# Semantic Analysis

TEACHING ASSISTANT: DAVID TRABISH

# Semantic Analysis

We need to check the following:

- Type checking
  - 1 + “1”
- Scopes
  - Undefined variables
- Additional:
  - Division by zero
  - Visibility semantics in classes (public, private, …)

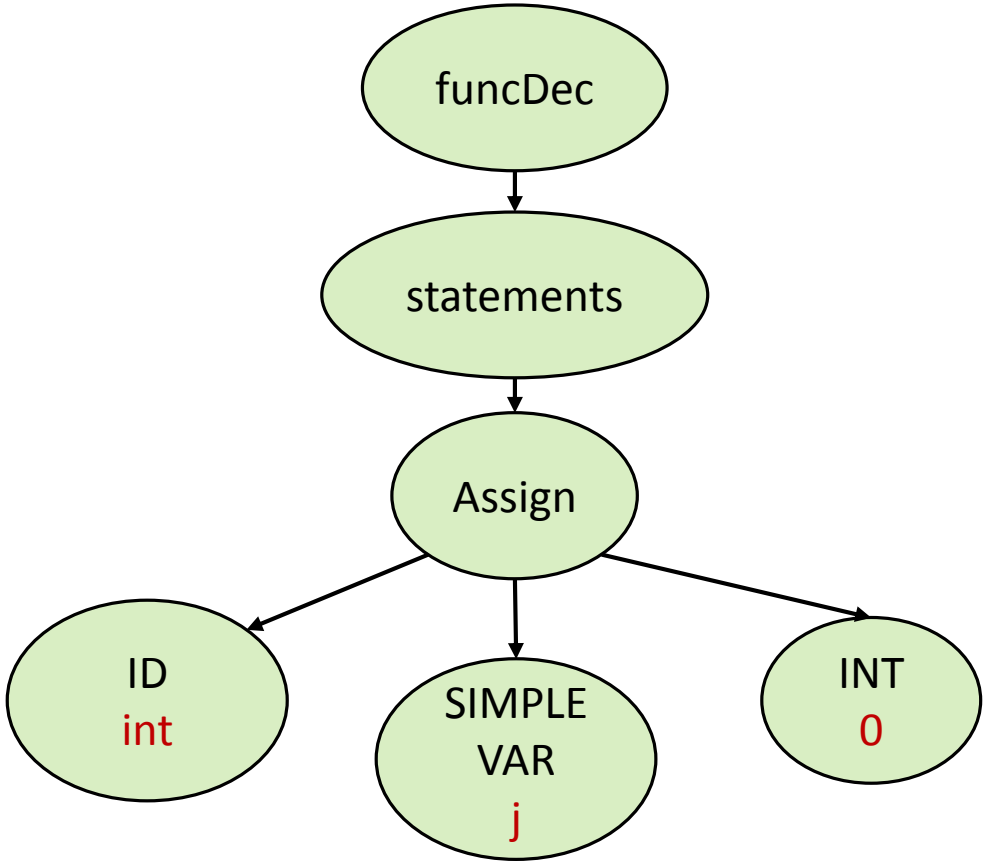
# Symbol Table

- Maintain a stack of scopes
- Each scope maps identifiers to their type information
- 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, …

