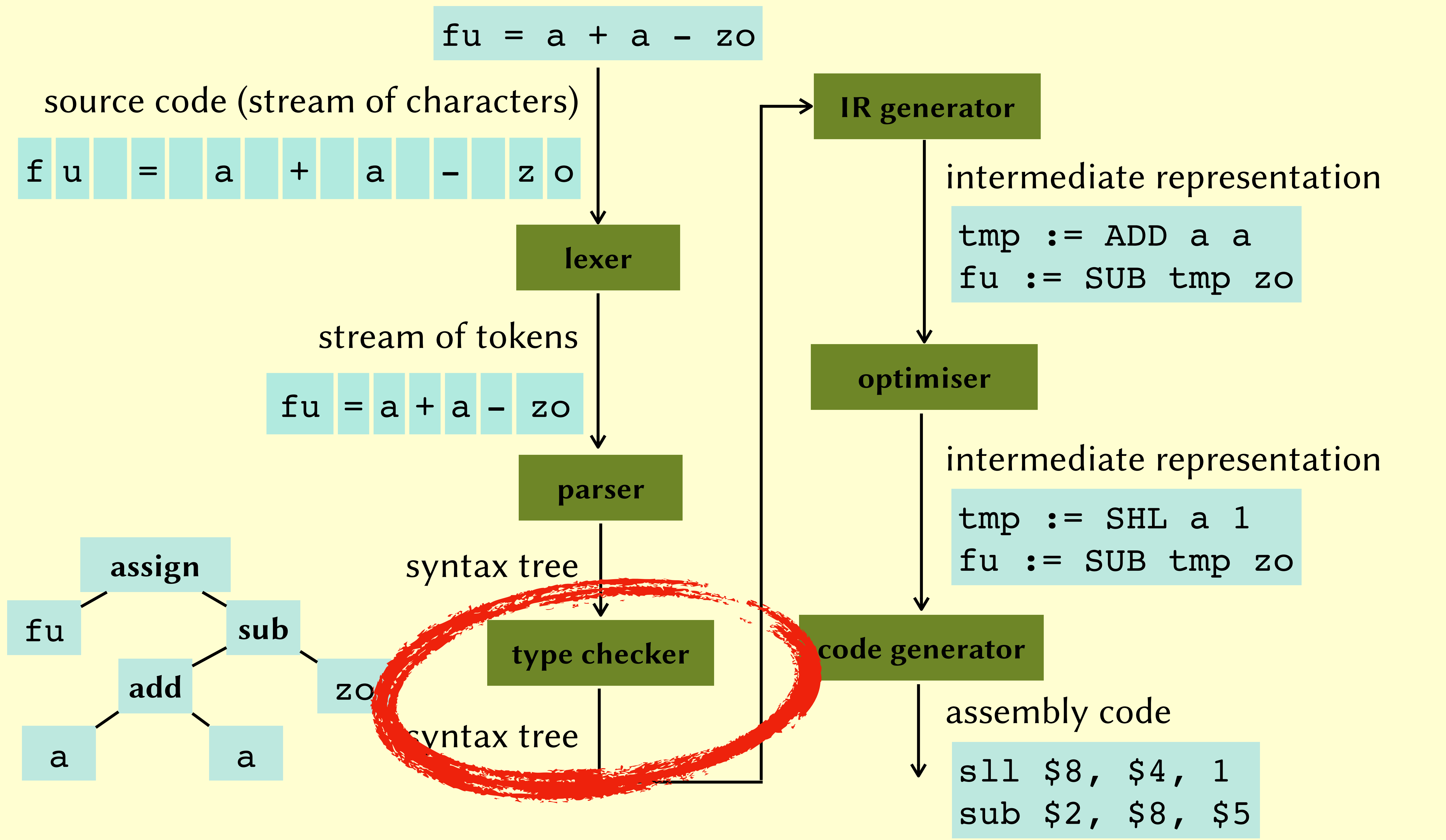
Lecture 6: Types

John Wickerson

Compilers

Anatomy of a compiler



Type checking

- Some programs are *syntactically valid* but *semantically invalid*.

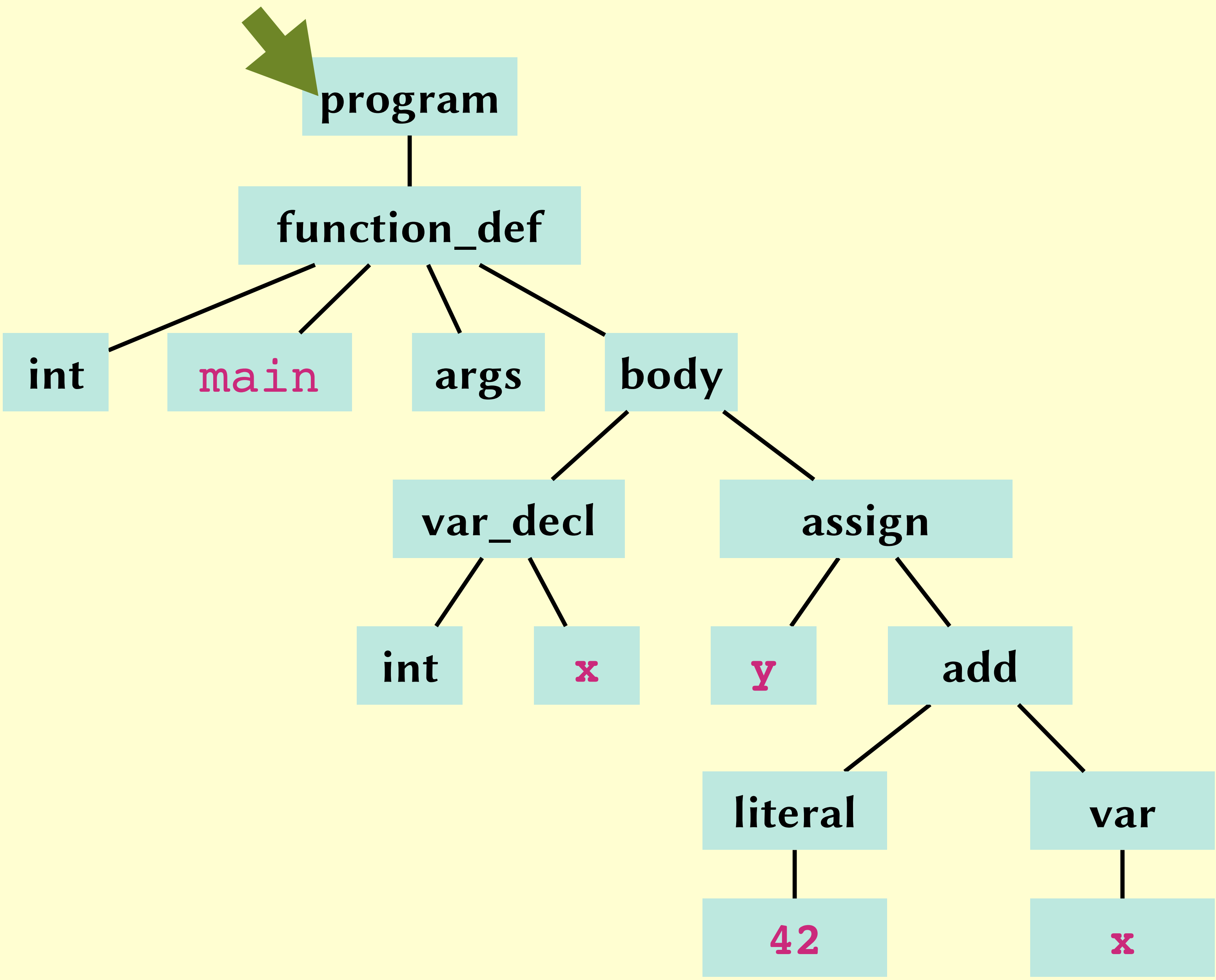
- Consider this (partial) grammar for C programs:

Prog	::=	Type **X** `(` Args `)` `{` Stmts `}` | …
Stmts	::=	ε | Stmt Stmts
Stmt	::=	Type **X** `;` | **X** `=` Expr `;` | …

- The program `int main () { int x; y = 42+x; }` would be accepted by this grammar, despite not being meaningful.

