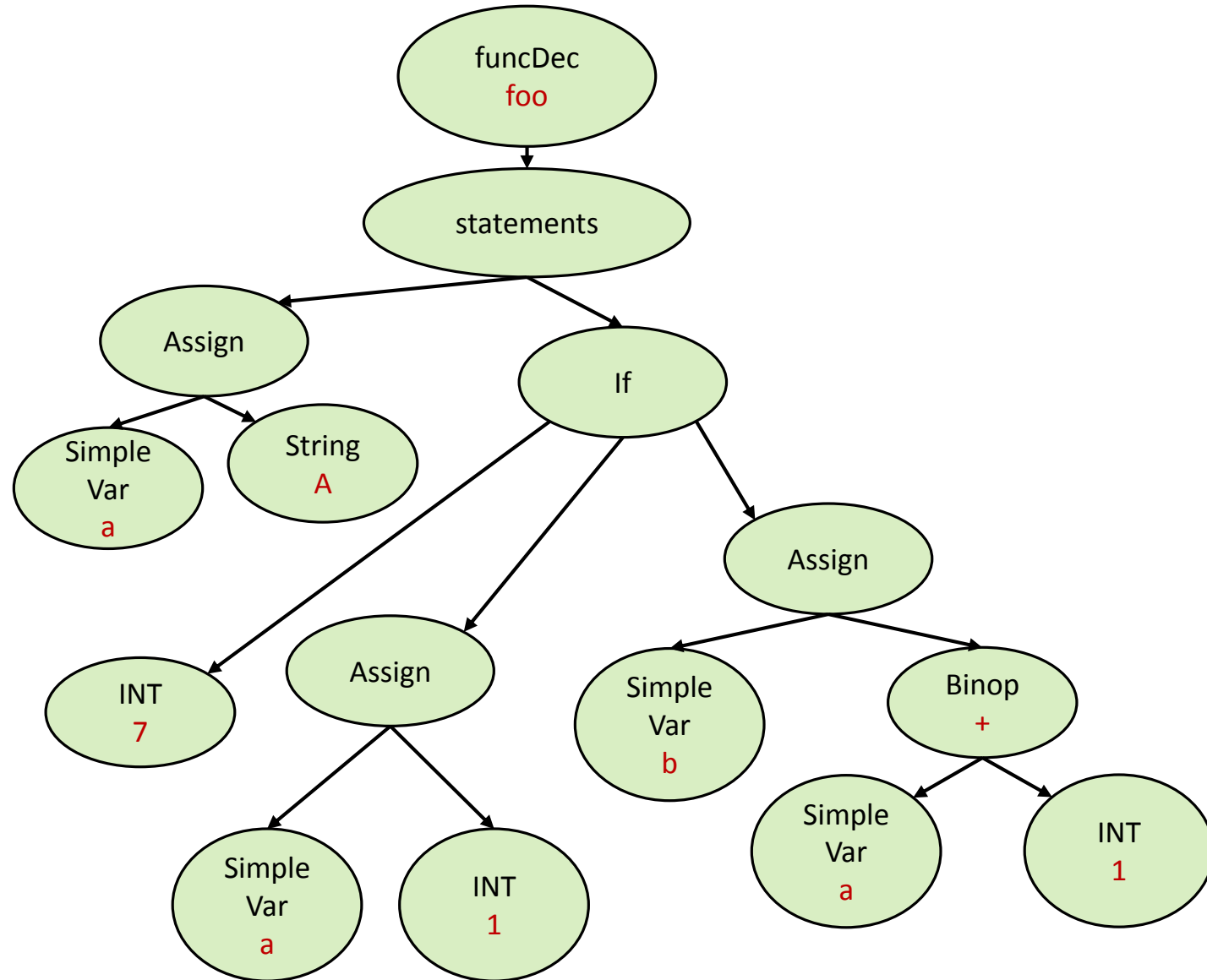
```
class A { };  
void foo(A a) { }  
void main() {  
    foo(null);  
}
```

Valid



Scopes

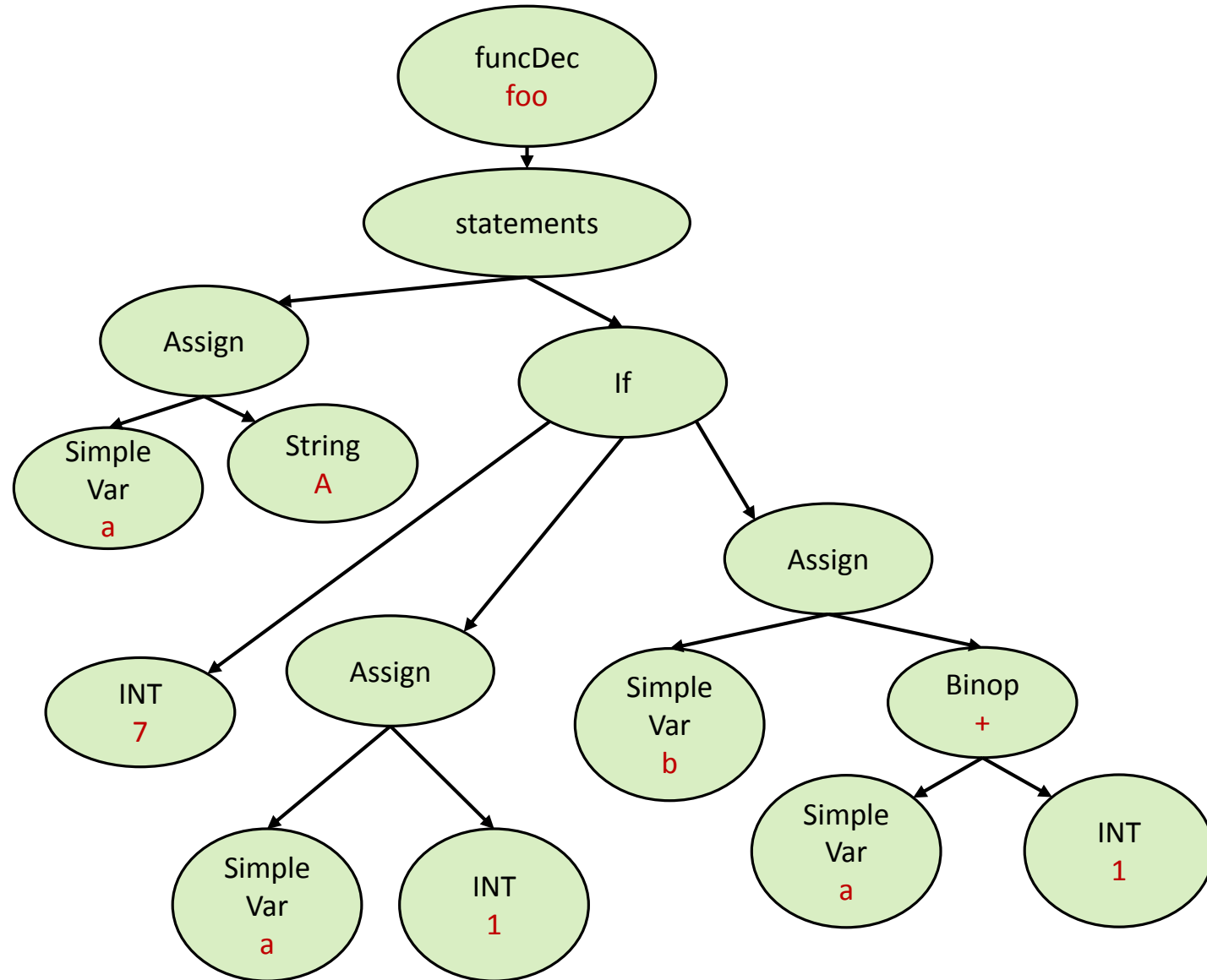
```
void foo(void) {  
  string a = "A";  
  if (7) {  
    int a = 1;  
    int b = a + 1;  
  }  
}
```



Scopes

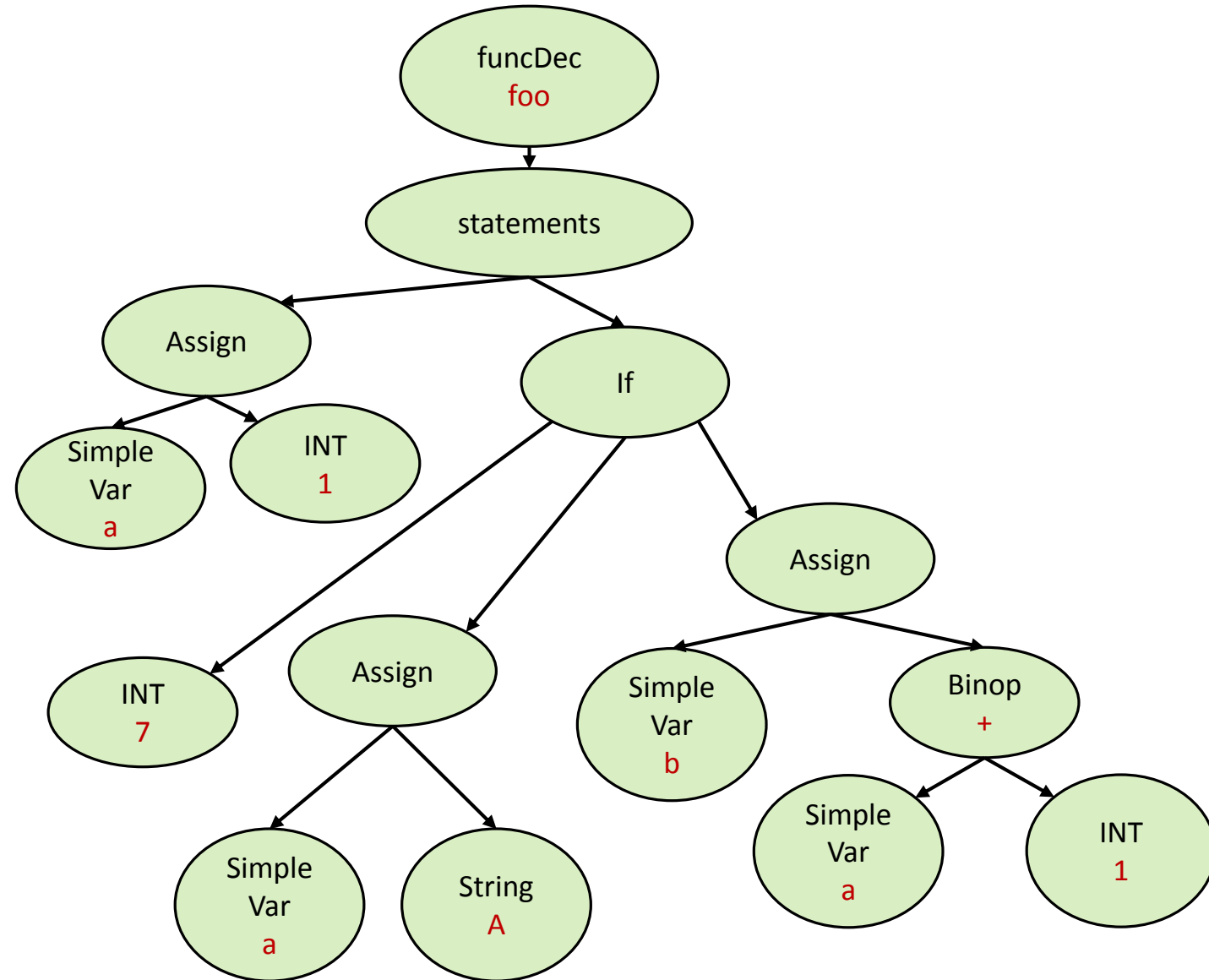
```
void foo(void) {  
  string a = "A";  
  if (7) {  
    int a = 1;  
    int b = a + 1;  
  }  
}
```