# Assignments

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

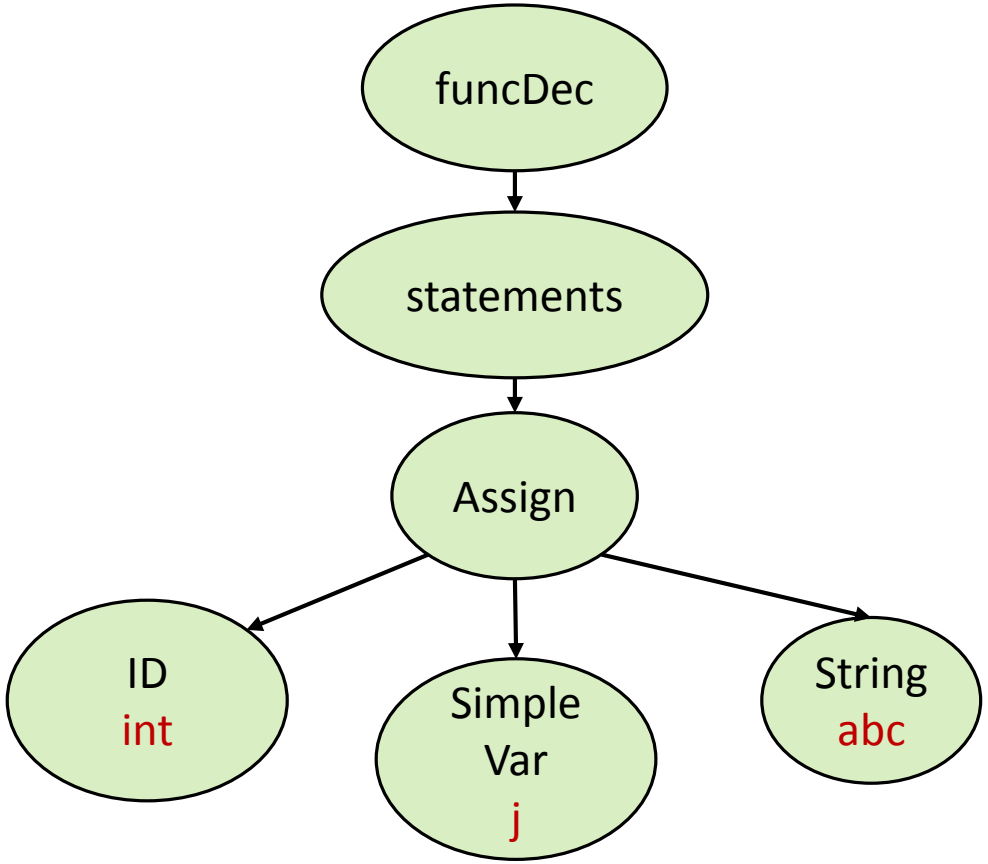
Valid



# Assignments

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

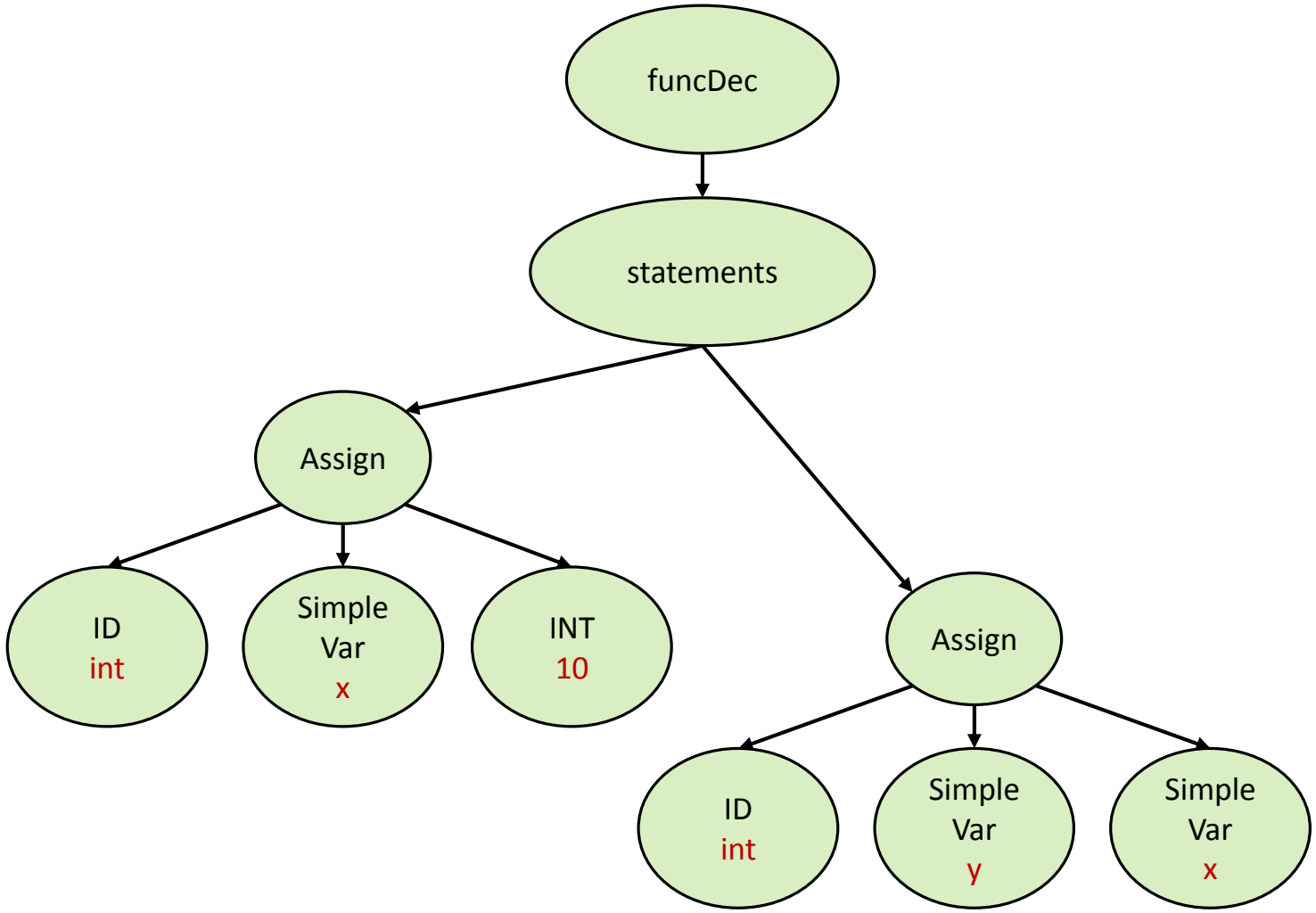
Invalid



# Assignments

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

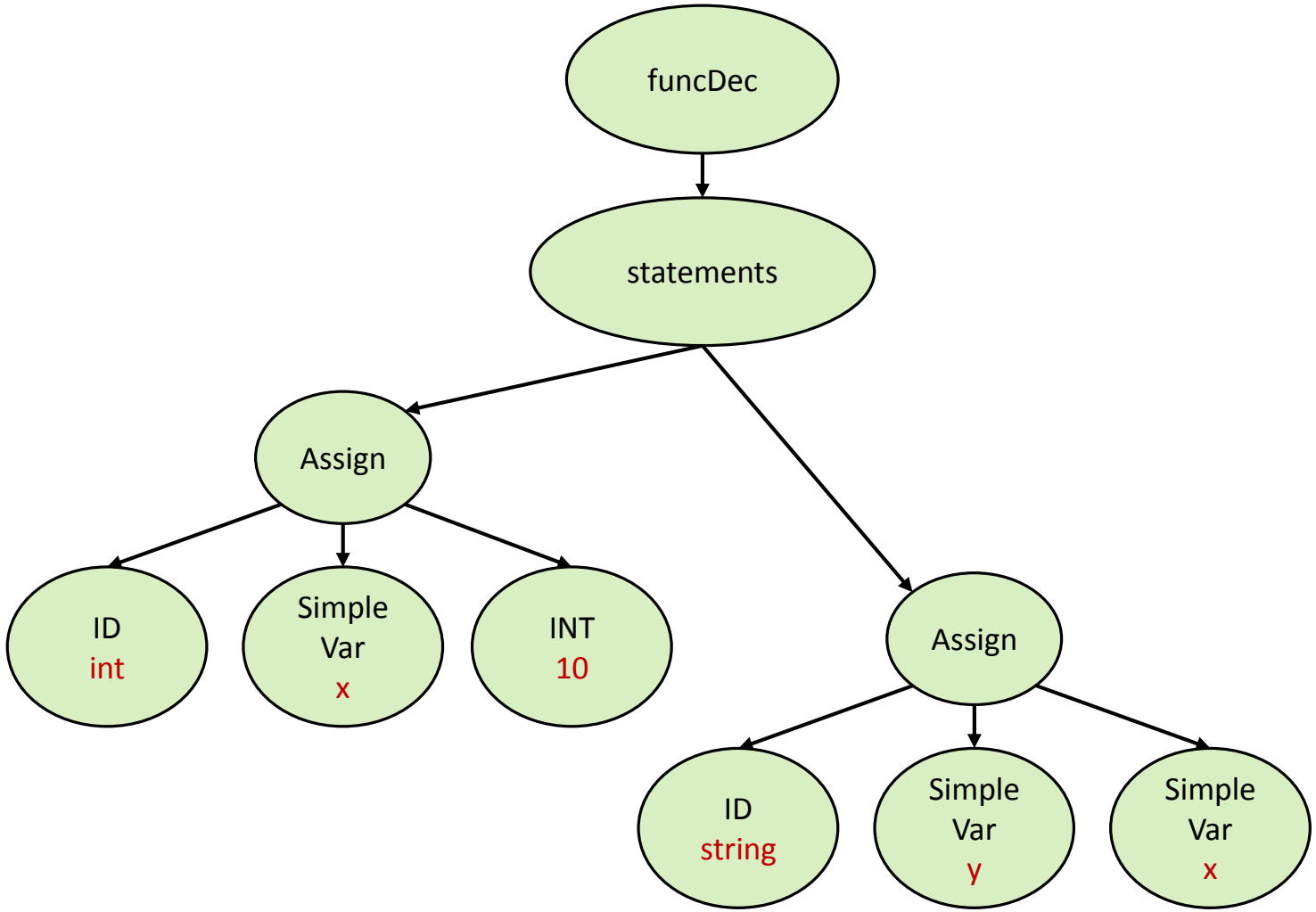
Valid