Type checking in C

```
int main () { int x; y = 42+x; }
```

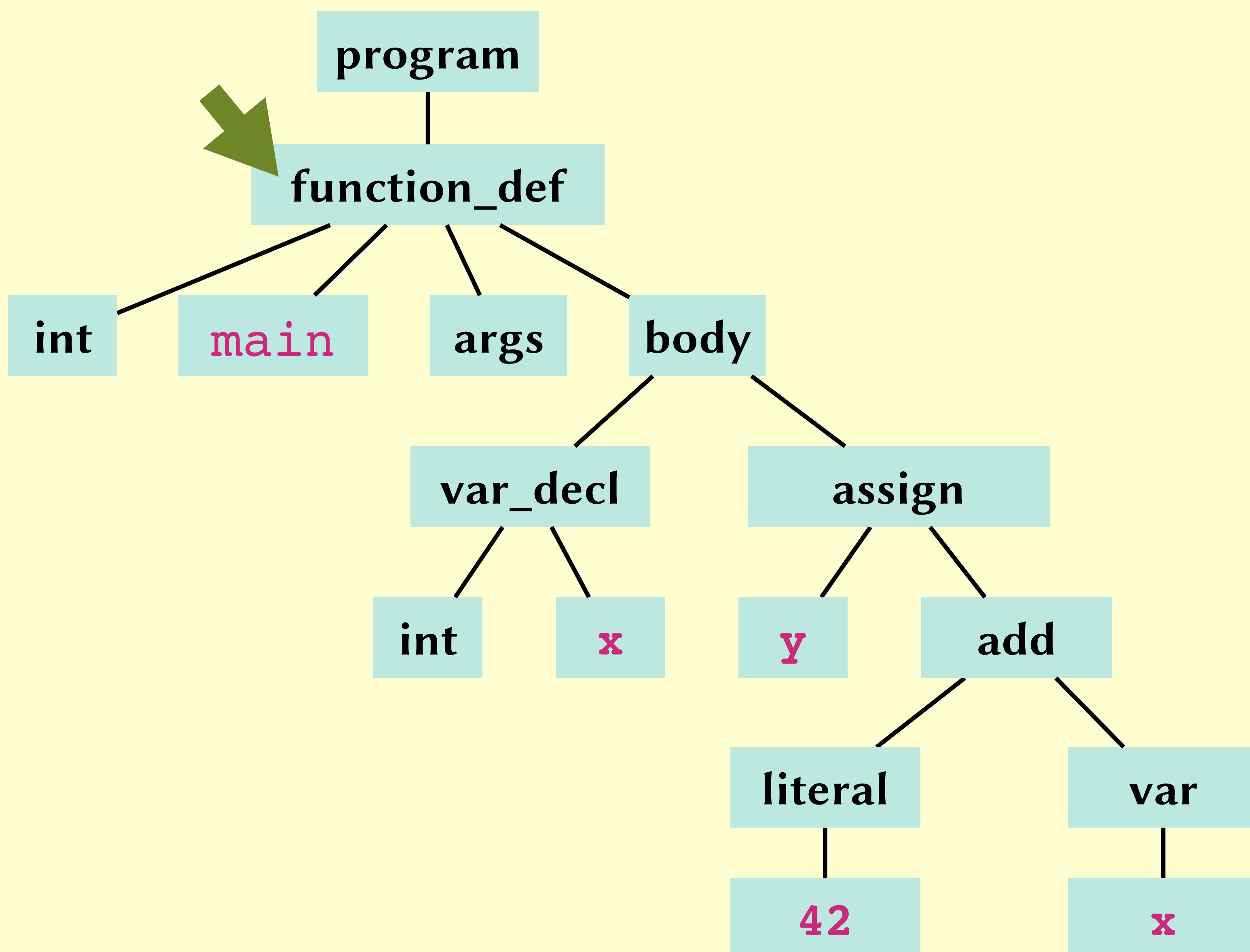


Symbol Table

Name	Type

Type checking in C

```
int main () { int x; y = 42+x; }
```

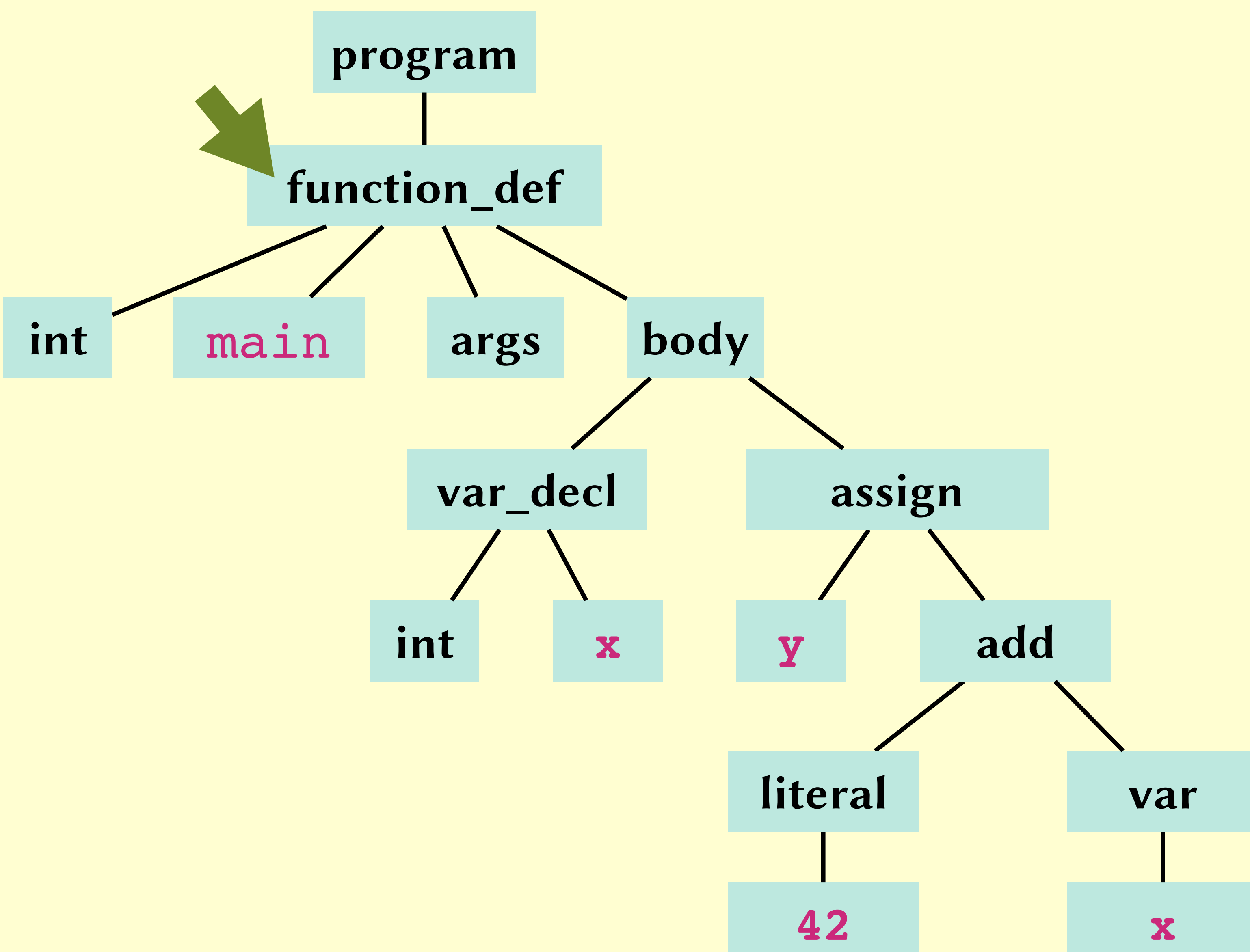


Symbol Table

Name	Type

Type checking in C

```
int main () { int x; y = 42+x; }
```

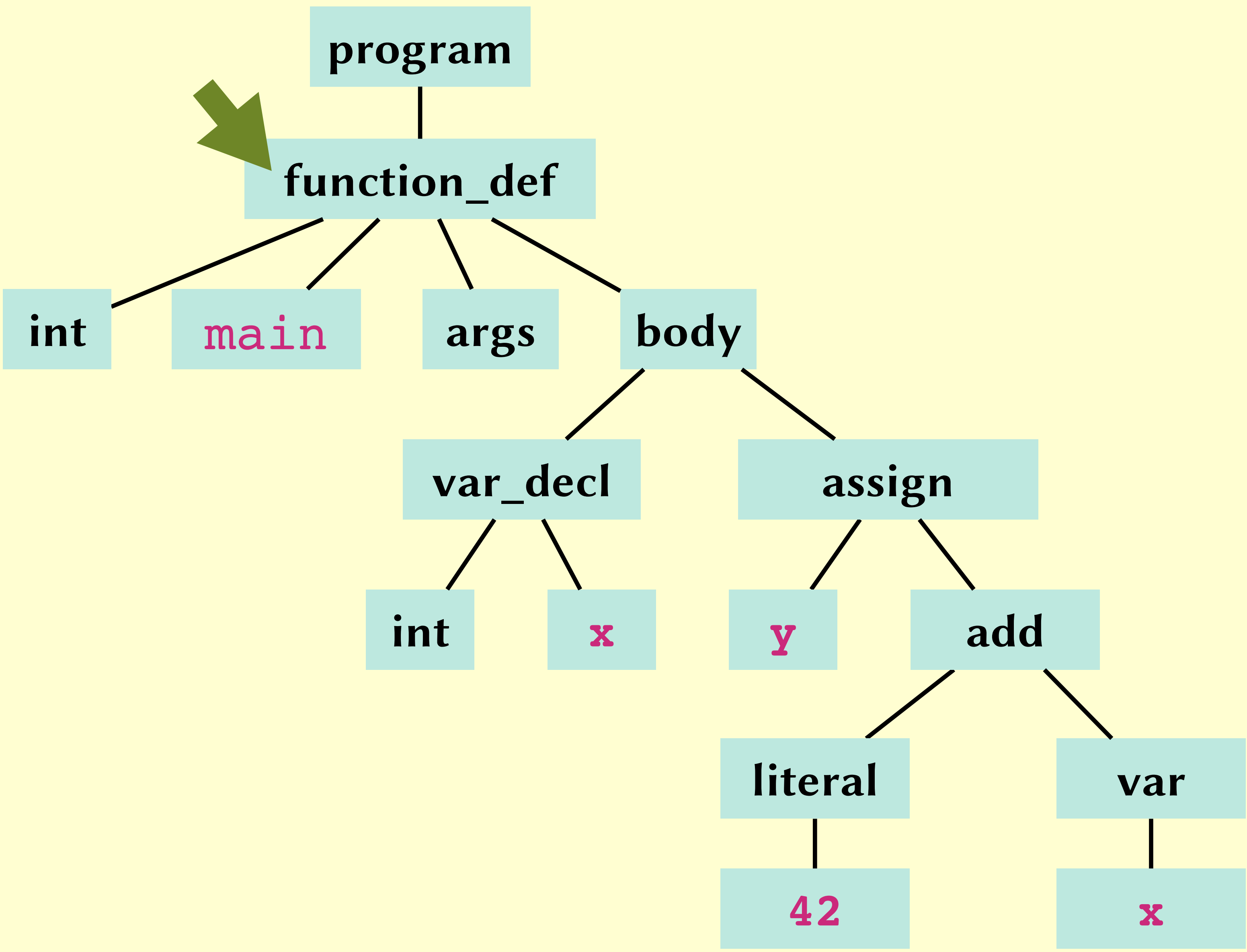


Symbol Table

Name	Type

Type checking in C

```
int main () { int x; y = 42+x; }
```

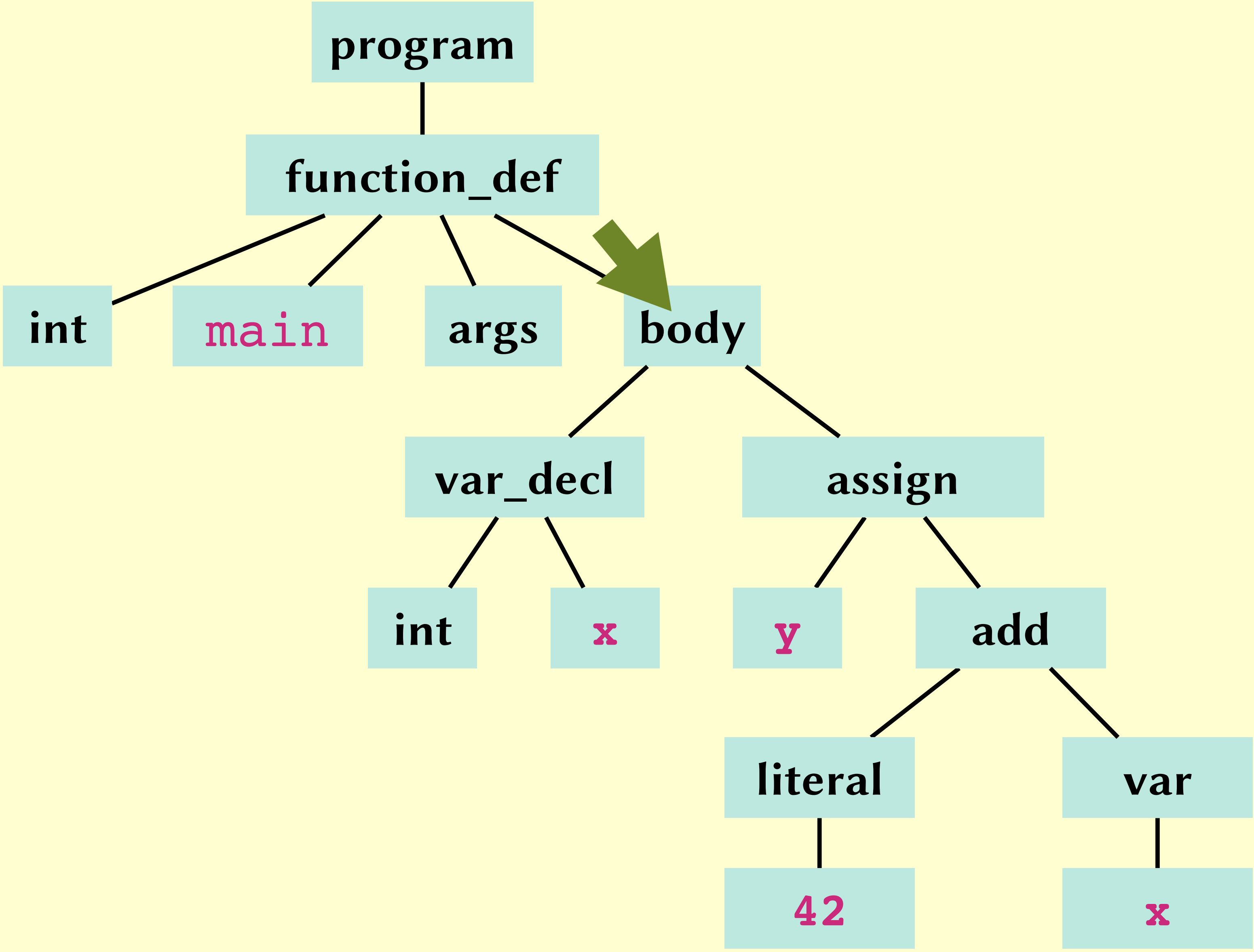


Symbol Table

Name	Type
