



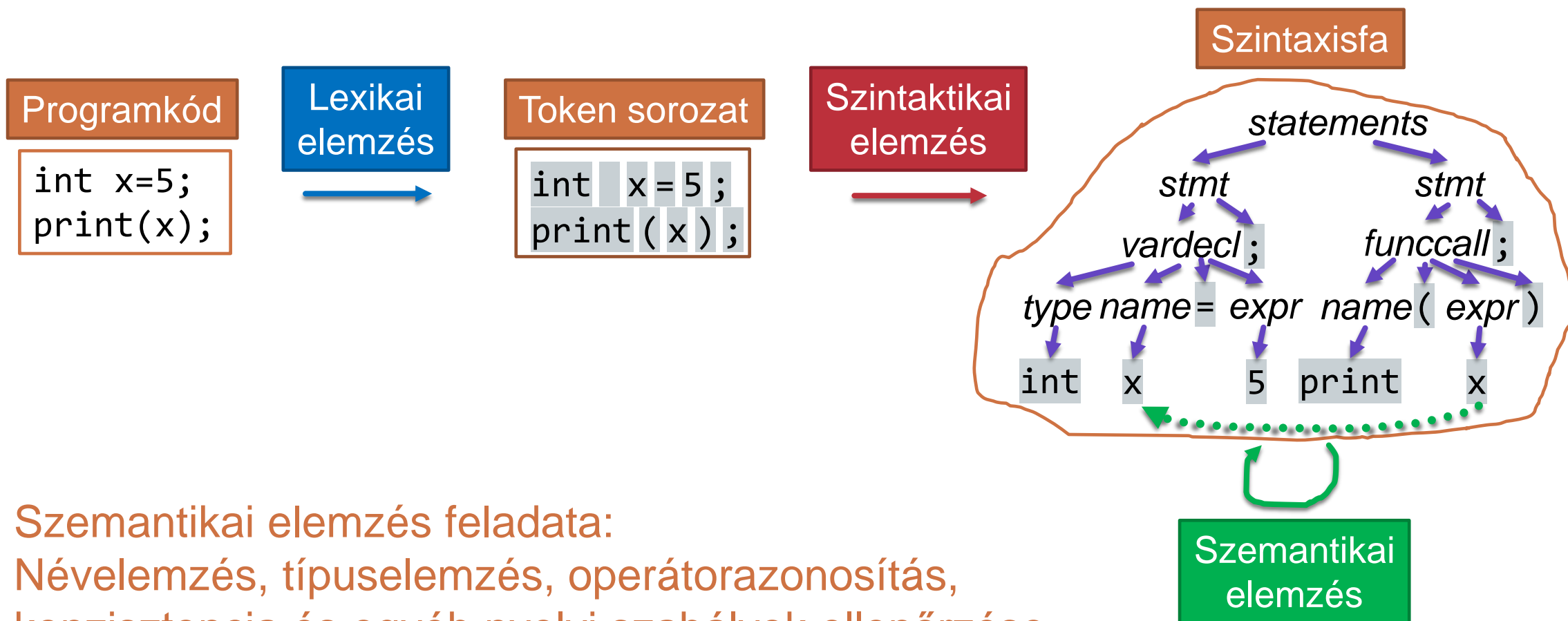
Modellalapú szoftverfejlesztés

IV. előadás

Szemantikai elemzés

Dr. Simon Balázs

Fordító front-end



A mai előadás: Szemantikai elemzés

I. Szemantikai elemzés

II. Attribútumnyelvtanok

III. Névelemzés

IV. Típuselemzés és operátorazonosítás

V. Egyéb nyelvi szabályok



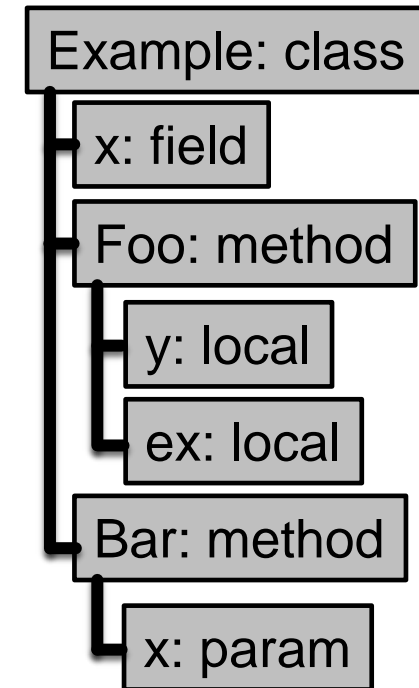
Szemantikai elemzés példa: szimbólumtábla felépítése

```
public abstract class Example
{
    private int x;

    public void Foo()
    {
        int y;
        if (!flag)
        {
            x = 3 + y;
            var ex = new Example();
        }
    }

    public int Bar(string x)
    {
        return "hello" + x;
    }
}
```

Szimbólumtábla:



Szemantikai elemzés példa: névelemzés

```
public abstract class Example
{
    private int x;

    public void Foo()
    {
        int y;
        if (!flag)
        {
            x = 3 + y;
            var ex = new Example();
        }
    }

    public int Bar(string x)
    {
        return "hello" + x;
    }
}
```

flag nincs deklarálva

Szimbólumtábla:

Example: class

x: field

Foo: method

y: local

ex: local

Bar: method

x: param

Szemantikai elemzés példa: típuselemzés

```
public abstract class Example
{
    private int x;

    public void Foo()
    {
        int y;
        if (!flag)
        {
            x = 3 + y;
            var ex = new Example();
        }
    }

    public int Bar(string x)
    {
        return "hello" + x;
    }
}
```

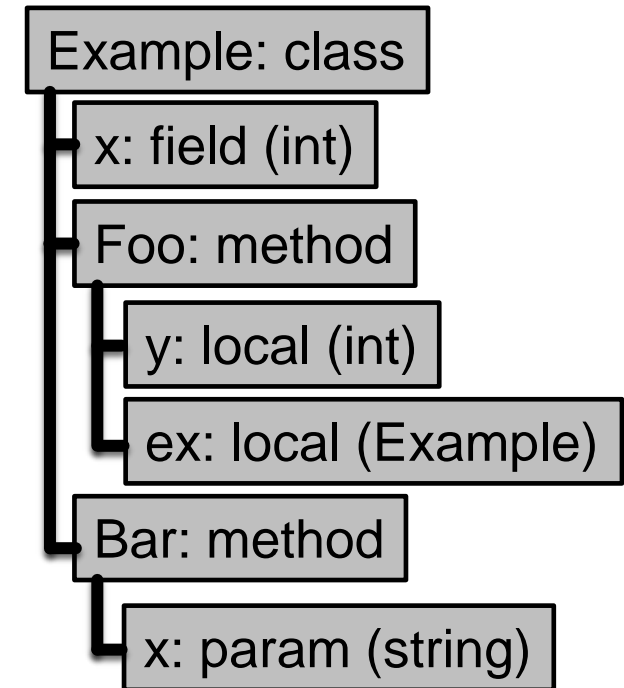
ex típusa Example



var ex = new Example();

visszatérési érték típusa hibás

Szimbólumtábla:



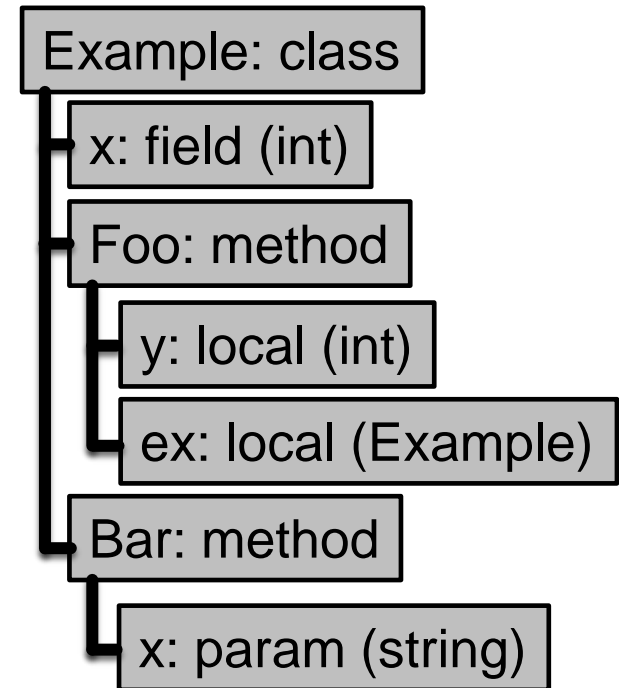
Szemantikai elemzés példa: operátorazonosítás

```
public abstract class Example
{
    private int x;

    public void Foo()
    {
        int y;
        if (!flag)
        {
            x = 3 + y;
            var ex = new Example();
        }
    }

    public int Bar(string x)
    {
        return "hello" + x;
    }
}
```

Szimbólumtábla:



int operator+(int,int)

string operator+(string,string)

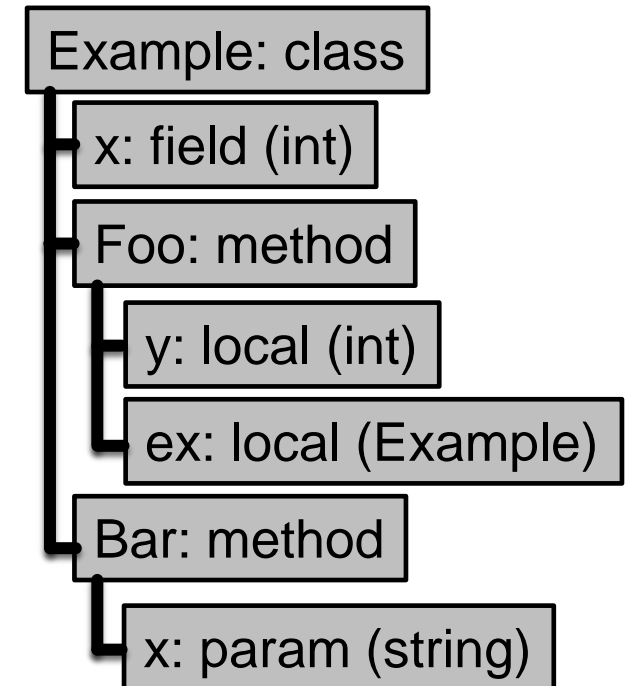
Szemantikai elemzés példa: egyéb nyelvi szabályok

```
public abstract class Example
{
    private int x;

    public void Foo()
    {
        int y;
        if (!flag)
        {
            x = 3 + y;
            var ex = new Exam
        }
    }

    public int Bar(string x)
    {
        return "hello" + x;
    }
}
```

Szimbólumtábla:



Example nem példányosítható

Szemantikai elemzés

■ Szemantikai elemzés

- > szimbólumtábla építése
- > névelemzés
- > típuselemzés
- > operátorok azonosítása
- > konzisztencia ellenőrzése
- > egyéb nyelvi szabályok ellenőrzése

■ Miért nem a szintaktikai elemzés része?

- > a programnyelvek kontextus-függőek (nem elég a CF nyelvtan a leírásukhoz)
- > kódban később szereplő definíciók, kereszthivatkozások, túl sok összefüggés

Szimbólumtábla

- A fordító adatbázisa
- Összes szimbólum definíciója
 - > név
 - > jelentés
 - > kontextus
- Cél:
 - > szimbólumok keresésének támogatása
 - > szimbólumok egyezőségének eldöntése

A mai előadás: Szemantikai elemzés

I. Szemantikai elemzés

II. Attribútumnyelvtanok

III. Névelemzés

IV. Típuselemzés és operátorazonosítás

V. Egyéb nyelvi szabályok



Attribútum nyelvtanok

- Szemantikai elemzéshez extra információra van szükség a szintaxisfában
- Ötlet: rendeljünk attribútumokat a szintaxisfa csúcsaihoz
 - > attribútum definíció: név és egy kifejezés
- Két attribútum fajta:
 - > örökölt (inherited): fentről lefelé (top-down) kiértékelés
 - a szabályok jobb oldalán található nemterminálisokra kell definiálni
 - > szintetizált (synthesized): alulról felfelé (bottom-up) kiértékelés
 - a szabályok bal oldalán található nemterminálisokra kell definiálni

Attribútum nyelvtan példa

CF nyelvtan:

```
A → T x = E
E → E+C | C
C → 1 | "a"
T → int | string
```

szintetizált (bottom-up)

$E \rightarrow C$

$E.type = C.type$

$C.expType = E.expType$

örökölt (top-down)

$C \rightarrow 1$

$C.type = int$

$C \rightarrow "a"$

$C.type = string$

$T \rightarrow int$

$T.type = int$

$T \rightarrow string$

$T.type = string$

Attribútum nyelvtan:

$A \rightarrow T x = E$

$E.expType = T.type$

$T.expType = any$

$E \rightarrow E+C$

$E[1].op = GetOperator(+, E[2].type, C.type)$

$E[1].type = E[1].op.type$

$E[2].expType = E[1].op.expType$

$C.expType = E[1].op.expType$

Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

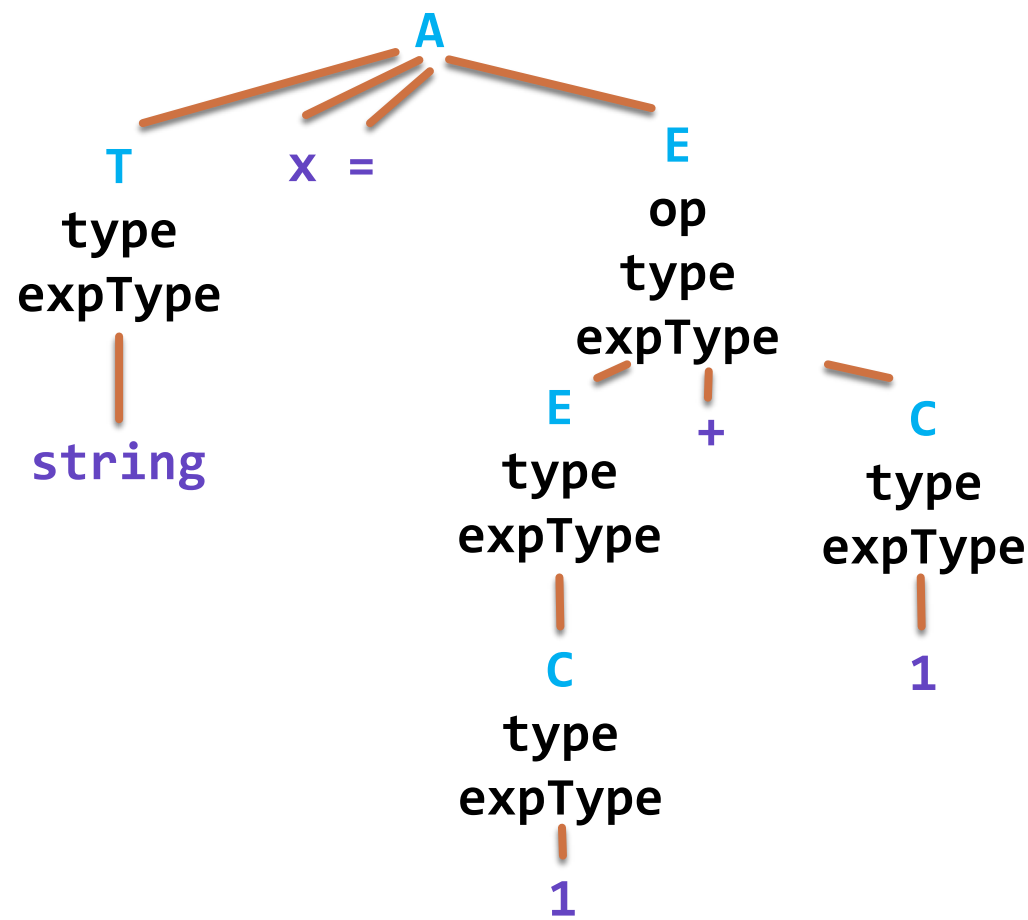
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

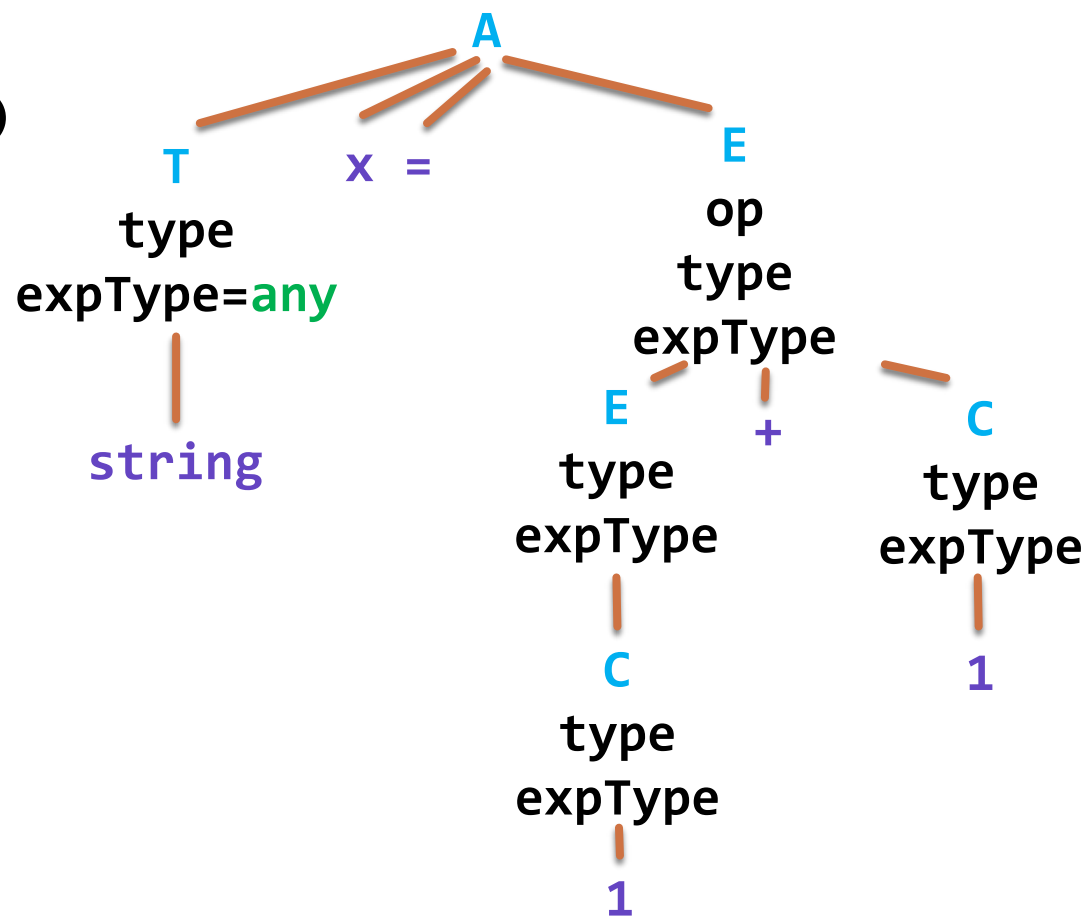
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