# Assignments

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

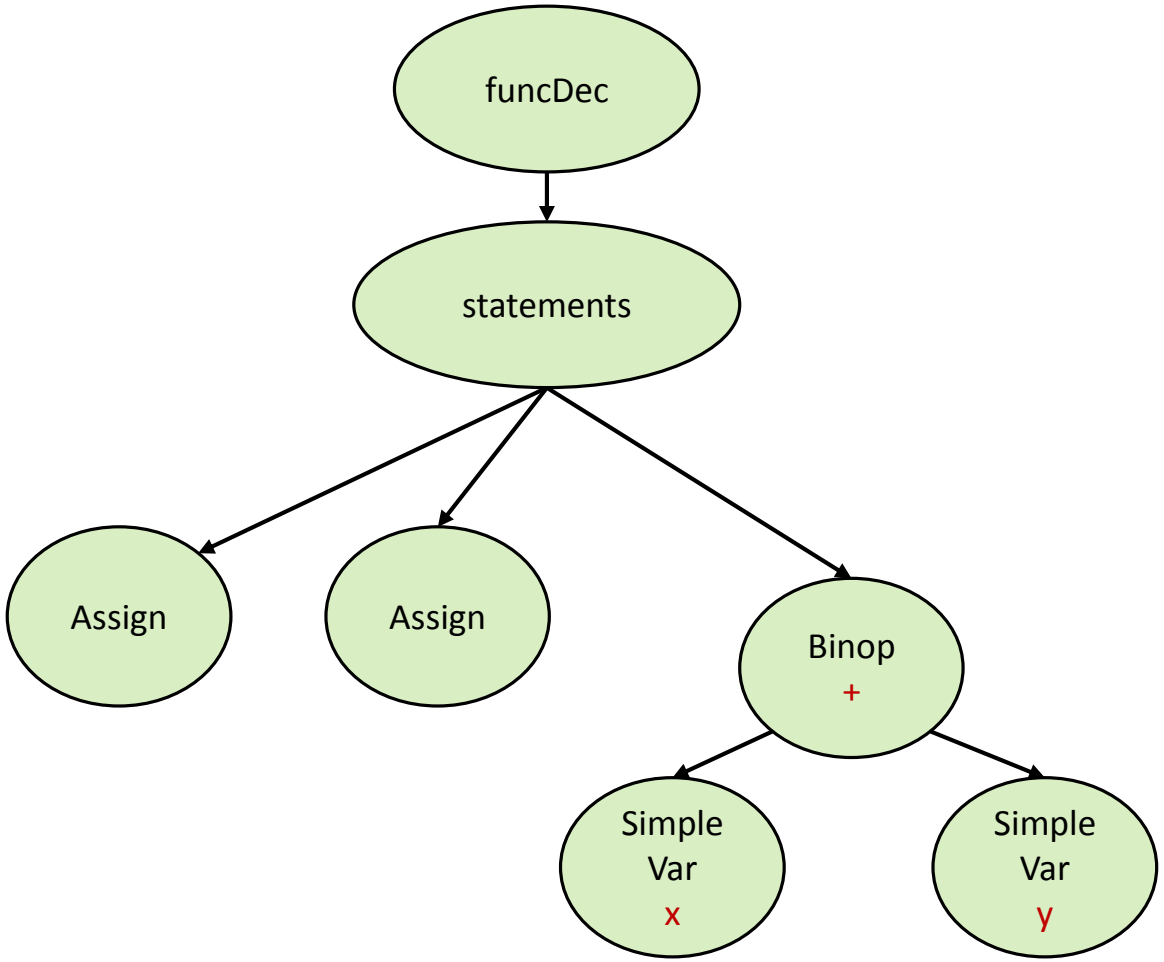
Invalid



# Binary Operations

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

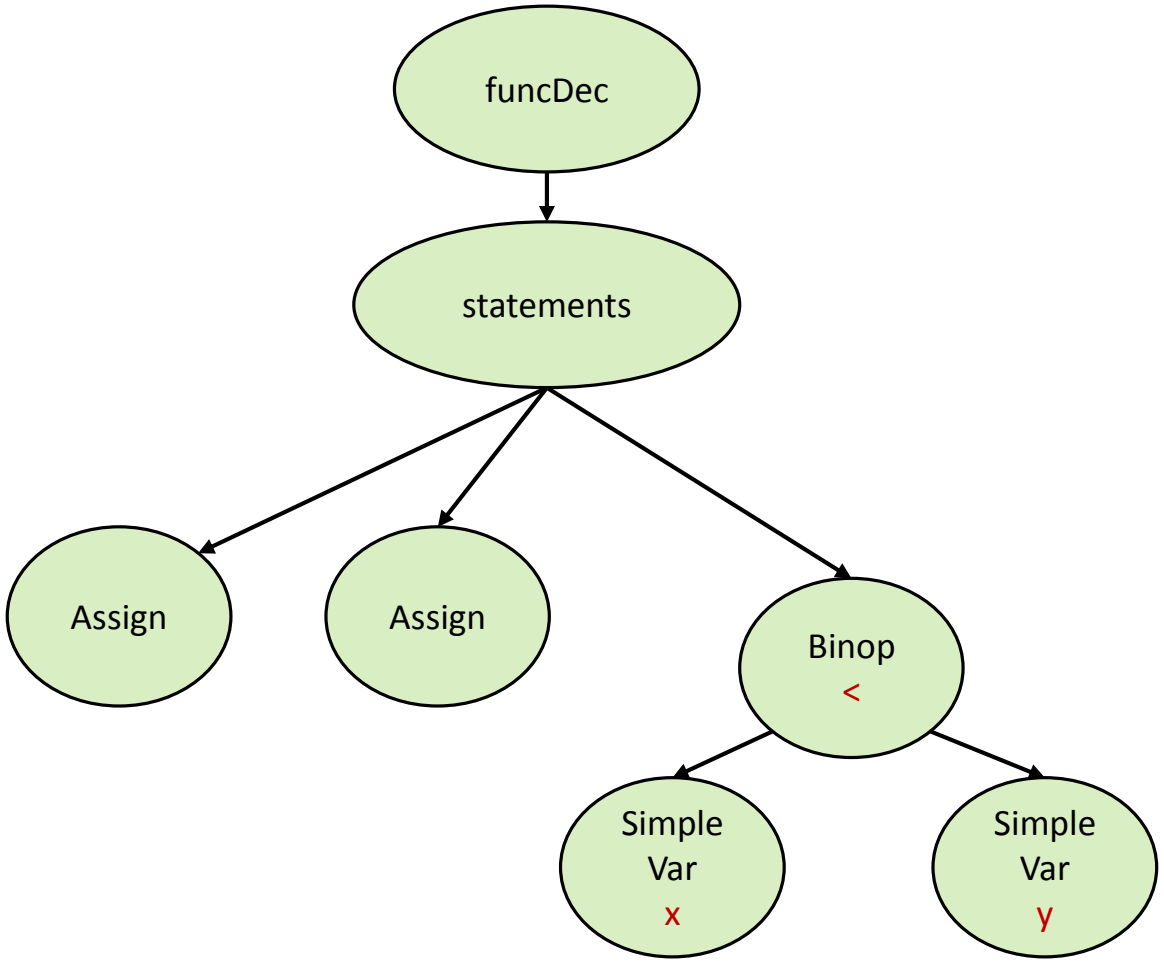
Valid



# Binary Operations

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

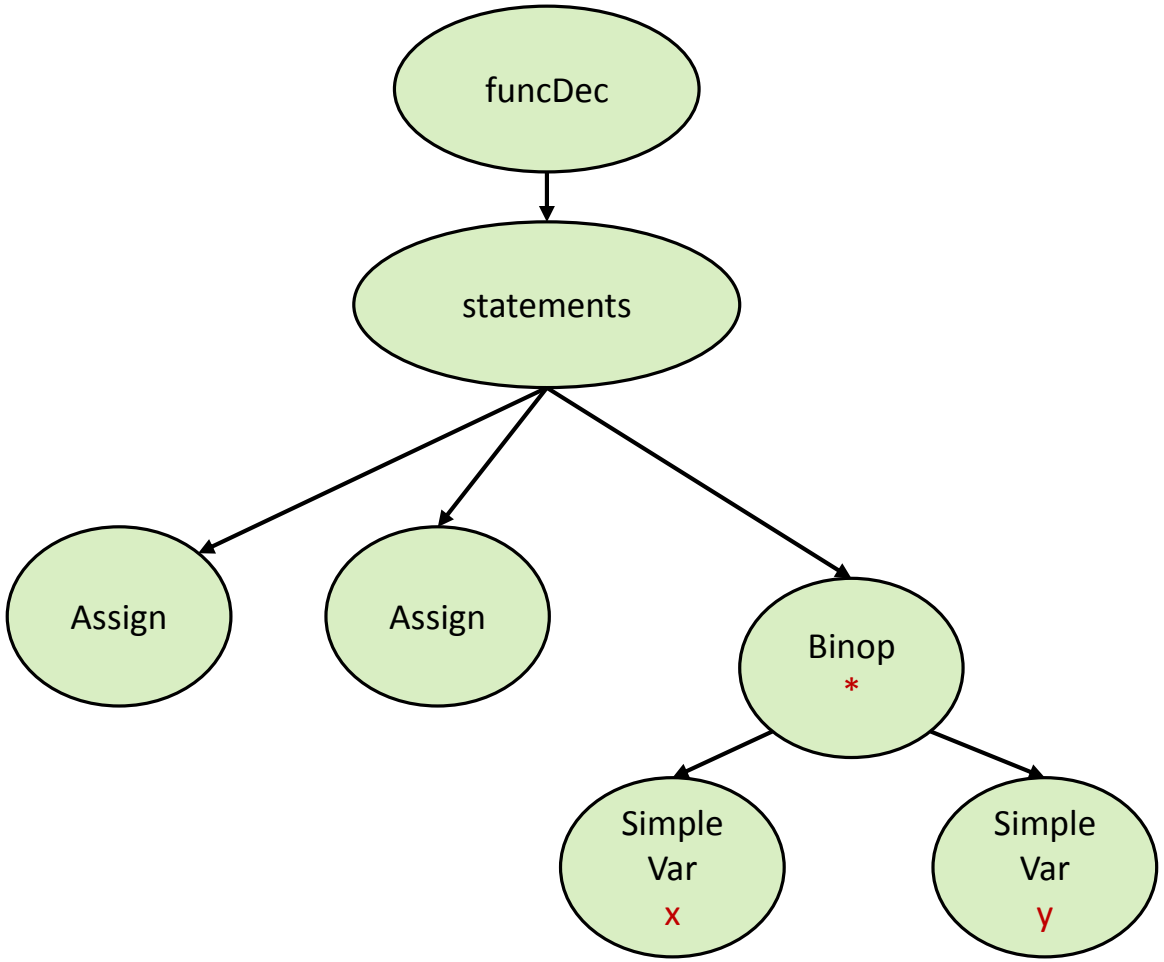
Invalid



