```
signed char *(*(*bar)[5])(long int );
int (*(*foo)(void ))[3];
typedef struct u U;
char (*(*x())[5])();
typedef char (*(*x[3])())[5];
x *(*(**foo[][8])())[];
struct u { long (*f)(U);};
struct v { x f; U a;};
double (*f)(int (U *, int ), int );
```

https://bbb.in.tum.de/mic-yho-pdh-xua

```
signed char *(*(*bar)[5])(long int );
int (*(*foo)(void ))[3];
typedef struct u U;
char (*(*x())[5])();
typedef char (*(*x[3])())[5];
x *(*(**foo[][8])())
struct u { long (*f)
struct v { x f; U a;};
double (*f)(int (U *, int ), int );
```

¿¿¿ Bruh, whattt ????

How to read C types

The boring *basic types*:

`struct` *tag*	`int`	`double`
`union` *tag*	`float`	`long`
`enum` *tag*	`char`	`short`
typedef_name	`void`	`...`

The leftout *type modifiers*:

`long`	`unsigned`
`short`	`signed`
`extern`	`static`
`register`	`...`

The inevitable *derived types* (in order of precedence):

- •`[]` array of...
  "Array of" can be undimensioned – [] – or dimensioned – [10] – but the sizes don't really play significantly into reading a declaration.
- •`()` function returning...
  Denoted by a matched pair of parentheses – () – possibly containing an abstract formal parameter list. Parameters lists (if present) are read separately.
- `*`• pointer to...
  This is denoted by the * character, and it always has to point to something.

We'll note that parens are also used for grouping: grouping parens surround the variable name, while "function returning" parens are always on the right.

How to read C types

https://bbb.in.tum.de/mic-yho-pdh-xua

```
char (*(*foo())[5])();
```

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

How to read C types

```
(*(* • ())[5])();
```

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

`foo` is ... `char`

How to read C types

```
(*(* •    )[5])();
```

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

`foo` is a function returning ... `char`

How to read C types

```
(*    •     [5])();
```

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

`foo` is a function returning a pointer to ... `char`

How to read C types

```
(*    •       )();
```

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

`foo` is a function returning a pointer to an array of 5 ... `char`

How to read C types

- `();`

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

`foo` is a function returning a pointer to an array of 5 pointers to ... `char`

How to read C types

https://bbb.in.tum.de/mic-yho-pdh-xua

1. Always start with the variable name:
   `foo` is ...
2. and always end with the *basic type*:
   `foo` is ... `char`
3. The "filling in the middle" part can be summarize with this rule:

"go right when you can, go left when you must"

Work your way out from the variable name and consume *derived-type* tokens *to the right as far as possible* without bumping into a grouping parenthesis. Then *go left to the matching parenthesis*. Rinse and repeat!

`foo` is a function returning a pointer to an array of 5 pointers to functions returning `char`

How to read C types

https://bbb.in.tum.de/mic-yho-pdh-xua

```
typedef double (*FOO)(int (*(*)())(), struct s *);
```

- `typedef` declares the variable name as a *typename*

How to read C types

```
(* • )(int (*(*)())(), struct s *);
```

- `typedef` declares the variable name as a *typename*

Declare typename `FOO` as
... `double`

How to read C types

- `(int (*(*)())(), struct s *);`

- `typedef` declares the variable name as a *typename*

Declare typename `FOO` as
pointer to ... `double`

How to read C types

- `typedef` declares the variable name as a *typename*
- `struct`s act like basic data types

Declare typename `FOO` as
pointer to a function (`int(*(*)())()`), pointer to `struct s`) returning `double`

How to read C types

https://bbb.in.tum.de/mic-yho-pdh-xua

```
int (*(*)())()
```

- `typedef` declares the variable name as a *typename*
- `struct`s act like basic data types
- abstract declarations for formal parameters – where does • go?
  1. to the right of all the "pointer to" derived type tokens
  2. to the left of all "array of" derived type tokens
  3. to the left of all "function returning" derived type tokens
  4. inside all the grouping parentheses