`main`	void $\longrightarrow$ int

Type checking in C

```
int main () { int x; y = 42+x; }
```

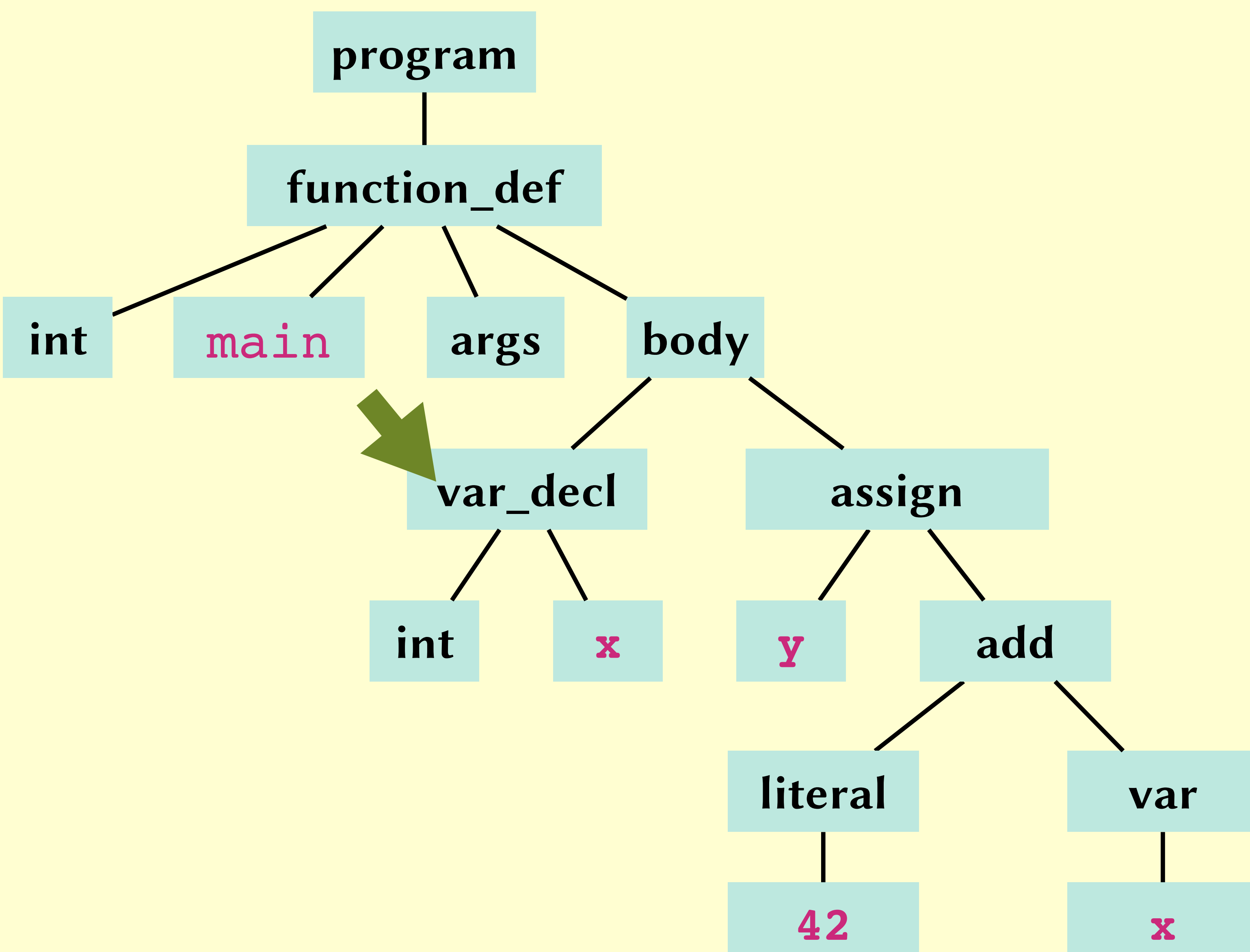


Symbol Table

Name	Type
main	void → int

Type checking in C

```
int main () { int x; y = 42+x; }
```

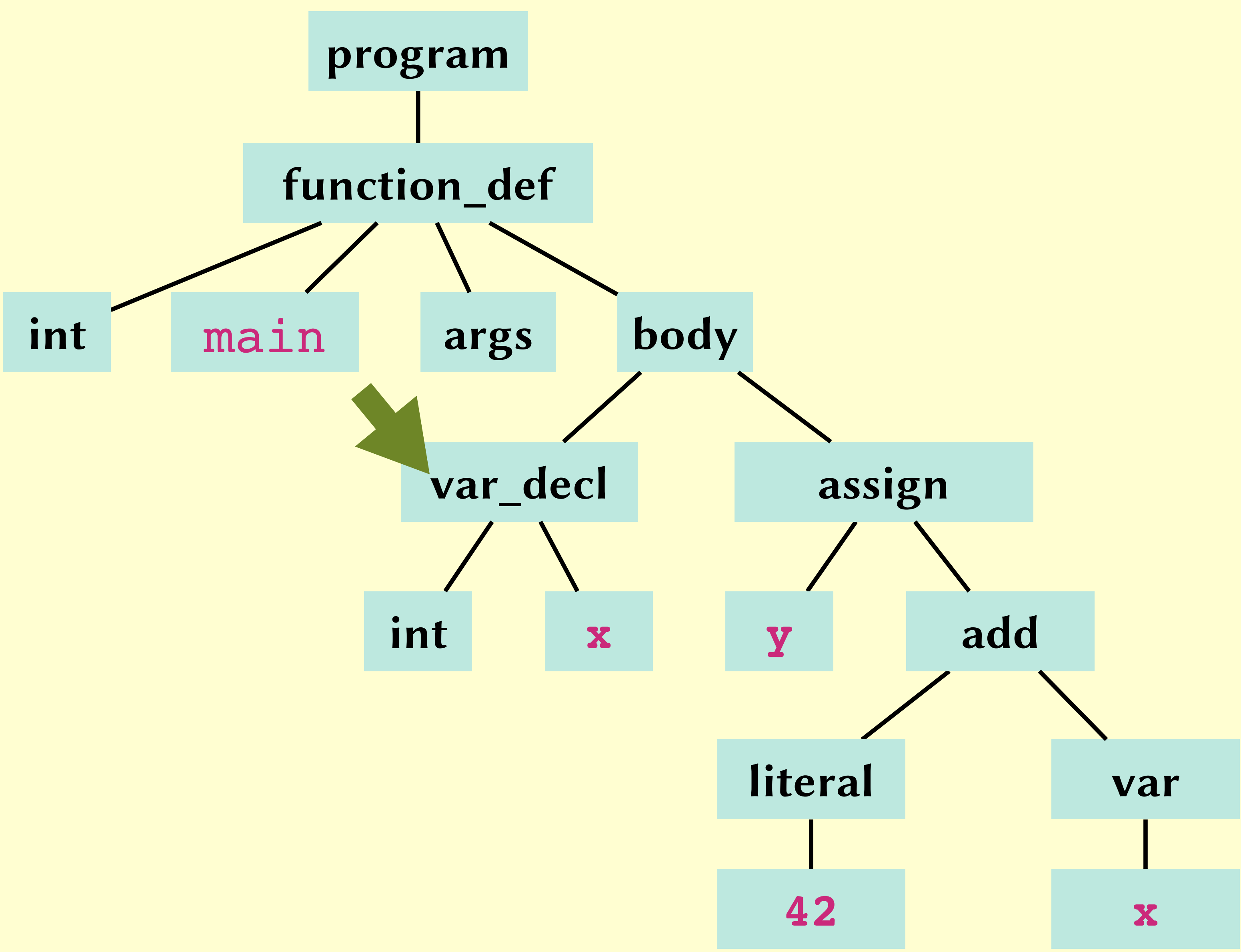


Symbol Table

Name	Type
`main`	void ⟶ int

Type checking in C

```
int main () { int x; y = 42+x; }
```

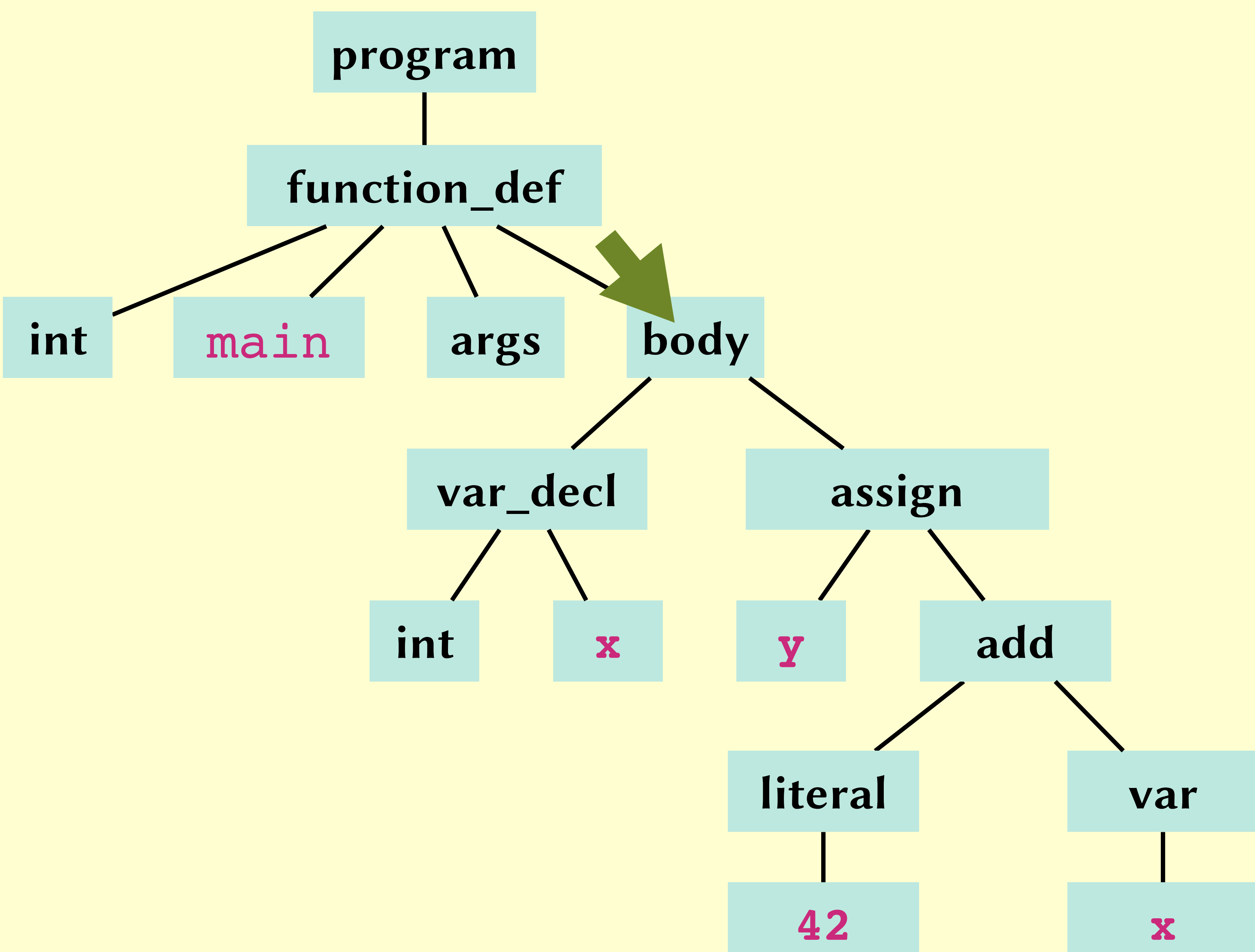


Symbol Table

Name	Type
main	void $\longrightarrow$ int
x	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