# Binary Operations

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

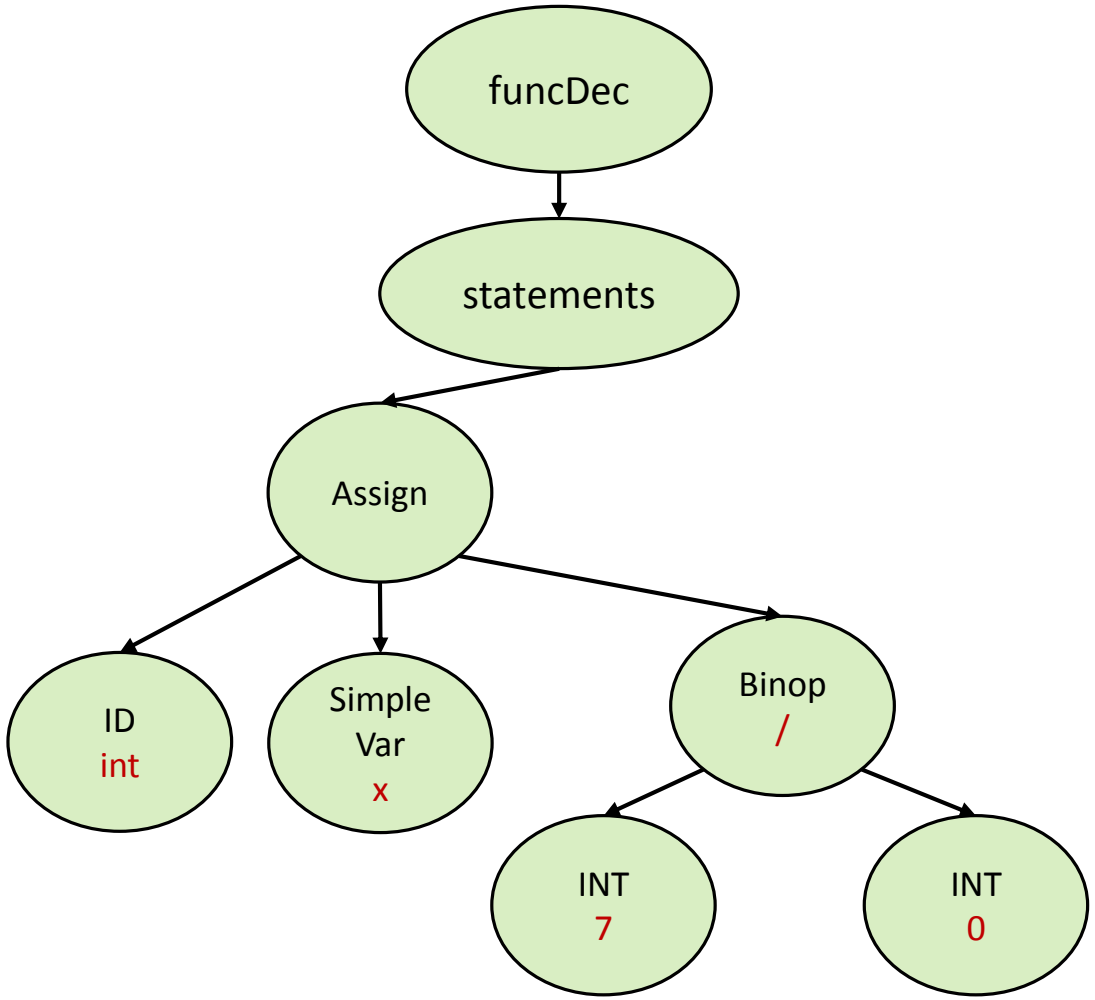
Invalid



# Binary Operations

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

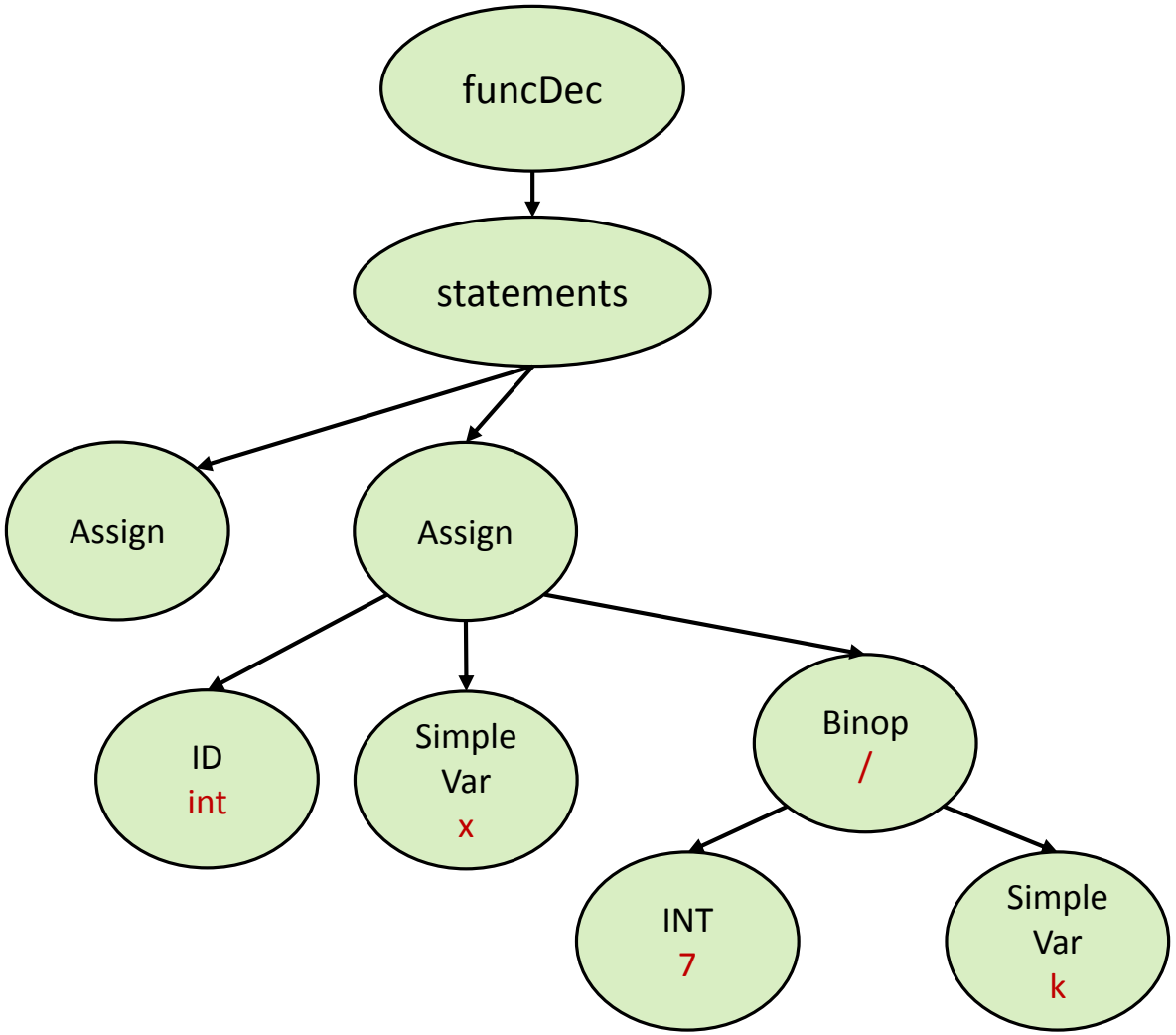
Invalid



# Binary Operations

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

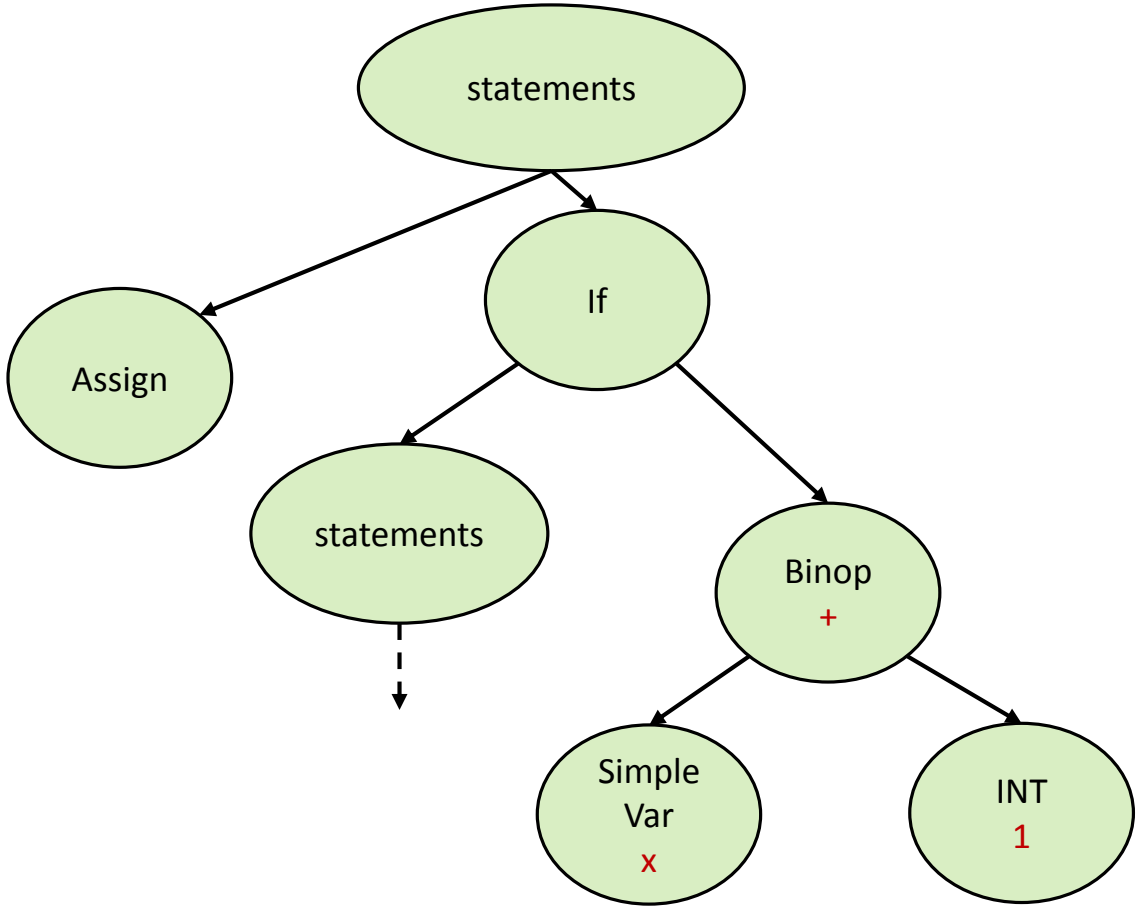
Depends