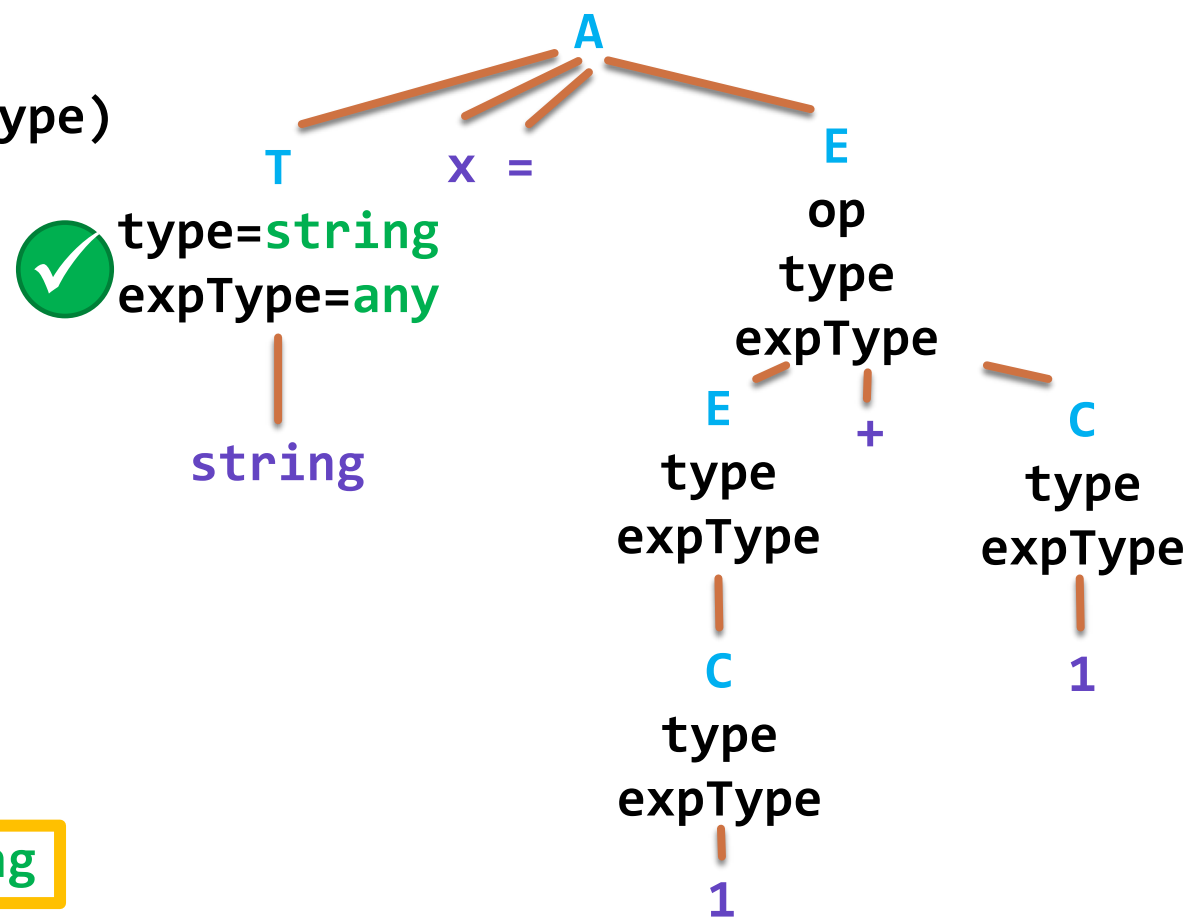
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

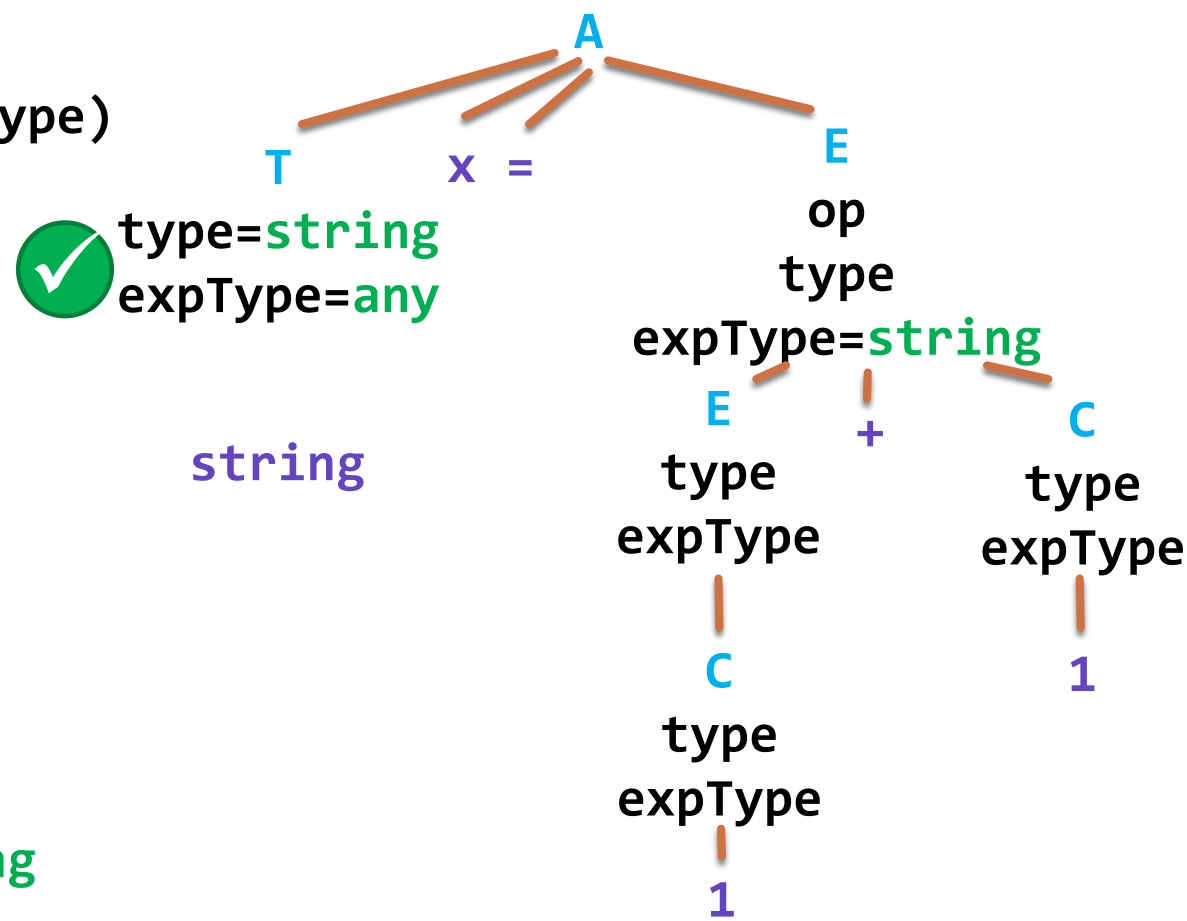
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

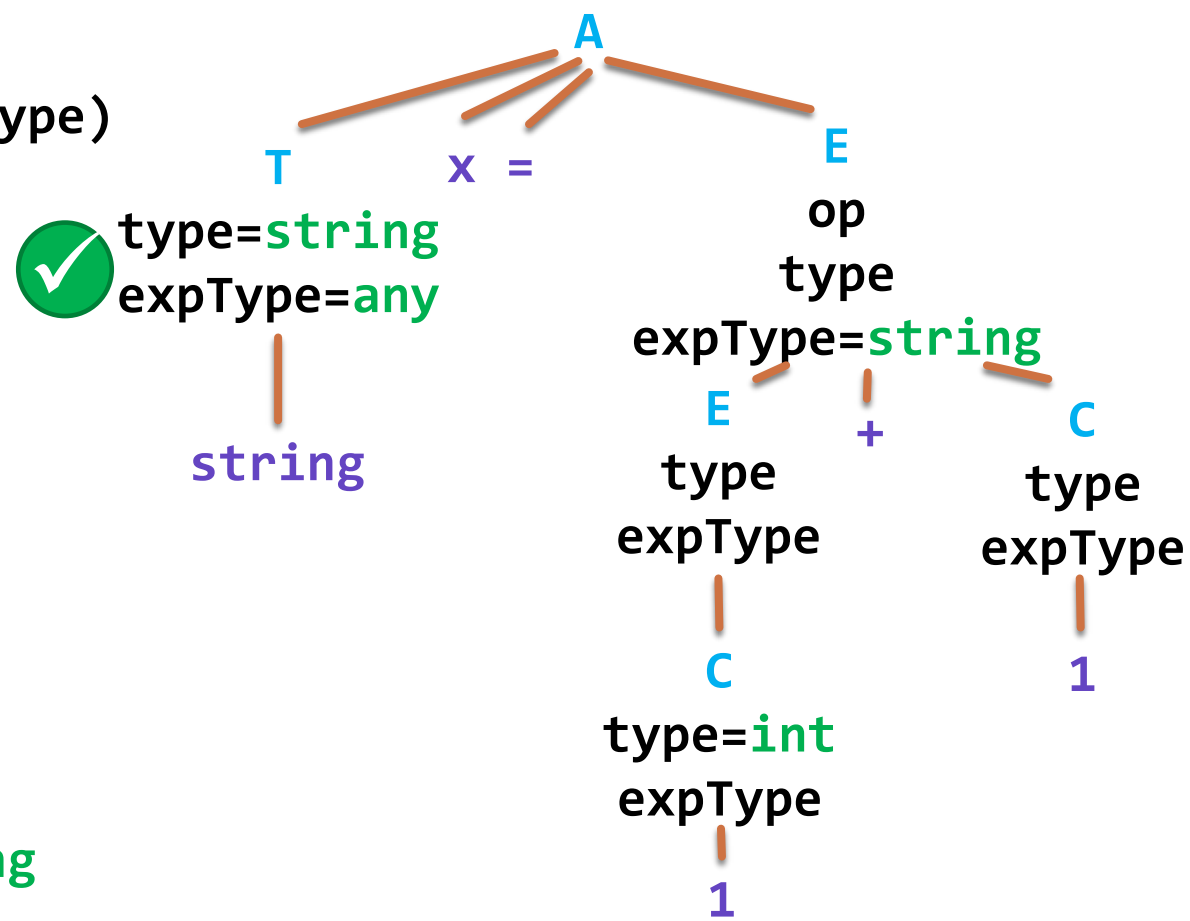
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

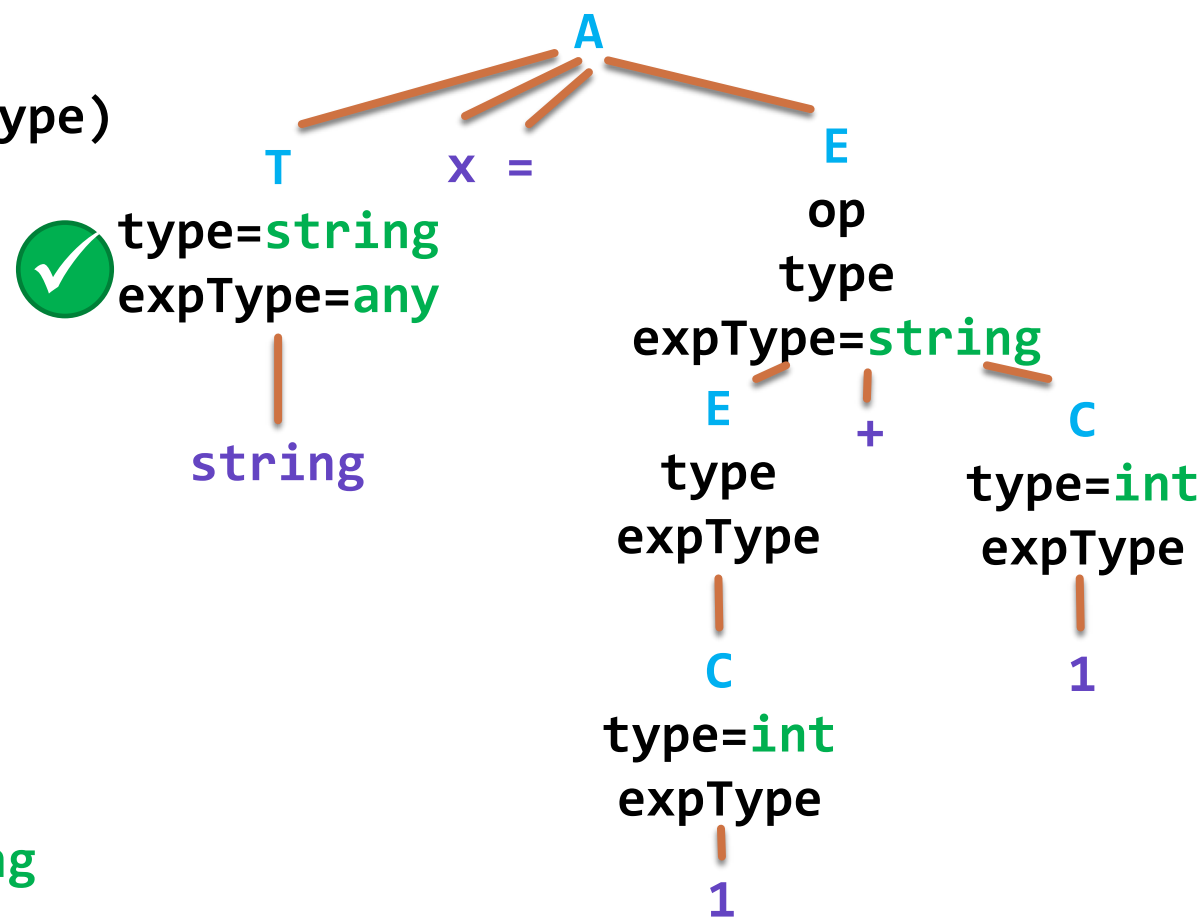
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

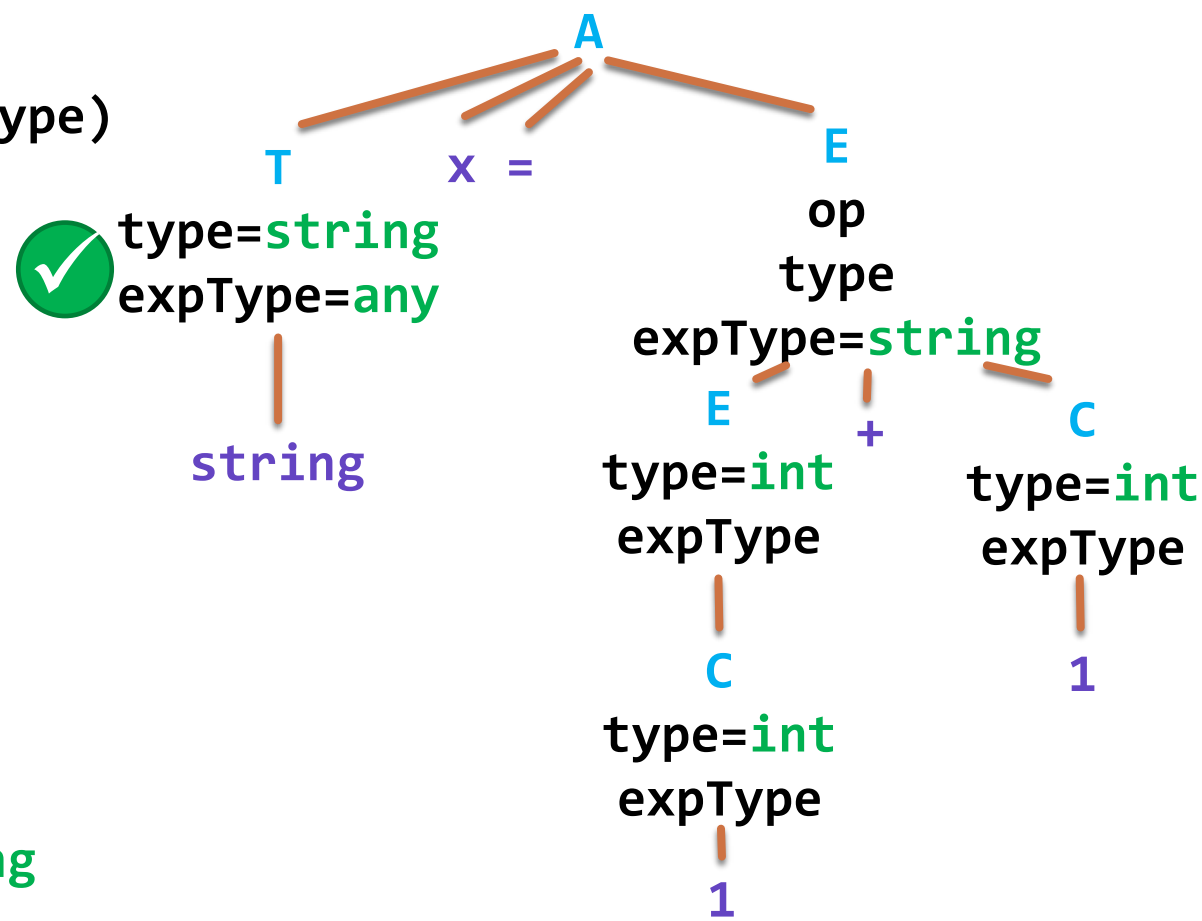
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

```
E.expType = T.type
```

$$E \rightarrow E + C$$

```
E[1].type = E[1].op.type
```

```
C.expType = E[1].op.expType
```

E.type = C.type

C → 1

c → "a"

T → int

T → string

```
T.type = string
```

```
string x = 1+1
```

```

graph TD
    A[A] --- T[T]
    A --- x["x ="]
    A --- E1[E]
    T --- string[string]
    E1 --- op["+"]
    E1 --- E2[E]
    E1 --- C1[C]
    E2 --- type1["type = int"]
    E2 --- expType1["expType = string"]
    type1 --- expType2["expType"]
    expType1 --- plus["+"]
    expType2 --- C2[C]
    C2 --- type2["type = int"]
    C2 --- expType3["expType"]
    type2 --- expType4["expType"]
    expType3 --- 1_1["1"]
    expType4 --- 1_2["1"]
    
```

The diagram illustrates a parse tree for the expression `x = string + (int + int)`. The root node `A` branches into `T`, `x =`, and `E`. The `T` node branches into `string`. The `E` node branches into `op = +`, `E`, and `C`. The first `E` node branches into `type = int` and `expType = string`. The `type = int` node branches into `expType`, which then branches into `C` and `1`. The `C` node branches into `type = int` and `expType`, which then branches into `1`. The `C` node branches into `type = int` and `expType`, which then branches into `1`.

Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op.type}$

$E[2].\text{expType} = E[1].\text{op.expType}$

$C.\text{expType} = E[1].\text{op.expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

$T \rightarrow \text{string}$

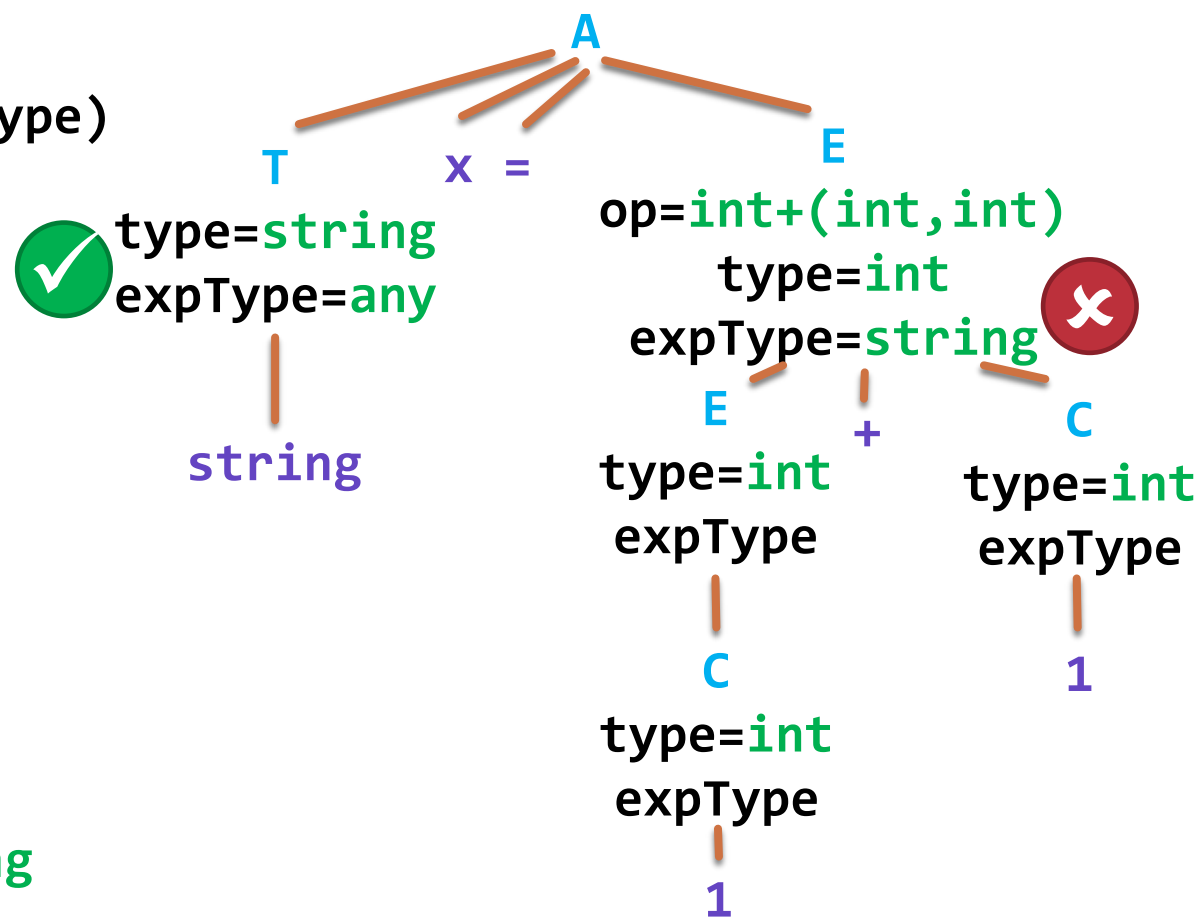
$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Hiba: string típusú kifejezést várunk!

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

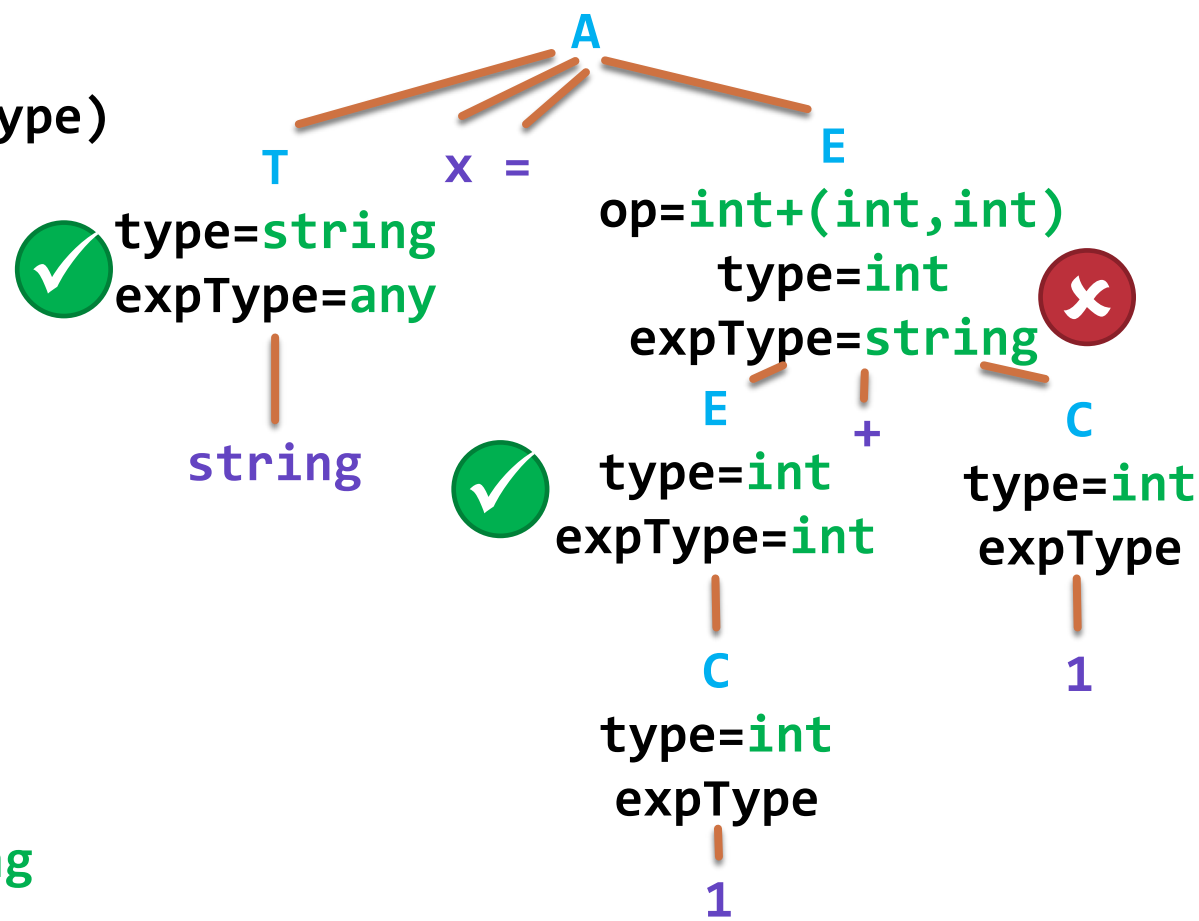
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

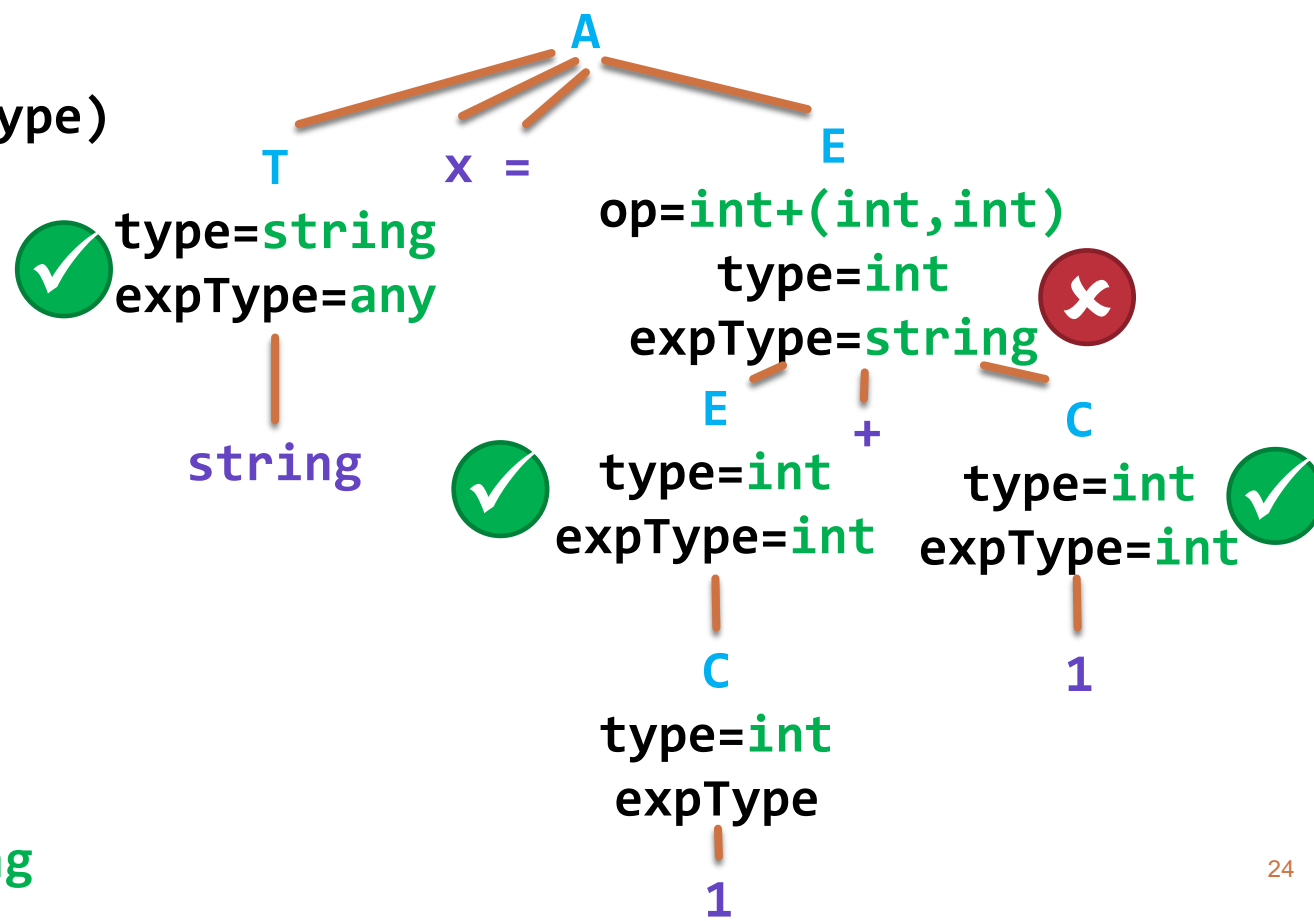
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Példa: attribútumok számítása

$A \rightarrow T \ x = E$

$E.\text{expType} = T.\text{type}$

$T.\text{expType} = \text{any}$

$E \rightarrow E + C$

$E[1].\text{op} = \text{GetOperator}(+, E[2].\text{type}, C.\text{type})$

$E[1].\text{type} = E[1].\text{op}.\text{type}$

$E[2].\text{expType} = E[1].\text{op}.\text{expType}$

$C.\text{expType} = E[1].\text{op}.\text{expType}$

$E \rightarrow C$

$E.\text{type} = C.\text{type}$

$C.\text{expType} = E.\text{expType}$

$C \rightarrow 1$

$C.\text{type} = \text{int}$

$C \rightarrow \text{"a"}$

$C.\text{type} = \text{string}$

$T \rightarrow \text{int}$

$T.\text{type} = \text{int}$

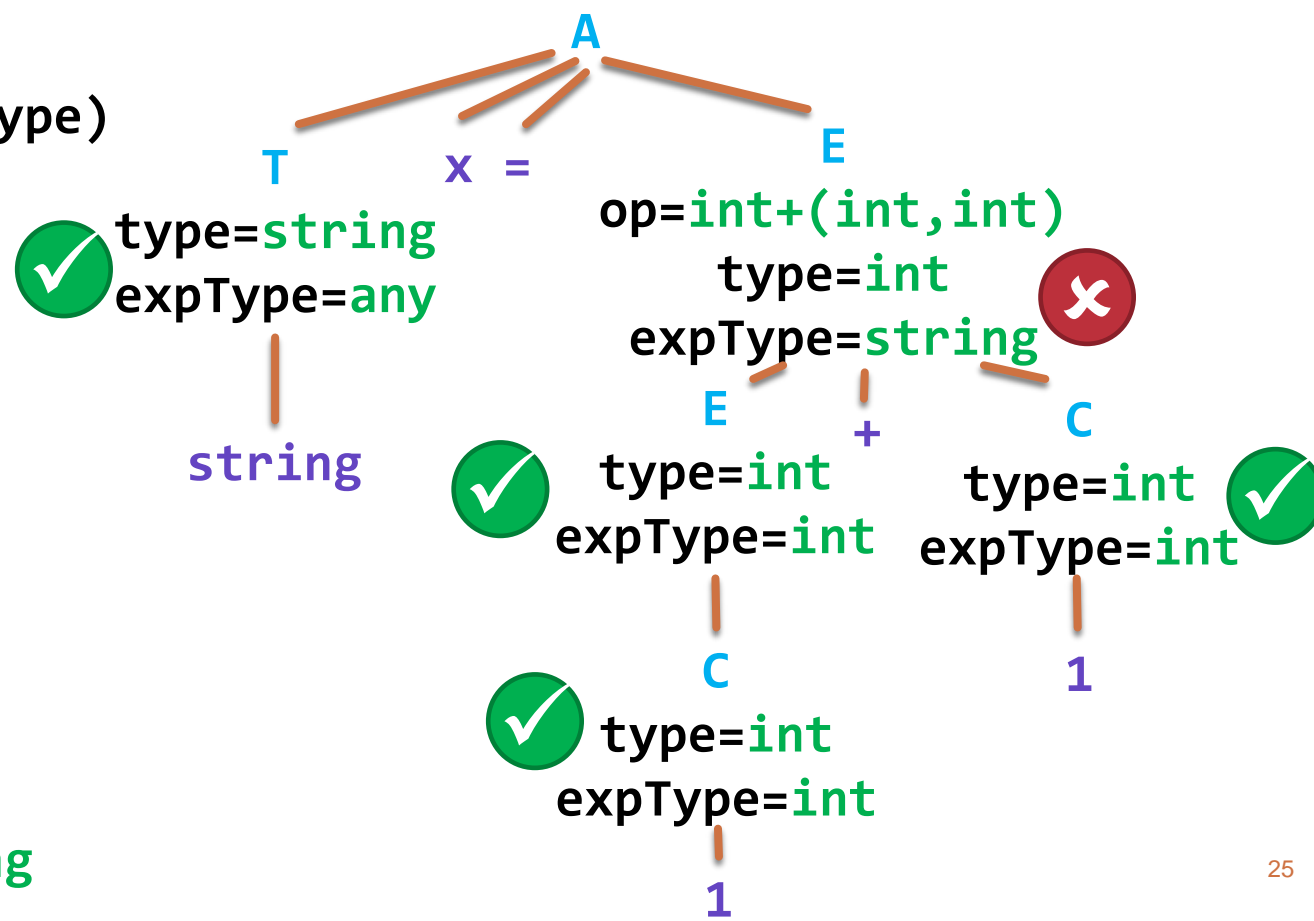
$T \rightarrow \text{string}$

$T.\text{type} = \text{string}$

Programkód:

`string x = 1+1`

Attribútumos szintaxisfa:



Alternatívák attribútumok kiértékelési sorrendjére

- Valahány balról jobbra ill. jobbról balra történő DFS
 - > Left-, Right-, Alternating Attribute Grammar: LAG(k), RAG(k), AAG(k)
- Attribútumhalmazok kiértékelése egymás után, globálisan (nemterminálistól függetlenül)
 - > Ordered Attribute Grammar: OAG (-> tulajdonság eldöntése polinomiális)
 - > topológiai rendezés
- Attribútumhalmazok kiértékelése egymás után, lokálisan (nemterminálistól függően)
 - > Partitionable Attribute Grammar: PAG (-> tulajdonság eldöntése NP-teljes)
 - > PAG-hoz néhány új függőséget felvéve OAG-t kapunk
- Attribútumok lusta kiértékelése
 - > Well-defined Attribute Grammar: WAG

Lényeg: ne legyen kör!

Attribútum nyelvtan alapú fordítás

- Attribútum nyelvtan: deklaratív leírás
- Léteznek attribútum nyelvtan alapú fordítók
 - > Ox: <https://sourceforge.net/projects/ox-attribute-grammar-compiler/>
 - > Silver: <https://github.com/melt-umn/silver>
- Könnyen programozható manuálisan is, imperatív módon
 - > pl. a Roslyn is hasonló módszert alkalmaz, de nem közvetlenül a szintaxisfán, hanem a szimbólumokon

A mai előadás: Szemantikai elemzés

I. Szemantikai elemzés

II. Attribútumnyelvtanok

III. Névelemzés

IV. Típuselemzés és operátorazonosítás

V. Egyéb nyelvi szabályok



Névelemzés

- Szimbólumok definícióinak bejegyzése a szimbólumtáblába
 - > pl. névterek, típusok, mezők, operációk, paraméterek, változók, stb.
 - > előre definiált szimbólumokat is
 - > névütközések felismerése (függhet a szimbólum fajtájától is)
 - > szimbólumok összefésülése (pl. névterek, partial class-ok)
- Szimbólumok használatának összekötése a definícióval
 - > pl. változó, konstans, paraméter, mező értékének olvasása-írása
 - > pl. névtérre v. típusnévre hivatkozás, függvényhívás, goto címkére hivatkozás
- Ugyanaz a név különböző szintaktikai pozíciókban más-mást jelenthet!
 - > fontos, hogy milyen kontextusban (scope-ban) szerepel

Példa: C-stílusú blokkok elemzése

Block → { **Decls** **Stmts** }

Stmts.scope = **append**(**Decls.scope**, **Block.scope**)

örökölt

Decls → **Decls** **Decl**

Decls[1].scope = **append**(**Decls[2].scope**, **Decl.scope**)

szintetizált

Decl → **Type** **Var** ;

Decl.scope = **new Scope**(**Var.symbol**)

Stmts → **Stmts** **Stmt**

Stmts[2].scope = **Stmts[1].scope**

Stmt.scope = **Stmts[1].scope**

Mi a probléma ezzel a megoldással?