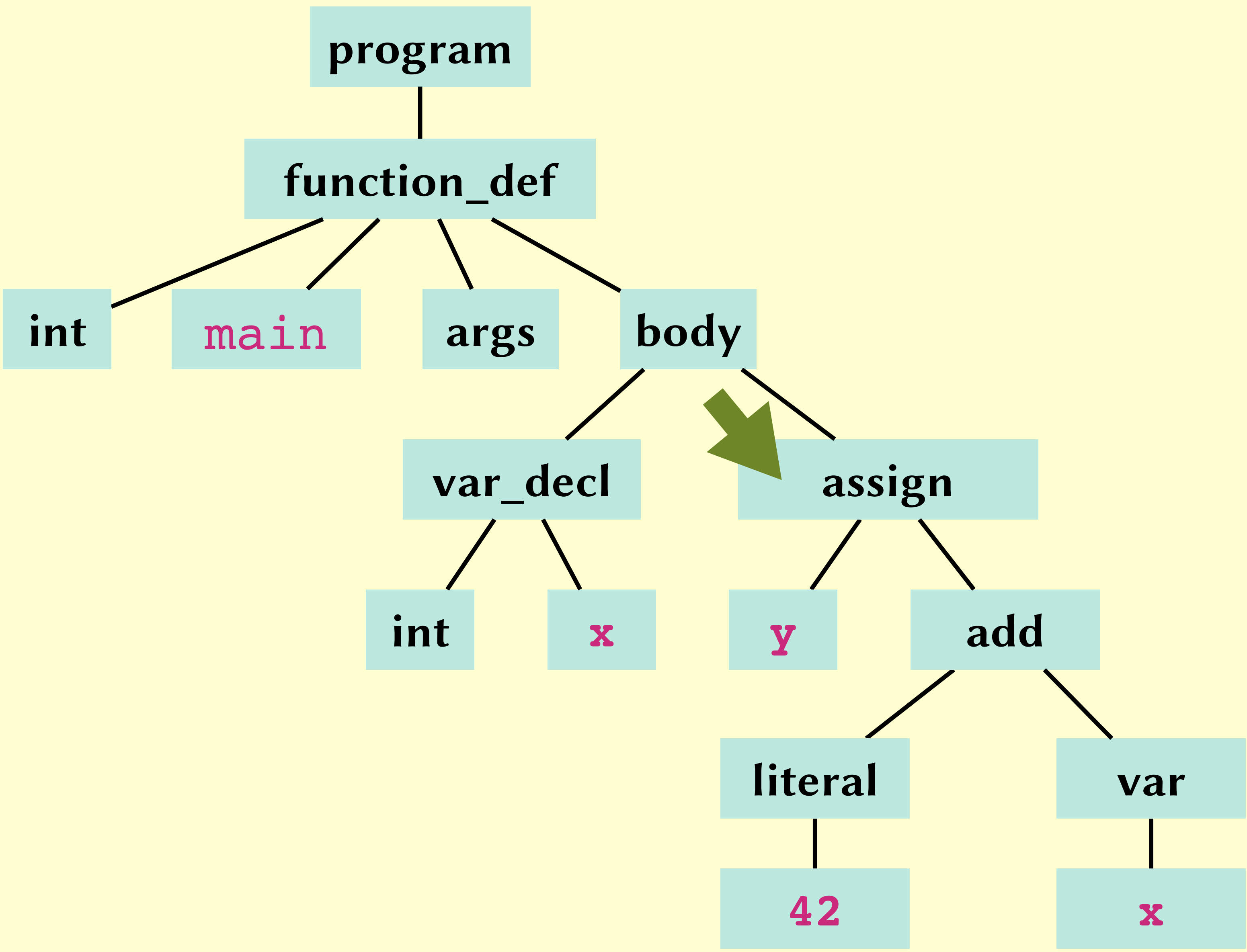


Symbol Table

Name	Type
`main`	void $\longrightarrow$ int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

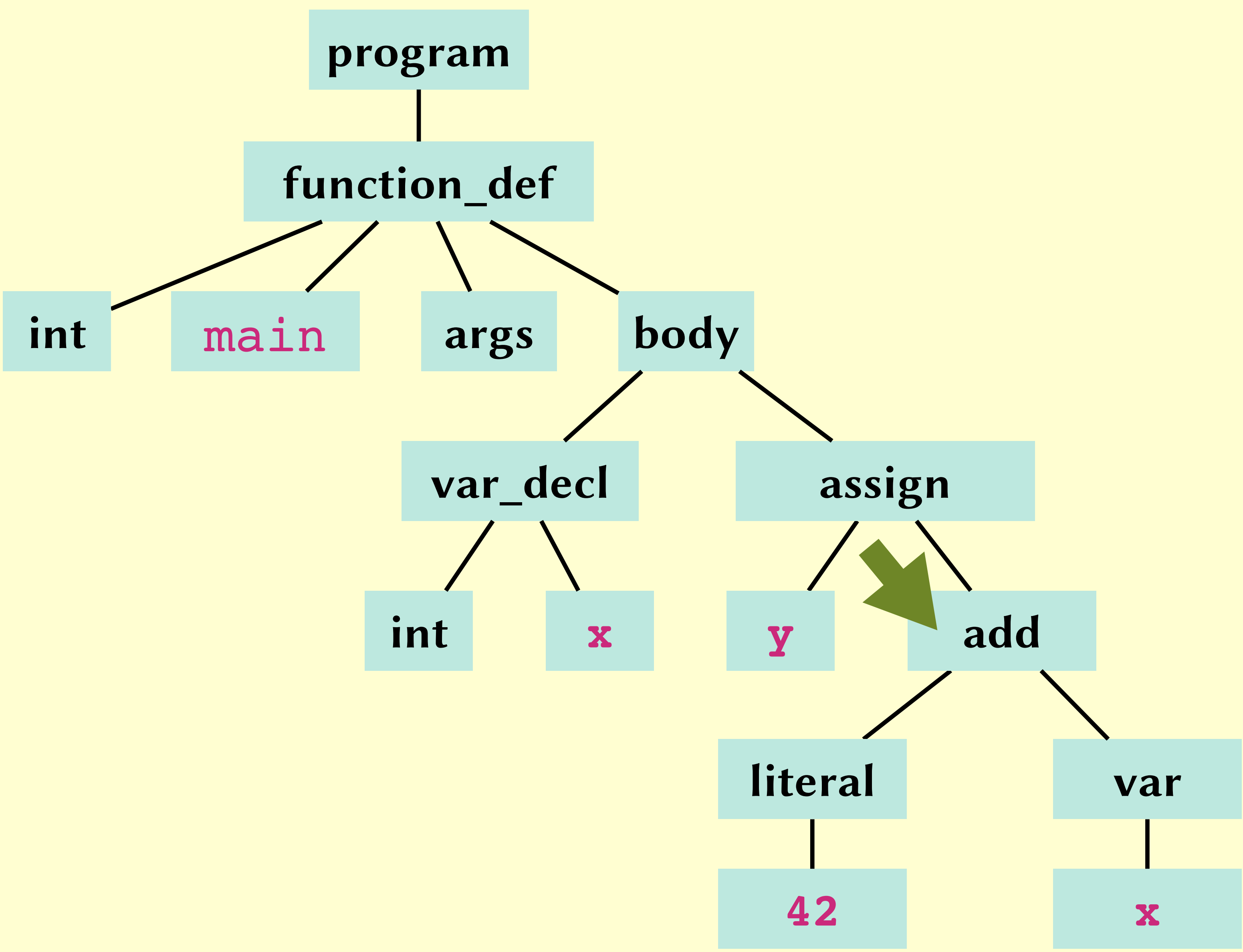


Symbol Table

Name	Type
`main`	void $\longrightarrow$ int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

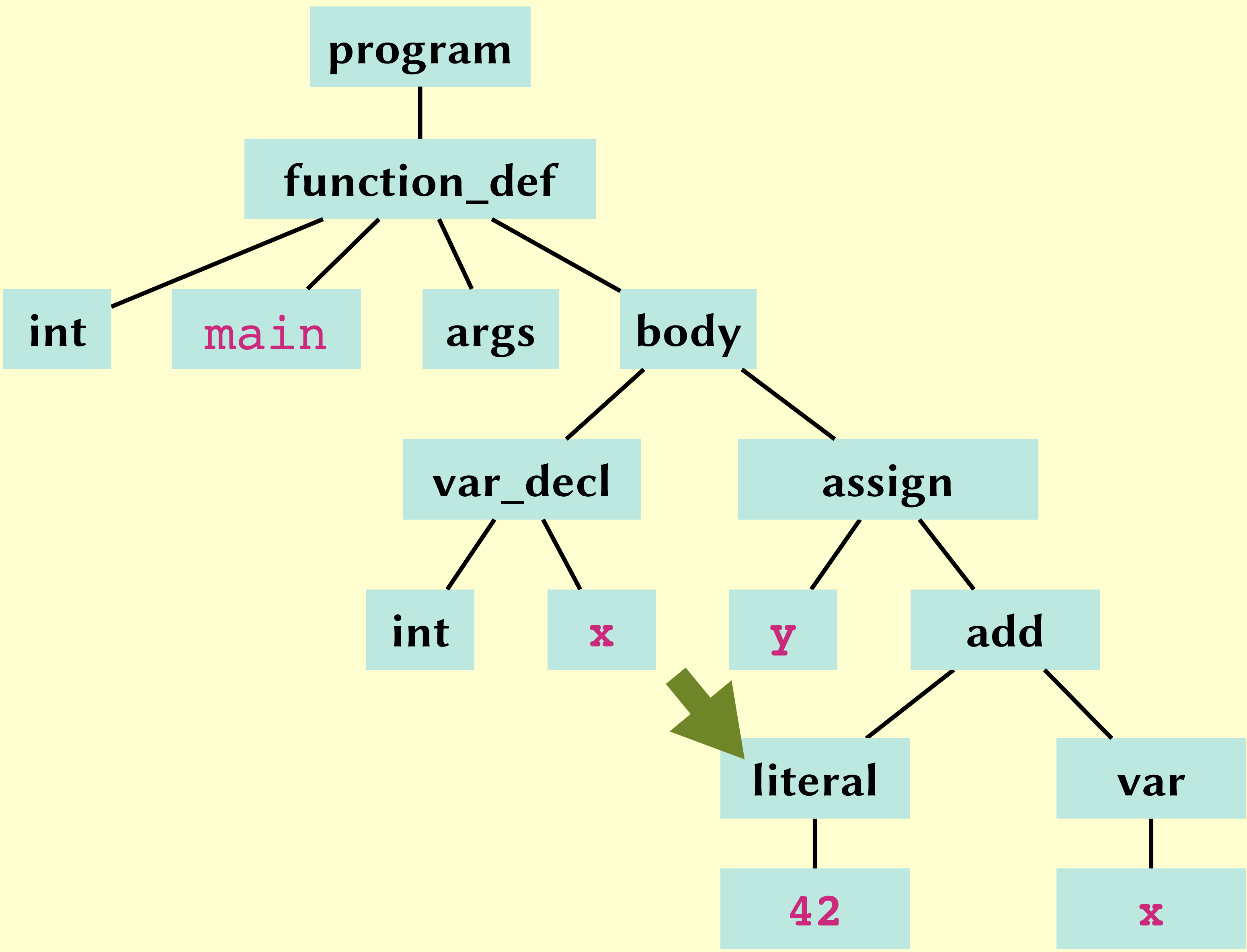


Symbol Table

Name	Type
main	void $\longrightarrow$ int
x	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