Declare typename `FOO` as
pointer to a function (`int(*(*)())()`), pointer to `struct s`) returning `double`

How to read C types

https://bbb.in.tum.de/mic-yho-pdh-xua

```
int (*(*•)())()
```

- `typedef` declares the variable name as a *typename*
- `struct`s act like basic data types
- abstract declarations for formal parameters
  1. to the right of all the "pointer to" derived type tokens
  2. to the left of all "array of" derived type tokens
  3. to the left of all "function returning" derived type tokens
  4. inside all the grouping parentheses

Declare typename `FOO` as
pointer to a function (`int(*(*)())()`), pointer to `struct s`) returning `double`

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

$$\dfrac{\textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash z :} \quad \textsc{App}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash f :} \quad \textsc{Array}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash a :} \quad \textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash y :} \quad \textsc{Const}\ \dfrac{}{\Gamma \vdash 1 :}}{\Gamma \vdash (y\ +\ 1) :}}{\Gamma \vdash a[(y\ +\ 1)] :}}{\Gamma \vdash f(a[(y\ +\ 1)]) :}}{\Gamma \vdash (z\ =\ f(a[(y\ +\ 1)])) :}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

$$\dfrac{\textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash z :} \quad \textsc{App}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash f :} \quad \textsc{Array}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash a :} \quad \textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash y : \textbf{int}\ \_} \quad \textsc{Const}\ \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}}{\Gamma \vdash (y\ +\ 1) :}}{\Gamma \vdash a[(y\ +\ 1)] :}}{\Gamma \vdash f(a[(y\ +\ 1)]) :}}{\Gamma \vdash (z\ =\ f(a[(y\ +\ 1)])) :}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

$$\textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash z :} \quad \textsc{App}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash f :} \quad \textsc{Array}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash a : \textbf{int}\,[]} \quad \textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash y : \textbf{int}} \quad \textsc{Const}\ \dfrac{}{\Gamma \vdash 1 : \textbf{int}}}{\Gamma \vdash (y\ +\ 1) : \textbf{int}}}{\Gamma \vdash a[(y\ +\ 1)] :}}{\Gamma \vdash f(a[(y\ +\ 1)]) :}}{\Gamma \vdash (z\ =\ f(a[(y\ +\ 1)])) :}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

$$\textsc{Op}\ \frac{\textsc{Var}\ \frac{}{\Gamma \vdash z :} \quad \textsc{App}\ \frac{\textsc{Var}\ \frac{}{\Gamma \vdash f : \textbf{int}\ \_(\textbf{double})} \quad \textsc{Array}\ \frac{\textsc{Var}\ \frac{}{\Gamma \vdash a : \textbf{int}\ []} \quad \textsc{Op}\ \frac{\textsc{Var}\ \frac{}{\Gamma \vdash y : \textbf{int}\ \_} \quad \textsc{Const}\ \frac{}{\Gamma \vdash 1 : \textbf{int}\ \_}}{\Gamma \vdash (y\ +\ 1) : \textbf{int}\ \_}}{\Gamma \vdash a[(y\ +\ 1)] : \textbf{int}\ \_}}{\Gamma \vdash f(a[(y\ +\ 1)]) :}}{\Gamma \vdash (z\ =\ f(a[(y\ +\ 1)])) :}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

$$\dfrac{\textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash z : \textbf{double}\ \_} \quad \textsc{App}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash f : \textbf{int}\ \_(\textbf{double})} \quad \lightning\ \text{parameter type mismatch:} \quad \textsc{Array}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash a : \textbf{int}\ []} \quad \textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash y : \textbf{int}\ \_} \quad \textsc{Const}\ \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}}{\Gamma \vdash (y\ +\ 1) : \textbf{int}\ \_}}{\Gamma \vdash a[(y\ +\ 1)] : \textbf{int}\ \_}}{\Gamma \vdash f(a[(y\ +\ 1)]) : \textbf{int}\ \_}}{\Gamma \vdash (z\ =\ f(a[(y\ +\ 1)])) :}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int } y, \\ \textbf{double } z, \\ \textbf{int } f(\textbf{double}), \\ \textbf{int } a[], \\ \textbf{int } x, \end{array} \right\}$$

```
(z = f(a[(y + 1)])) ;
```