# If, While, …

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

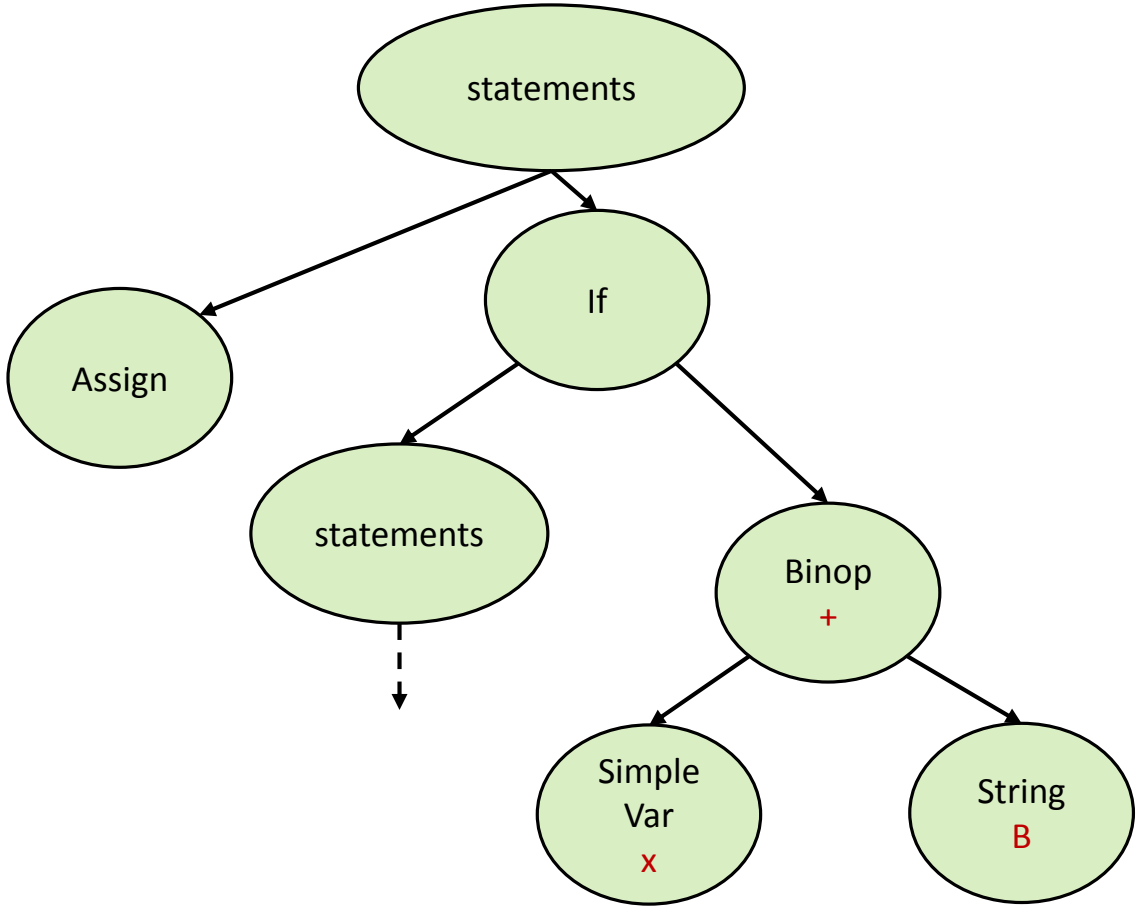
Valid



# If, While, …

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

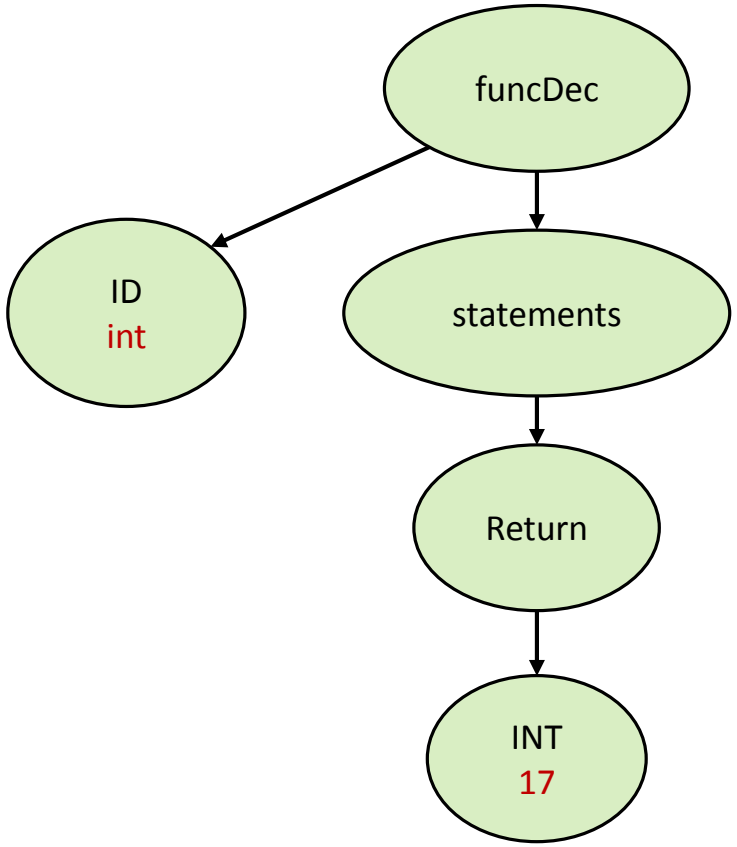
Invalid



# Return Statement

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

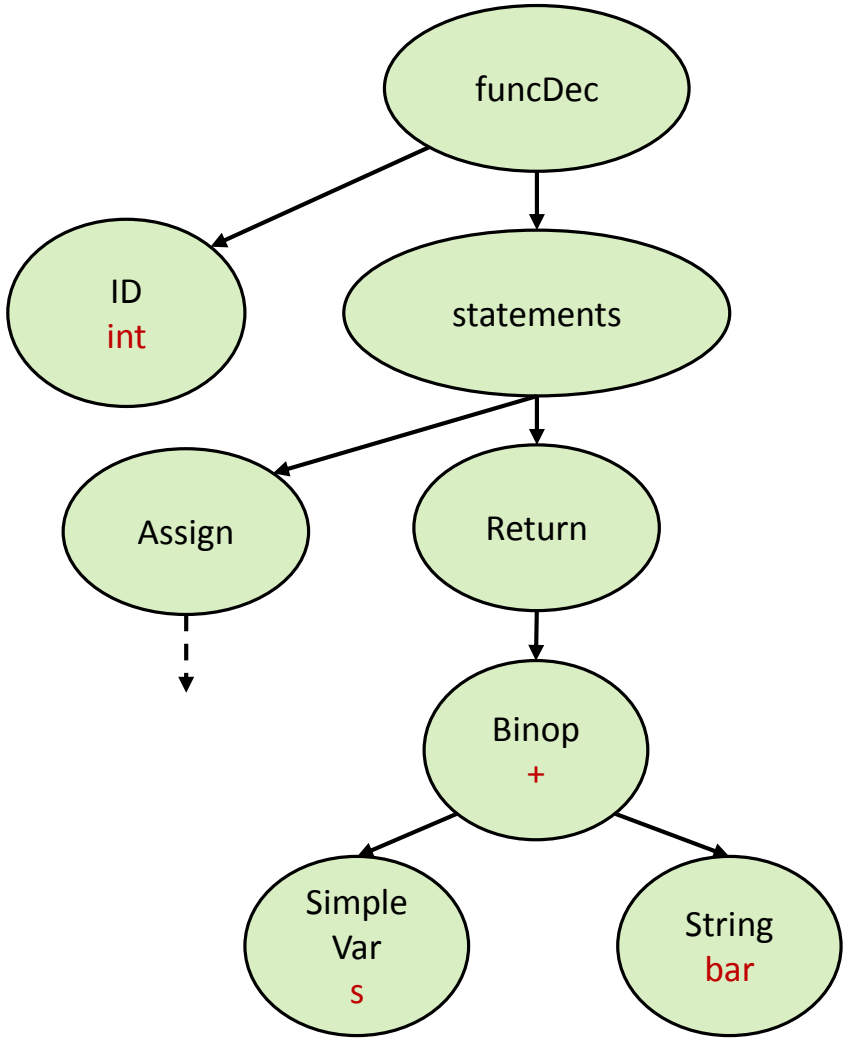
Valid