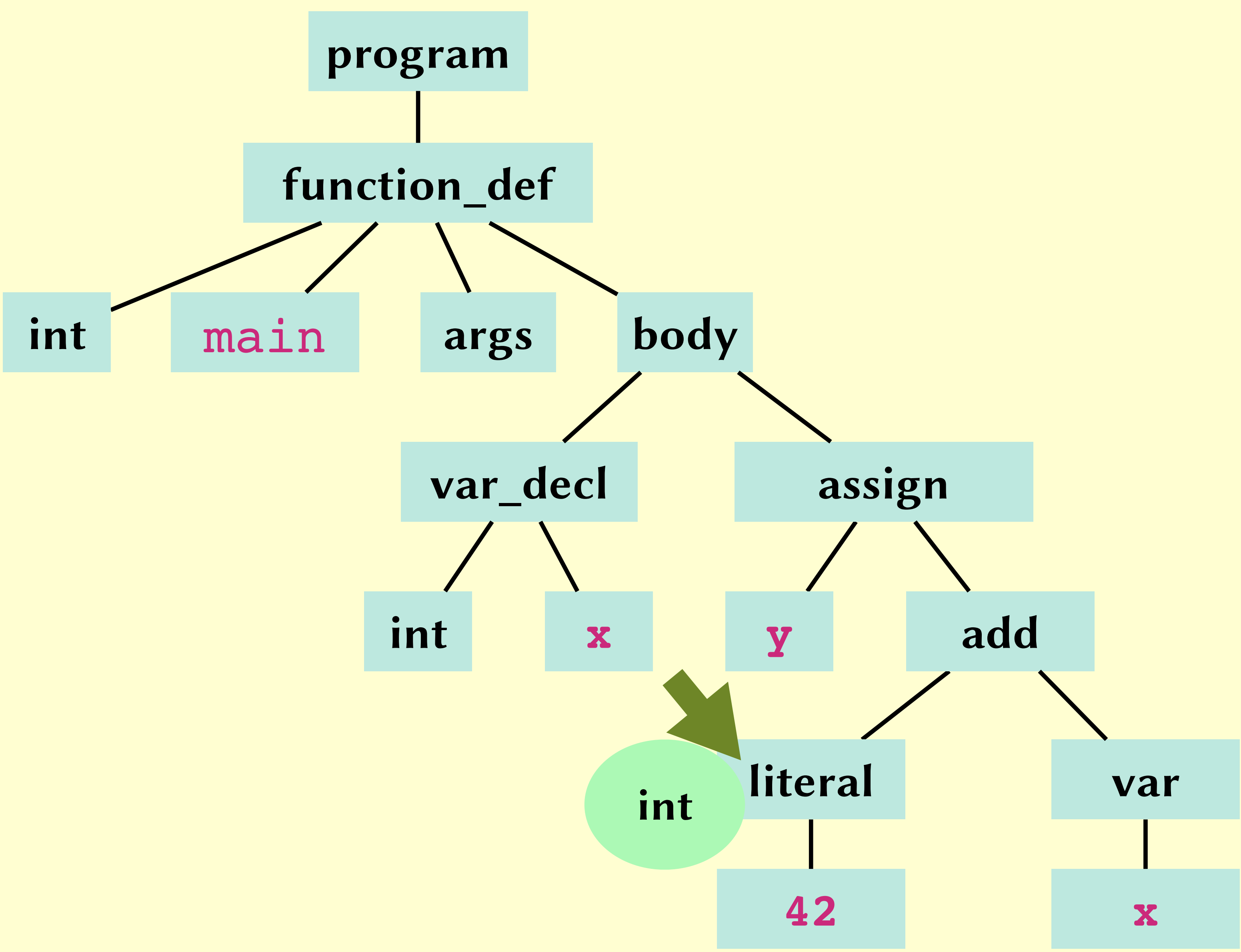


Symbol Table

Name	Type
main	void → int
x	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

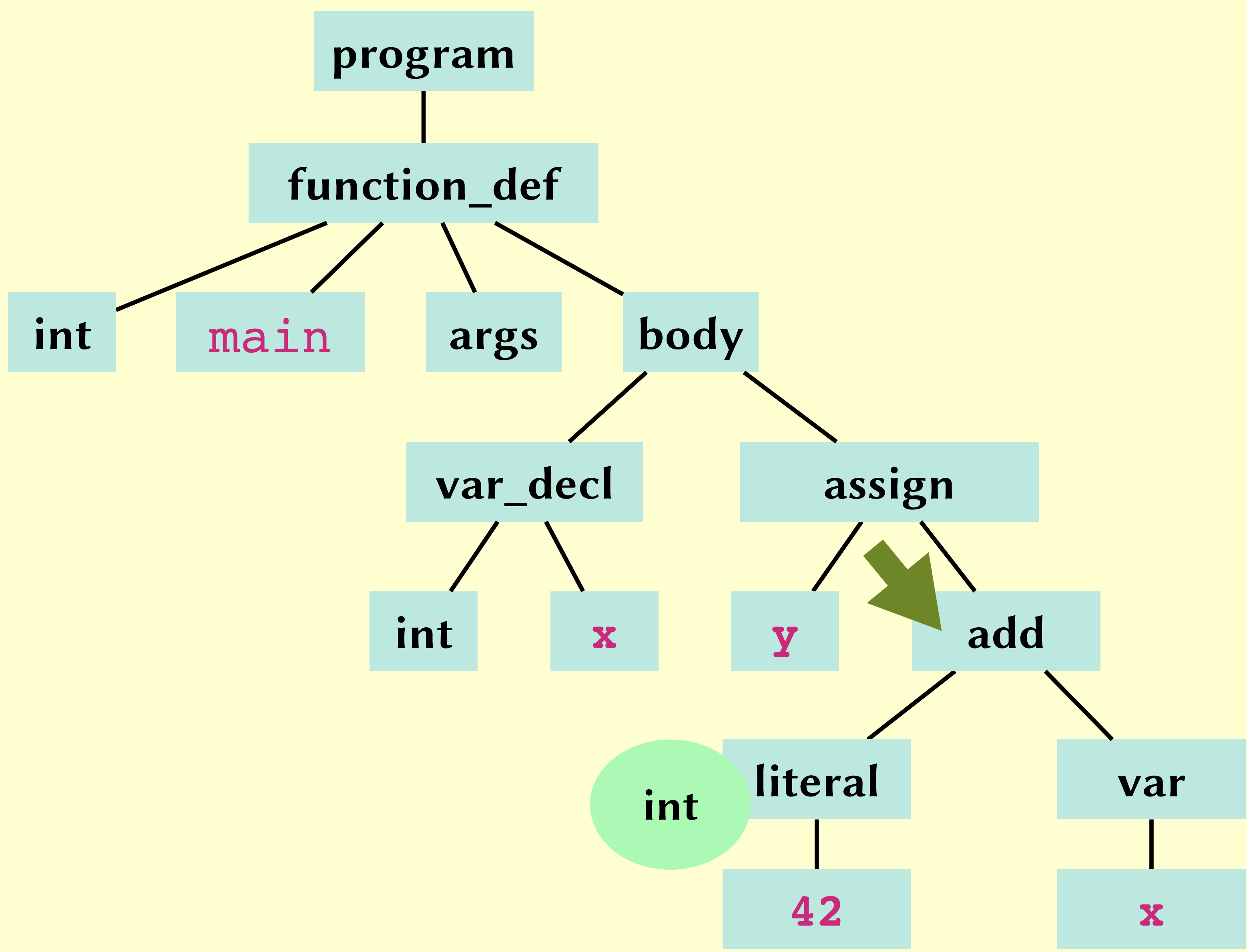


Symbol Table

Name	Type
`main`	void → int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

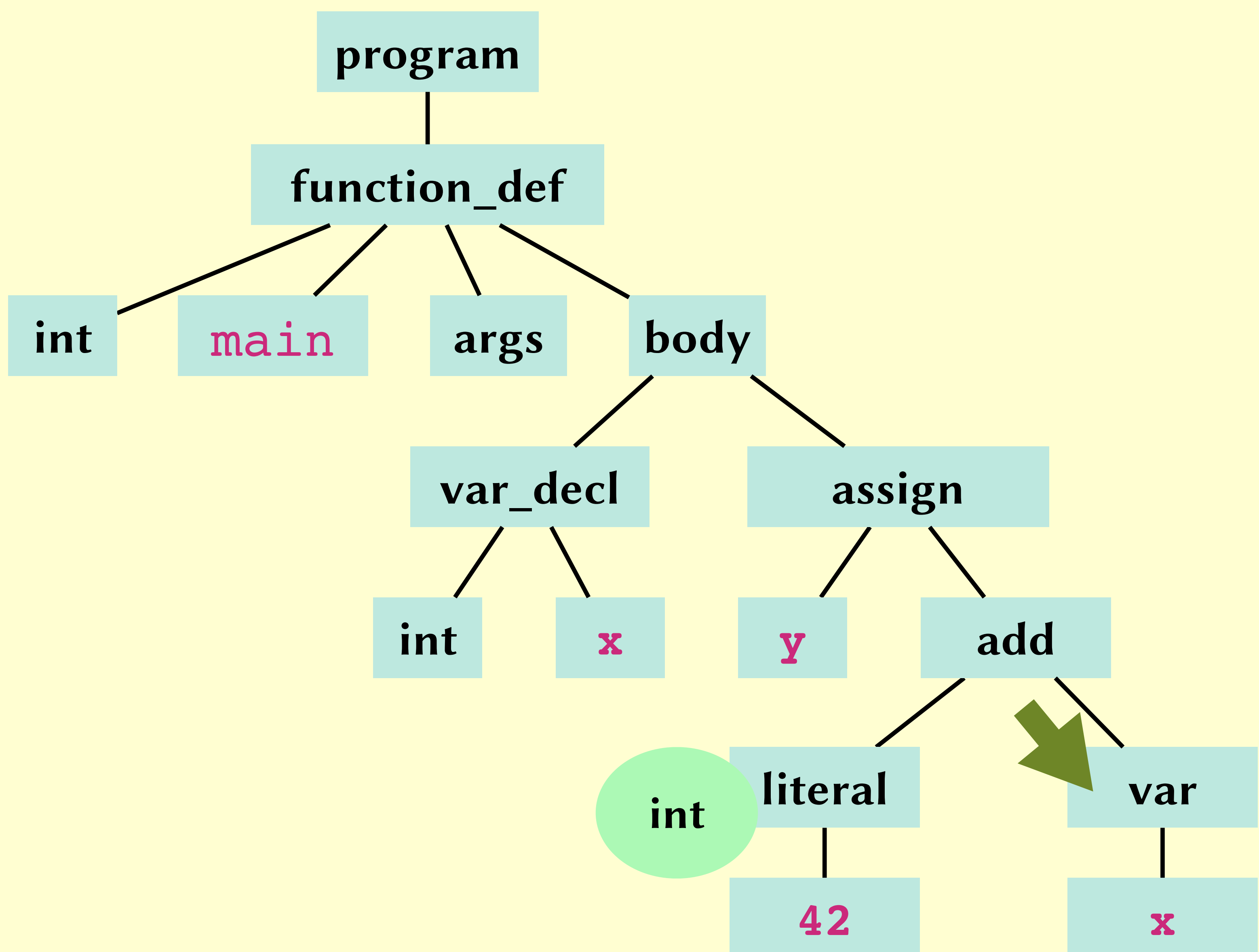


Symbol Table

Name	Type
main	void → int
x	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