Valid



Scopes

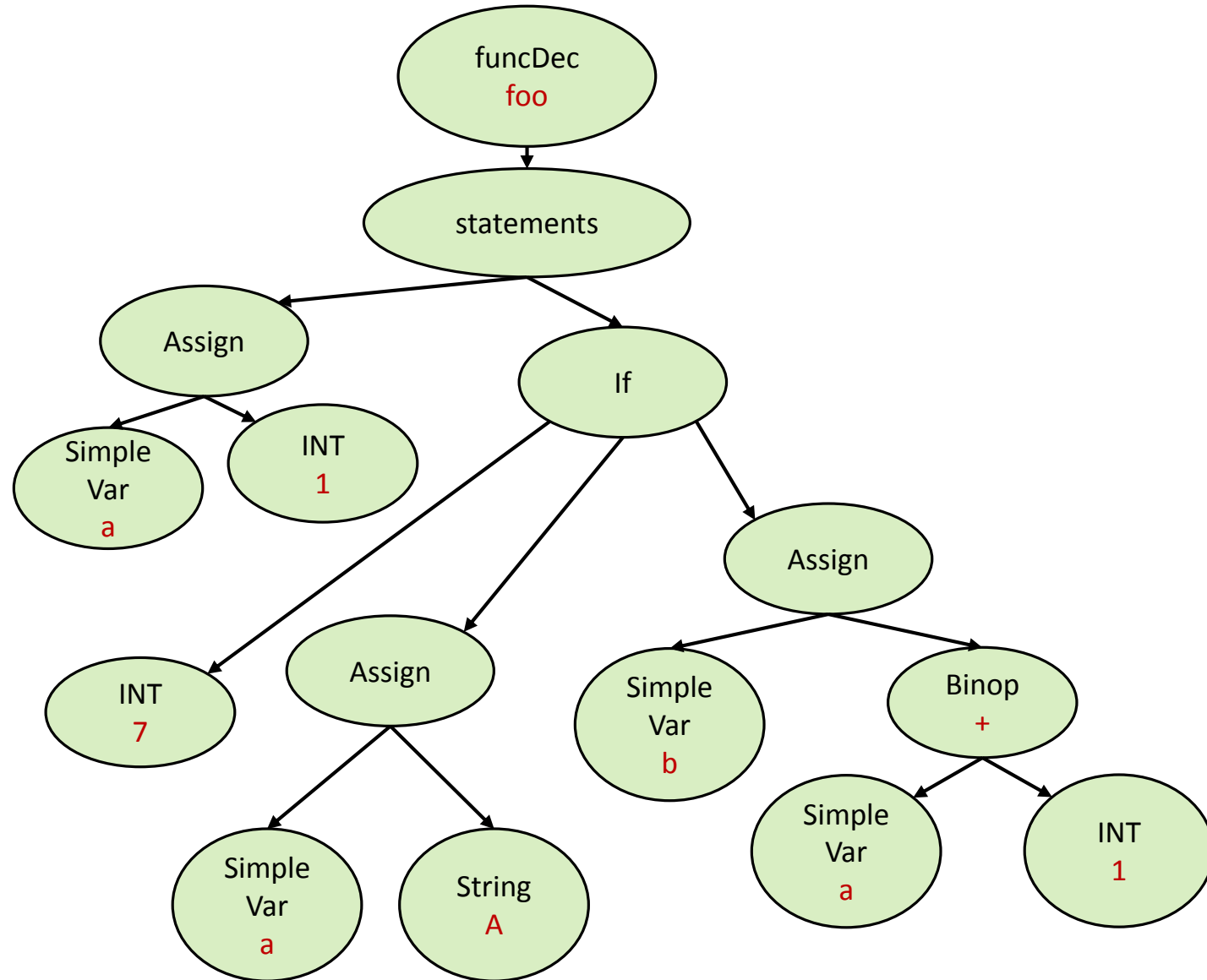
```
void foo(void) {  
  int a = 1;  
  if (7) {  
    string a = "A";  
    int b = a + 1;  
  }  
}
```



Scopes

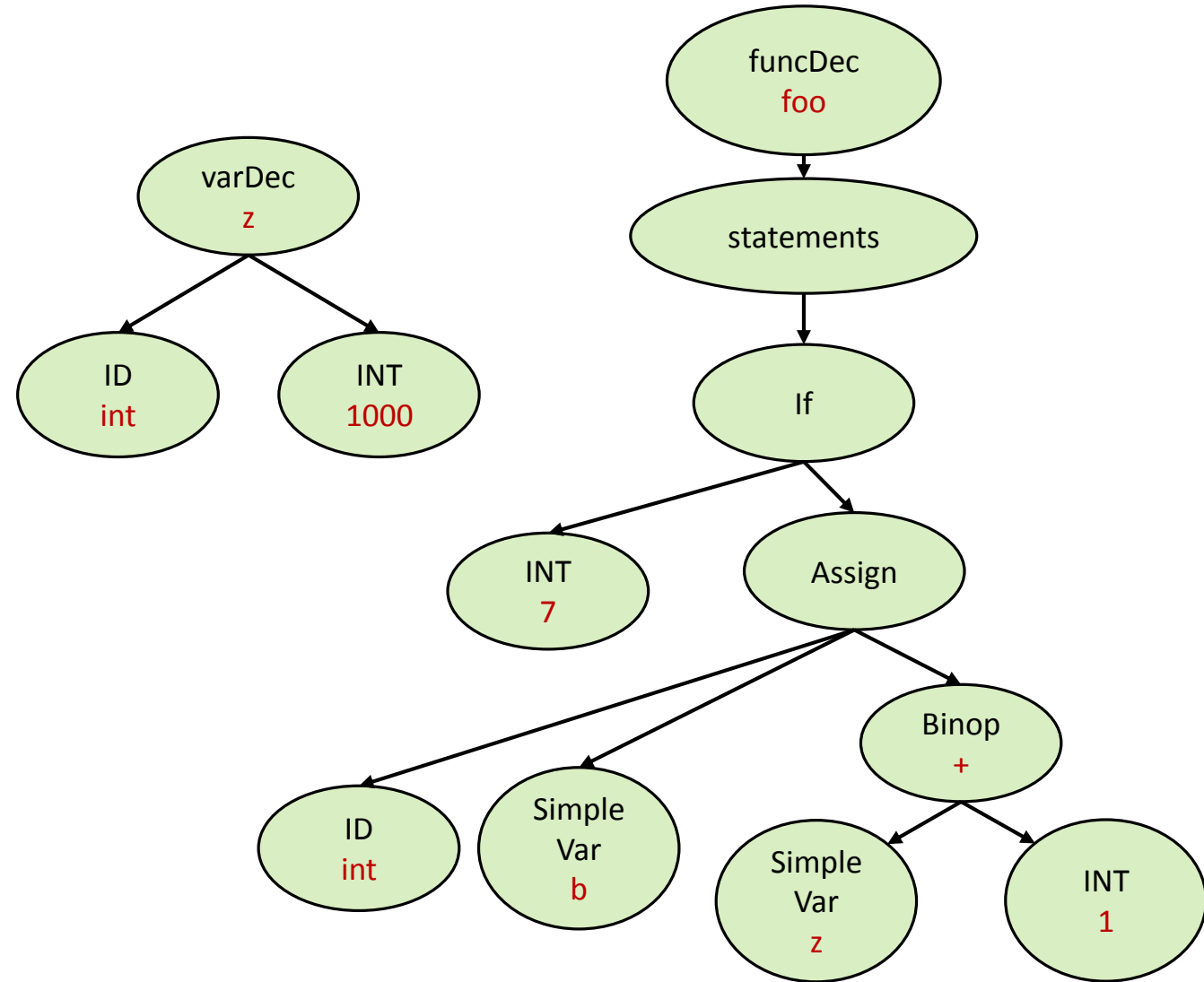
```
void foo(void) {  
  int a = 1;  
  if (7) {  
    string a = "A";  
    int b = a + 1;  
  }  
}
```

Invalid



Scopes

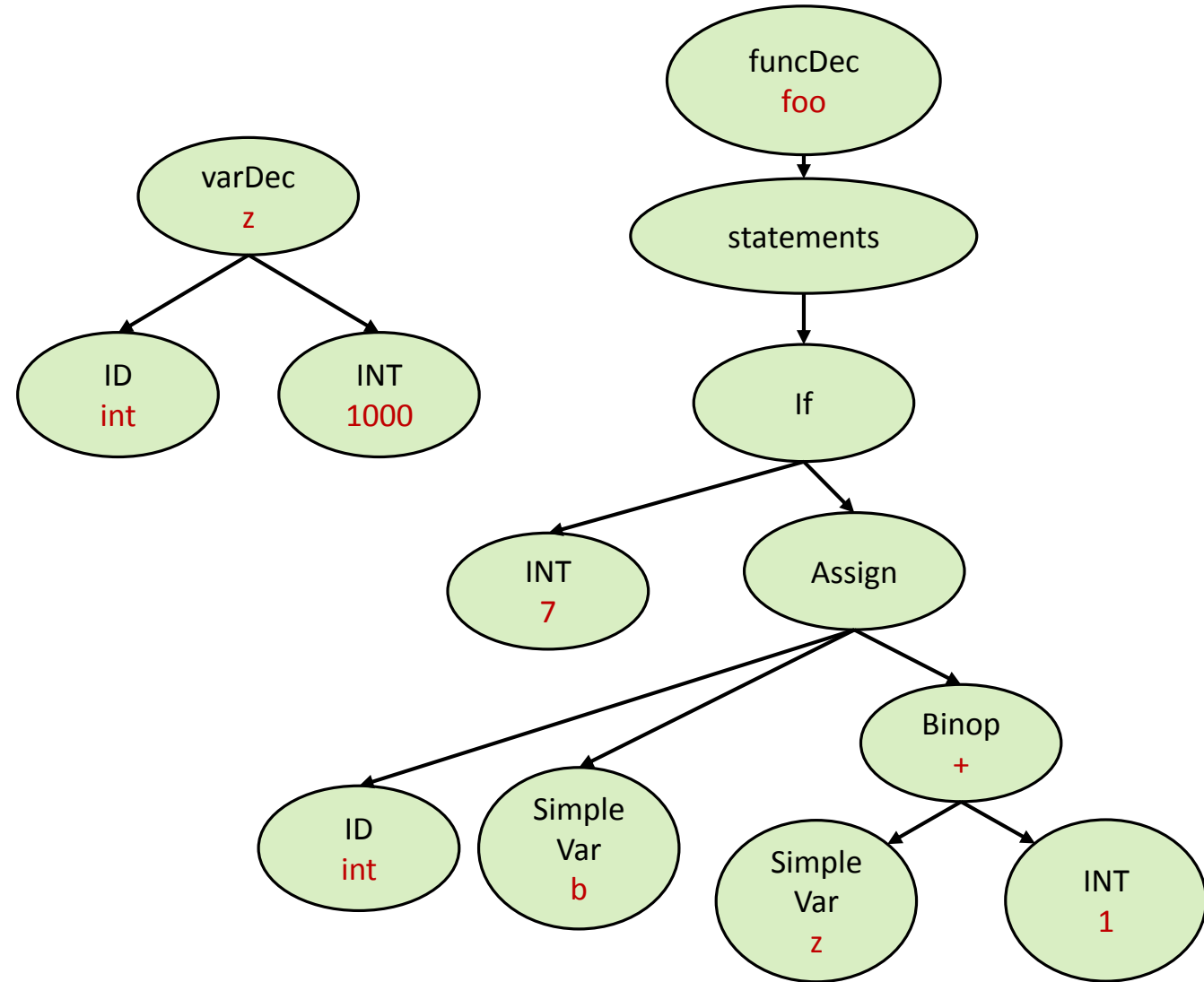
```
int z = 1000;  
void foo(int z) {  
    if (7) {  
        int b = z + 1;  
    }  
}
```