$$\dfrac{\textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash z : \textbf{double}\ \_} \quad \textsc{App}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash f : \textbf{int}\ \_(\textbf{double})} \quad \lightning\ \text{parameter type mismatch:} \quad \textsc{Array}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash a : \textbf{int}\ []} \quad \textsc{Op}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash y : \textbf{int}\ \_} \quad \textsc{Const}\ \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}}{\Gamma \vdash (y\ +\ 1) : \textbf{int}\ \_}}{\Gamma \vdash a[(y\ +\ 1)] : \textbf{int}\ \_}}{\Gamma \vdash f(a[(y\ +\ 1)]) : \textbf{int}\ \_}}{\Gamma \vdash (z\ =\ f(a[(y\ +\ 1)])) : \text{error}\ \lightning\ \_}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s:}\,\textsc{Var}}{\Gamma \vdash s.a2:}\,\textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f:}\,\textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a:}\,\textsc{Var} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y:}\,\textsc{Var}}{\Gamma \vdash *y:}\,\textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1:}\,\textsc{Const}}{\Gamma \vdash (*y\ +\ 1):}\,\textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)]:}\,\textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]):}\,\textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])):}\,\textsc{Op}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s:}\,\textsc{Var}}{\Gamma \vdash s.a2:}\,\textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f:}\,\textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a:}\,\textsc{Var} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \textbf{int}\ \_(\textbf{int})}\,\textsc{Var}}{\Gamma \vdash *y:}\,\textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1:}\,\textsc{Const}}{\Gamma \vdash (*y\ +\ 1):}\,\textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)]:}\,\textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]):}\,\textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])):}\,\textsc{Op}$$

Type Checking

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\ \{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s:}\,\textsc{Var}}{\Gamma \vdash s.a2:}\,\textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f:}\,\textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a:}\,\textsc{Var} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \textbf{int}\ \_(\textbf{int})}\,\textsc{Var}}{\Gamma \vdash *y : \text{error } f\ \_}\,\textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}\,\textsc{Const}}{\Gamma \vdash (*y\ +\ 1):}\,\textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)]:}\,\textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]):}\,\textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])):}\,\textsc{Op}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s :}\,\textsc{Var}}{\Gamma \vdash s.a2 :}\,\textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f :}\,\textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a : \textbf{int}*}\,\textsc{Var} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \textbf{int}}\,\textsc{Var}}{\Gamma \vdash *y :\ \text{error}\not\in}\,\textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1 : \textbf{int}}\,\textsc{Const}}{\Gamma \vdash (*y\ +\ 1) :\ \text{error}\not\in}\,\textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)] :}\,\textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]) :}\,\textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])) :}\,\textsc{Op}$$

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s : \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}}\,\textsc{Var}}{\Gamma \vdash s.a2 :}\,\textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f : \textbf{int}\ \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a : \textbf{int}*}\,\textsc{Var} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \textbf{int}\ \_(\textbf{int})}\,\textsc{Var}}{\Gamma \vdash *y :\ \text{error } \lightning\ \_}\,\textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}\,\textsc{Const}}{\Gamma \vdash (*y\ +\ 1) :\ \text{error } \lightning\ \_}\,\textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)] : \textbf{int}\ \_}\,\textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]) :}\,\textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])) :}\,\textsc{Op}$$

Non-simple index type:

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\ \{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s : \texttt{struct}\ \{\textbf{double}\ a1; \textbf{int}*\ a2;\}}\ \textsc{Var}}{\Gamma \vdash s.a2 : \textbf{int}*}\ \textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f : \textbf{int}\ \_(\textbf{int})}\ \textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a : \textbf{int}*}\ \textsc{Var} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \textbf{int}\ \_(\textbf{int})}\ \textsc{Var}}{\Gamma \vdash *y : \ \text{error } \lightning\ \_}\ \textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}\ \textsc{Const}}{\Gamma \vdash (*y\ +\ 1) : \ \text{error } \lightning\ \_}\ \textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)] : \textbf{int}\ \_}\ \textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]) : \textbf{int}\ \_}\ \textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])) :}\ \textsc{Op}$$

Non-simple index type:

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \textbf{int}\ y(\textbf{int}), \\ \textbf{double}\ z, \\ \textbf{int}\ f(\textbf{int}), \\ \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}\ s, \\ \textbf{int}*\ a, \\ \textbf{int}(*\ arr[])(), \\ \textbf{int}\ x, \end{array} \right\}$$

```
(s.a2 = f(a[(*y + 1)])) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash s : \texttt{struct}\,\{\textbf{double}\ a1; \textbf{int}*\ a2;\}}\,\textsc{Var}}{\Gamma \vdash s.a2 : \textbf{int}*}\,\textsc{Struct} \quad \dfrac{\dfrac{}{\Gamma \vdash f : \textbf{int}\ \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{\dfrac{}{\Gamma \vdash a : \textbf{int}*}\,\textsc{Var} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \textbf{int}\ \_(\textbf{int})}\,\textsc{Var}}{\Gamma \vdash *y : \text{ error } \lightning\ \_}\,\textsc{Deref} \quad \dfrac{}{\Gamma \vdash 1 : \textbf{int}\ \_}\,\textsc{Const}}{\Gamma \vdash (*y\ +\ 1) : \text{ error } \lightning\ \_}\,\textsc{Op}}{\Gamma \vdash a[(*y\ +\ 1)] : \textbf{int}\ \_}\,\textsc{Array}}{\Gamma \vdash f(a[(*y\ +\ 1)]) : \textbf{int}\ \_}\,\textsc{App}}{\Gamma \vdash (s.a2\ =\ f(a[(*y\ +\ 1)])) : \text{ error } \lightning\ \_}\,\textsc{Op}$$

Non-simple index type:

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\}$$

```
((&(*g(i))).f)(i) ;
```

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\} \qquad \texttt{((\&(*g(i))).f)(i) ;}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{VAR}\ \overline{\Gamma \vdash g :}}{\Gamma \vdash g(i) :}\ \text{APP}}{\Gamma \vdash (*g(i)) :}\ \text{DEREF}}{\Gamma \vdash (\&(*g(i))) :}\ \text{REF}}{\Gamma \vdash ((\&(*g(i))).f) :}\ \text{STRUCT} \qquad \text{VAR}\ \overline{\Gamma \vdash i :}}{\Gamma \vdash ((\&(*g(i))).f)(i) :}\ \text{APP}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\}$$