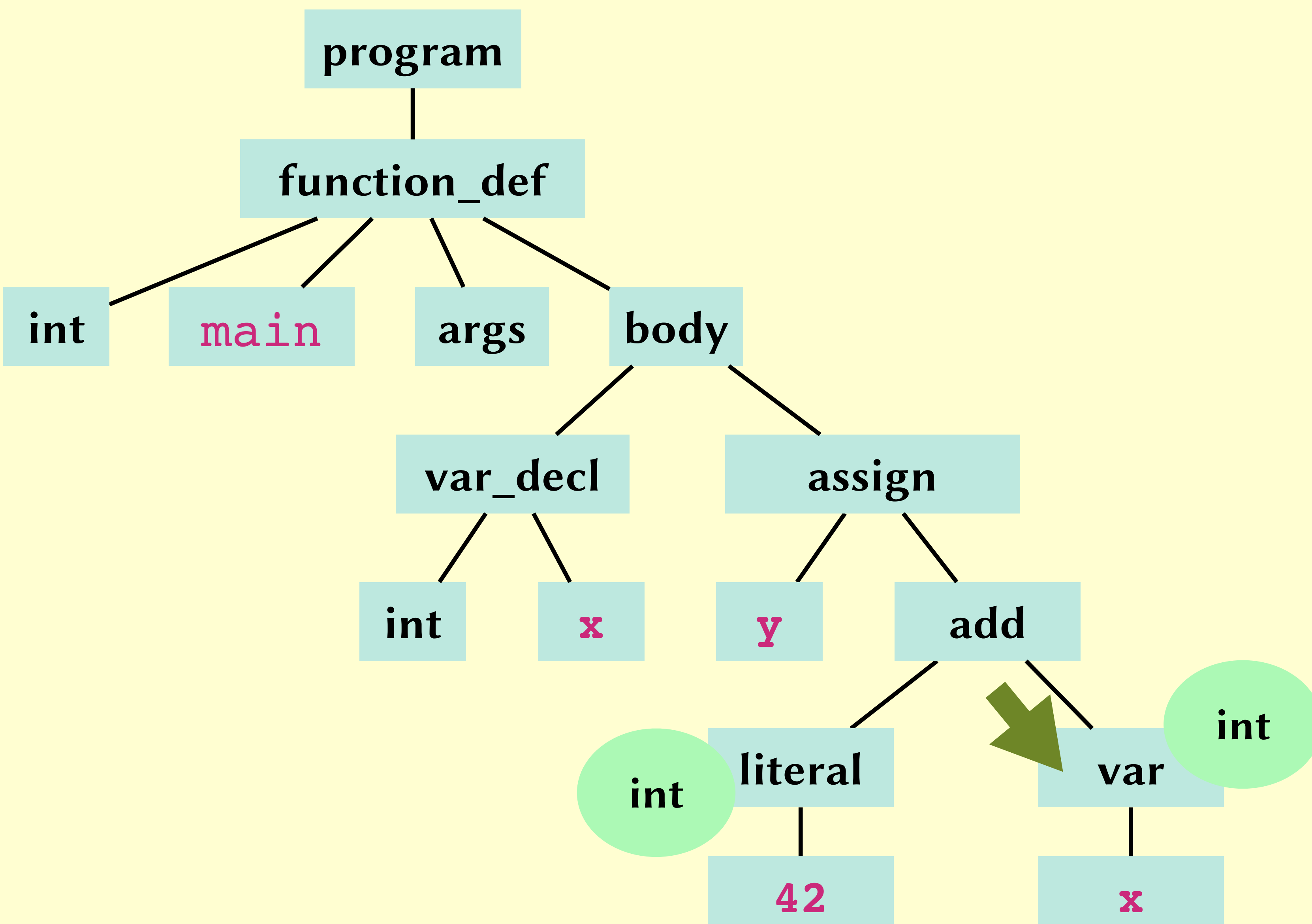


Symbol Table

Name	Type
`main`	void $\longrightarrow$ int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

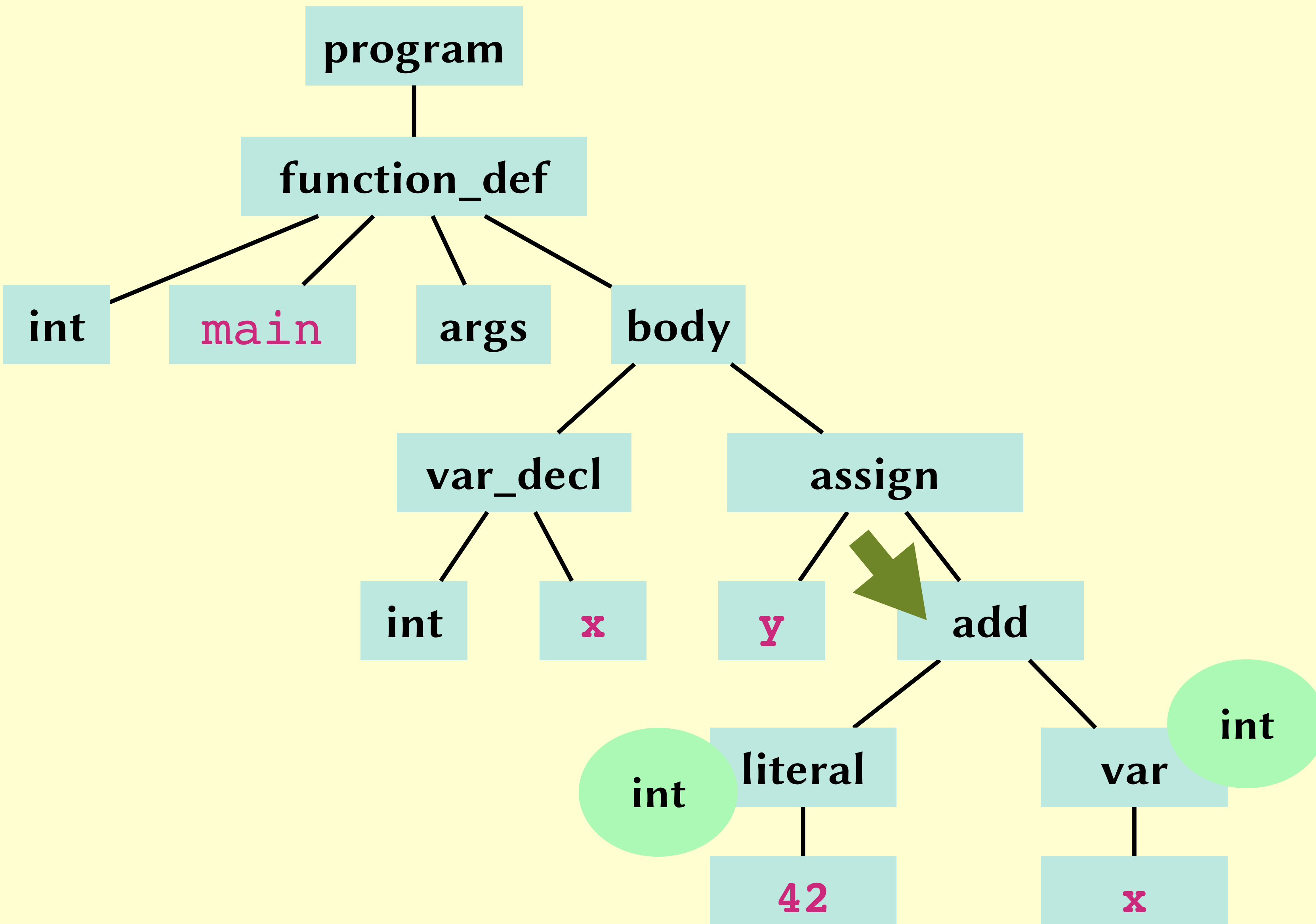


Symbol Table

Name	**Type**
`main`	void → int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

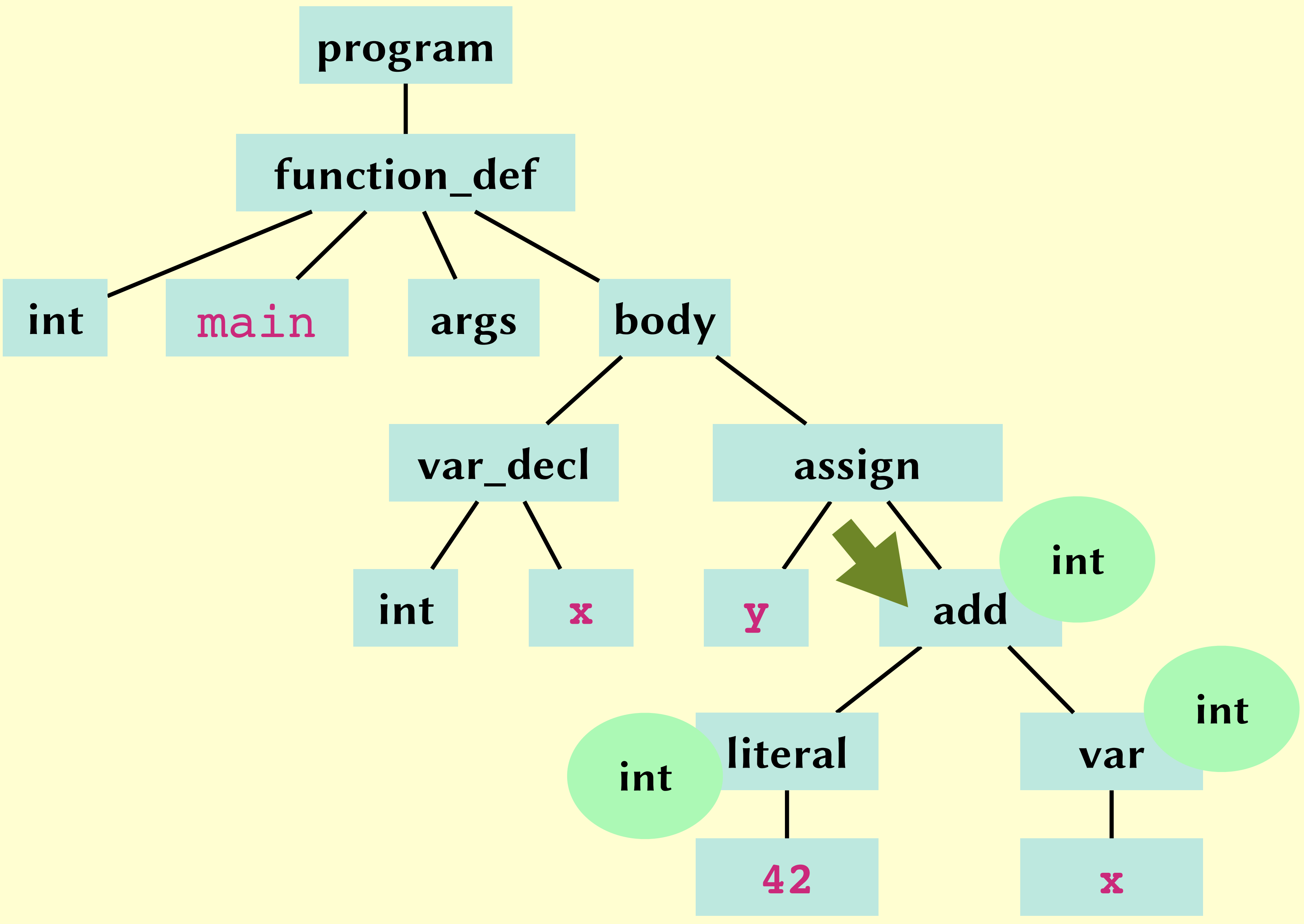


Symbol Table

Name	Type
main	void → int
x	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