# Return Statement

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

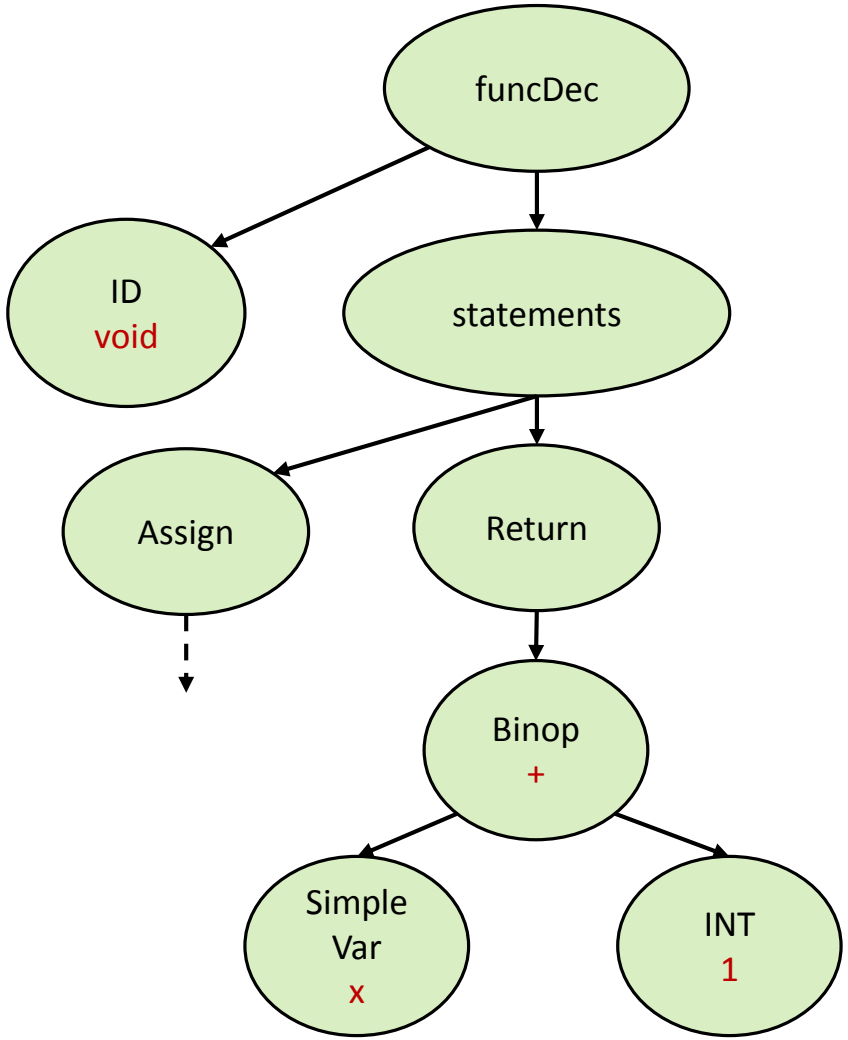
Invalid



# Return Statement

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

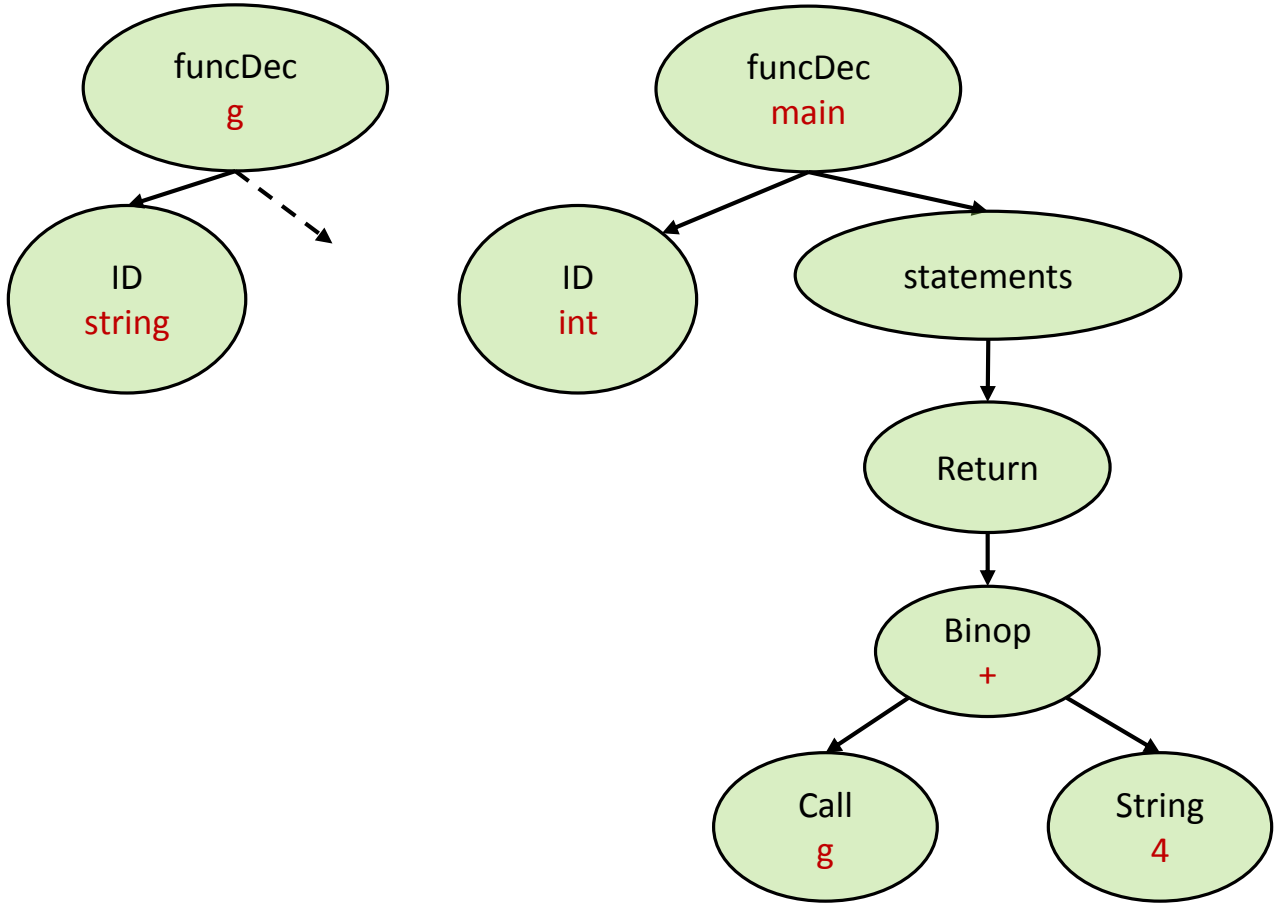
Invalid



# Return Statement

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

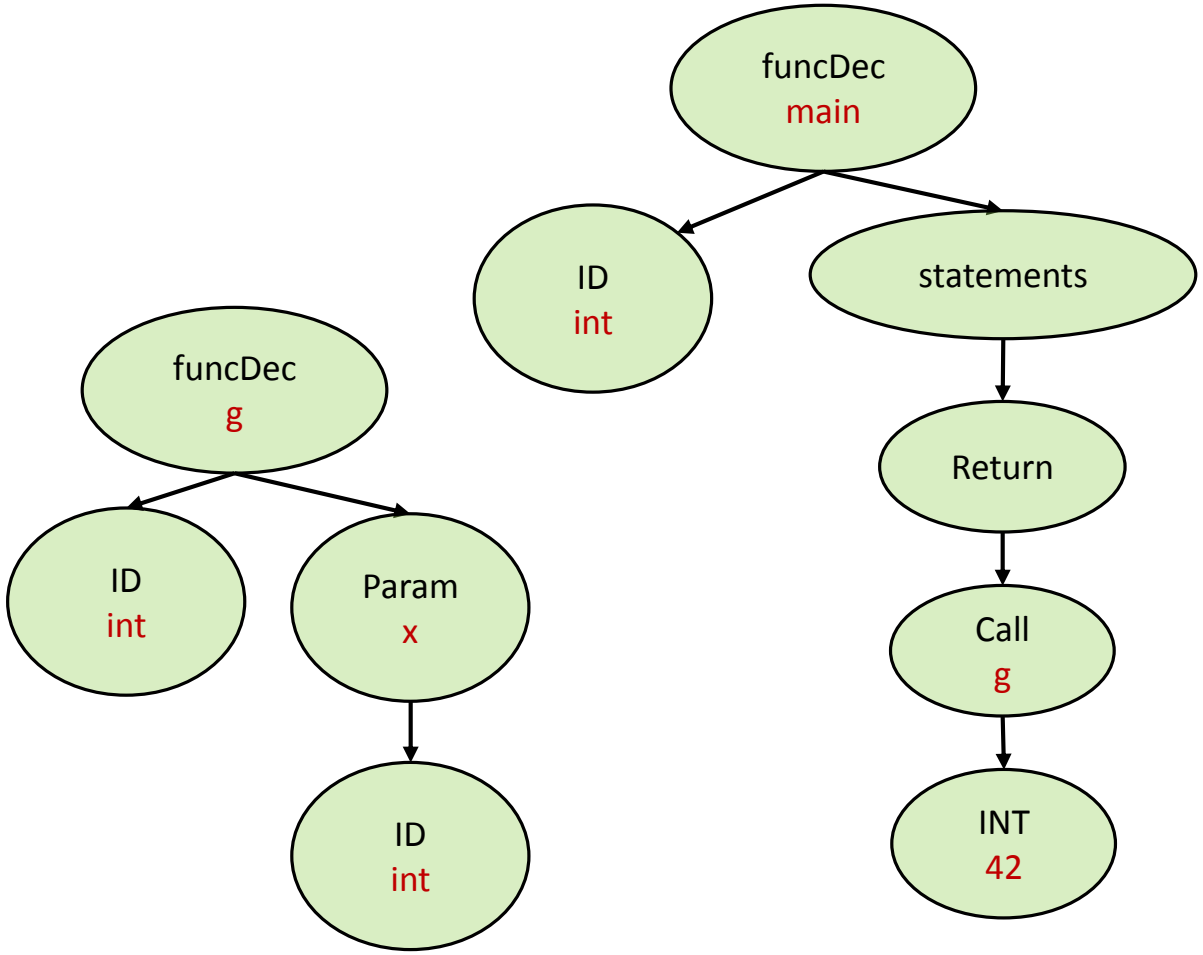
Invalid