Stmt → ... **Var** ... ;

Var.symbol = **Stmt.scope.find**(**Var.name**)

Problémák

- Probléma: egy attribútum egyszerre csak egy fajta lehet (örökölt v. szintetizált)
- Megoldás: attribútum szétválasztása
- A definíciók hatóköre is különbözhet:
 - > 1. csak a definíció pozíciója után hivatkozható (pl. lokális változók)
 - scopeIn (örökölt): korábban definiált szimbólumok
 - scopeOut (szintetizált): összes eddig definiált szimbólum
 - > 2. a definíció pozíciója előtt is hivatkozható (pl. ugyanazon osztályon belüli függvények)
 - scopeAcc (szintetizált): ebben gyűjtjük a szimbólumokat
 - scopeLkp (örökölt): ebből keressük a szimbólumokat

Példa: csak a definíció pozíciója után hivatkozható

Block → { **Decls** **Stmts** }

Decls.scopeIn = **Block**.scopeIn

Stmts.scopeIn = **Decls**.scopeOut

Block.scopeOut = **Block**.scopeIn

Decls → **Decls** **Decl**

Decls[2].scopeIn = **Decls**[1].scopeIn

Decl.scopeIn = **Decls**[2].scopeOut

Decls[1].scopeOut = **Decl**.scopeOut

Decl → **Type** **Var** = **Var** ;

Type.symbol = **Decl**.scopeIn.find(**Type**.name)

Var[2].symbol = **Decl**.scopeIn.find(**Var**[2].name)

Decl.scopeOut = append(new **Scope**(**Var**[1].symbol), **Decl**.scopeIn)

Példa: a definíció pozíciója előtt is hivatkozható

Block → { **Decls** **Stmts** }

Decls.scopeLkp = **append**(**Block**.scopeLkp, **Decls**.scopeAcc)

Stmts.scopeLkp = **Decls**.scopeLkp

Block.scopeAcc = **new Scope**()

Decls → **Decls Decl**

Decls[2].scopeLkp = **Decl**[1].scopeLkp

Decl.scopeLkp = **Decls**[1].scopeLkp

Decls[1].scopeAcc = **append**(**Decls**[2].scopeAcc, **Decl**.scopeAcc)

Decl → **Type Var = Var ;**

Type.symbol = **Decl**.scopeLkp.**find**(**Type**.name)

Var[2].symbol = **Decl**.scopeLkp.**find**(**Var**[2].name)

Decl.scopeAcc = **new Scope**(**Var**[1].symbol)

Példa: minősített névhivatkozás

```
QualifiedName → QualifiedName . Name  
Name.symbol = QualifiedName[2].type.find(Name.name)  
QualifiedName[1].type = Name.symbol.type
```

```
QualifiedName → Name  
Name.symbol = QualifiedName.scopeLkp.find(Name.name)  
QualifiedName.type = Name.symbol.type
```



Honnan jön?

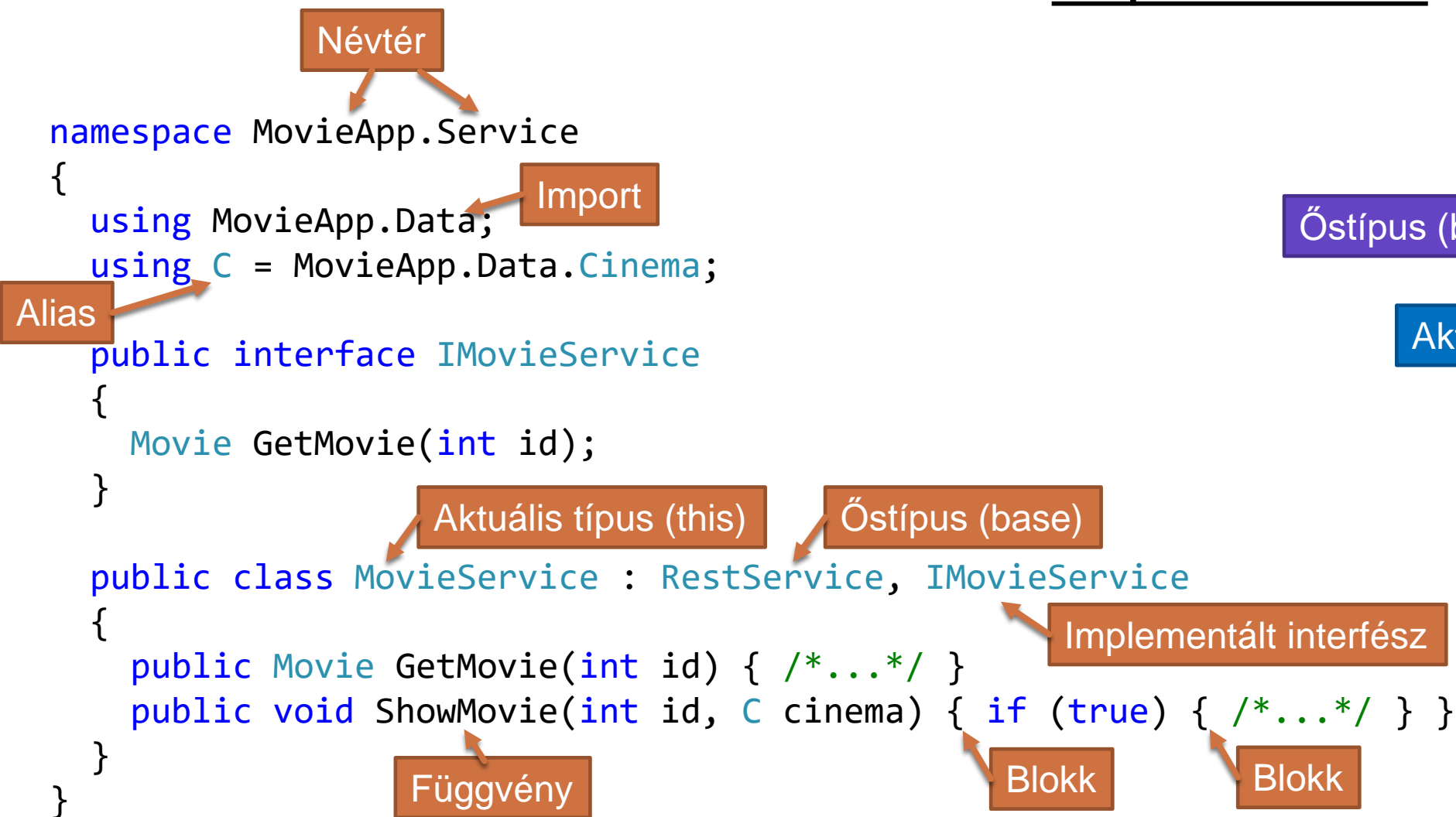
A név- és a típuselemzés kölcsönösen hatnak egymásra!

Alternatívák scope kezelésre: imperatív manuális megoldások

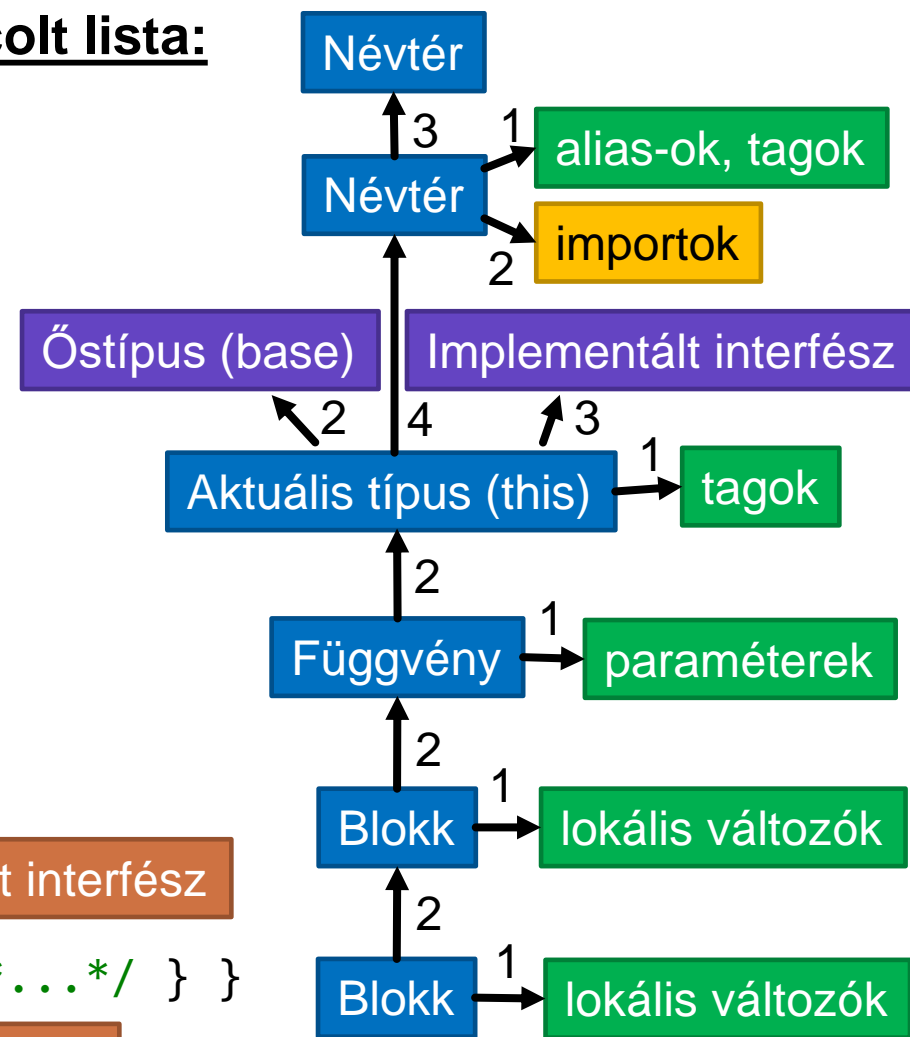
- 1. Verem (symbol stack)
 - > dinamikusan változik
 - > kevés memóriát igényel
 - > egy menetben fel kell oldania mindent
- 2. Láncolt lista
 - > statikus struktúra, nem változik
 - > több memóriát igényel
 - > bonyolultabb eseteket is tud kezelni
 - > (az előző, attribútum nyelvtanos példák is láncolt listát építettek: deklaratívan)

Névkeresés tipikus scope-jai láncolt listával

Példa kód:



Scope láncolt lista:



Névelemzés során felderíthető hibák

■ Többszörös definíció

- > ugyanaz a név többször van definiálva ugyanabban a scope-ban:
nem lehet egyértelműen bejegyezni a szimbólumtáblába
- > nem mindig hiba (pl. overloading vagy különböző fajta szimbólumok)

■ Szimbólum nem létezik

- > hivatkozás helyén megadott név nem szerepel a scope hierarchiában

■ Nem egyértelmű szimbólum

- > hivatkozás helyén megadott név nem oldható fel egyértelműen a scope hierarchia alapján
- > pl. többszörös definíció miatt, vagy egy importált és egy definiált szimbólum közötti névütközés miatt

A mai előadás: Szemantikai elemzés

I. Szemantikai elemzés

II. Attribútumnyelvtanok

III. Névelemzés

IV. Típuselemzés és operátorazonosítás

V. Egyéb nyelvi szabályok



Típuselemzés

- Típus: értékek és azokon értelmezett műveletek egy halmaza
 - > primitív típusok (int, float, double, char, bool, stb.)
 - > összetett típusok (struct, union, class, array, stb.)
- Típuselemzés feladata:
 - > nevek, operandusok és kifejezések típusának meghatározása
 - > típuskikövetkeztetés (type inference), pl. C#-ban a **var** kulcsszó
- Típuselemzés szükséges:
 - > névelemzéshez és operátorok azonosításához
 - > érték- és típuskonverziók ellenőrzéséhez
 - > konzisztencia-ellenőrzéshez

Típusellenőrzés

- Típushiba:
 - > egy szimbólumon olyan műveletet akarunk végrehajtani, amelyet a típusa nem támogat
- Típusellenőrzés:
 - > típushibák elkerülésére a szimbólumok típusát a műveletek végrehajtása előtt ellenőrizzük
 - > fordítási időben: statikus típusellenőrzés -> gyors, de nagyon nehéz minden esetet kezelni
 - > futási időben: dinamikus típusellenőrzés -> lassú, de precízebb
- Nyelv típusossága:
 - > erősen típusos nyelv: minden típushiba felderíthető fordítási időben
 - > gyengén típusos nyelv: felderítés futási időben -> futási idejű típushiba előfordulhat

Típusekvivalencia

- Névekvivalencia:

- > két típus egyforma, ha ugyanazzal a típusdefinícióval lettek definiálva
- > pl. osztályok C#-ban

- Strukturális ekvivalencia:

- > két típus egyforma, ha ugyanazzal a típuskonstruktorral és ugyanolyan típusargumentumokkal lettek létrehozva
- > pl. tömbök, tuple-ök, generikus típusok
- > vigyázat: létezhetnek rekurzív típusok is!

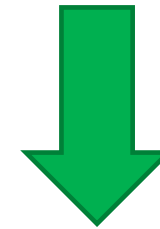
Kovariancia és Kontravariancia

`virtual Base Method(Derived obj)`

Kovariancia



Kontravariancia



`override Derived Method(Base obj)`

Típusellenőrzés attribútumai

- Tényleges típus: `type` (szintetizált: alulról felfelé kiértékelve)
- Elvárt típus: `expectedType` (örökölt: fentről lefelé kiértékelve)
- Köztük implicit típuskonverzió
- Példa:

```
int x = 5;
```

```
double y = x;
```

`x.expectedType = double`
`x.type = int`



```
if (x)  
{
```

```
    ...
```

```
}
```

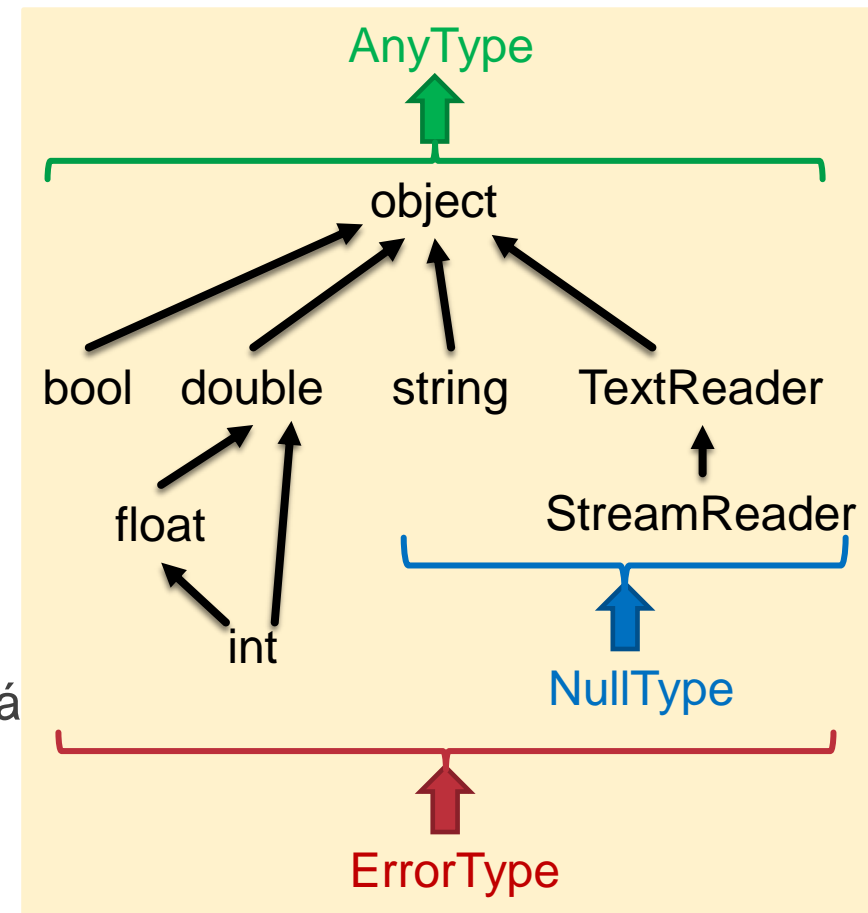
`x.expectedType = bool`
`x.type = int`



Típushierarchia

- Implicit típuskonverzió (type coercion): automatikus
 - > pl. `int` -> `double`, leszármazott -> ős
- Explicit típuskonverzió (type cast): nem automatikus
 - > nem biztonságos vagy információvesztéssel járhat
 - > pl. ős -> leszármazott, `double` -> `int`
- Típushierarchia:
 - > típusok között egy rendezés
 - > megadja, melyik típus mely másikkra konvertálható
- Speciálisan kezelt típusok:
 - > AnyType: bármi megengedett -> minden típus konvertálható rá
 - > NullType: `null` típusa -> referencia típusokra konvertálható
 - > ErrorType: típushiba esetén -> minden típusra konvertálható

Típushierarchia példa (C#):



Műveletek típuselemzéshez

- `GetBaseType: Type -> Type`
 - > legbelső alaptípus megadása, pl. `int?[] -> int`
- `GetInnerType: Type -> Type`
 - > közvetlenül tartalmazott típus megadása, pl. `int?[] -> int?`
- `AreEquivalent: Type x Type -> bool`
 - > két típus strukturálisan ekvivalens-e, pl. `(int,bool),ValueTuple<int,bool> -> true`
- `GetImplicitConversion: Type x Type -> Operator`
 - > első típust a második típusra konvertáló implicit operátor megadása
- `GetExplicitConversion: Type x Type -> Operator`
 - > első típust a második típusra konvertáló explicit operátor megadása

lehet beépített
vagy felhasználói



Műveletek operátorazonosításhoz

- $\text{GetOperator: OpKind} \times \text{Type} \rightarrow \text{Operator}$
 - > adott fajtájú és adott típusú operandussal rendelkező unáris operátor megadása
- $\text{GetOperator: OpKind} \times \text{Type} \times \text{Type} \rightarrow \text{Operator}$
 - > adott fajtájú és két adott típusú operandussal rendelkező bináris operátor megadása

lehet beépített
vagy felhasználói



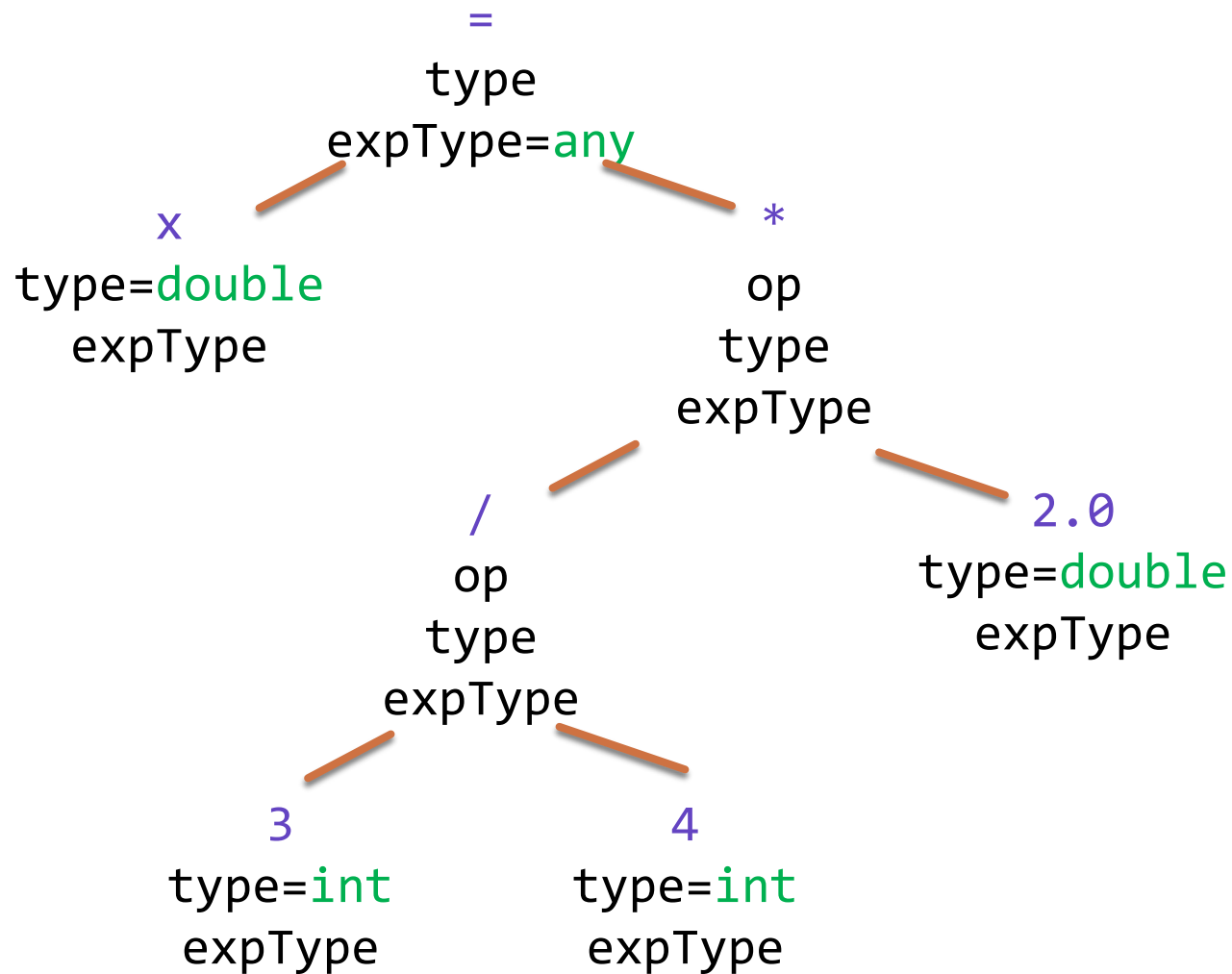
Operátor kiválasztásának lépései

- 1. operandusok típusának meghatározása
- 2. szóba jöhető operátordefiníciók összegyűjtése
- 3. a definíciók közül a leginkább megfelelő kiválasztása
 - > hibajelzés: ha nincs illeszkedő definíció, vagy több alternatíva is illeszkedik
- 4. operandusok elvárt típusának meghatározása
- 5. tényleges típusok és elvárt típusok kompatibilitásának vizsgálata
 - > hibajelzés: inkompatibilitás esetén
- Függvénytúlterhelés feloldása hasonló:
 - > operandus -> argumentum
 - > operátor -> függvény