Scopes

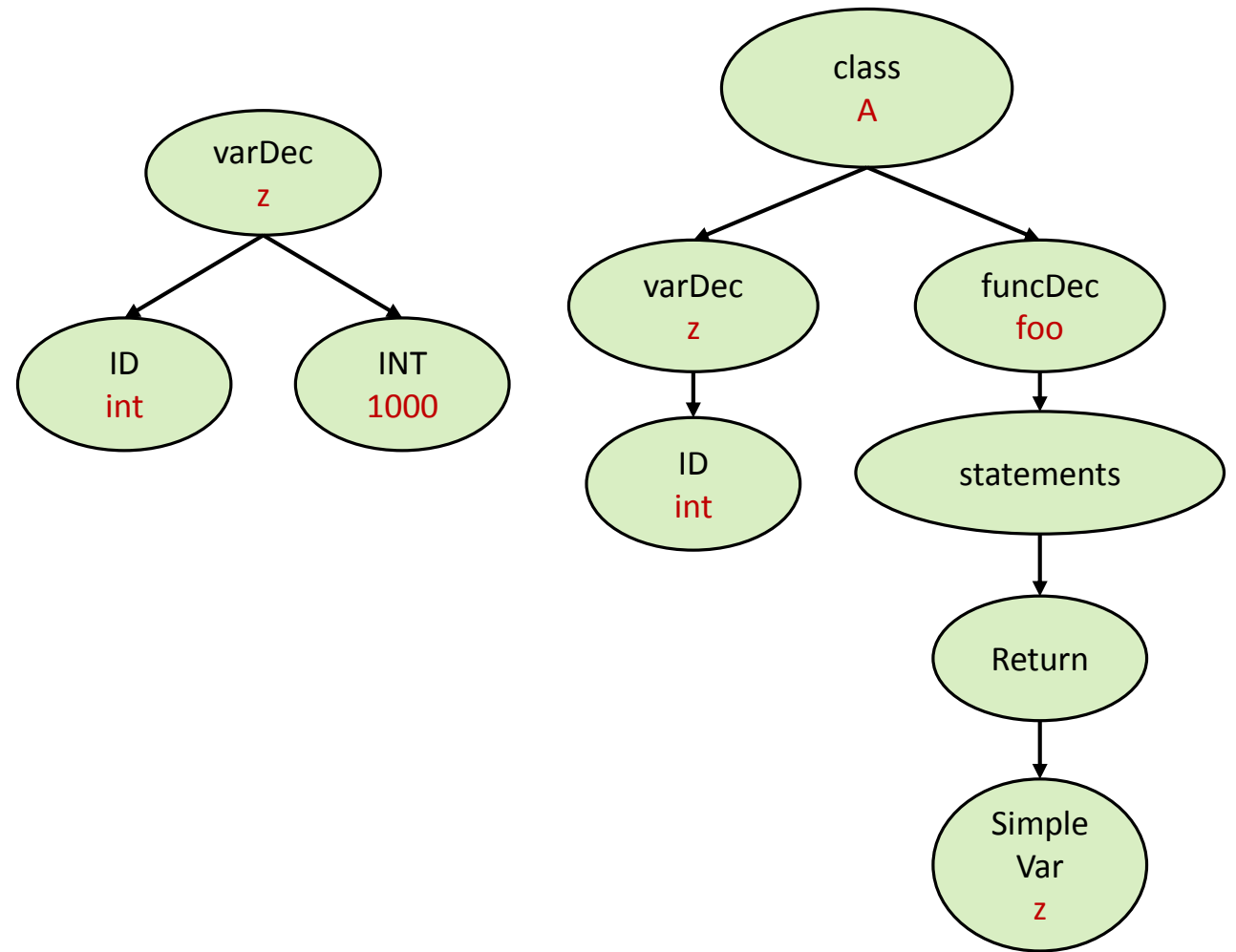
```
int z = 1000;  
void foo(int z) {  
    if (7) {  
        int b = z + 1;  
    }  
}
```

Valid



Scopes

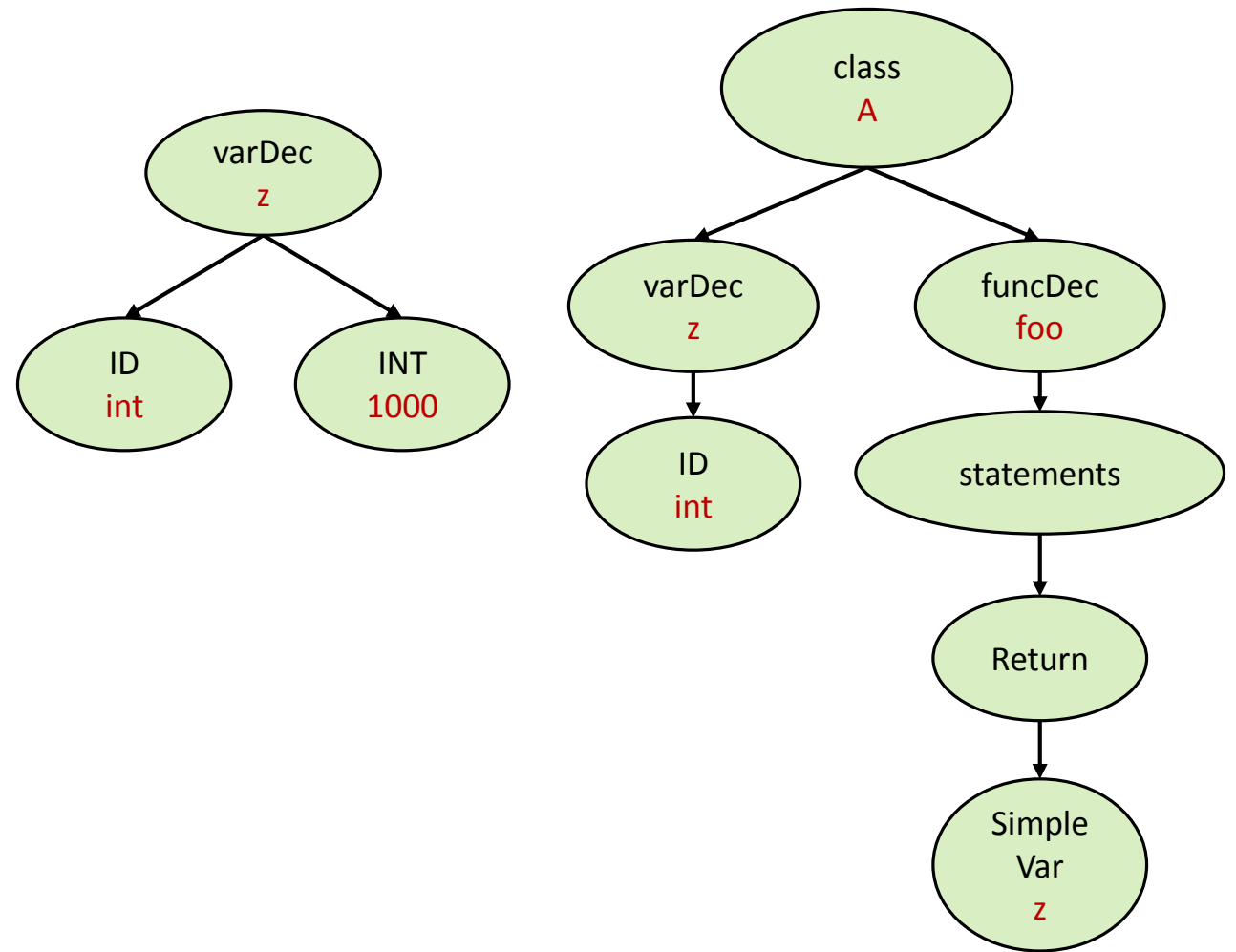
```
int z = 1000;  
class A {  
    int z;  
    int foo() {  
        return z;  
    }  
}
```



Scopes

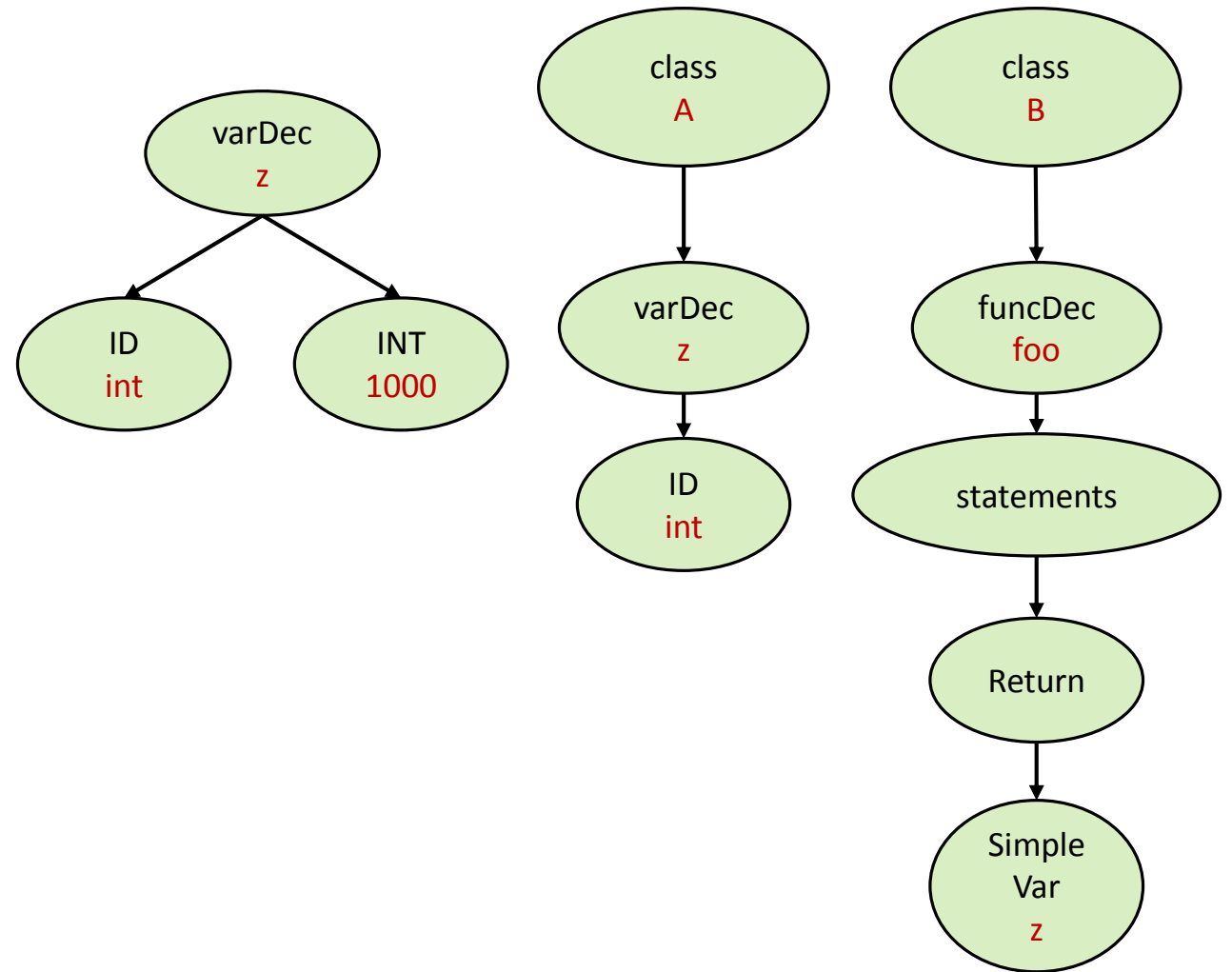
```
int z = 1000;  
class A {  
    int z;  
    int foo() {  
        return z;  
    }  
}
```