# Function Calls

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

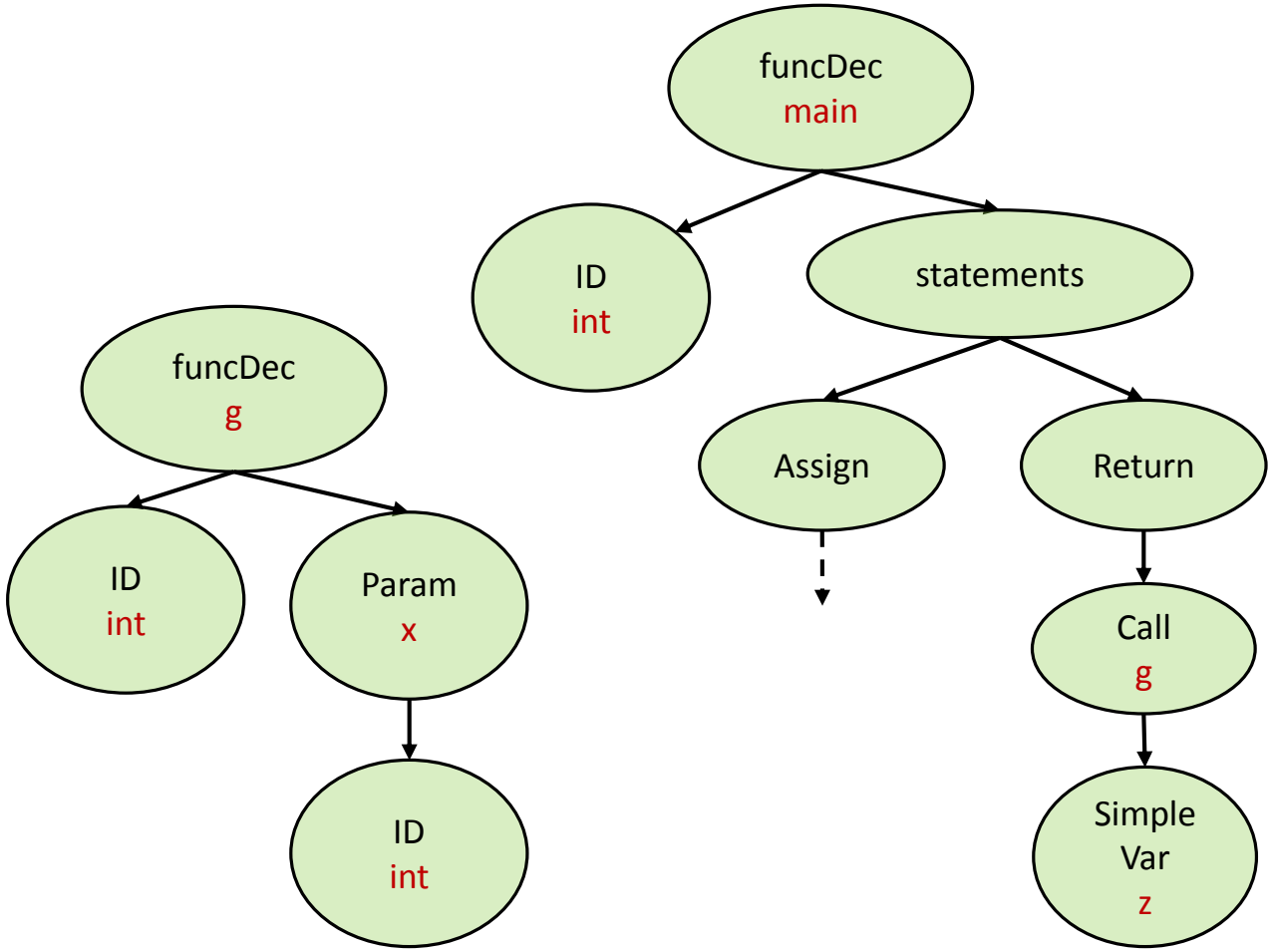
Valid



# Function Calls

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

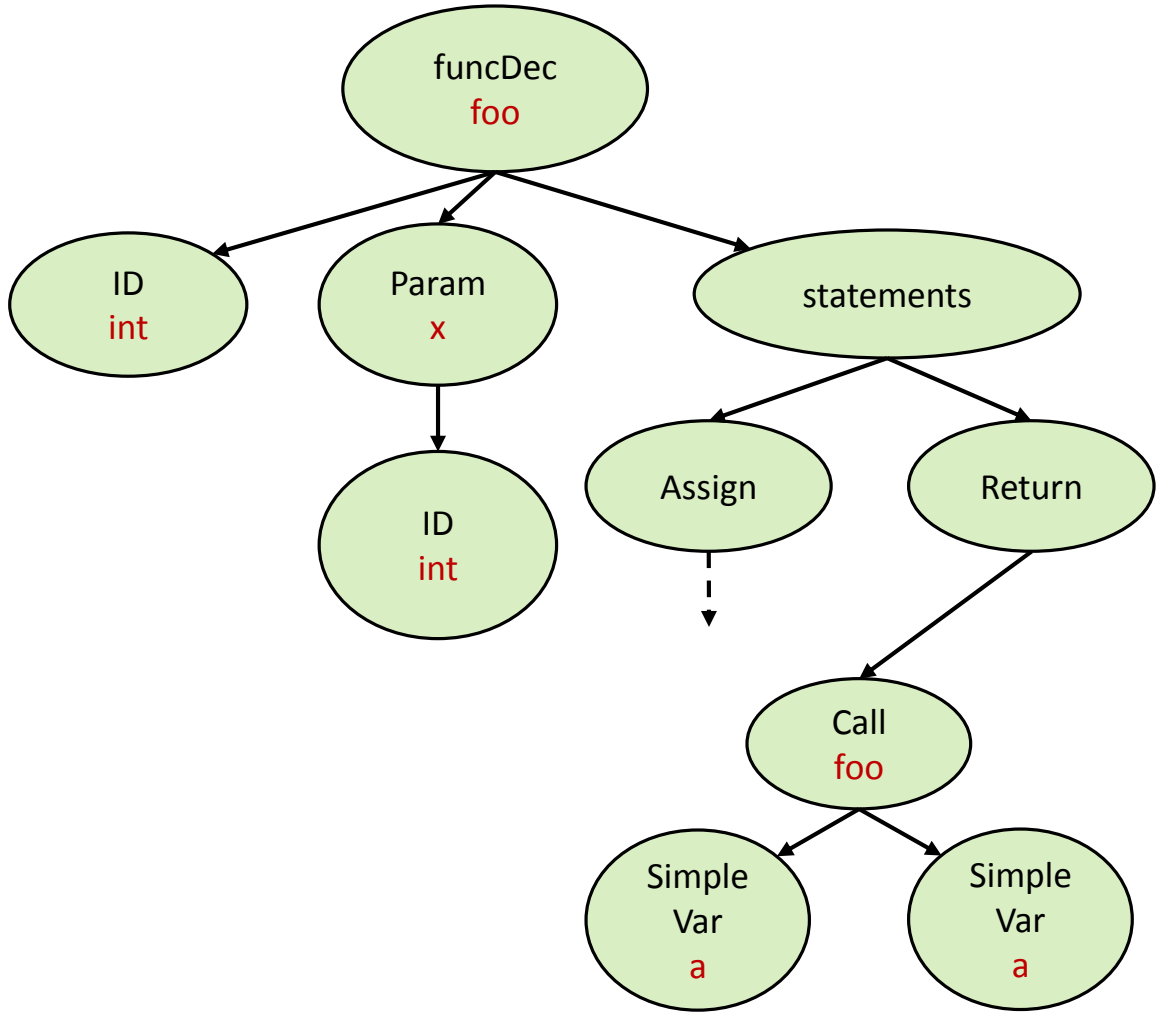
Invalid



# Function Calls

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

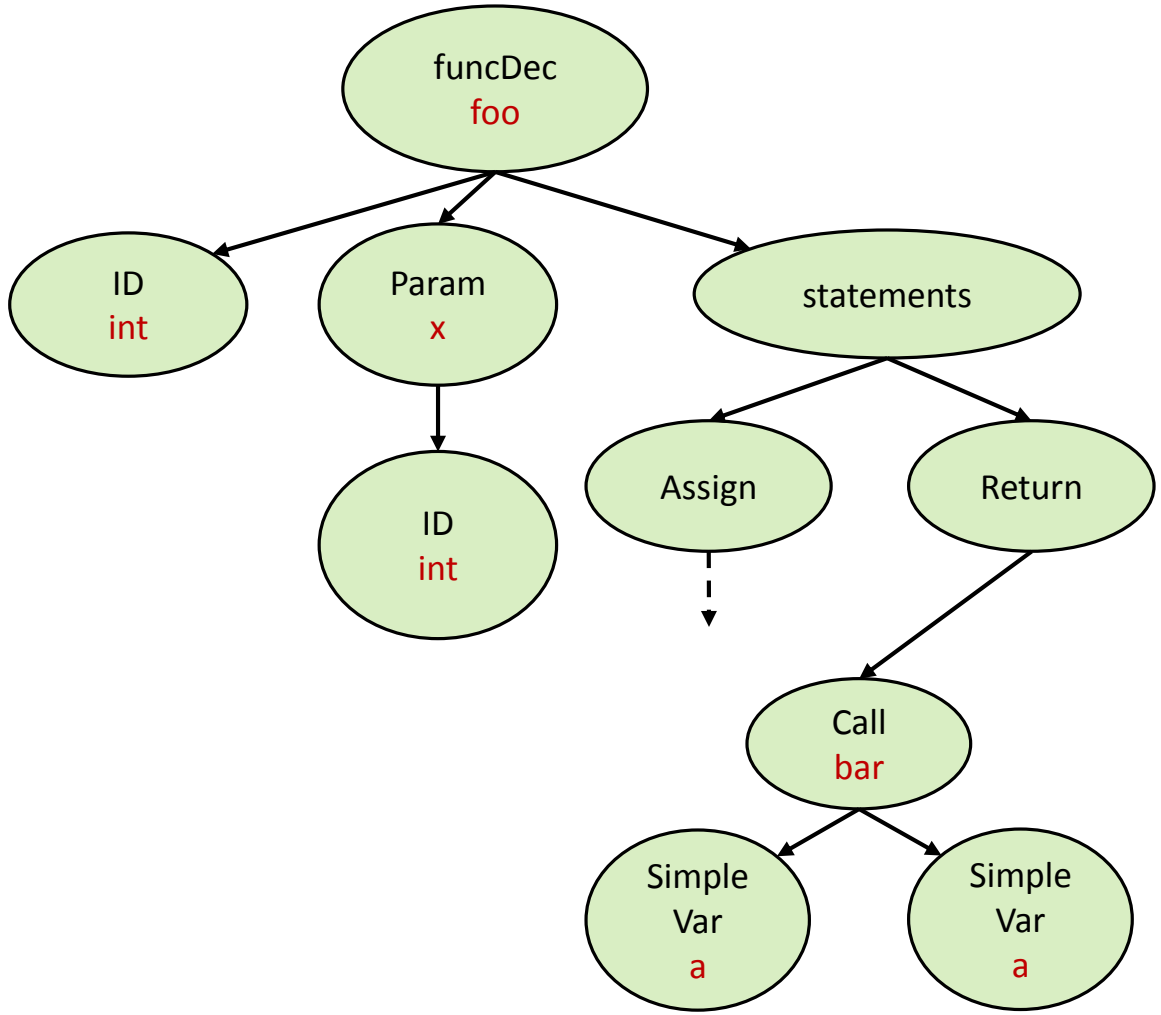
Invalid