Példa: operátor kiválasztása

Utasítás:

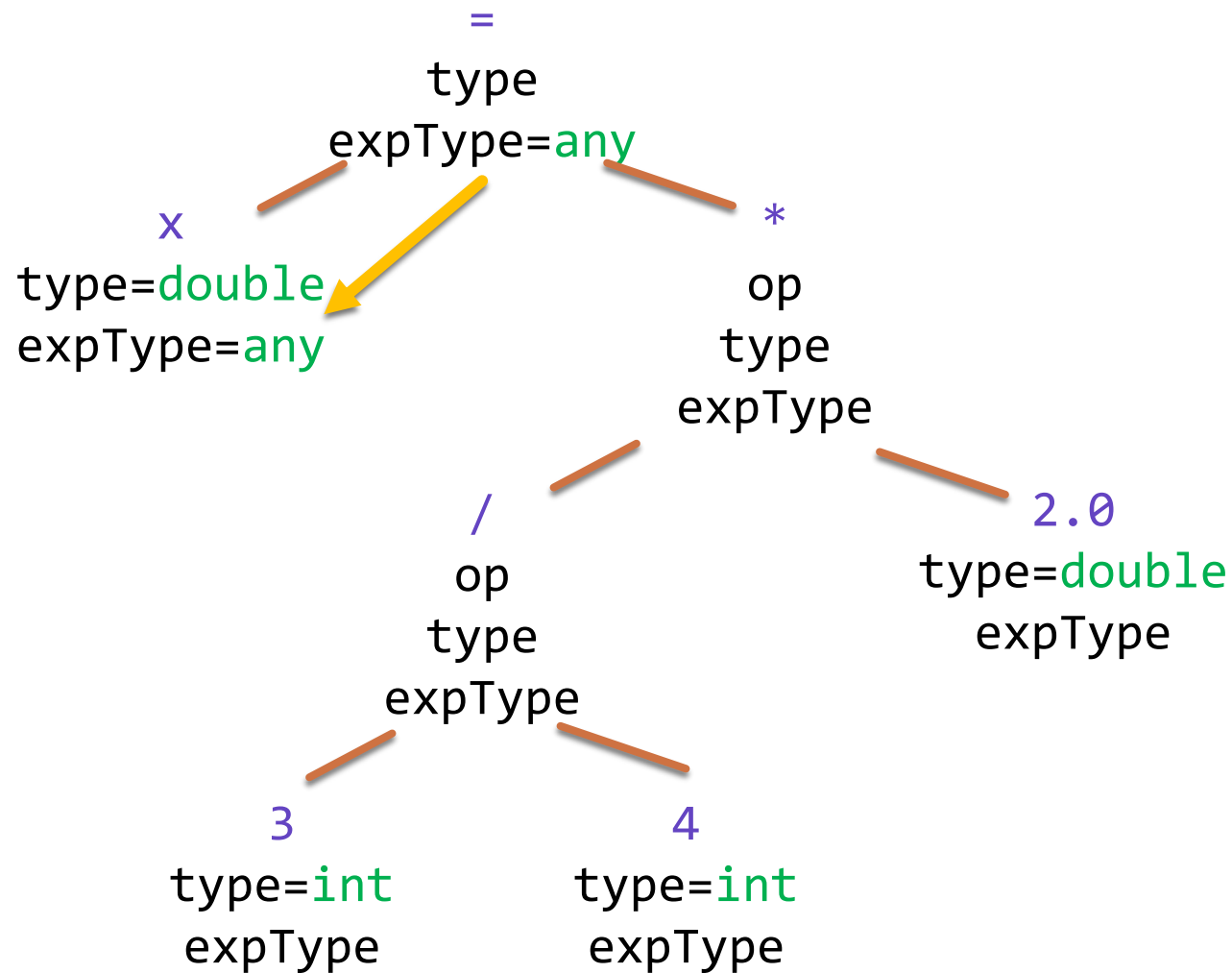
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

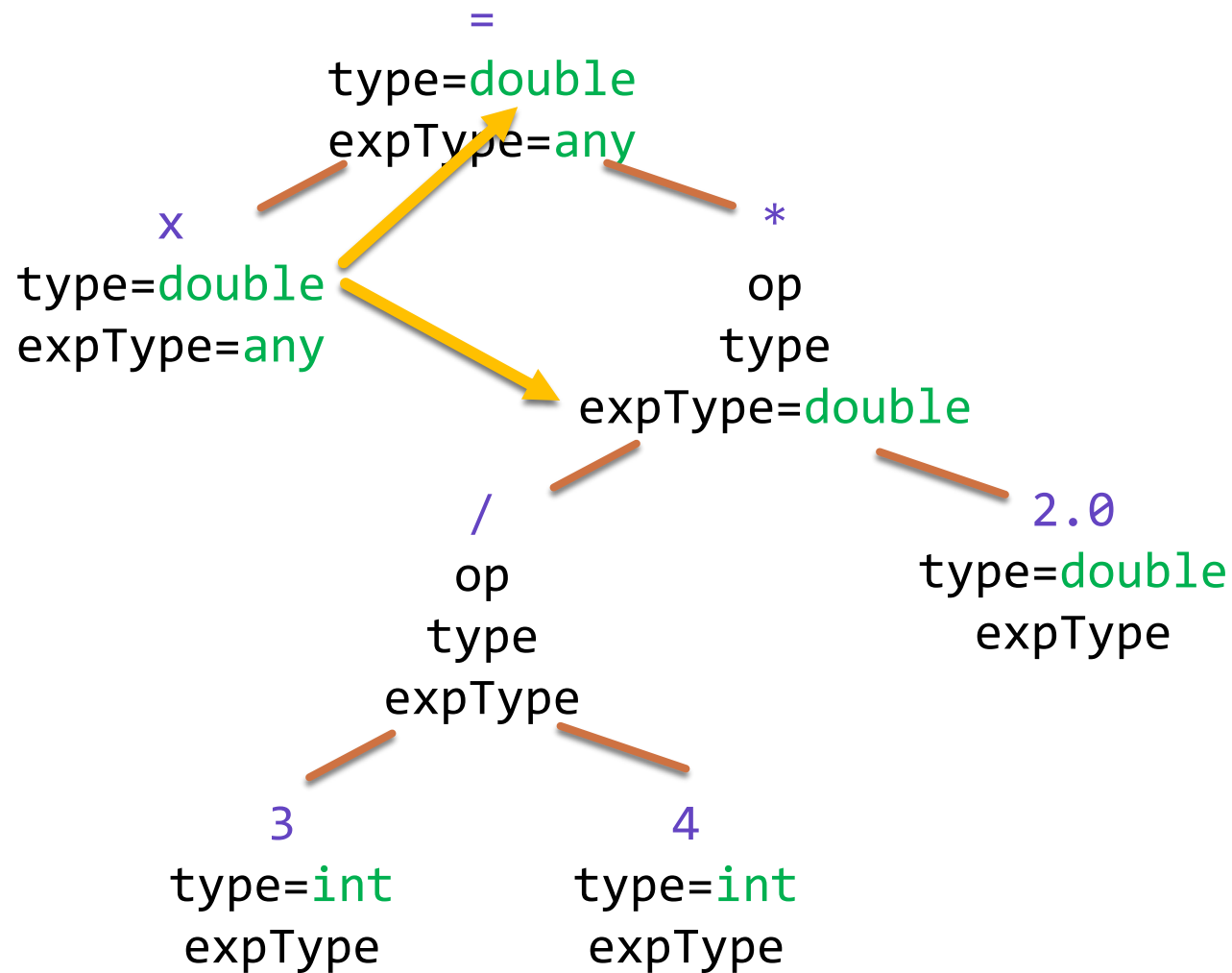
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

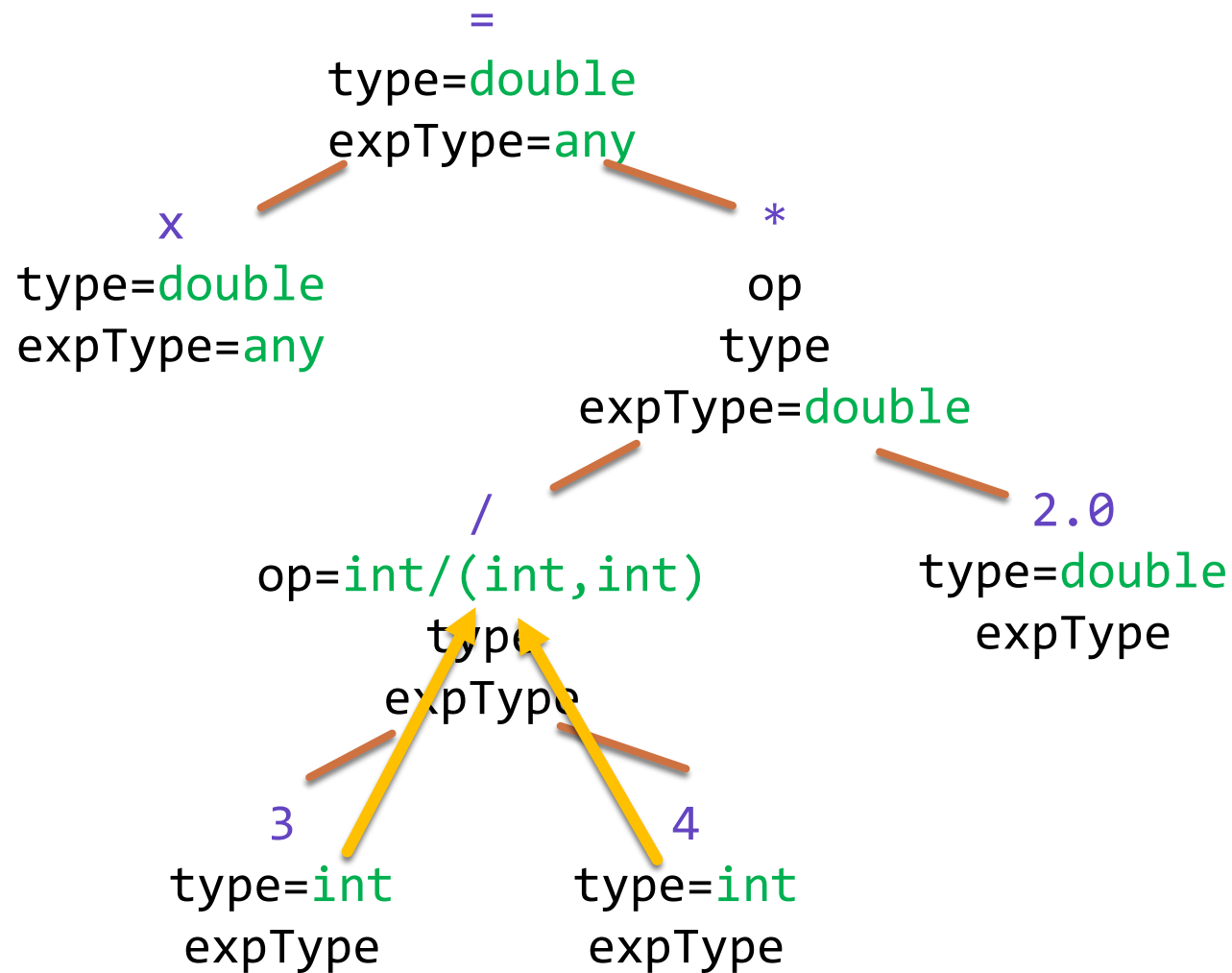
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

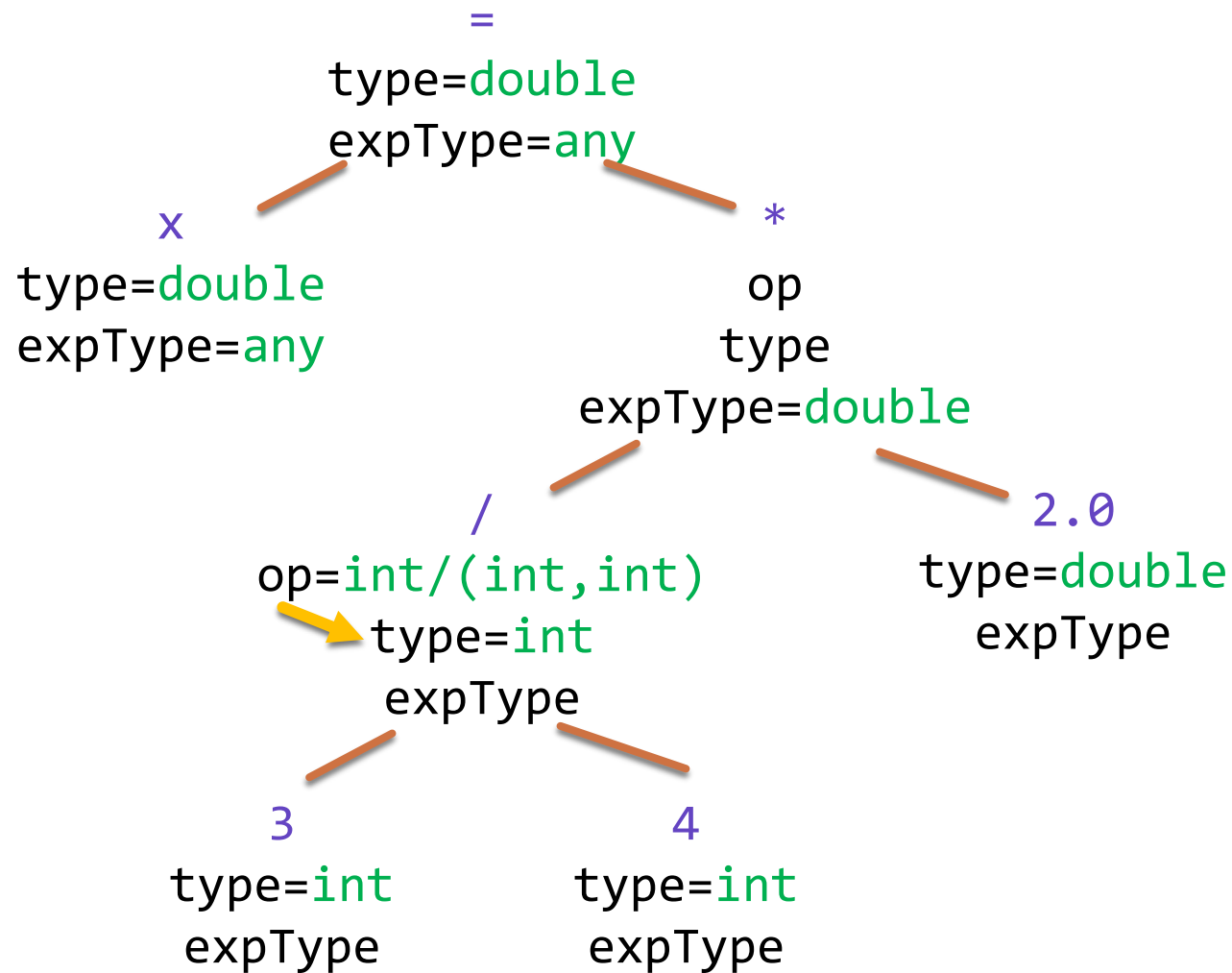
double x = 3/4*2.0;



Példa: operátor kiválasztása

Utasítás:

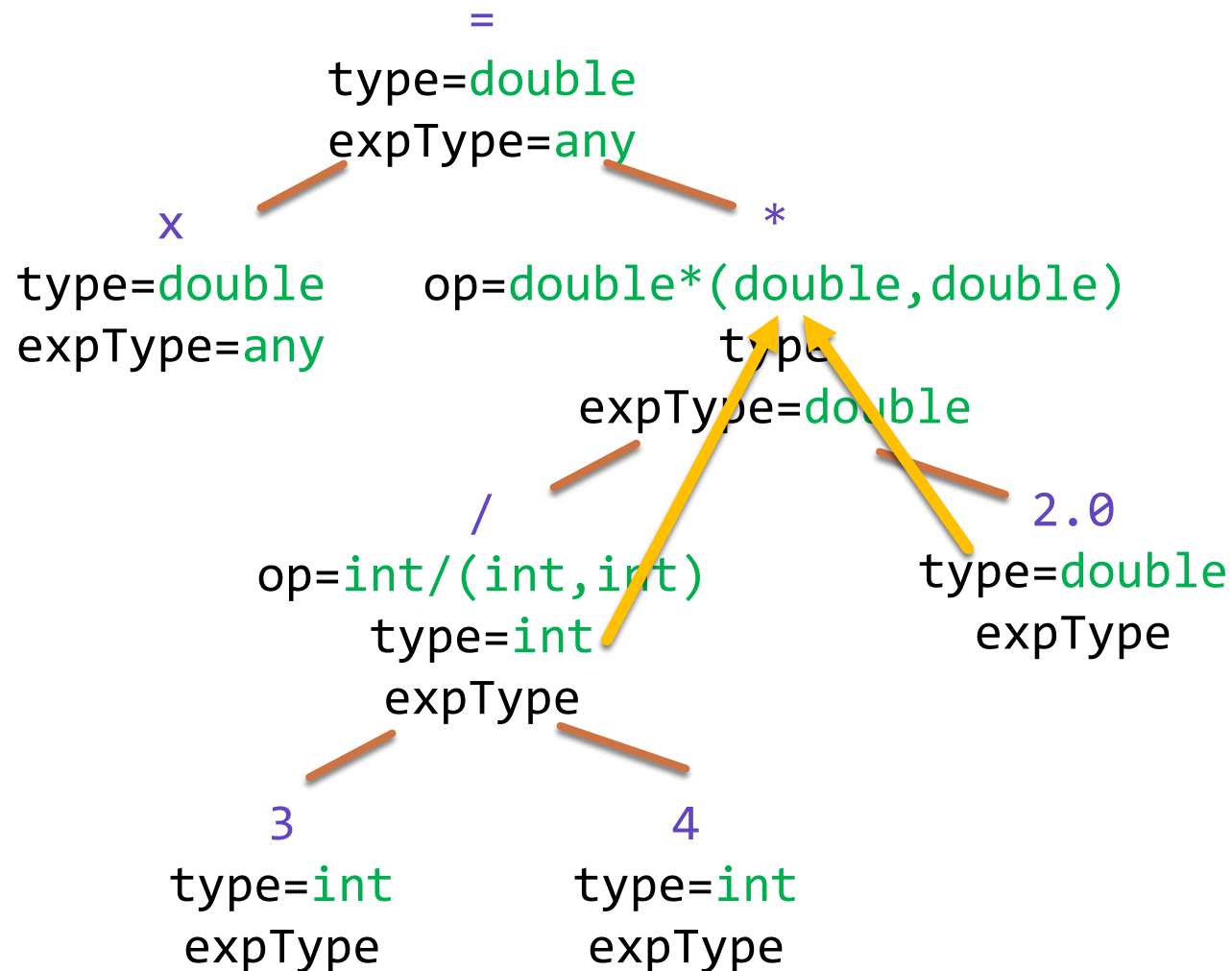
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

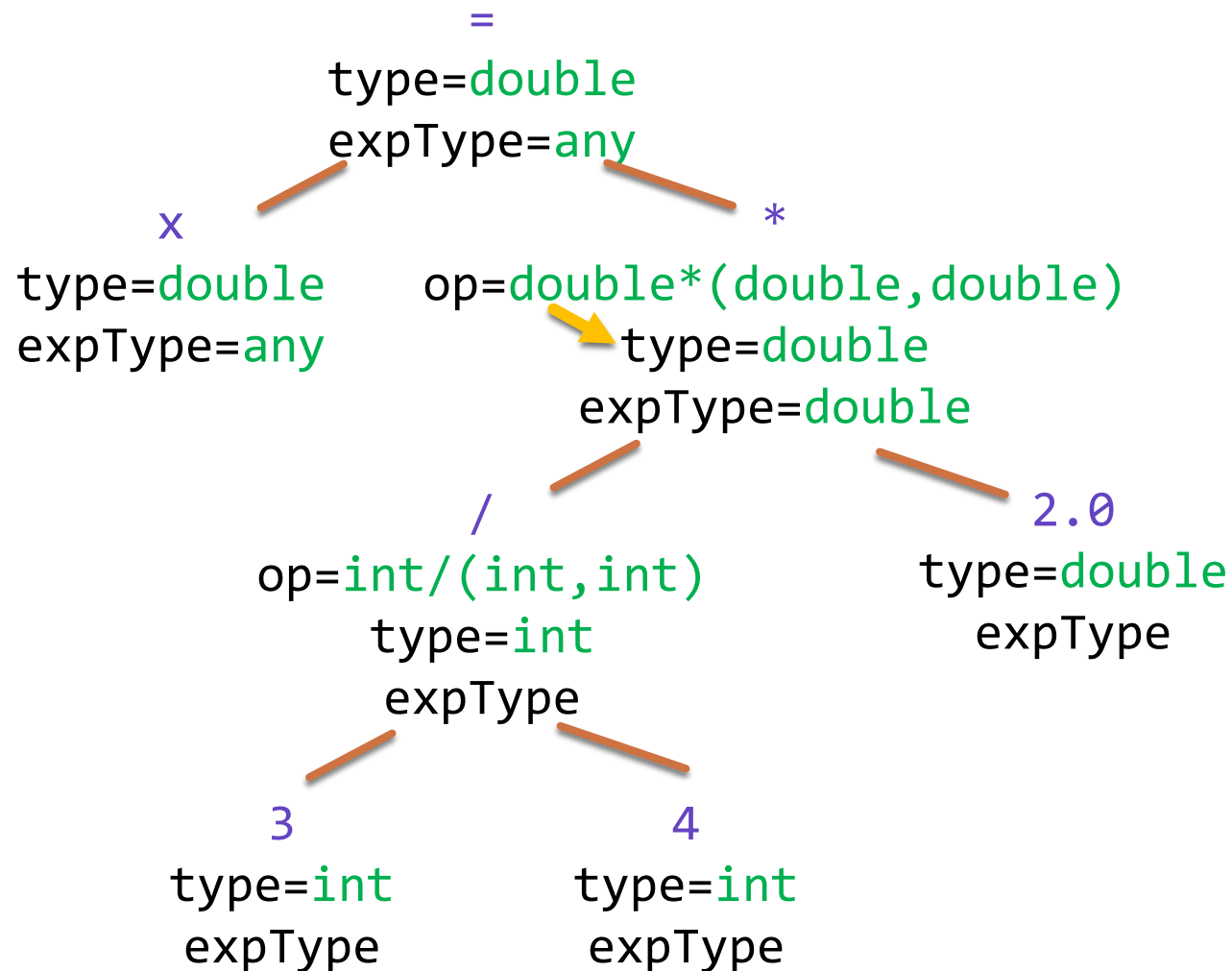
double x = 3/4*2.0;



Példa: operátor kiválasztása

Utasítás:

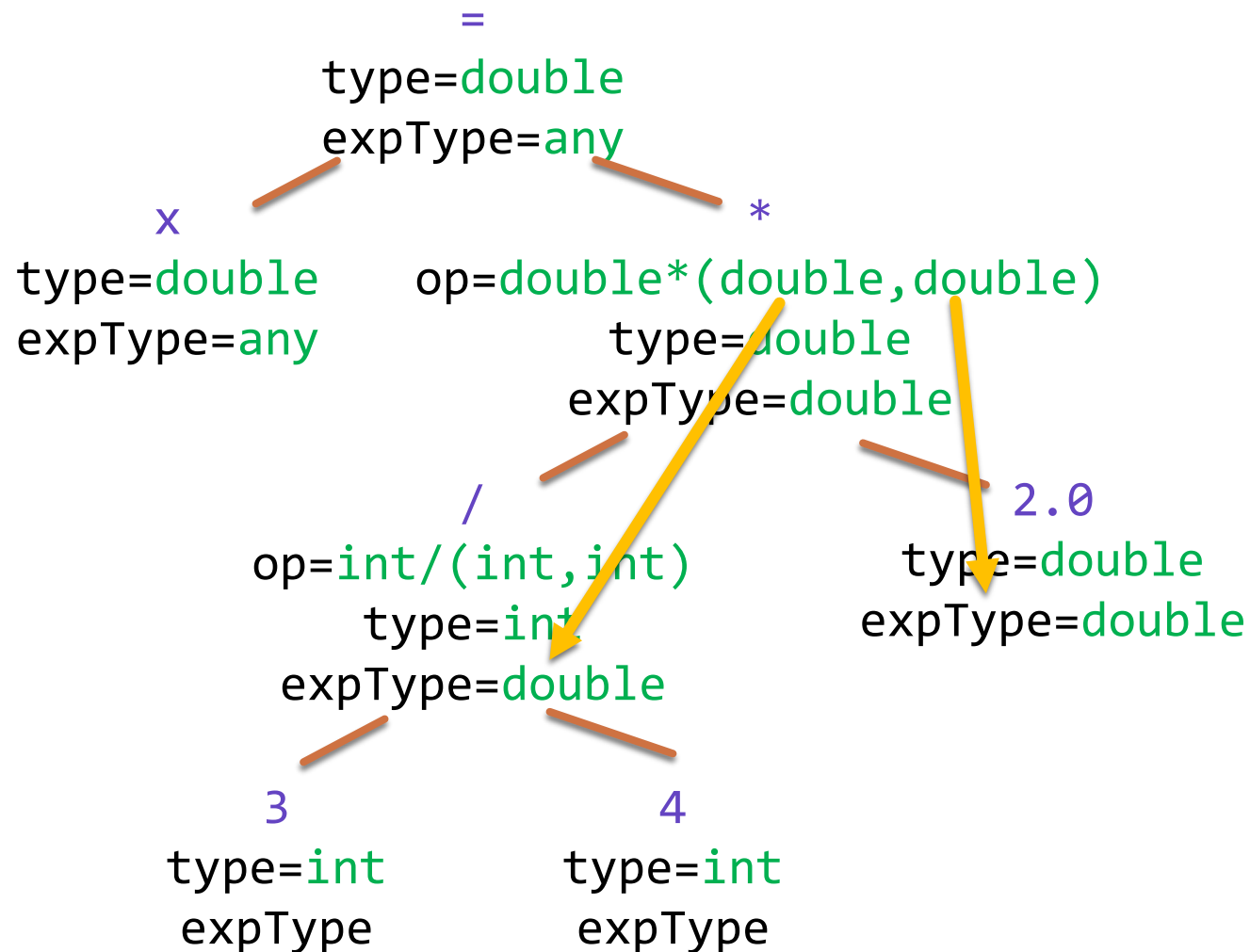
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

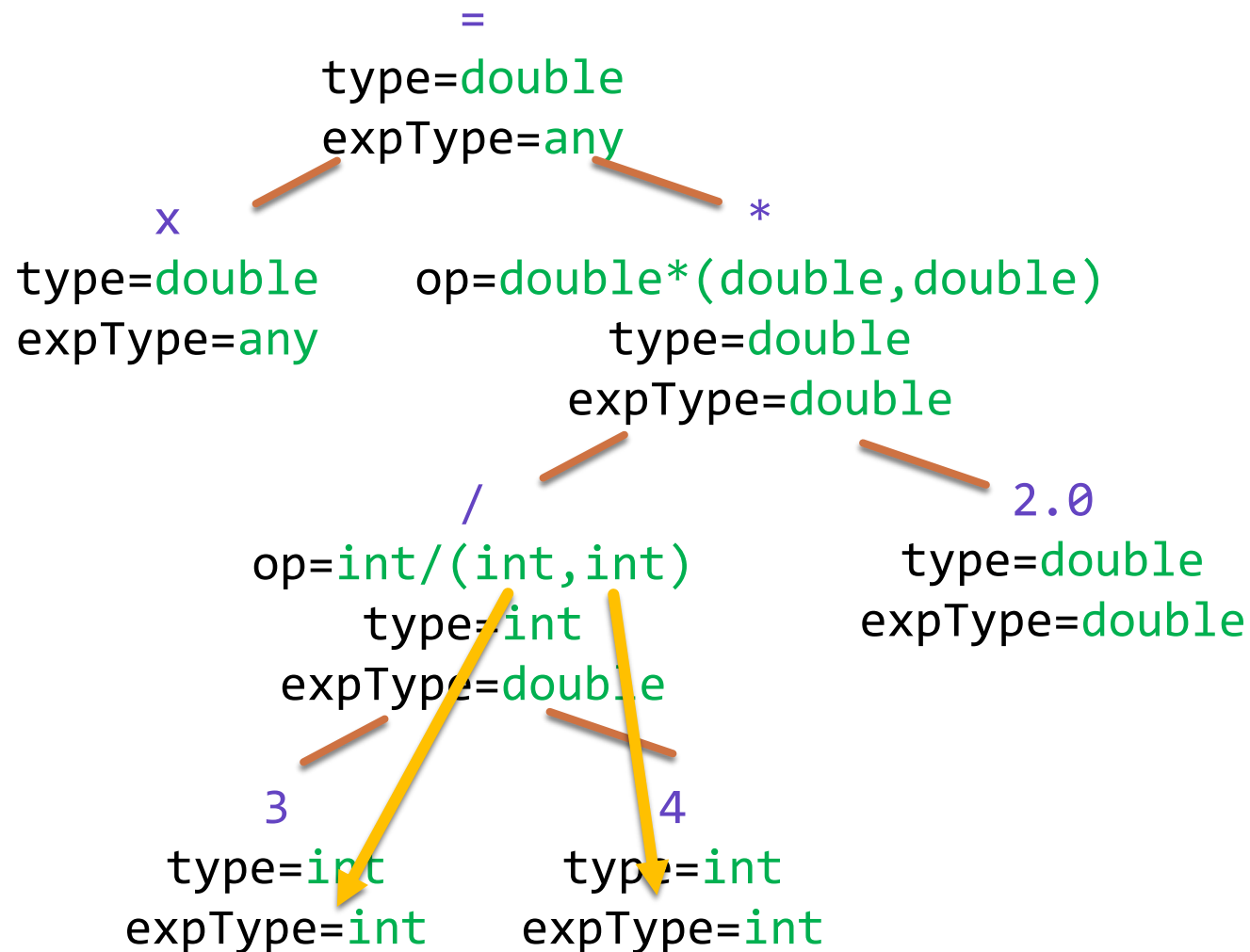
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

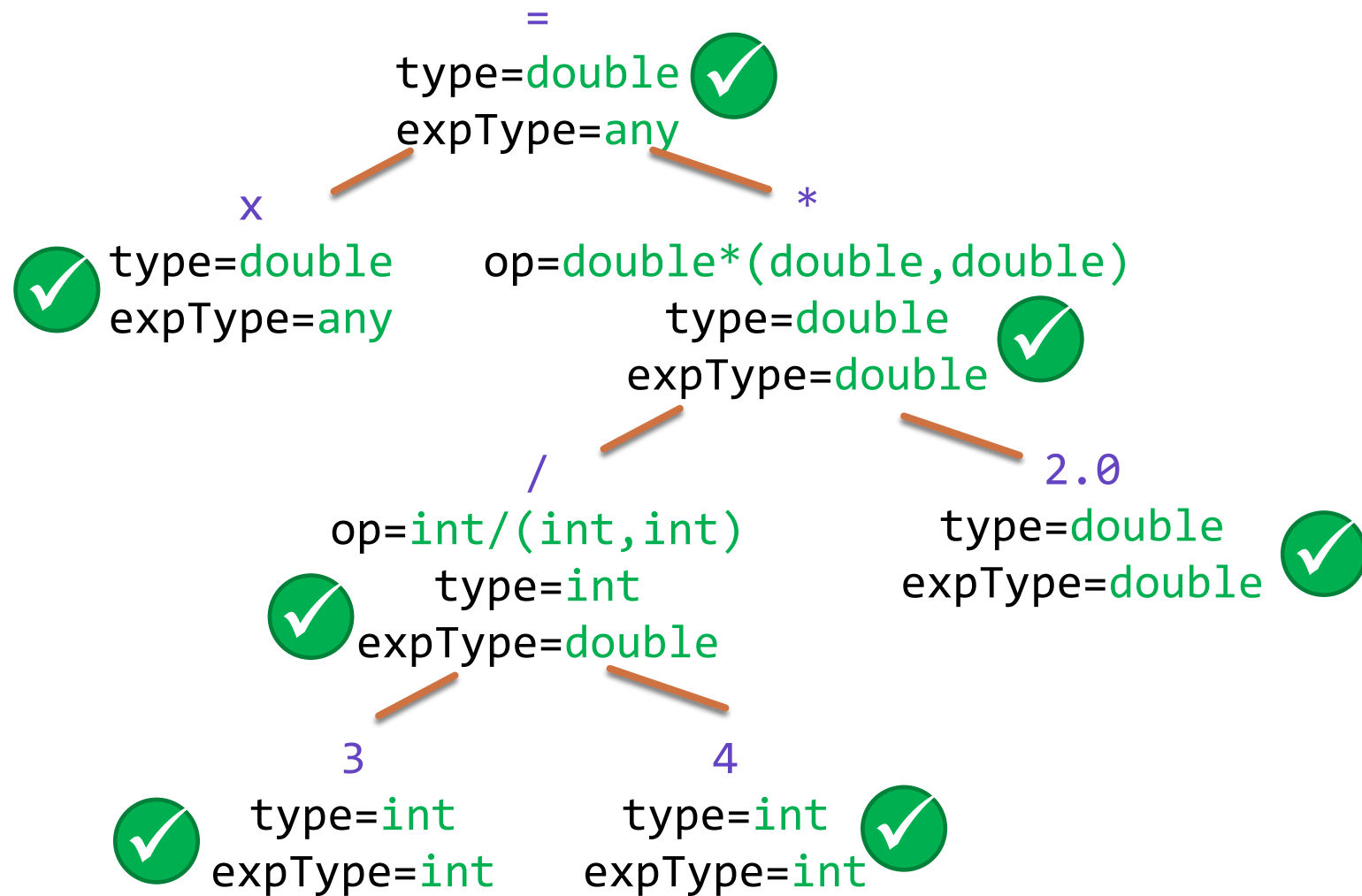
`double x = 3/4*2.0;`



Példa: operátor kiválasztása

Utasítás:

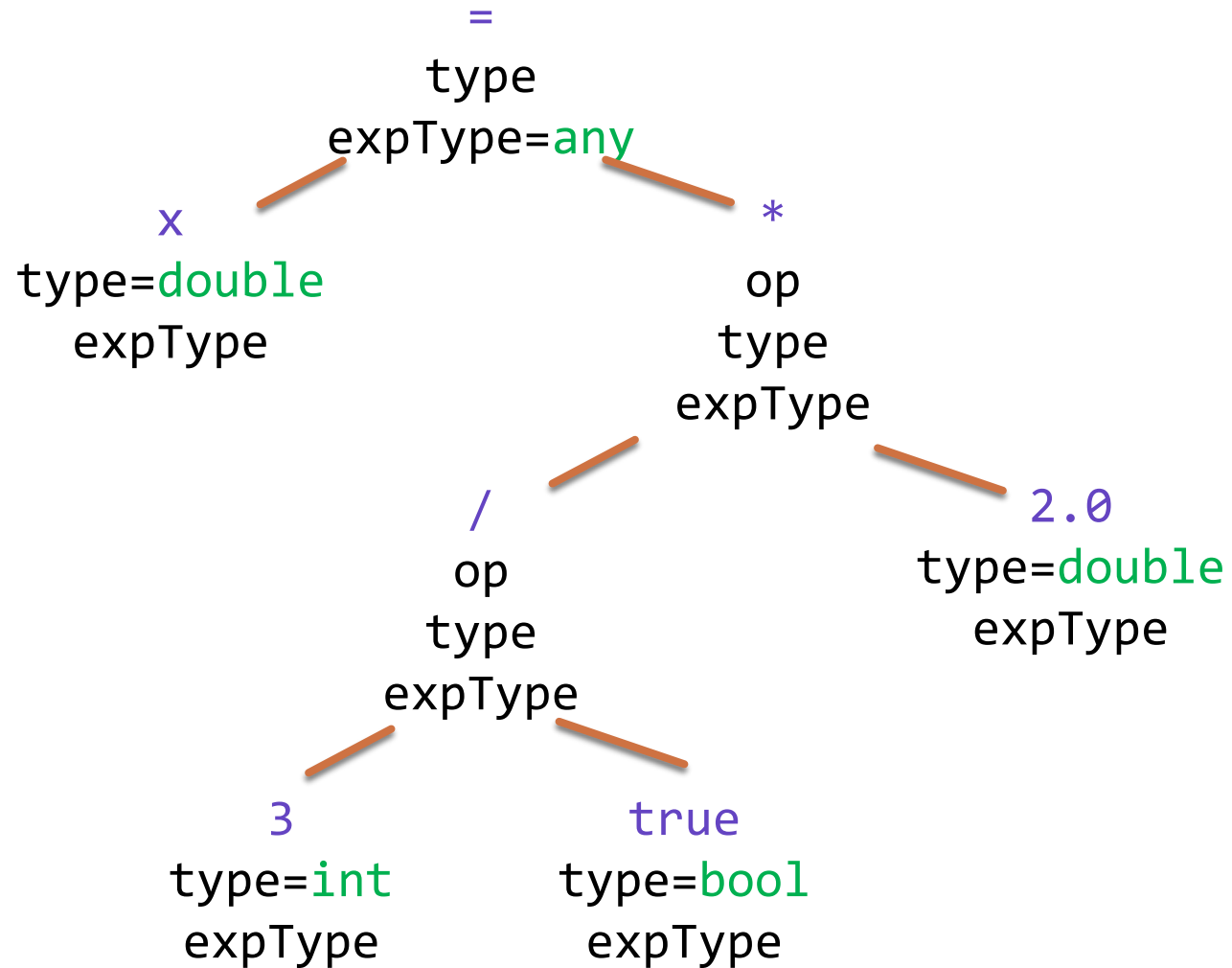
`double x = 3/4*2.0;`



Példa: típushiba

Utasítás:

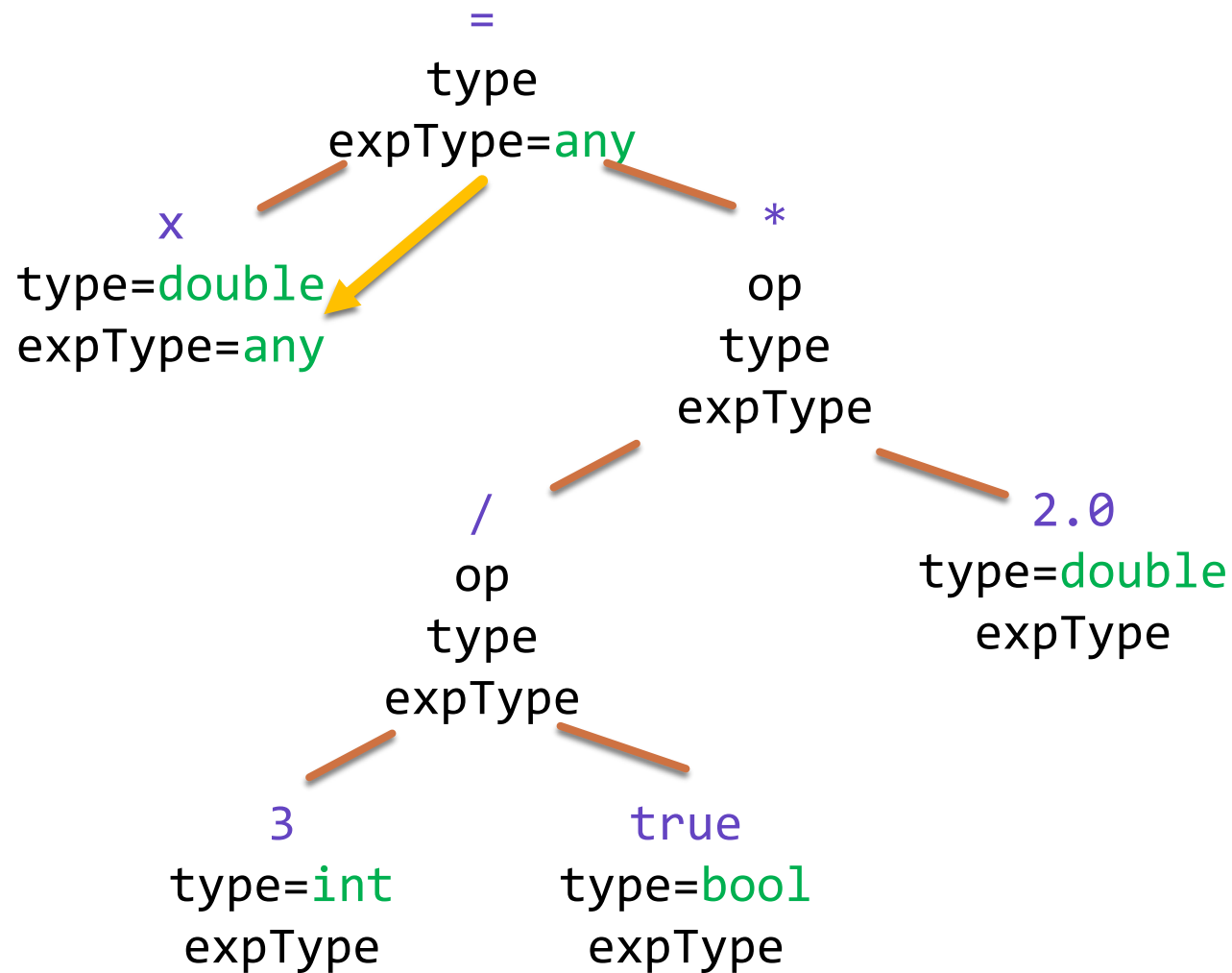
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

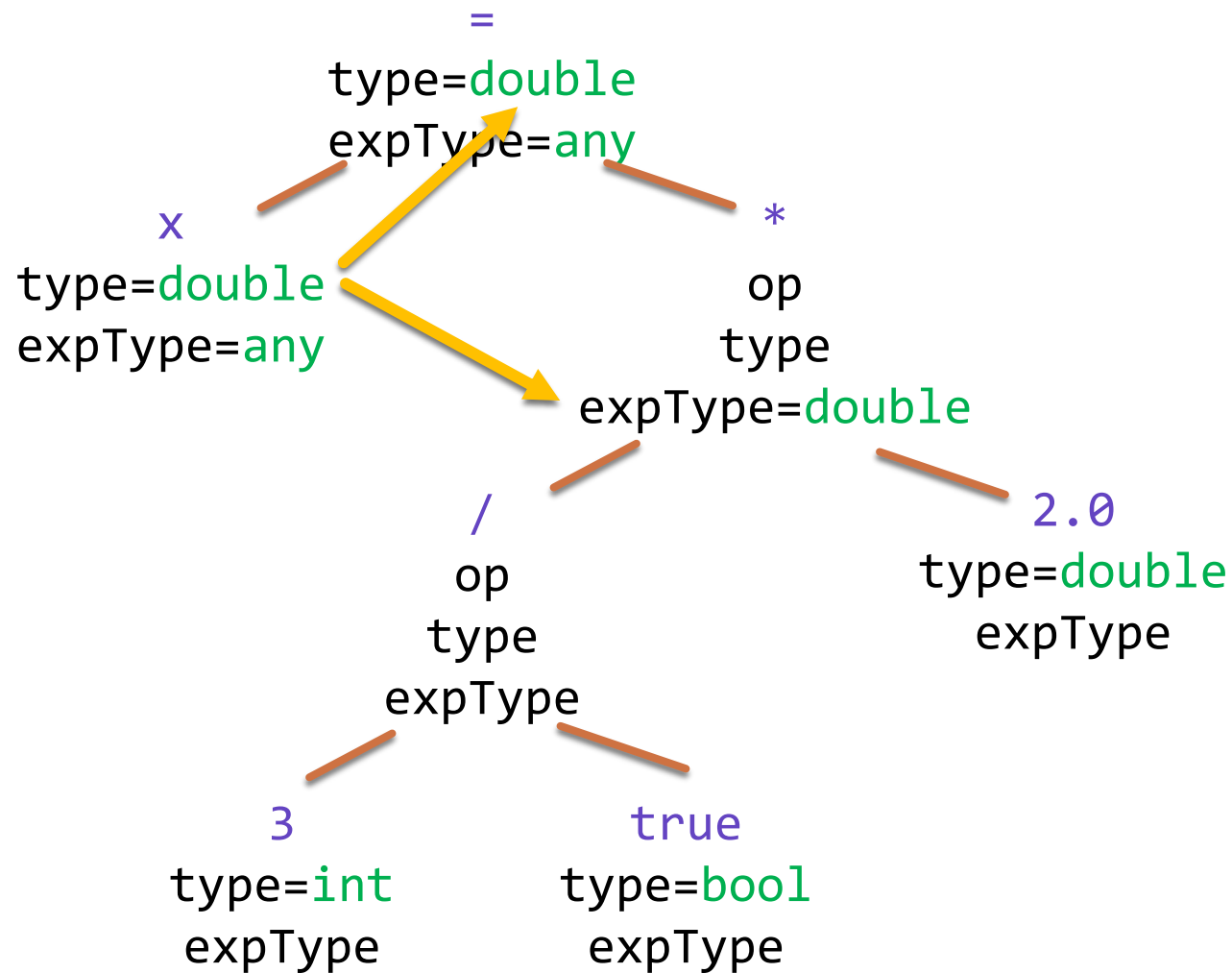
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

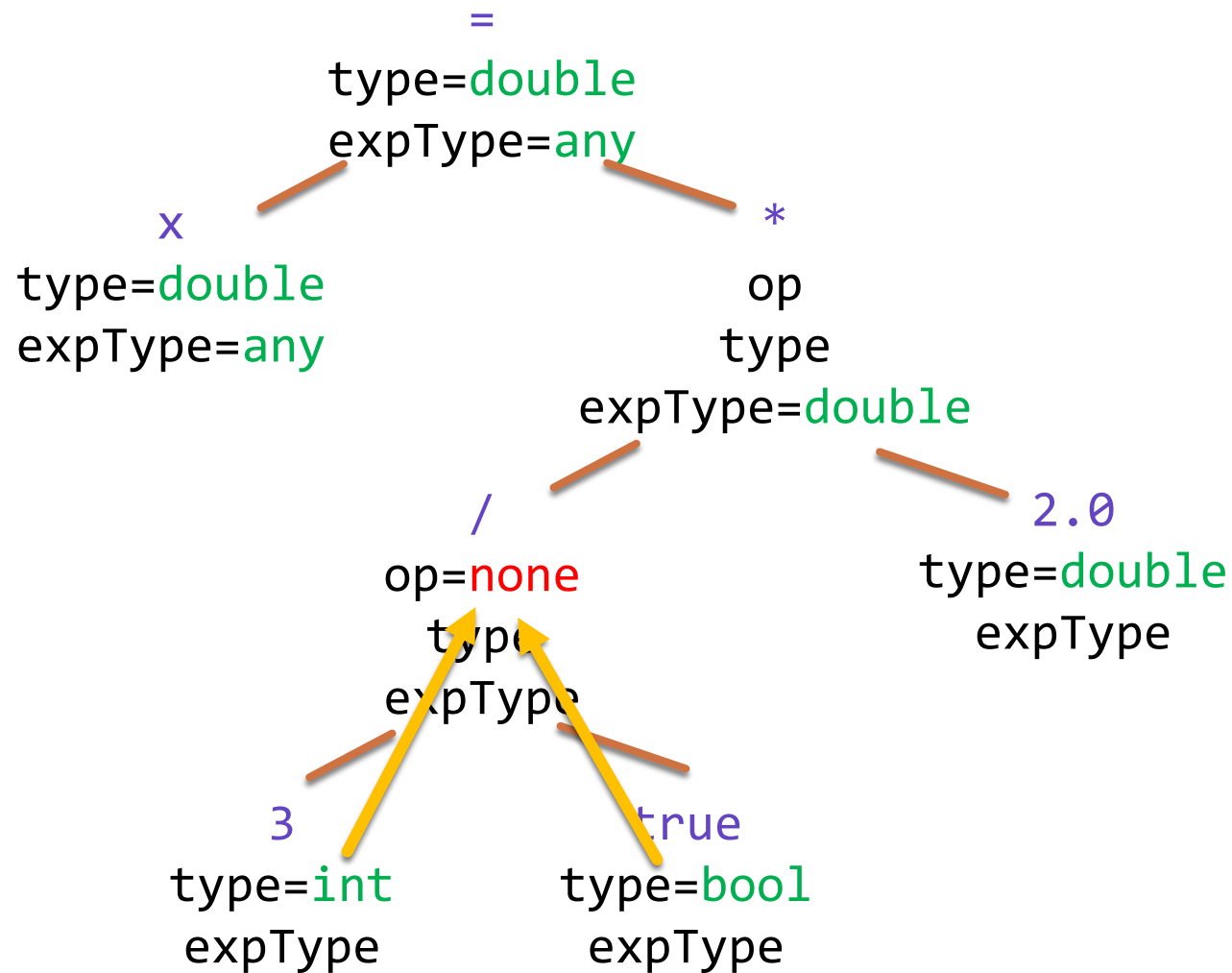
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

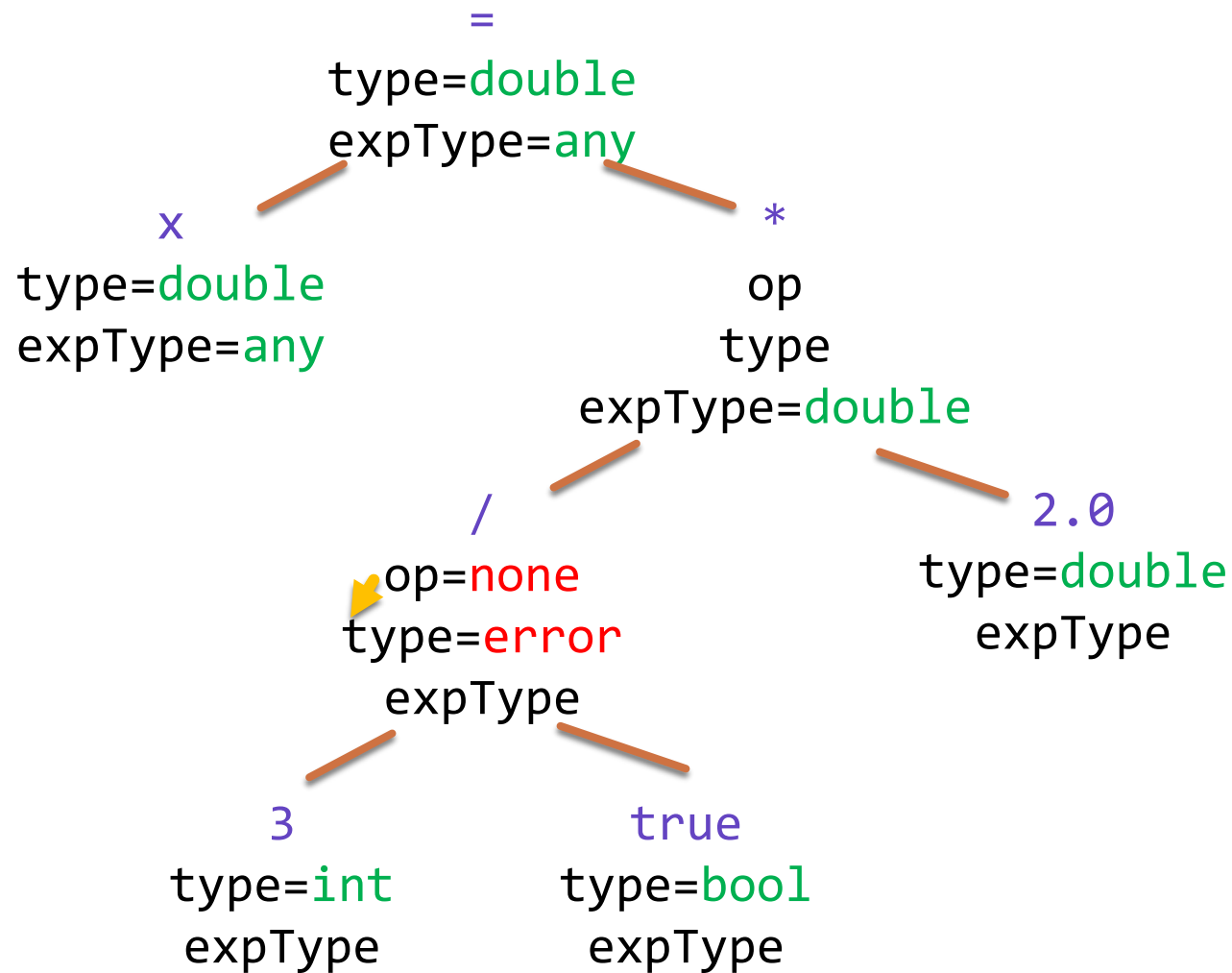
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