Valid



Scopes

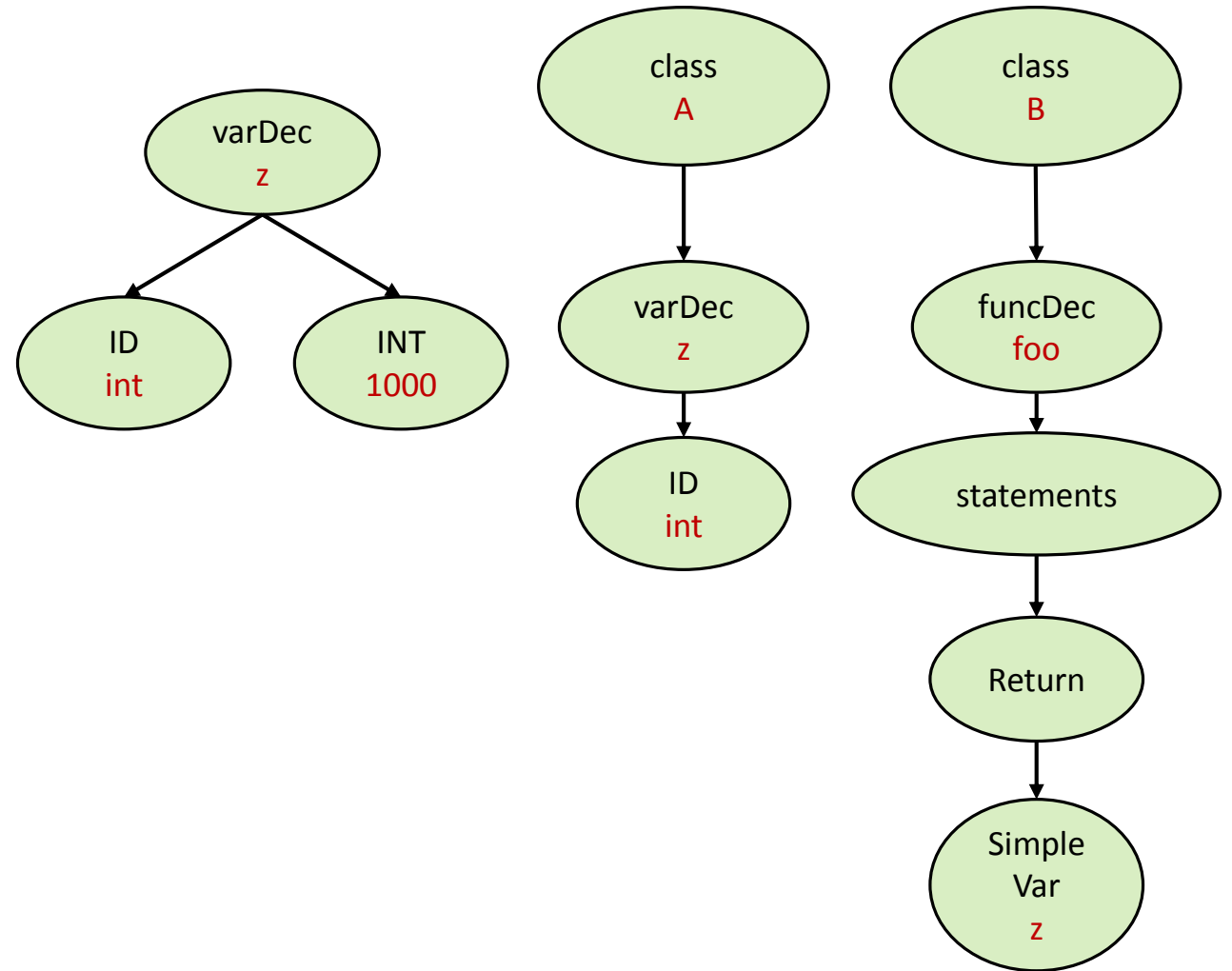
```
int z = 1000;  
class A {  
    int z;  
}  
class B extends A {  
    int foo() {  
        return z;  
    }  
}
```



Scopes

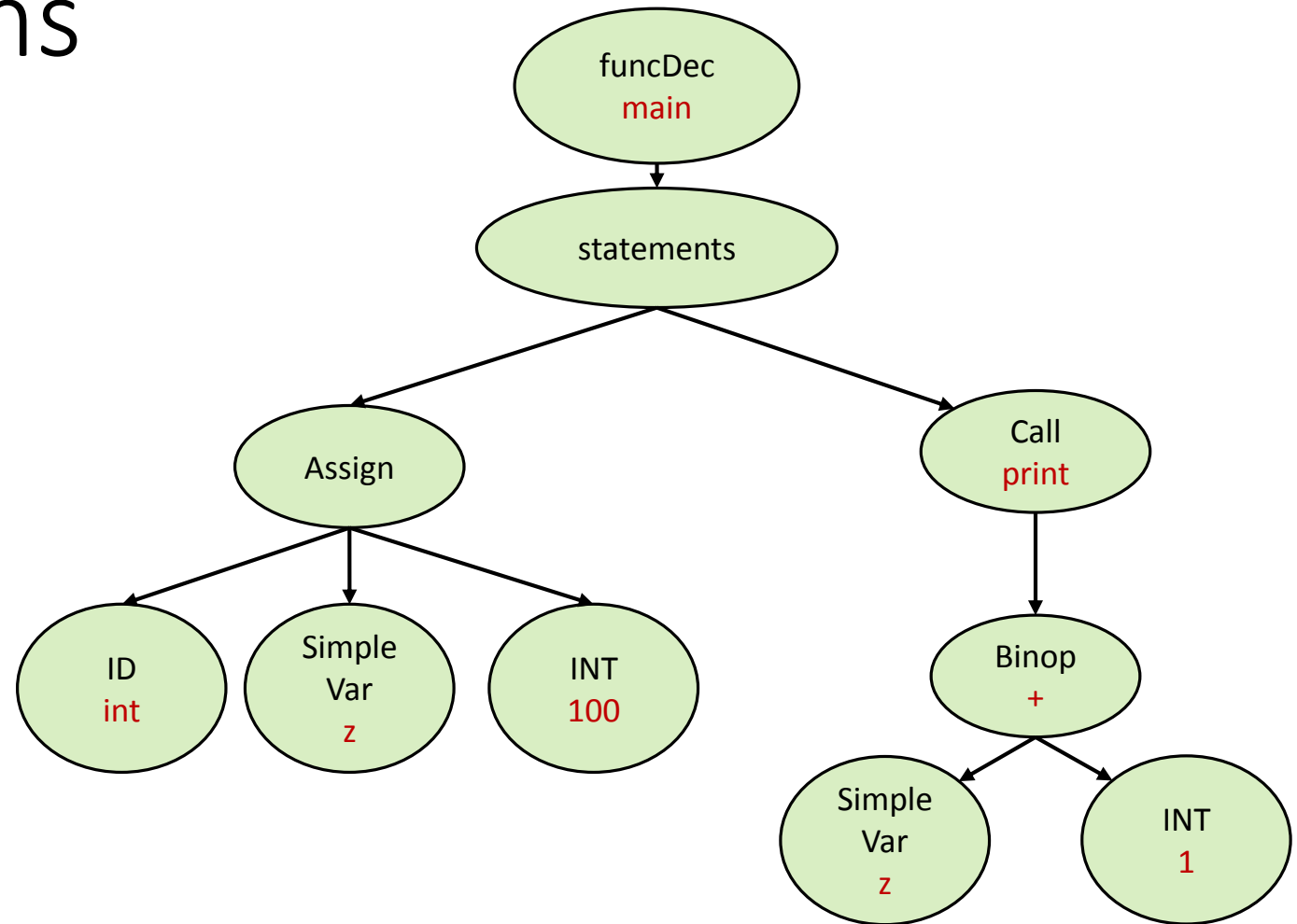
```
int z = 1000;  
class A {  
    int z;  
}  
class B extends A {  
    int foo() {  
        return z;  
    }  
}
```

Valid



Library Functions

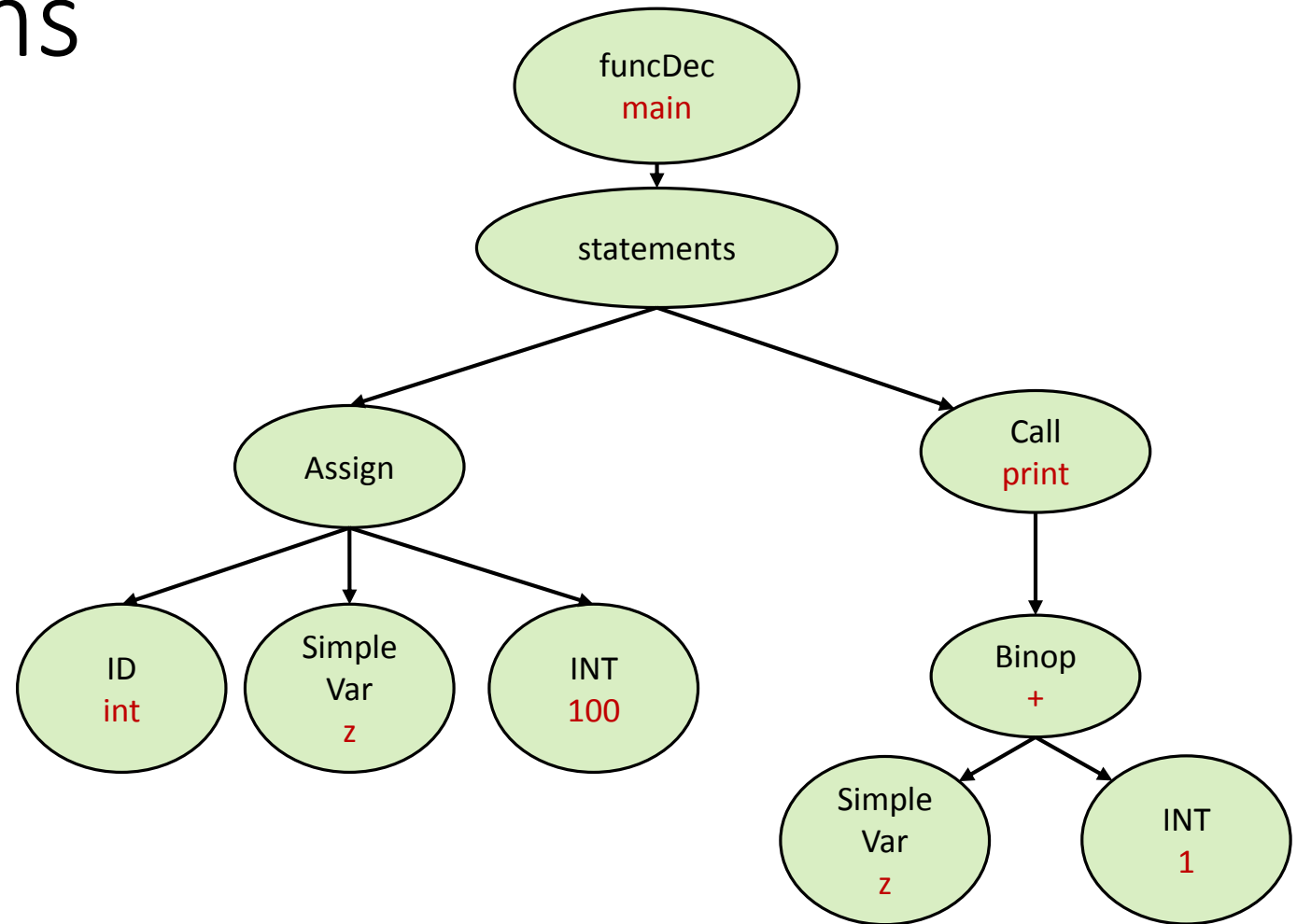
```
void main() {  
    int z = 100;  
    print(z + 1);  
}
```



Library Functions

```
void main() {  
    int z = 100;  
    print(z + 1);  
}
```

Valid



Implementation

The AST is traversed in a top-down manner

- Each AST node class, has a **visit** API:
 - Performs the relevant semantic checks
 - May call the visitors of the node's children
 - In the skeleton it's called *semantMe*
- The traversal starts from the root node

Implementation

```
Class ASTExpBinOp {  
    public ASTExp left;  
    public ASTExp right;  
    ...  
    public Type visit() {  
        Type t1 = left.visit();  
        Type t2 = right.visit();  
        if (t1 != t2) {  
            // error  
        }  
        ...  
    }  
}
```