```
((&(*g(i))).f)(i) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \texttt{struct}\ t *\ \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash g(i) :}\,\textsc{App}}{\Gamma \vdash (*g(i)) :}\,\textsc{Deref}}{\Gamma \vdash (\&(*g(i))) :}\,\textsc{Ref}}{\Gamma \vdash ((\&(*g(i))).f) :}\,\textsc{Struct} \quad \dfrac{}{\Gamma \vdash i :}\,\textsc{Var}}{\Gamma \vdash ((\&(*g(i))).f)(i) :}\,\textsc{App}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\}$$

```
((&(*g(i))).f)(i) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \texttt{struct}\ t *\ \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash g(i) : \texttt{struct}\ t *}\,\textsc{App}}{\Gamma \vdash (*g(i)) :}\,\textsc{Deref}}{\Gamma \vdash (\&(*g(i))) :}\,\textsc{Ref}}{\Gamma \vdash ((\&(*g(i))).f) :}\,\textsc{Struct} \quad \dfrac{}{\Gamma \vdash i :}\,\textsc{Var}}{\Gamma \vdash ((\&(*g(i))).f)(i) :}\,\textsc{App}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\} \qquad \texttt{((\&(*g(i))).f)(i) ;}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \texttt{struct}\ t * \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash g(i) : \texttt{struct}\ t *}\,\textsc{App}}{\Gamma \vdash (*g(i)) : \texttt{struct}\ t}\,\textsc{Deref}}{\Gamma \vdash (\&(*g(i))) :}\,\textsc{Ref}}{\Gamma \vdash ((\&(*g(i))).f) :}\,\textsc{Struct} \quad \dfrac{}{\Gamma \vdash i :}\,\textsc{Var}}{\Gamma \vdash ((\&(*g(i))).f)(i) :}\,\textsc{App}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Type Checking

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct } t\{\textbf{int } d; \textbf{int}(*f)(\textbf{int}); \} s, \\ \textbf{double } d, \\ \texttt{struct } t * g(\textbf{int}), \\ \textbf{int } i, \end{array} \right\} \qquad \texttt{((\&(*g(i))).f)(i) ;}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \texttt{struct } t * \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash g(i) : \texttt{struct } t *}\,\textsc{App}}{\Gamma \vdash (*g(i)) : \texttt{struct } t}\,\textsc{Deref}}{\Gamma \vdash (\&(*g(i))) : \texttt{struct } t *}\,\textsc{Ref}}{\Gamma \vdash ((\&(*g(i))).f) :}\,\textsc{Struct} \quad \dfrac{}{\Gamma \vdash i :}\,\textsc{Var}}{\Gamma \vdash ((\&(*g(i))).f)(i) :}\,\textsc{App}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Type Checking

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\}$$

```
((&(*g(i))).f)(i) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \texttt{struct}\ t *\ \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash g(i) : \texttt{struct}\ t *}\,\textsc{App}}{\Gamma \vdash (*g(i)) : \texttt{struct}\ t}\,\textsc{Deref}}{\Gamma \vdash (\&(*g(i))) : \texttt{struct}\ t *}\,\textsc{Ref}}{\Gamma \vdash ((\&(*g(i))).f) :\ \text{error}\ \lightning\ \_}\,\textsc{Struct} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash ((\&(*g(i))).f)(i) :}\,\textsc{App}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Type Checking