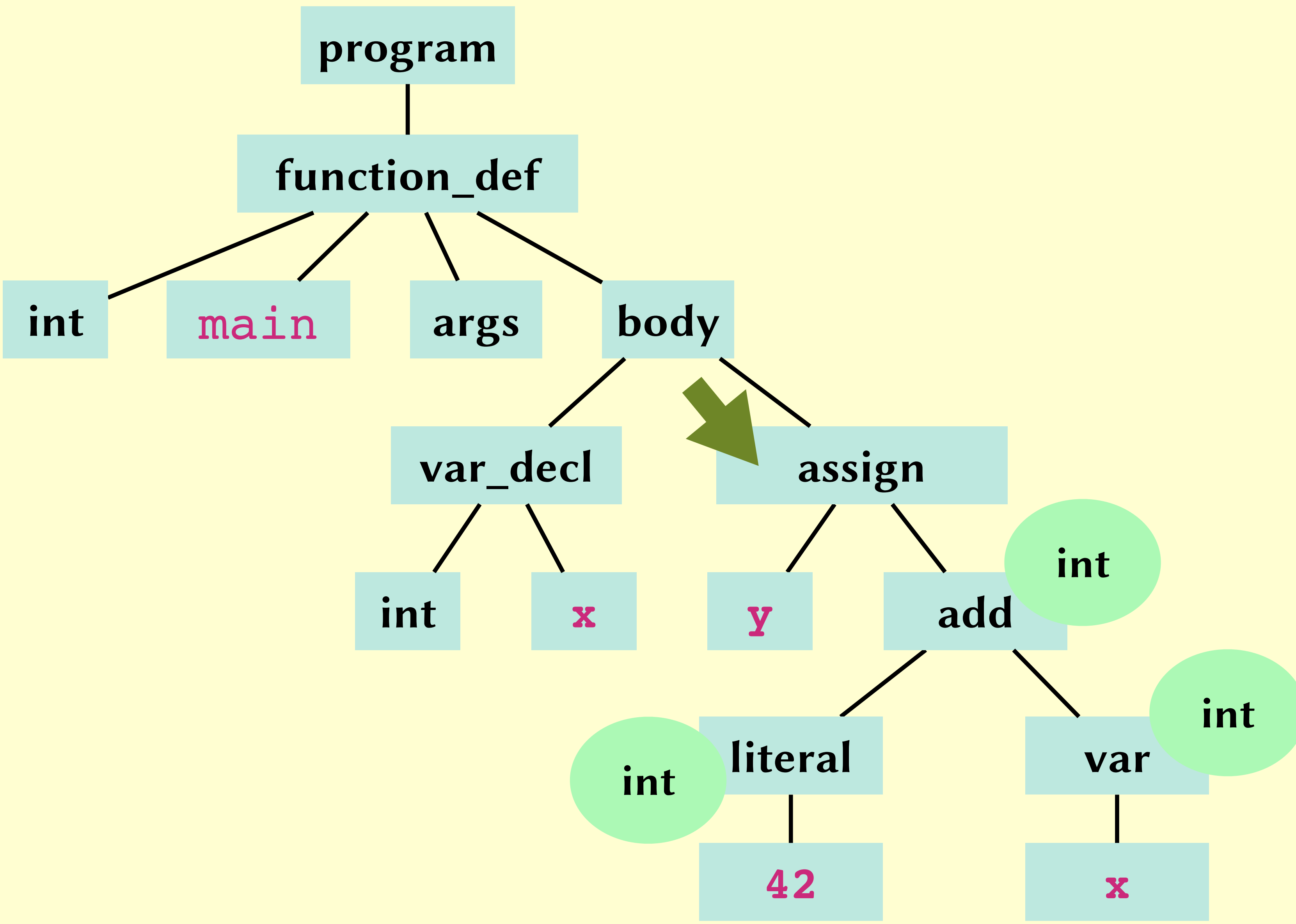


Symbol Table

Name	Type
`main`	void $\longrightarrow$ int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```

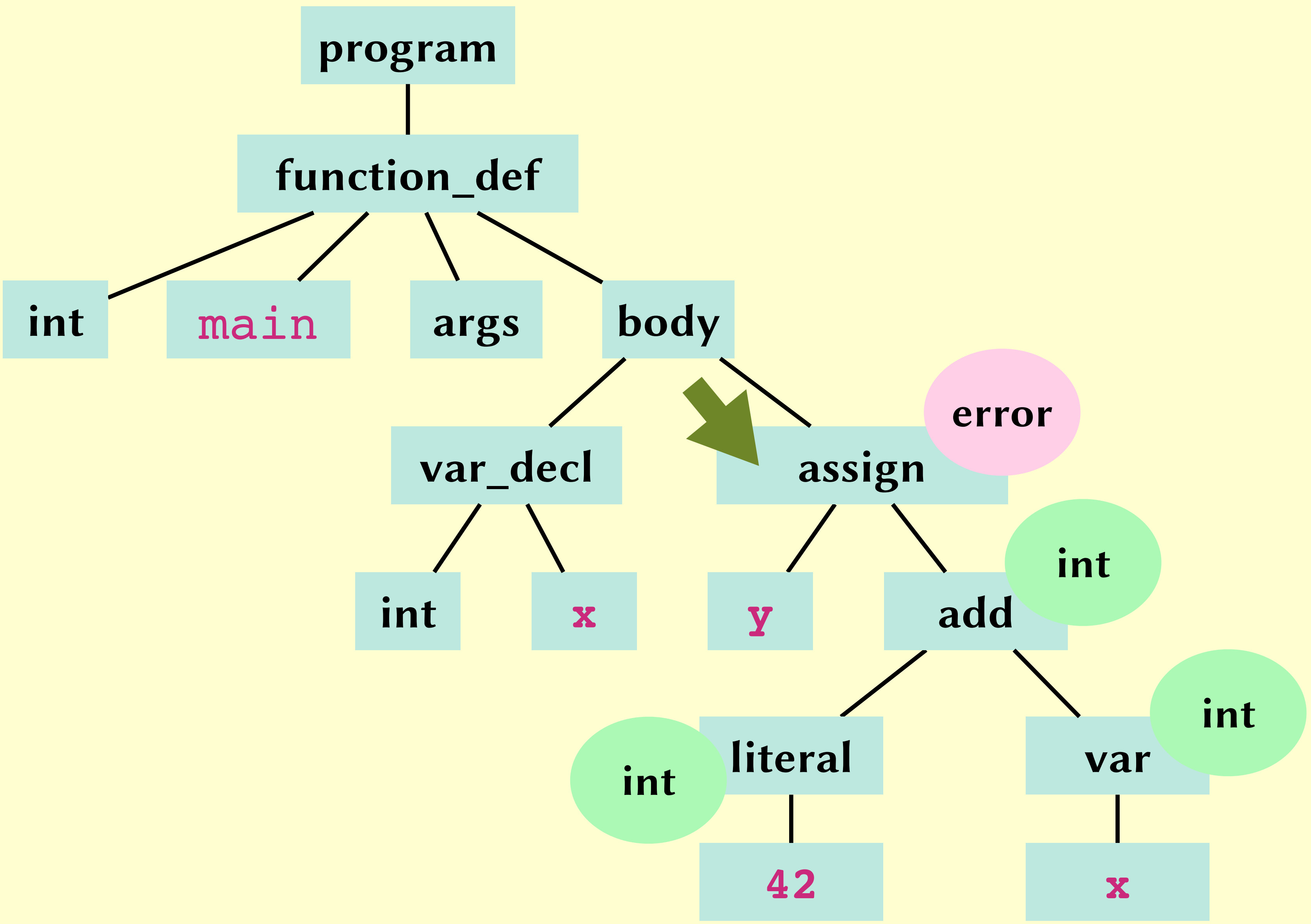


Symbol Table

Name	Type
`main`	void $\rightarrow$ int
`x`	int

Type checking in C

```
int main () { int x; y = 42+x; }
```



Symbol Table

Name	Type
`main`	void $\longrightarrow$ int
`x`	int

Type checking in C

- Another example, featuring *function calls*.

```
void foo(int a) {...}

int baz(int b, char c) {...}

int main() {
  foo(42);
  return baz(17,'g');
}
```

Name	Type
foo	int $\longrightarrow$ void
baz	(int $\times$ char) $\longrightarrow$ int
main	void $\longrightarrow$ int

Type checking in C

- Convenient time to check for other programming errors.

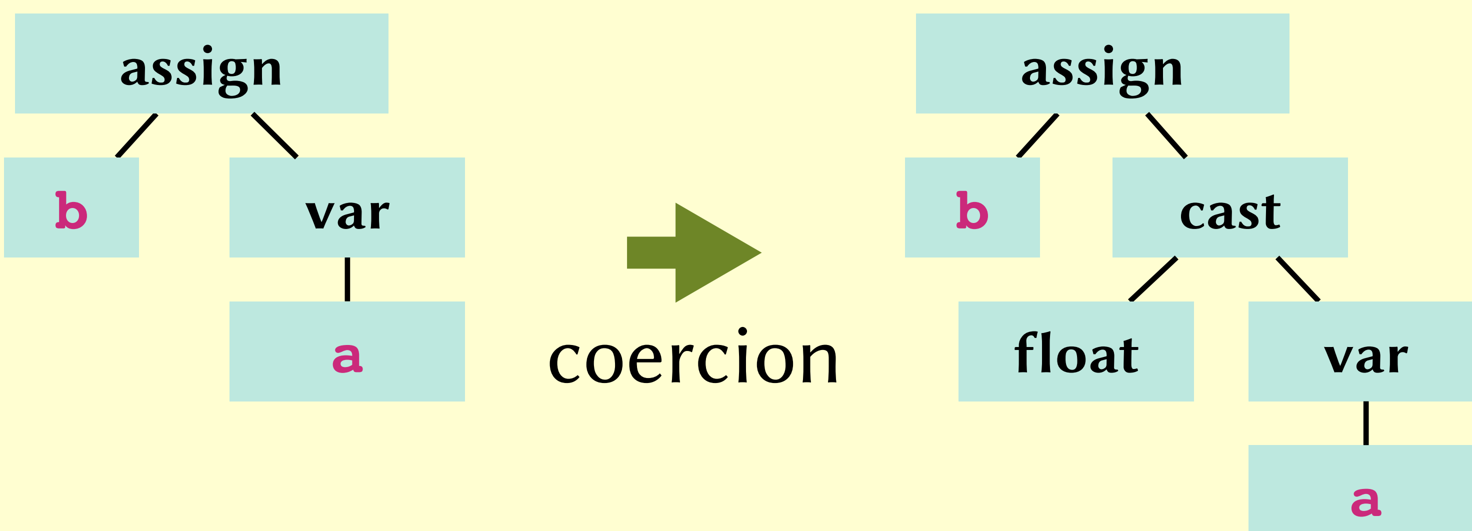
```
switch(x) {
case 1: y=42;
case 3: y=45;
case 1: y=0;
}
```

```
int main() {
  break;
  return 0;
}
```

```
int main() {
  int x;
  int x;
  return 0;
}
```

Type checking in C

- Types don't always need to match *exactly*.



Name	Type
`a`	`int`
`b`	`float`

Type systems

✔ Type checking
- Type inference
- Polymorphic typing
- Subtyping
- Even fancier type systems

A little language

Type ::= `int` | `bool` | Type → Type

`int` → `bool`

`bool` → (`int` → `bool`)

(`int` → `int`) → `int`

A little language

Type ::= `int` | `bool` | Type → Type

Expr ::= **x** *// variables*

`foo` `baz`

A little language

$$\text{Type} ::= \texttt{int} \mid \texttt{bool} \mid \text{Type} \rightarrow \text{Type}$$

$$\begin{array}{lll} \text{Expr} ::= & \mathbf{X} & \textit{// variables} \\ \mid & \mathbf{N} & \textit{// integer literals} \end{array}$$

`2` `42`

A little language

$$\begin{array}{lcll}
\text{Type} & ::= & \texttt{int} \mid \texttt{bool} \mid \text{Type} \rightarrow \text{Type} & \\
\\
\text{Expr} & ::= & \mathbf{X} & \textit{// variables} \\
 & \mid & \mathbf{N} & \textit{// integer literals} \\
 & \mid & \text{Expr} + \text{Expr} & \textit{// integer addition}
\end{array}$$

```
foo + 42
```

A little language

Type ::= `int` | `bool` | Type → Type

Expr ::= **X** *// variables*
| **N** *// integer literals*
| Expr + Expr *// integer addition*
| Expr > Expr *// integer comparison*

```
(foo + 42) > 59
```

A little language

$$\begin{array}{lcll} \text{Type} & ::= & \texttt{int} \mid \texttt{bool} \mid \text{Type} \rightarrow \text{Type} & \\ \\ \text{Expr} & ::= & \mathbf{X} & \textit{// variables} \\ & \mid & \mathbf{N} & \textit{// integer literals} \\ & \mid & \text{Expr} + \text{Expr} & \textit{// integer addition} \\ & \mid & \text{Expr} > \text{Expr} & \textit{// integer comparison} \\ & \mid & \textbf{if}\ \text{Expr}\ \textbf{then}\ \text{Expr}\ \textbf{else}\ \text{Expr} & \textit{// if-expressions} \end{array}$$

```
if (foo + 42) > 59 then 1 else 0
```

A little language

$$\text{Type} ::= \texttt{int} \mid \texttt{bool} \mid \text{Type} \rightarrow \text{Type}$$

Expr	::=	**X**	*// variables*
	|	**N**	*// integer literals*
	|	Expr + Expr	*// integer addition*
	|	Expr > Expr	*// integer comparison*
	|	**`if`** Expr **`then`** Expr **`else`** Expr	*// if-expressions*
	|	**`true`**	*// boolean literal*
	|	**`false`**	*// boolean literal*

A little language

Type ::= `int` | `bool` | Type → Type

Expr ::= **X** // *variables*
| **N** // *integer literals*
| Expr + Expr // *integer addition*
| Expr > Expr // *integer comparison*
| **if** Expr **then** Expr **else** Expr // *if-expressions*
| **true** // *boolean literal*
| **false** // *boolean literal*
| **let** **X** = Expr **in** Expr // *assignment*

```
let a = 42 in
let b = 17+a in
a+b
```