Implementation

```
Class ASTStatmentList {  
    public ASTStatement head;  
    public ASTStatmentList tail;  
    ...  
    public Type visit() {  
        if (head)  
            head.visit();  
        if (tail)  
            tail.visit();  
        return null;  
    }  
}
```

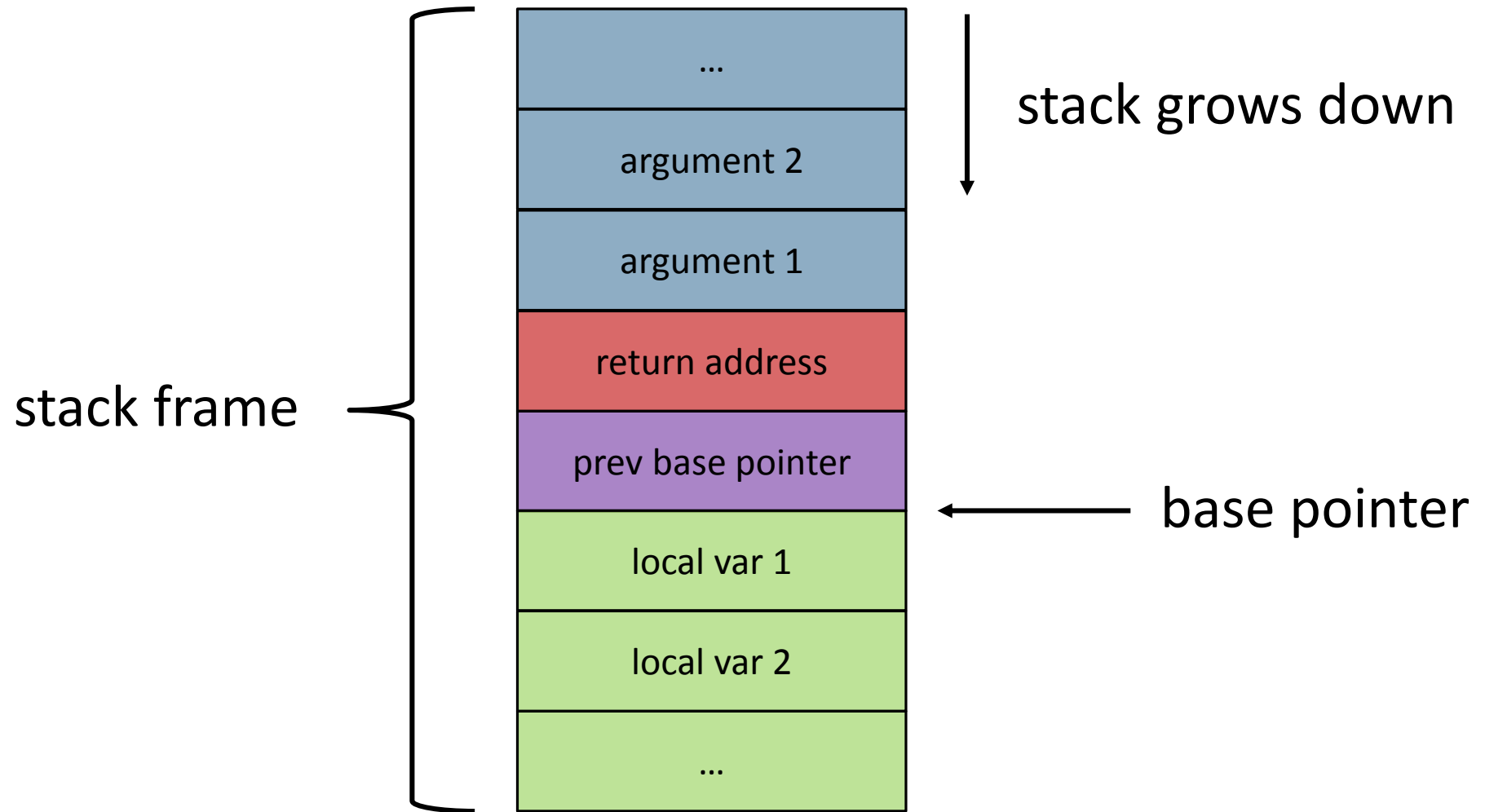
AST Annotations

AST Annotations

While analyzing the AST, we can extend it with useful information:

- Variable offsets
- Parameter offsets
- Class layout
- Type size

Stack



Stack

```
int f(int x, int y){  
    int z = x + y;  
    return z;  
}  
int g() {  
    int x = f(10, 20)  
}
```

```
f:  
push    %ebp  
mov     %esp,%ebp  
sub     $0x10,%esp  
mov     0x8(%ebp),%edx  
mov     0xc(%ebp),%eax  
add     %edx,%eax  
mov     %eax,-0x4(%ebp)  
mov     -0x4(%ebp),%eax  
leave  
ret
```

```
g:  
...  
push    $0x14  
push    $0x0a  
call    24 <g+0xe>  
...
```

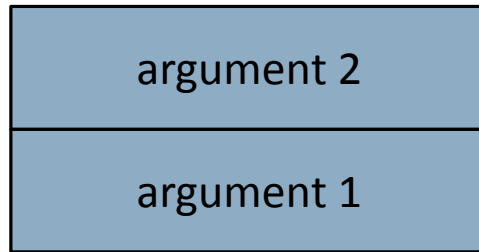

Stack

argument 2

```
f:  
push    %ebp  
mov     %esp,%ebp  
sub     $0x10,%esp  
mov     0x8(%ebp),%edx  
mov     0xc(%ebp),%eax  
add     %edx,%eax  
mov     %eax,-0x4(%ebp)  
mov     -0x4(%ebp),%eax  
leave  
ret
```

```
g:  
...  
push    $0x14  
push    $0x0a  
call    24 <g+0xe>  
...
```

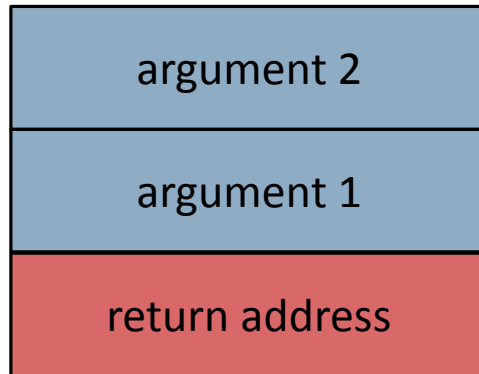
Stack



```
f:  
push    %ebp  
mov     %esp,%ebp  
sub     $0x10,%esp  
mov     0x8(%ebp),%edx  
mov     0xc(%ebp),%eax  
add     %edx,%eax  
mov     %eax,-0x4(%ebp)  
mov     -0x4(%ebp),%eax  
leave  
ret
```

```
g:  
...  
push    $0x14  
push    $0x0a  
call    24 <g+0xe>  
...
```

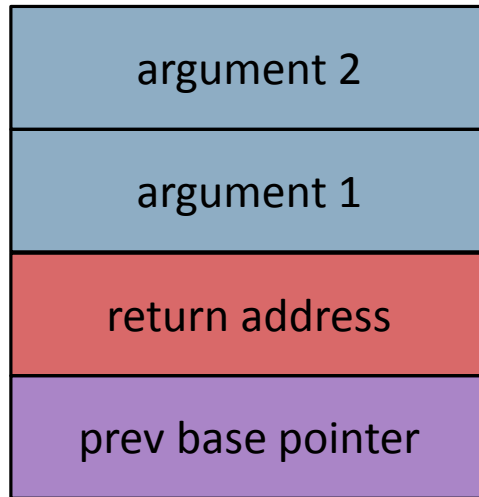
Stack



```
f:  
push    %ebp  
mov     %esp,%ebp  
sub     $0x10,%esp  
mov     0x8(%ebp),%edx  
mov     0xc(%ebp),%eax  
add     %edx,%eax  
mov     %eax,-0x4(%ebp)  
mov     -0x4(%ebp),%eax  
leave  
ret
```

```
g:  
...  
push    $0x14  
push    $0x0a  
call   24 <g+0xe>  
...
```

Stack



f:

push **%ebp**

mov %esp, %ebp

sub \$0x10, %esp

mov 0x8(%ebp), %edx

mov 0xc(%ebp), %eax

add %edx, %eax

mov %eax, -0x4(%ebp)

mov -0x4(%ebp), %eax

leave

ret

g:

...

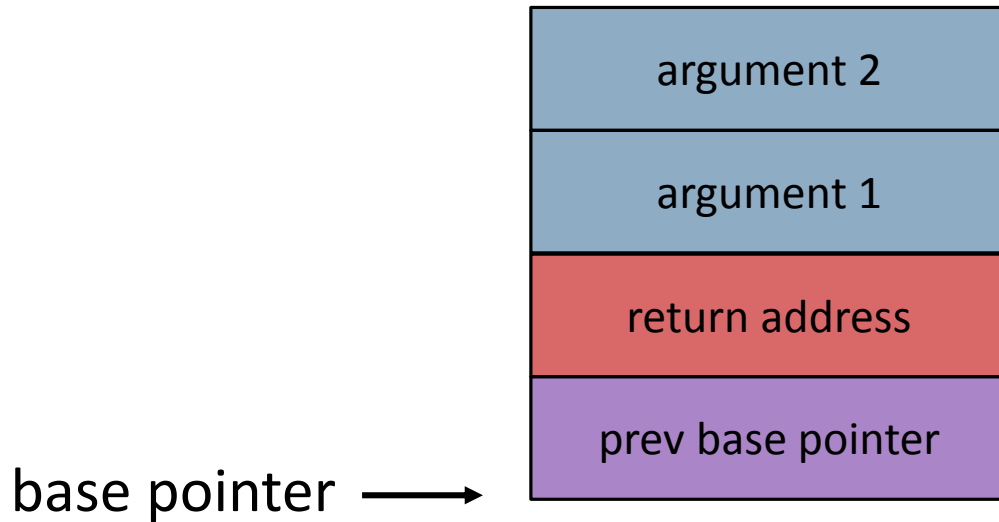
push \$0x14

push \$0x0a

call 24 <g+0xe>

...