$$\Gamma = \left\{ \begin{array}{l} \texttt{struct}\ t\{\textbf{int}\ d; \textbf{int}(*\ f)(\textbf{int}); \}\ s, \\ \textbf{double}\ d, \\ \texttt{struct}\ t *\ g(\textbf{int}), \\ \textbf{int}\ i, \end{array} \right\}$$

```
((&(*g(i))).f)(i) ;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash g : \texttt{struct}\ t *\ \_(\textbf{int})}\,\textsc{Var} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash g(i) : \texttt{struct}\ t *}\,\textsc{App}}{\Gamma \vdash (*g(i)) : \texttt{struct}\ t}\,\textsc{Deref}}{\Gamma \vdash (\&(*g(i))) : \texttt{struct}\ t *}\,\textsc{Ref}}{\Gamma \vdash ((\&(*g(i))).f) :\ \text{error}\ \lightning\ \_}\,\textsc{Struct} \quad \lightning\ \text{no function} \quad \dfrac{}{\Gamma \vdash i : \textbf{int}\ \_}\,\textsc{Var}}{\Gamma \vdash ((\&(*g(i))).f)(i) :\ \text{error}\ \lightning\ \_}\,\textsc{App}$$

Complete the type check!

In case of a type error, mark the erroneous subexpression!

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

Prove or disprove:

$$B \leq A$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

Prove or disprove:

$$B \leq A$$

$B \leq A$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

Prove or disprove:

$$B \leq A$$

$$\frac{}{\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \} \leq A}$$

$$B \leq A$$

Structural Subtyping

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b;\};$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b;\};$$

Prove or disprove:

$$B \leq A$$

$$B \leq A$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b;\} \leq A$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b;\} \leq \texttt{struct } s1\{\mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b;\}$$

Structural Subtyping

$$\texttt{typedef struct } s1\ A;$$

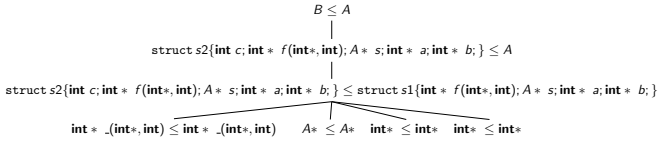
$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

$$\texttt{struct } s2\{\mathbf{int}\ c; \mathbf{int} * \ f(\mathbf{int}*, \mathbf{int}); A * \ s; \mathbf{int} * \ a; \mathbf{int} * \ b; \};$$

Prove or disprove:

$$B \leq A$$



Structural Subtyping

$$\texttt{typedef struct } s1\ A;$$

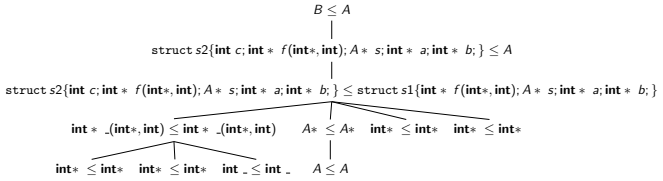
$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\mathbf{int}*\ f(\mathbf{int}*,\mathbf{int});A*\ s;\mathbf{int}*\ a;\mathbf{int}*\ b;\};$$