A little language

Type	::=	`int` | `bool` | Type → Type	
Expr	::=	X	*// variables*
	|	N	*// integer literals*
	|	Expr + Expr	*// integer addition*
	|	Expr > Expr	*// integer comparison*
	|	**`if`** Expr **`then`** Expr **`else`** Expr	*// if-expressions*
	|	**`true`**	*// boolean literal*
	|	**`false`**	*// boolean literal*
	|	**`let`** X = Expr **`in`** Expr	*// assignment*
	|	**`fun`** X => Expr	*// anonymous function*

```
(fun a => a + 1)
```

A little language

Type ::= `int` | `bool` | Type → Type

Expr	::=	**X**	*// variables*
	|	**N**	*// integer literals*
	|	Expr + Expr	*// integer addition*
	|	Expr > Expr	*// integer comparison*
	|	**`if`** Expr **`then`** Expr **`else`** Expr	*// if-expressions*
	|	**`true`**	*// boolean literal*
	|	**`false`**	*// boolean literal*
	|	**`let`** **X** = Expr **`in`** Expr	*// assignment*
	|	**`fun`** **X** => Expr	*// anonymous function*
	|	Expr (Expr)	*// function call*

```
(fun a => a + 1)(2)
```

A little language

```
Type ::= int | bool | Type → Type

Expr ::= X                              // variables
       | N                              // integer literals
       | Expr + Expr                    // integer addition
       | Expr > Expr                    // integer comparison
       | if Expr then Expr else Expr    // if-expressions
       | true                           // boolean literal
       | false                          // boolean literal
       | let X = Expr in Expr           // assignment
       | fun X => Expr                  // anonymous function
       | Expr ( Expr )                  // function call
```

```
let i = (fun a => a) in
let d = (fun a => a+a) in
d(i(2))
```

Inference rules

$$\dfrac{\text{If I'm a man, then I'm mortal} \qquad \text{I'm a man}}{\text{I'm mortal}}\ \textit{(Modus Ponens)}$$

Theophrastus
371BC – 287BC

$$\dfrac{\text{All mortals are green} \qquad \text{Socrates is mortal}}{\text{Socrates is green}}$$

John Wickerson
1987–

$$\dfrac{\text{likes}(X, Z) \qquad \text{cancook}(Y, Z)}{\text{wouldgetonwith}(X, Y)}$$

$$\textit{(transitivity)}\ \dfrac{\textit{(distrib)}\ \dfrac{}{a\times(b+c) = a\times b + a\times c} \qquad \textit{(congr)}\ \dfrac{\textit{(commut)}\ \dfrac{}{a\times b = b\times a} \qquad \dfrac{}{a\times c = c\times a}\ \textit{(commut)}}{a\times b + a\times c = b\times a + c\times a}}{a\times(b+c) = b\times a + c\times a}$$

Type inference

$$\frac{}{\mathrm{n}\ \textit{has type}\ \texttt{int}}$$

$$\frac{\mathrm{e1}\ \textit{has type}\ \texttt{int} \qquad \mathrm{e2}\ \textit{has type}\ \texttt{int}}{\mathrm{e1} + \mathrm{e2}\ \textit{has type}\ \texttt{int}}$$

$$\frac{\mathrm{e1}\ \textit{has type}\ \texttt{int} \qquad \mathrm{e2}\ \textit{has type}\ \texttt{int}}{\mathrm{e1} < \mathrm{e2}\ \textit{has type}\ \texttt{bool}}$$

Type inference

$$\frac{}{\Gamma \vdash n : \texttt{int}} \qquad \frac{\Gamma \vdash e1 : \texttt{int} \qquad \Gamma \vdash e2 : \texttt{int}}{\Gamma \vdash e1 + e2 : \texttt{int}} \qquad \frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e1 : \texttt{int} \qquad \Gamma \vdash e2 : \texttt{int}}{\Gamma \vdash e1 < e2 : \texttt{bool}} \qquad \frac{\Gamma \vdash e1 : \texttt{bool} \qquad \Gamma \vdash e2 : \tau \qquad \Gamma \vdash e3 : \tau}{\Gamma \vdash \textbf{\texttt{if}}\ e1\ \textbf{\texttt{then}}\ e2\ \textbf{\texttt{else}}\ e3 : \tau}$$

$$\Gamma = \{\, \texttt{foo} : \texttt{int},\ \texttt{baz} : \texttt{bool} \,\}$$

Type inference

$$\frac{}{\Gamma \vdash \mathrm{n} : \texttt{int}} \qquad \frac{\Gamma \vdash \mathrm{e1} : \texttt{int} \qquad \Gamma \vdash \mathrm{e2} : \texttt{int}}{\Gamma \vdash \mathrm{e1} \mathbin{\texttt{+}} \mathrm{e2} : \texttt{int}} \qquad \frac{(\mathrm{x} : \tau) \in \Gamma}{\Gamma \vdash \mathrm{x} : \tau}$$

$$\frac{\Gamma \vdash \mathrm{e1} : \texttt{int} \qquad \Gamma \vdash \mathrm{e2} : \texttt{int}}{\Gamma \vdash \mathrm{e1} \mathbin{\texttt{<}} \mathrm{e2} : \texttt{bool}} \qquad \frac{\Gamma \vdash \mathrm{e1} : \texttt{bool} \qquad \Gamma \vdash \mathrm{e2} : \tau \qquad \Gamma \vdash \mathrm{e3} : \tau}{\Gamma \vdash \textbf{if}\ \mathrm{e1}\ \textbf{then}\ \mathrm{e2}\ \textbf{else}\ \mathrm{e3} : \tau}$$

$$\frac{}{\Gamma \vdash \textbf{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \textbf{false} : \texttt{bool}}$$

$$\frac{\Gamma \vdash \mathrm{e1} : \tau' \qquad \Gamma[\mathrm{x} : \tau'] \vdash \mathrm{e2} : \tau}{\Gamma \vdash \textbf{let}\ \mathrm{x} = \mathrm{e1}\ \textbf{in}\ \mathrm{e2} : \tau} \qquad \frac{\emptyset \vdash \texttt{5} : \texttt{int} \qquad \{\texttt{a} : \texttt{int}\} \vdash \texttt{a > 3} : \texttt{bool}}{\emptyset \vdash \textbf{let}\ \texttt{a = 5}\ \textbf{in}\ \texttt{a > 3} : \texttt{bool}}$$

Type inference

$$\frac{}{\Gamma \vdash n : \texttt{int}} \qquad \frac{\Gamma \vdash e1 : \texttt{int} \qquad \Gamma \vdash e2 : \texttt{int}}{\Gamma \vdash e1 \; \texttt{+} \; e2 : \texttt{int}} \qquad \frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e1 : \texttt{int} \qquad \Gamma \vdash e2 : \texttt{int}}{\Gamma \vdash e1 \; \texttt{<} \; e2 : \texttt{bool}} \qquad \frac{\Gamma \vdash e1 : \texttt{bool} \qquad \Gamma \vdash e2 : \tau \qquad \Gamma \vdash e3 : \tau}{\Gamma \vdash \textbf{if} \; e1 \; \textbf{then} \; e2 \; \textbf{else} \; e3 : \tau}$$

$$\frac{}{\Gamma \vdash \textbf{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \textbf{false} : \texttt{bool}} \qquad \frac{\Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \textbf{fun} \; x \; \texttt{=>} \; e : \tau \rightarrow \tau'}$$

$$\frac{\Gamma \vdash e1 : \tau' \qquad \Gamma[x : \tau'] \vdash e2 : \tau}{\Gamma \vdash \textbf{let} \; x \; \texttt{=} \; e1 \; \textbf{in} \; e2 : \tau} \qquad \frac{\Gamma \vdash e1 : \tau \rightarrow \tau' \qquad \Gamma \vdash e2 : \tau}{\Gamma \vdash e1 \; \texttt{(} e2 \texttt{)} : \tau'}$$

DEMO

An interesting connection

$$\frac{\Gamma \vdash e1 : \tau \rightarrow \tau' \qquad \Gamma \vdash e2 : \tau}{\Gamma \vdash e1\ (\ e2\ )\ : \tau'}$$

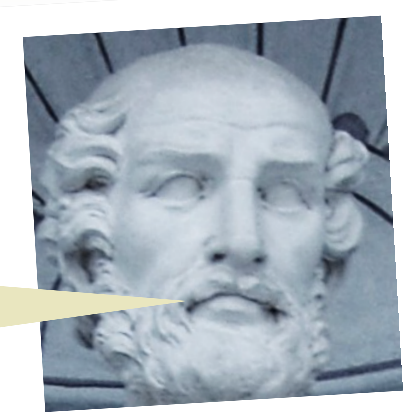
$$\frac{\tau \rightarrow \tau' \qquad \tau}{\tau'}$$

$$\frac{\text{man} \rightarrow \text{mortal} \qquad \text{man}}{\text{mortal}}$$

$$\frac{\text{If I'm a man, then I'm mortal} \qquad \text{I'm a man}}{\text{I'm mortal}}$$ *(Modus Ponens)*

Haskell B. Curry
1900–1982

William Howard
1926–



Theophrastus
371BC – 287BC

Type systems

- ✔ Type checking
- ✔ Type inference
- Polymorphic typing
- Subtyping
- Even fancier type systems

Polymorphic typing

- The above approach to type-inference fails if given:

```
let i = (fun a => a) in
let d = (fun a => a+a) in
i(d)(i(2))
```

  because the type system does not support *polymorphism*.

- Even polymorphic type inference would fail if given:

```
if false then 5 else true
```

Type systems

- ✔ Type checking
- ✔ Type inference
- ✔ Polymorphic typing
- Subtyping
- Even fancier type systems

Subtyping

$$\frac{\Gamma \vdash e : \tau' \qquad \tau' <: \tau}{\Gamma \vdash e : \tau}$$

- `int` <: `float`
- `Labrador` <: `Dog` <: `Animal`
- `Tshirt` <: `Clothing`
- **`struct`** `{int a; int b;}` <: **`struct`** `{int a;}`
- `Dog → Tshirt`

Subtyping

- `int` <: `float`
- `Labrador` <: `Dog` <: `Animal`
- `Tshirt` <: `Clothing`
- **`struct`** `{int a; int b;}` <: **`struct`** `{int a;}`
- `Dog → Tshirt` <: `Labrador → Clothing`

:> <:

$$\frac{\Gamma \vdash e : \tau' \qquad \tau' <: \tau}{\Gamma \vdash e : \tau}$$

"Functions are *contravariant* in the input type and *covariant* in the output type."

- int
- Lab
- Tsh
- **stru**
- Dog

$$\frac{\tau' <: \tau}{\tau}$$

}

avariant in

covariant in

the output type."

Type systems

- ✔ Type checking
- ✔ Type inference
- ✔ Polymorphic typing
- ✔ Subtyping
- Even fancier type systems

Units of measure

- ```
  float<m> distance;
  float<s> time;
  float<m/s> speed;
  ```

- System is implemented in the F# language.

- Would have been handy for the Mars Climate Orbiter in 1999.


  ```

Dependent types

- `int[][] mult (int[][] A, int[][] B);`
- `int[n][p] mult (int[n][m] A, int[m][p] B);`
- `int[len] makeArray(int len);`
- Type-checking now gives much stronger guarantees.
- But type-checking becomes much more complicated.

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases:

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking, type inference

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking, type inference, coercion

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking, type inference, coercion, polymorphism

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking, type inference, coercion, polymorphism, subtype

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking, type inference, coercion, polymorphism, subtype, covariance and contravariance

Summary

- Designing type systems involves a three-way trade-off:
  - Type system **should not restrict** programmers.
  - Type system **should detect many errors**.
  - Type checking/inference **should run quickly**.
- Some key phrases: type checking, type inference, coercion, polymorphism, subtype, covariance and contravariance, dependent type.