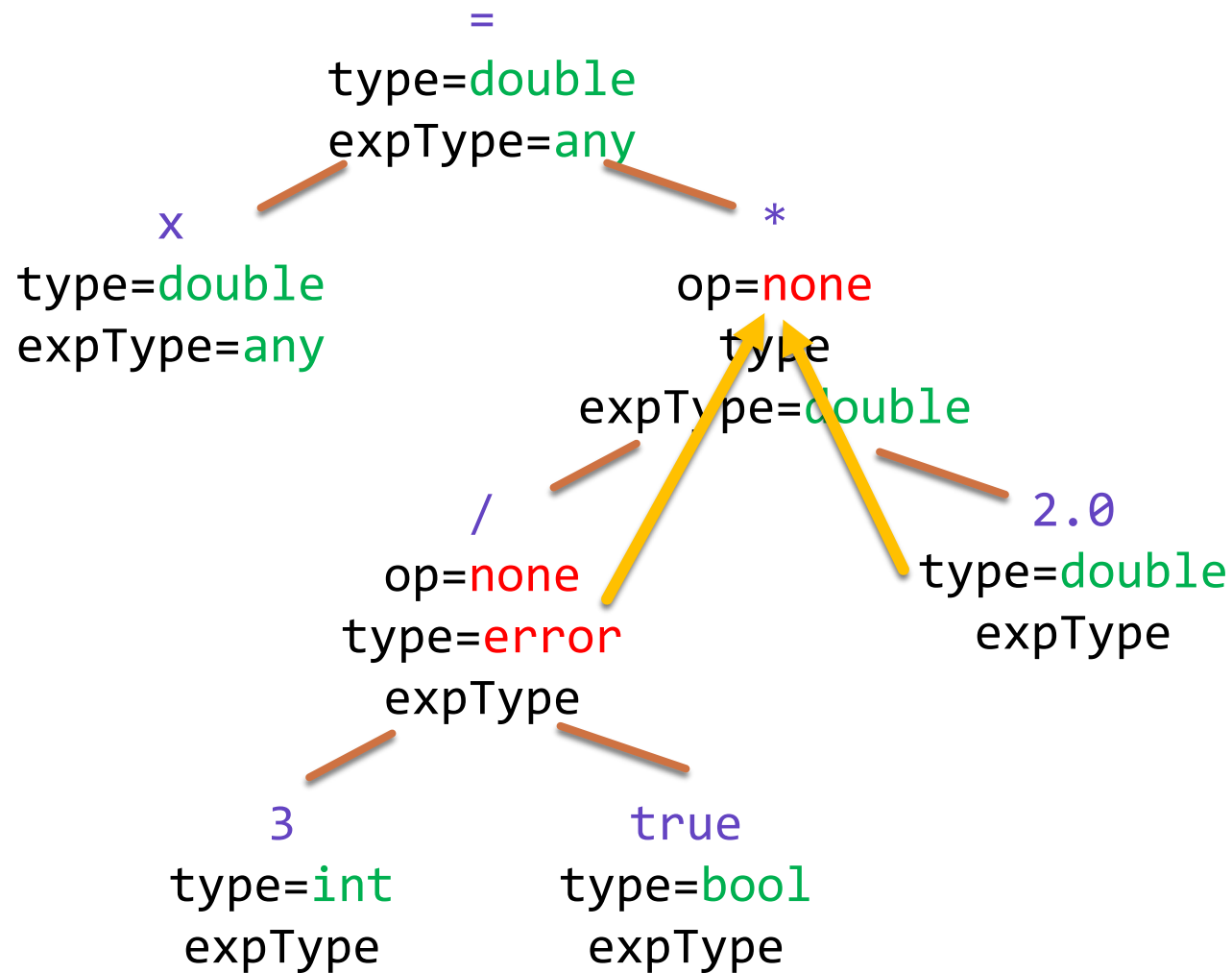
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

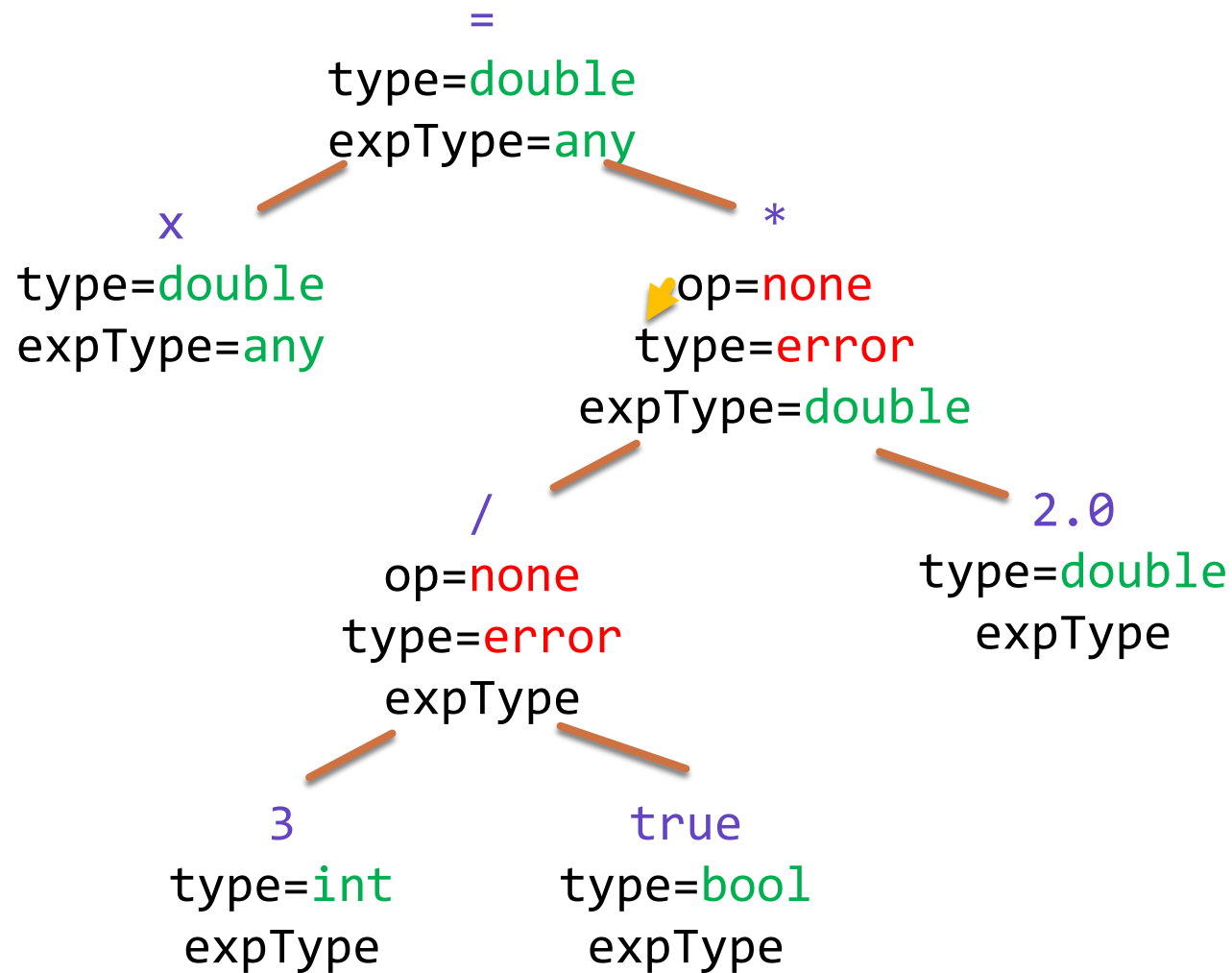
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

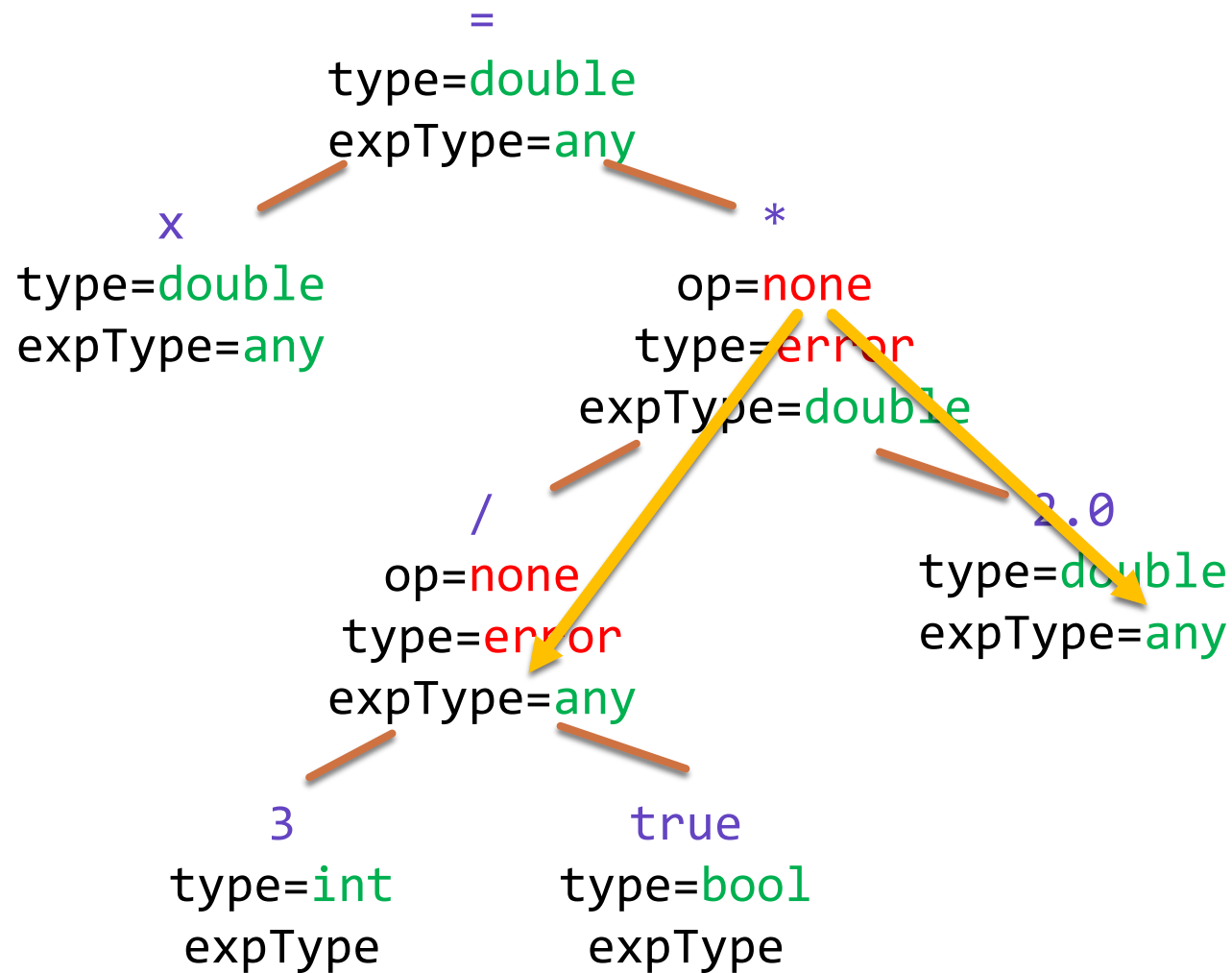
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

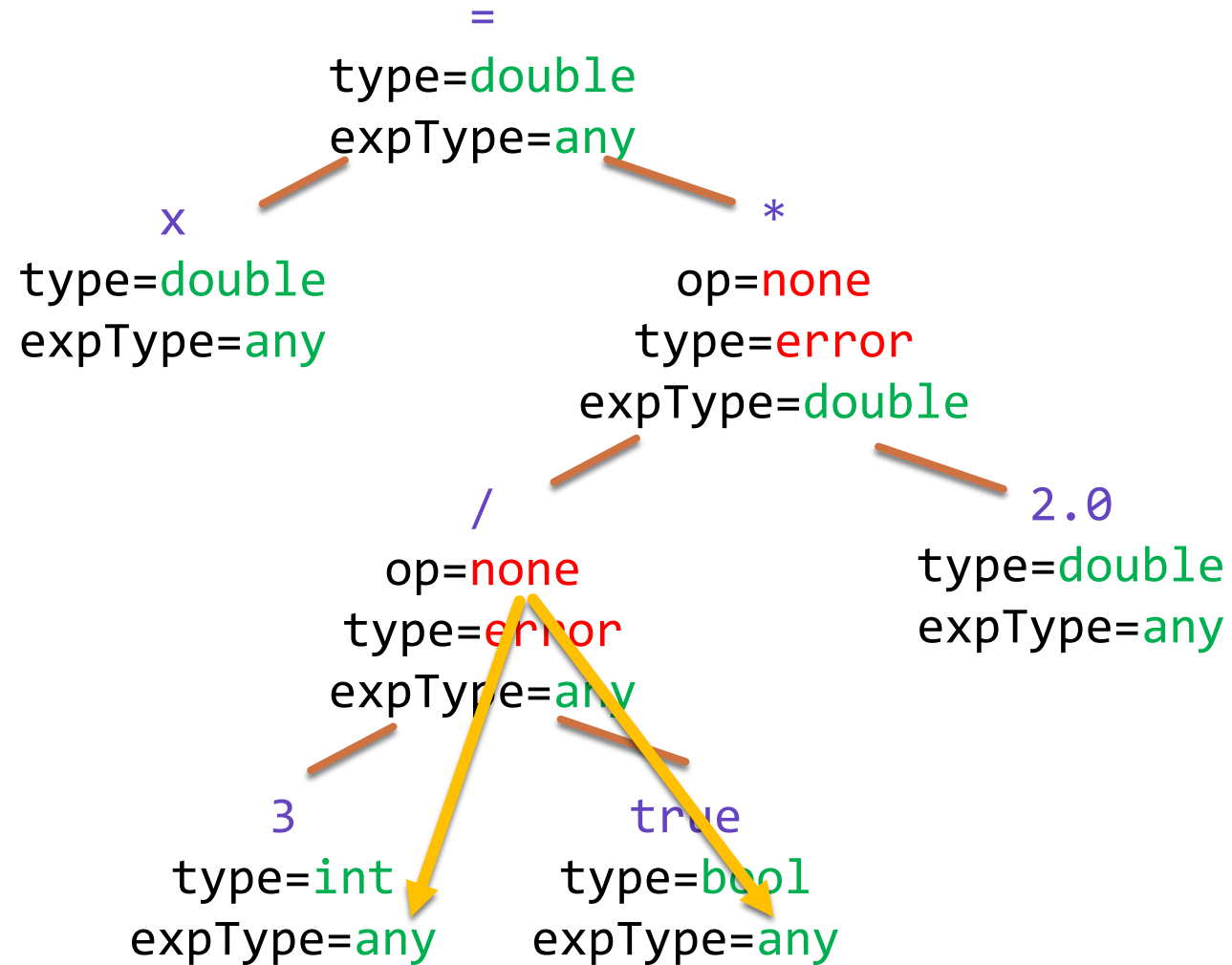
```
double x = 3/true*2.0;
```



Példa: típushiba

Utasítás:

```
double x = 3/true*2.0;
```

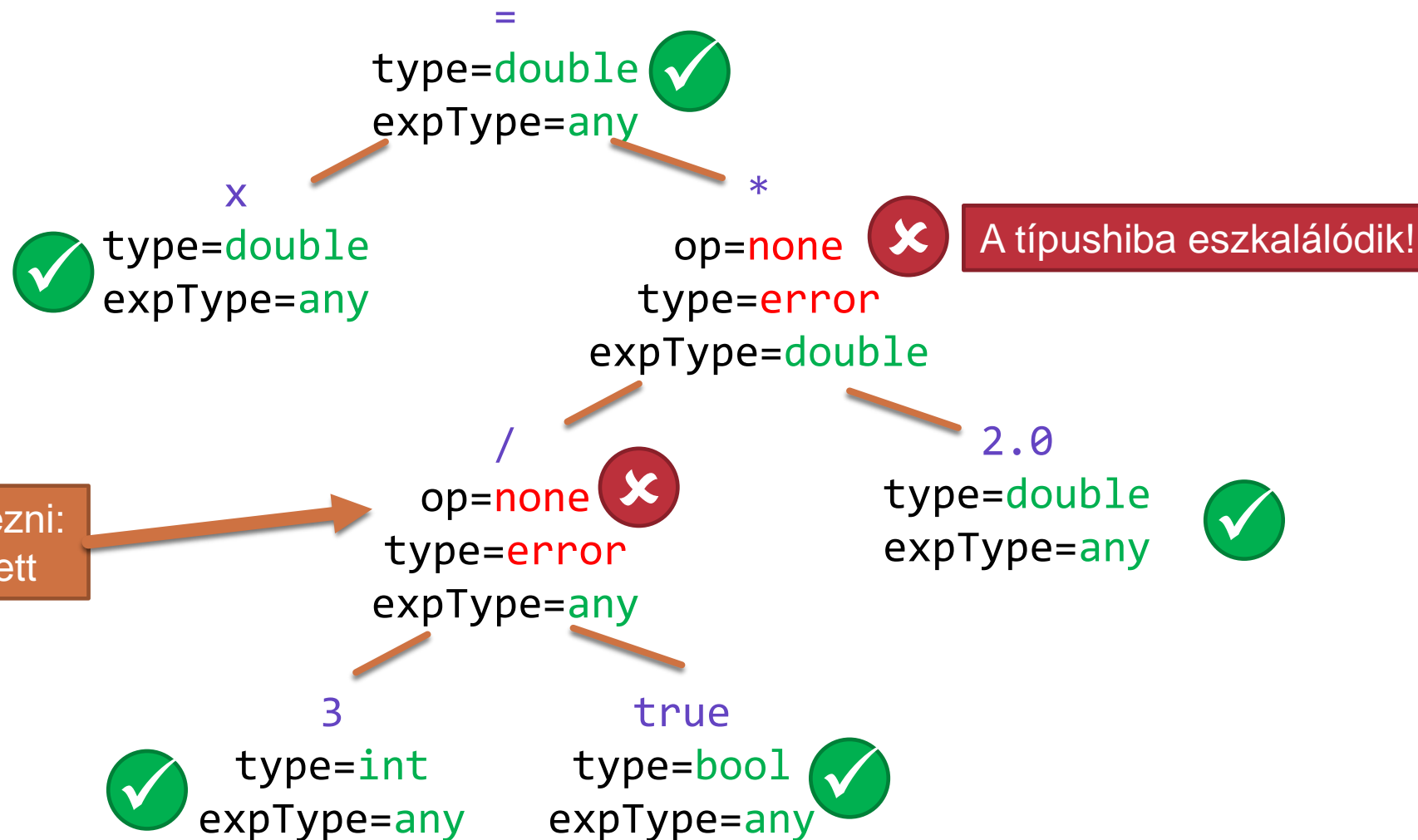


Példa: típushiba

Utasítás:

```
double x = 3/true*2.0;
```

Elegendő csak egy helyen jelezni:
ahol a kiindulási hiba keletkezett



A mai előadás: Szemantikai elemzés

I. Szemantikai elemzés

II. Attribútumnyelvtanok

III. Névelemzés

IV. Típuselemzés

V. Egyéb nyelvi szabályok



Egyéb nyelvi szabályok

- Adott programnyelvre specifikusak
- Példák:
 - > láthatóság (`private`, `protected`, `public`, stb.)
 - > virtuális függvények (`abstract`, `virtual`, `override`, `sealed`, stb.)
 - > interfész implementációja, többszörös öröklődés (multiple inheritance)
 - > `this`, `base`, `super` kulcsszavak
 - > memóriakezelés
 - > tömbök
 - > nullable típusok
 - > generikus típusok (generic types)
 - > típuskikövetkeztetés (type inference)
 - > dinamikus típusellenőrzés (dynamic type checking)

Szemantikai elemzés összefoglalás

- Attribútum nyelvtanok
- Szimbólumtábla
- Névelemzés
- Típuselemzés
- Operátorok azonosítása
- Konzisztencia-ellenőrzés
- Egyéb nyelvi szabályok

Ha a szintaktikai és a szemantikai elemző nem talál hibát:
a programkód megfelel a nyelvi specifikációnak -> lefordítható!



Köszönöm a figyelmet!