$$\texttt{struct } s2\{\mathbf{int}\ c;\mathbf{int}*\ f(\mathbf{int}*,\mathbf{int});A*\ s;\mathbf{int}*\ a;\mathbf{int}*\ b;\};$$

Prove or disprove:

$$B \leq A$$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

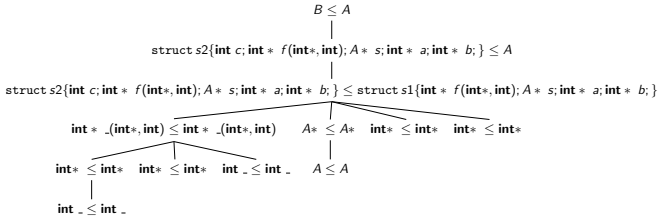
$$\texttt{typedef struct } s2\ B;$$

$$\texttt{struct } s1\{\textbf{int} * \ f(\textbf{int}*, \textbf{int}); A * \ s; \textbf{int} * \ a; \textbf{int} * \ b; \};$$

$$\texttt{struct } s2\{\textbf{int}\ c; \textbf{int} * \ f(\textbf{int}*, \textbf{int}); A * \ s; \textbf{int} * \ a; \textbf{int} * \ b; \};$$

Prove or disprove:

$$B \leq A$$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$

$C \leq B$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$

$$\frac{C \leq B}{\texttt{struct } s3\{C\ f(C); C*\ g;\} \leq B}$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$

$$\dfrac{\dfrac{C \leq B}{\texttt{struct } s3\{C\ f(C); C*\ g;\} \leq B}}{\texttt{struct } s3\{C\ f(C); C*\ g;\} \leq \texttt{struct } s2\{B\ f(A);\}}$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$

$$\frac{\dfrac{\dfrac{}{C \leq B}}{\texttt{struct } s3\{C\ f(C); C*\ g;\} \leq B}}{\texttt{struct } s3\{C\ f(C); C*\ g;\} \leq \texttt{struct } s2\{B\ f(A);\}}$$

$$C \_(C) \leq B \_(A)$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

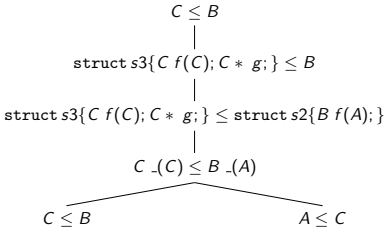
$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$



Prove or disprove:

$$C \leq B$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

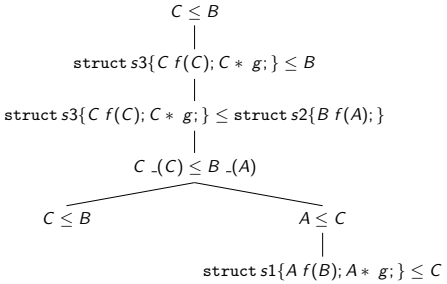
$$\texttt{struct } s3\{C\ f(C); C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B); A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

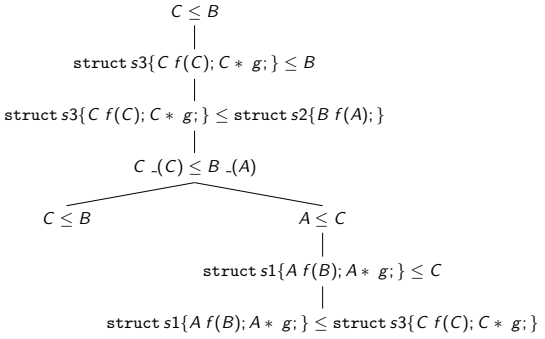
$$\texttt{struct } s3\{C\ f(C);\ C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B);\ A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

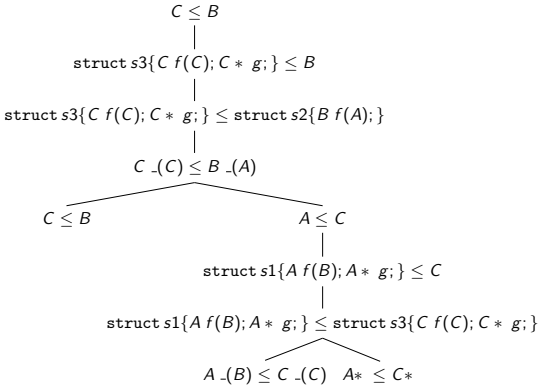
$$\texttt{struct } s3\{C\ f(C);\ C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B);\ A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s1\ A;$$

$$\texttt{typedef struct } s2\ B;$$

$$\texttt{typedef struct } s3\ C;$$

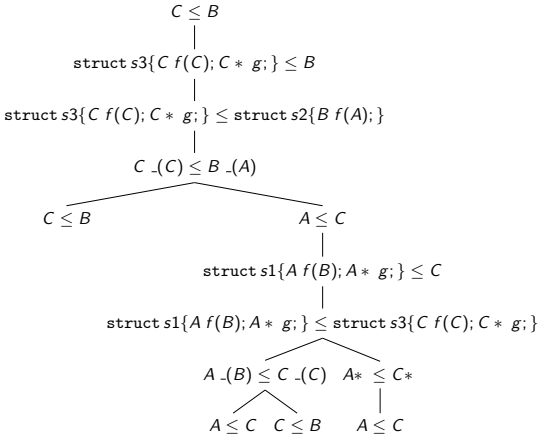
$$\texttt{struct } s3\{C\ f(C);\ C*\ g;\};$$

$$\texttt{struct } s1\{A\ f(B);\ A*\ g;\};$$

$$\texttt{struct } s2\{B\ f(A);\};$$

Prove or disprove:

$$C \leq B$$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