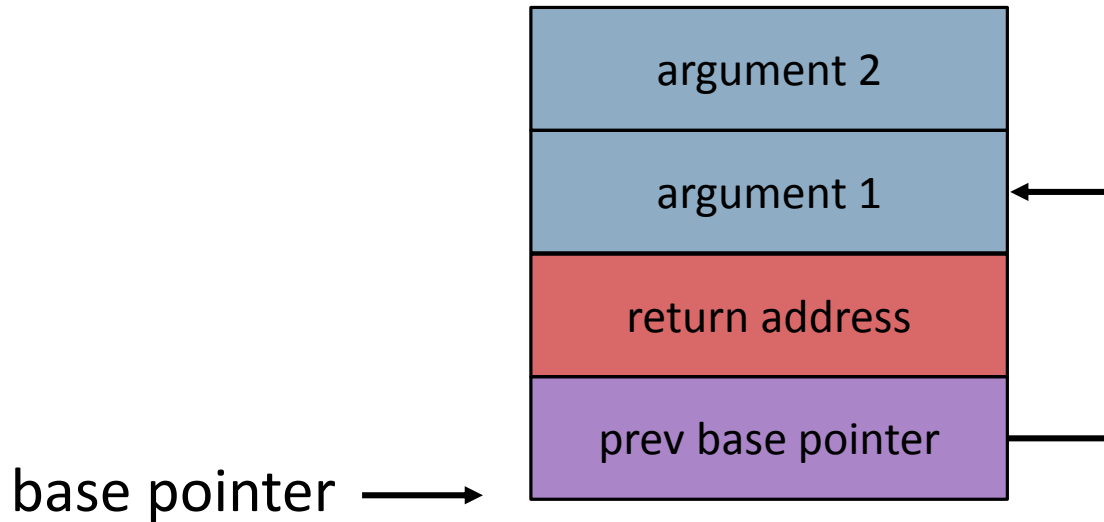
Stack



```
f:
push    %ebp
mov     %esp, %ebp
sub     $0x10, %esp
mov     0x8(%ebp), %edx
mov     0xc(%ebp), %eax
add     %edx, %eax
mov     %eax, -0x4(%ebp)
mov     -0x4(%ebp), %eax
leave
ret
```

```
g:
...
push    $0x14
push    $0x0a
call    24 <g+0xe>
...
```

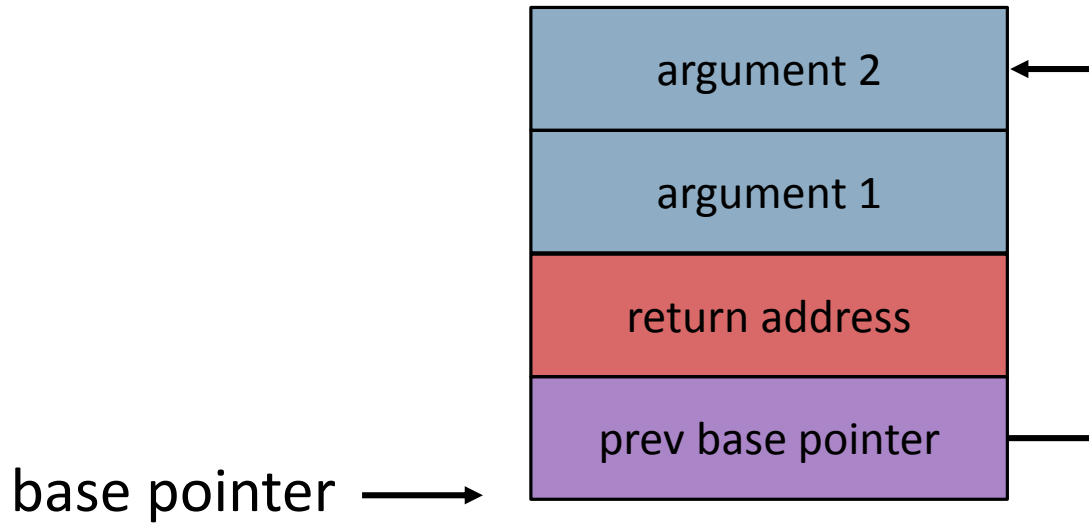
Stack



```
f:
push    %ebp
mov     %esp,%ebp
sub     $0x10,%esp
mov     0x8(%ebp),%edx
mov     0xc(%ebp),%eax
add     %edx,%eax
mov     %eax,-0x4(%ebp)
mov     -0x4(%ebp),%eax
leave
ret
```

```
g:
...
push    $0x14
push    $0x0a
call    24 <g+0xe>
...
```

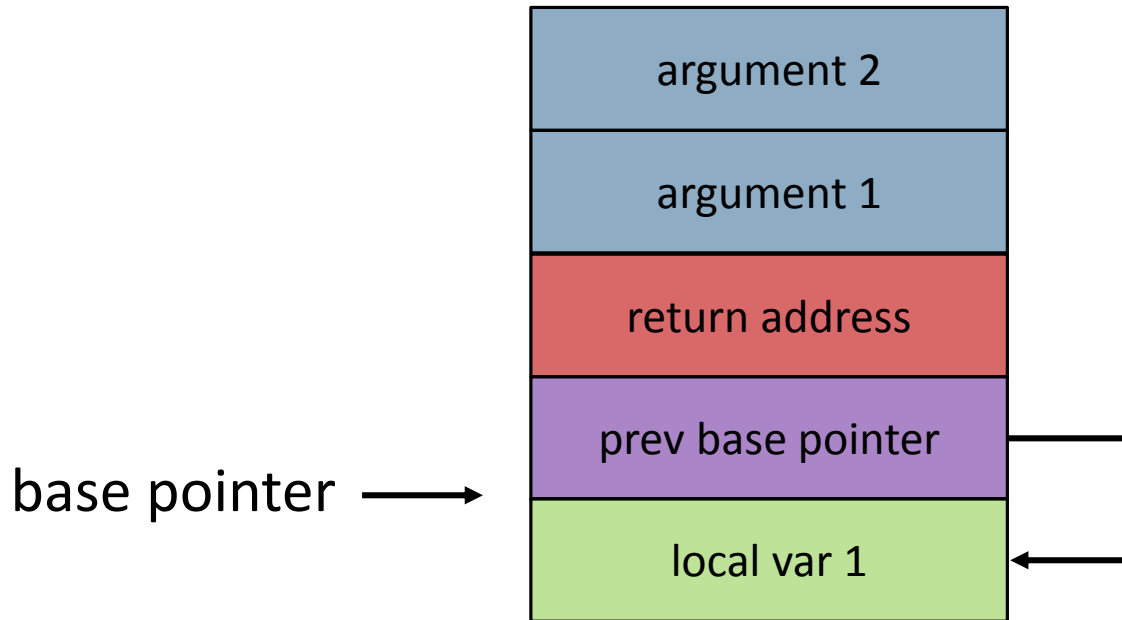
Stack



```
f:
push    %ebp
mov     %esp,%ebp
sub     $0x10,%esp
mov     0x8(%ebp),%edx
mov     0xc(%ebp),%eax
add     %edx,%eax
mov     %eax,-0x4(%ebp)
mov     -0x4(%ebp),%eax
leave
ret
```

```
g:
...
push    $0x14
push    $0x0a
call    24 <g+0xe>
...
```

Stack



```
f:
push    %ebp
mov     %esp,%ebp
sub     $0x10,%esp
mov     0x8(%ebp),%edx
mov     0xc(%ebp),%eax
add     %edx,%eax
mov     %eax,-0x4(%ebp)
mov     -0x4(%ebp),%eax
leave
ret
```

```
g:
...
push    $0x14
push    $0x0a
call    24 <g+0xe>
...
```


Variable Offsets

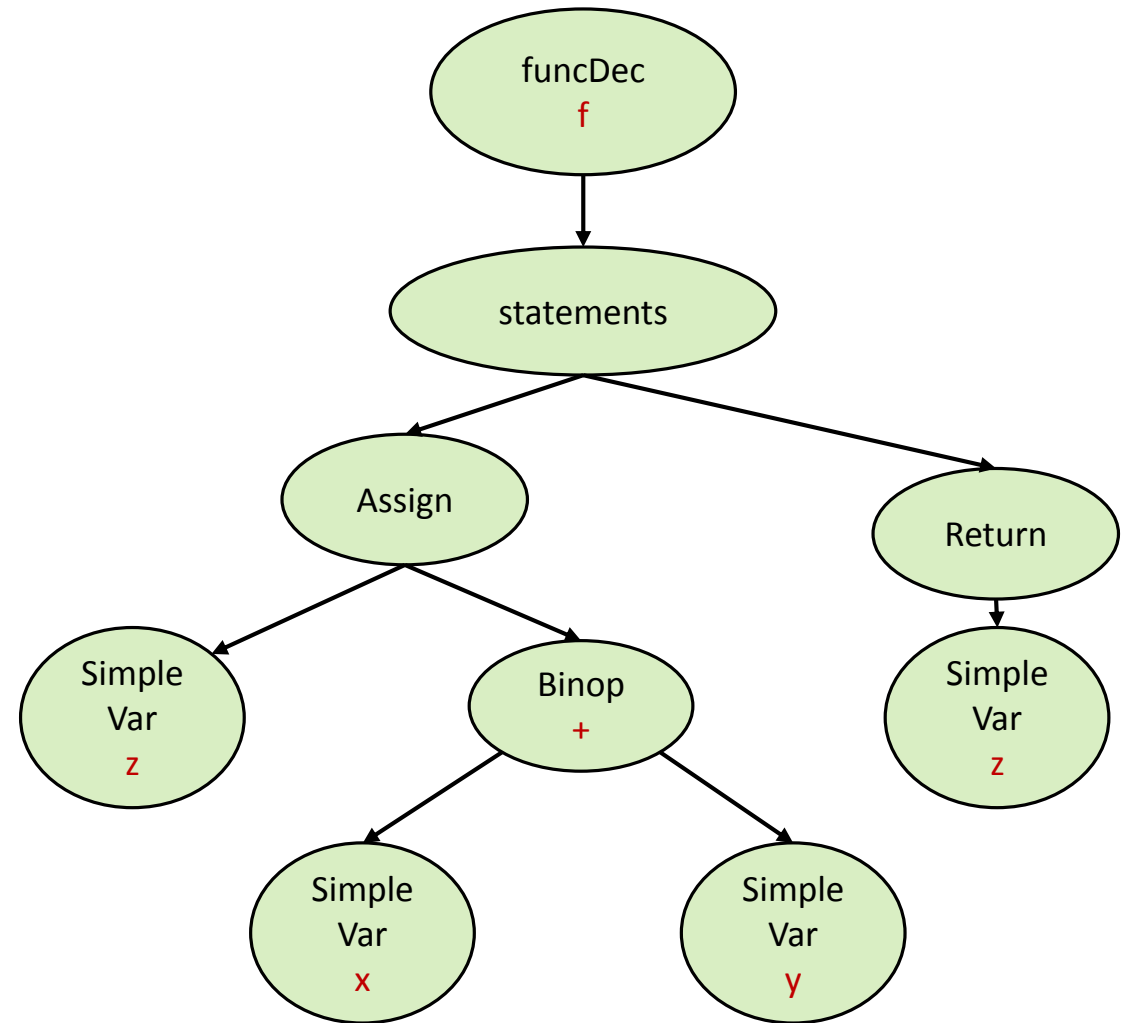
Machine code **does not** contain names of:

- Local variables
- Parameters

Instead, we use offsets **relatively** from the **stack base pointer**

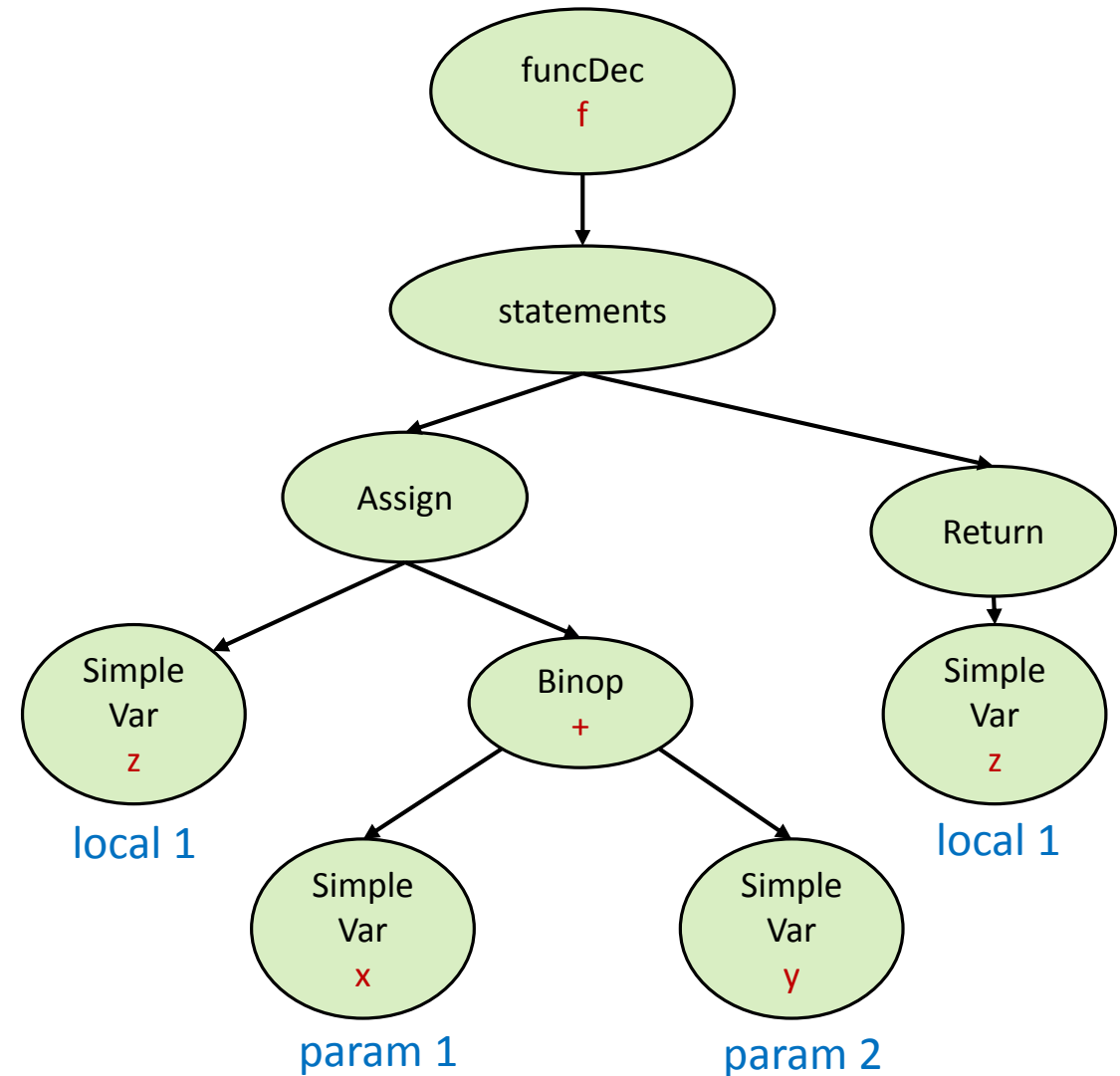
Variable Offsets

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



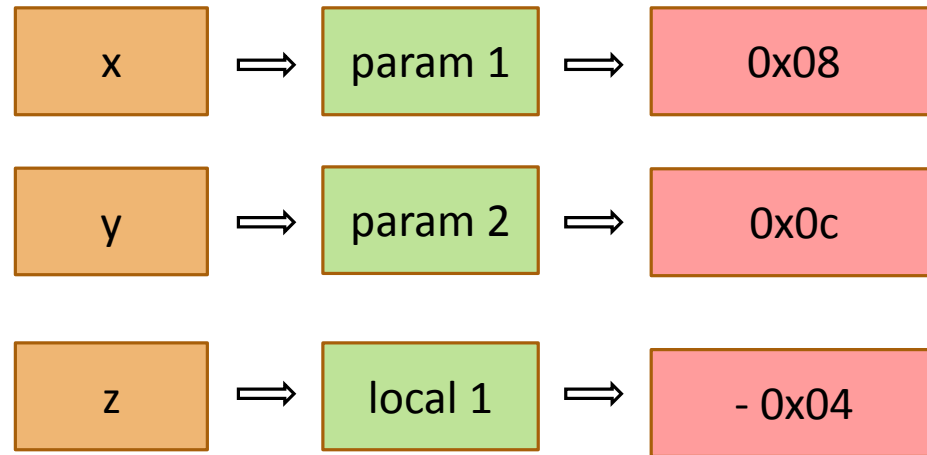
Variable Offsets

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



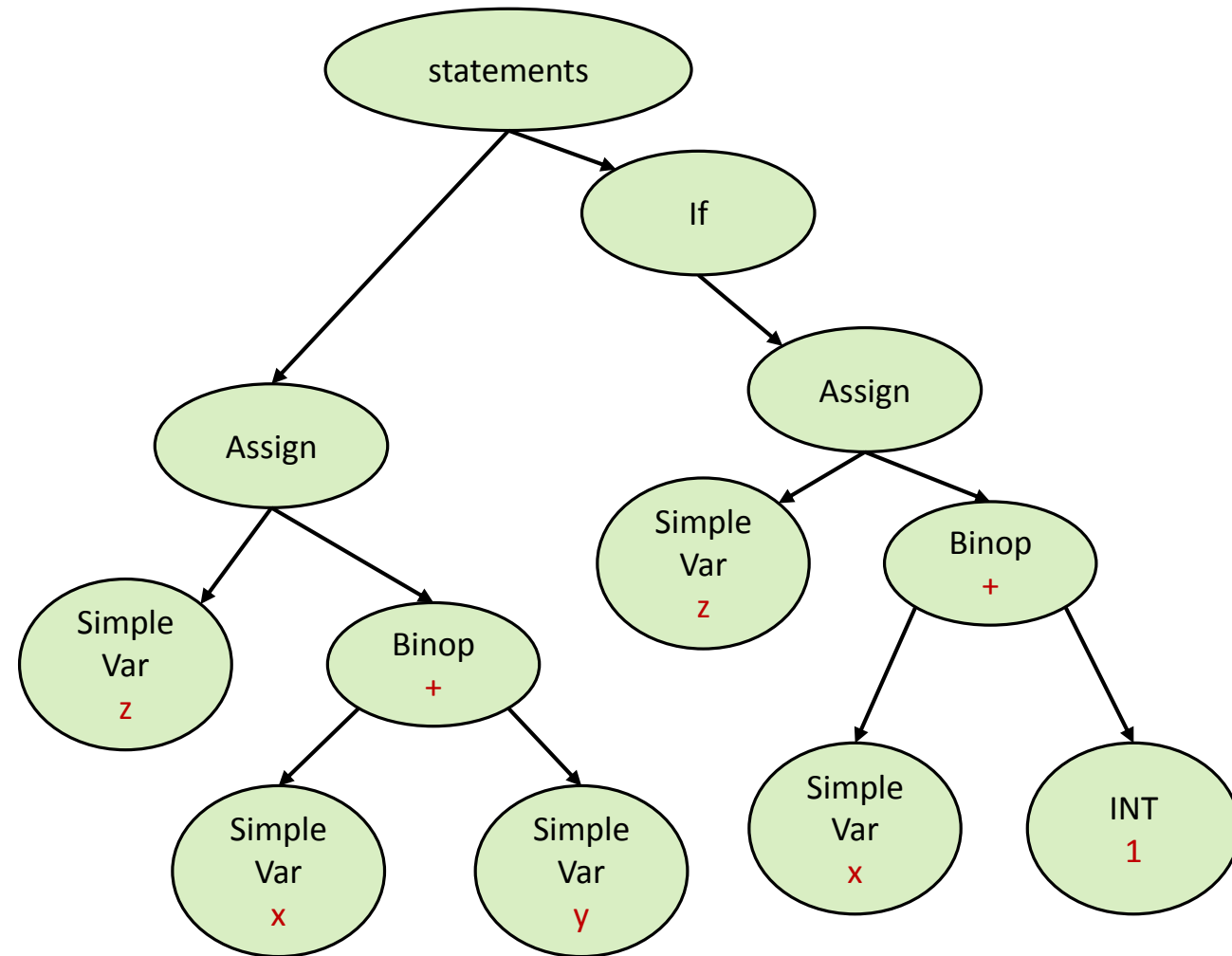
Variable Offsets

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



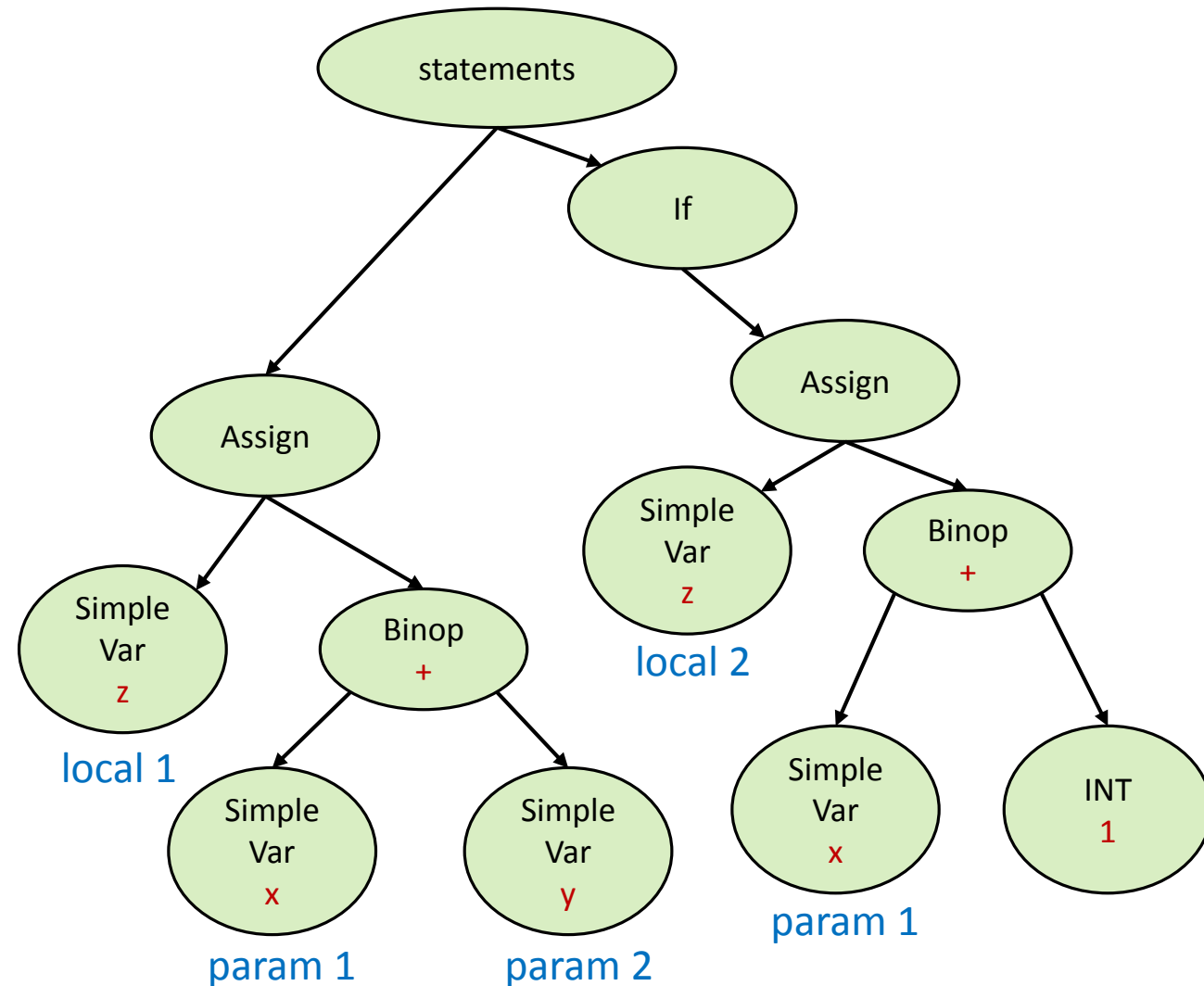
Variable Offsets

```
void f(int x, int y) {  
    int z = x + y;  
    if (z > 1) {  
        int z = x + 1;  
    }  
}
```



Variable Offsets

```
void f(int x, int y) {  
    int z = x + y;  
    if (z > 1) {  
        int z = x + 1;  
    }  
}
```