# Function Calls

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

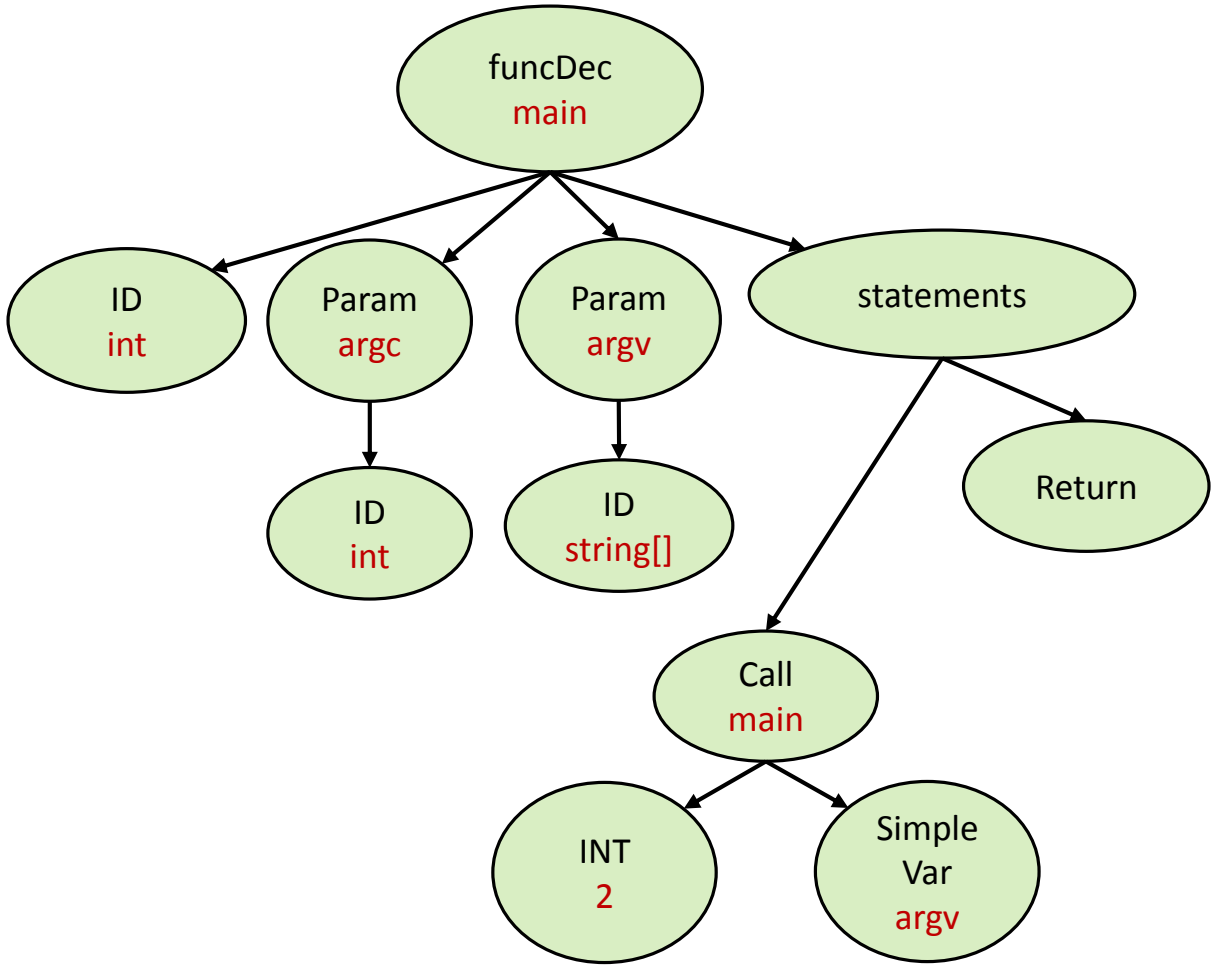
Invalid



# Function Calls

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

Valid

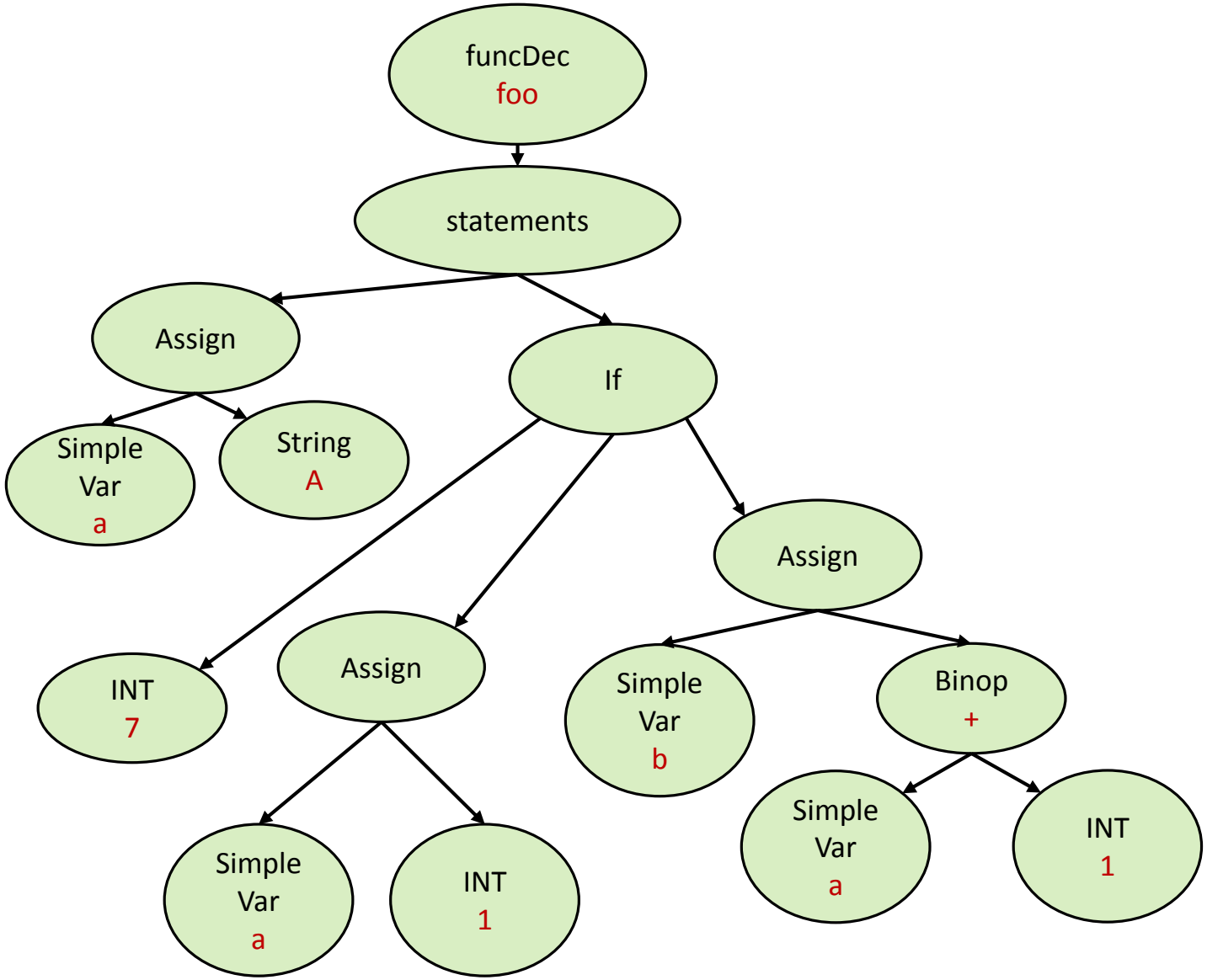


# Arrays

# Scopes

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

Valid



# Scopes

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

Invalid