Field Offsets

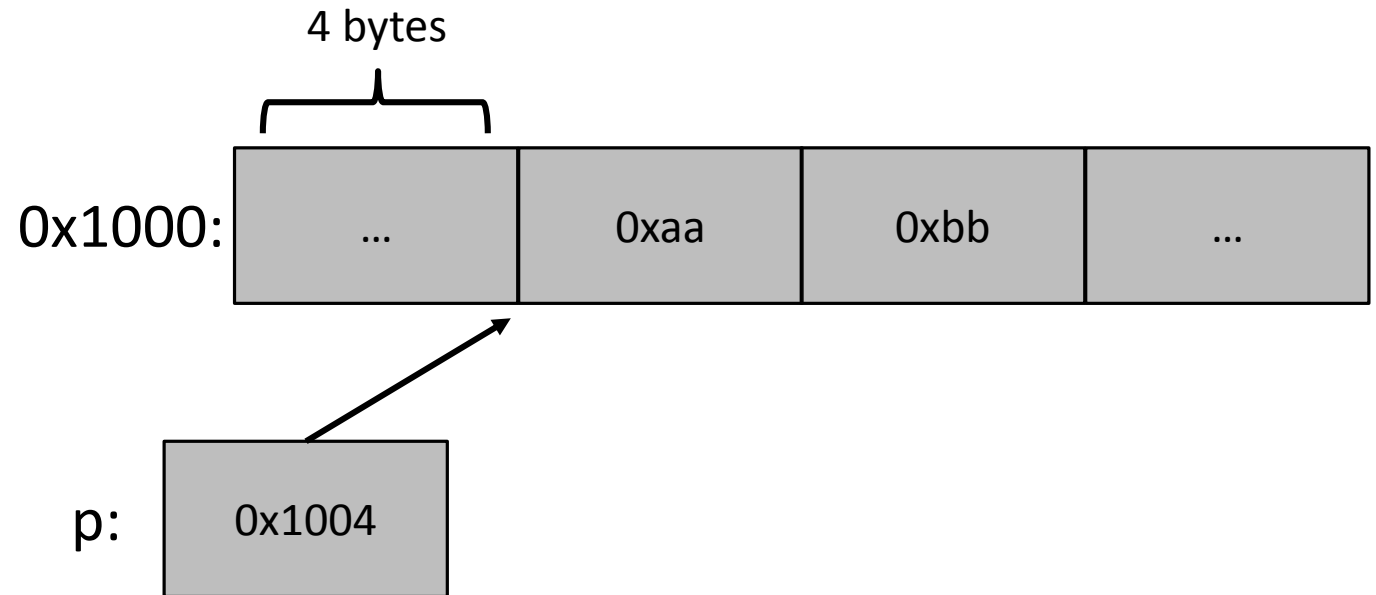
How does an object of this class look in memory?

```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```

Field Offsets

How does an object of this class look in memory?

```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```



Field Offsets

How does an object of this class look in memory?

```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
mov     0x8(%ebp), %eax  
movl    $0xaa, (%eax)  
mov     0x8(%ebp), %eax  
movl    $0xbb, 0x4(%eax)  
pop     %ebp  
ret
```

Field Offsets

How does an object of this class look in memory?

```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
mov     0x8(%ebp), %eax  
movl    $0xaa, (%eax)  
mov     0x8(%ebp), %eax  
movl    $0xbb, 0x4(%eax)  
pop     %ebp  
ret
```

Field Offsets

How does an object of this class look in memory?

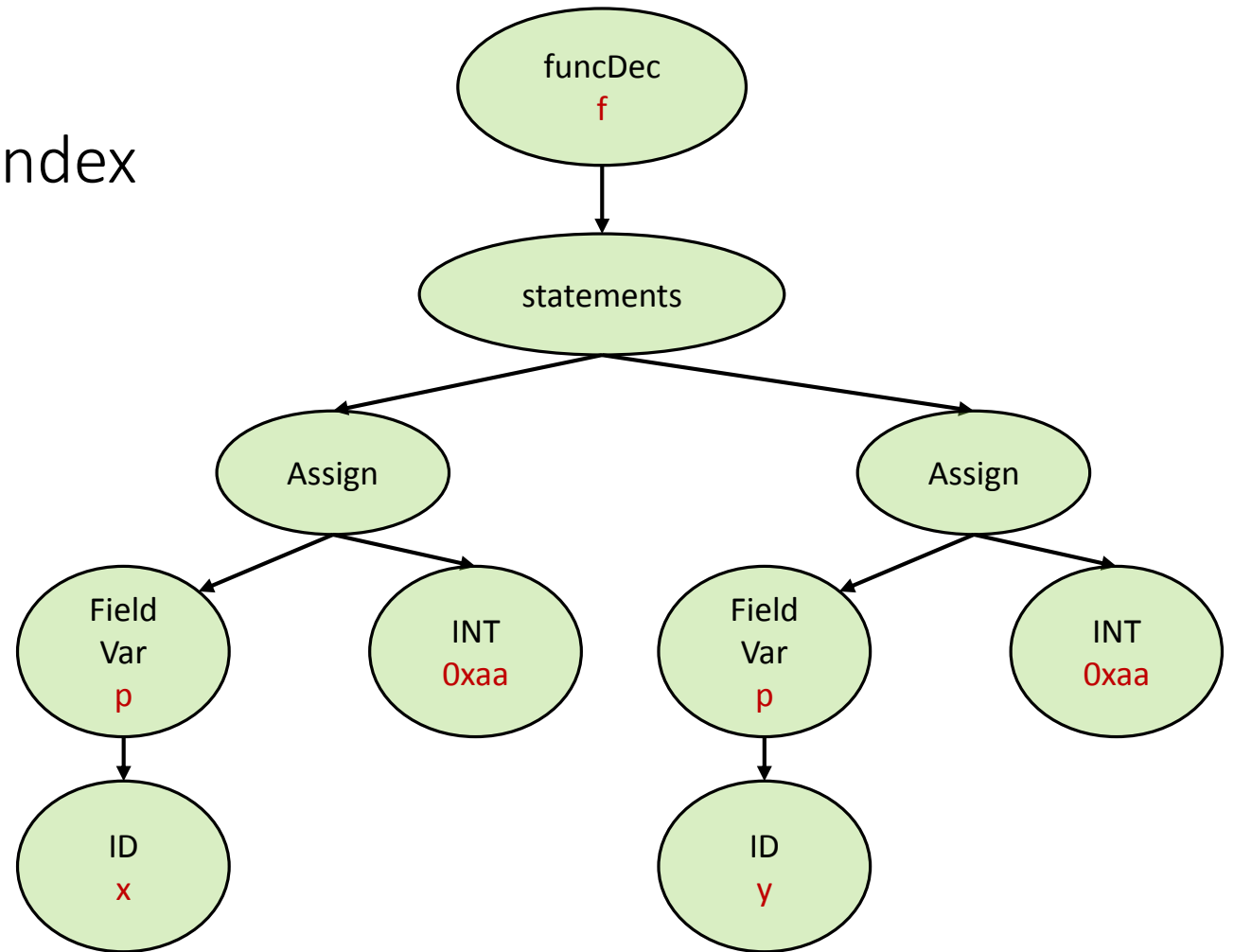
```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
mov     0x8(%ebp), %eax  
movl    $0xaa, (%eax)  
mov     0x8(%ebp), %eax  
movl    $0xbb, 0x4(%eax)  
pop     %ebp  
ret
```

Field Offsets

Each class field should have an index

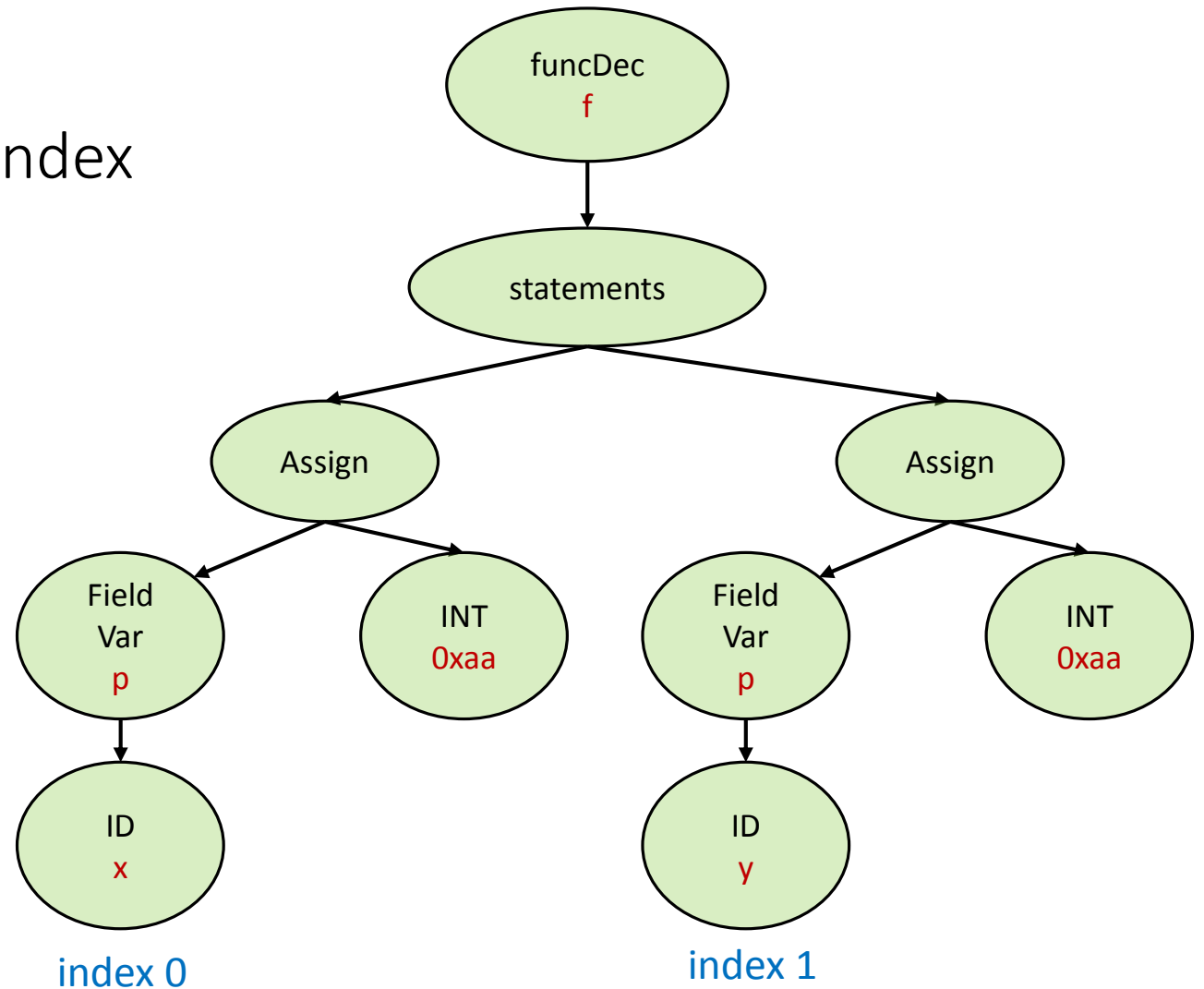
```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```



Field Offsets

Each class field should have an index

```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    p.x = 0xaa;  
    p.y = 0xbb;  
}
```



Type Sizes

```
class Point {  
    int x;  
    int y;  
}  
  
void f(Point p) {  
    Point p = new Point;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
sub     $0x18, %esp  
sub     $0xc, %esp  
push    $0x8  
call    56 <foo+0xc>  
add     $0x10, %esp  
mov     %eax, -0xc(%ebp)  
leave  
ret
```

Type Sizes

```
class Point {  
    int x;  
    int y;  
}  
void f(Point p) {  
    Point p = new Point;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
sub     $0x18, %esp  
sub     $0xc, %esp  
push    $0x8  
call    56 <foo+0xc>  
add     $0x10, %esp  
mov     %eax, -0xc(%ebp)  
leave  
ret
```

Type Sizes

```
class Point {  
    int x;  
    int y;  
    string name;  
}  
void f(Point p) {  
    Point p = new Point;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
sub     $0x18, %esp  
sub     $0xc, %esp  
push    ?  
call    56 <foo+0xc>  
add     $0x10, %esp  
mov     %eax, -0xc(%ebp)  
leave  
ret
```


Type Sizes

```
class Point {  
    int x;  
    int y;  
    string name;  
}  
void f(Point p) {  
    Point p = new Point;  
}
```

```
f:  
push    %ebp  
mov     %esp, %ebp  
sub     $0x18, %esp  
sub     $0xc, %esp  
push    $0xc  
call    56 <foo+0xc>  
add     $0x10, %esp  
mov     %eax, -0xc(%ebp)  
leave  
ret
```