```
typedef struct s S;
typedef struct t T;
typedef T(* F)(S);
struct s{F f; T t;};
struct t{S(* f)(T); S t;};
```

Prove or disprove: $S \leq T$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$S \leq T$

```
typedef struct s S;
typedef struct t T;
typedef T(* F)(S);
struct s{F f; T t;};
struct t{S(* f)(T); S t;};
```

Prove or disprove: $S \leq T$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$

$$\frac{S \leq T}{\texttt{struct } s\{F\ f;\ T\ t;\} \leq T}$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$

$$\frac{\dfrac{S \leq T}{\texttt{struct } s\{F\ f;\ T\ t;\} \leq T}}{\texttt{struct } s\{F\ f;\ T\ t;\} \leq \texttt{struct } t\{S(*f)(T);\ S\ t;\}}$$

Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

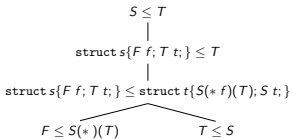
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

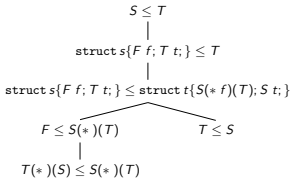
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

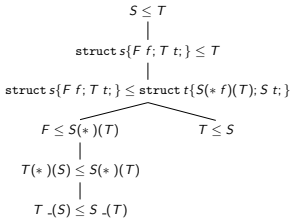
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

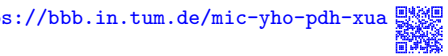
$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua



$$\texttt{typedef struct } s\ S;$$

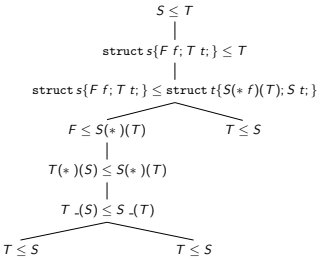
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

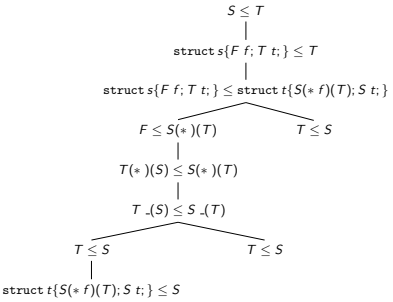
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

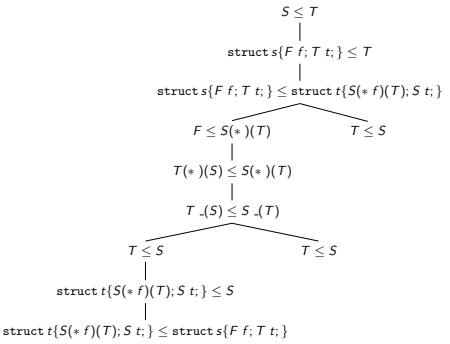
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$\texttt{typedef struct } s\ S;$

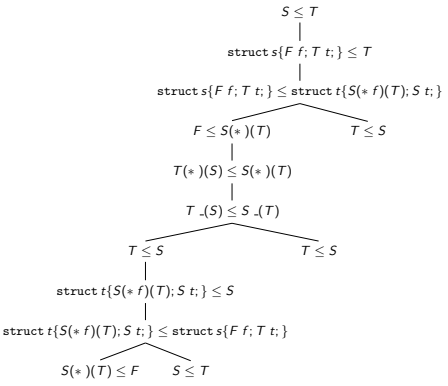
$\texttt{typedef struct } t\ T;$

$\texttt{typedef } T(*F)(S);$

$\texttt{struct } s\{F\ f;\ T\ t;\};$

$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$\texttt{typedef struct } s\ S;$

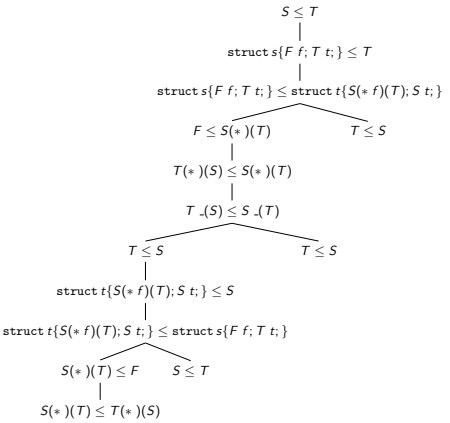
$\texttt{typedef struct } t\ T;$

$\texttt{typedef } T(*F)(S);$

$\texttt{struct } s\{F\ f;\ T\ t;\};$

$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

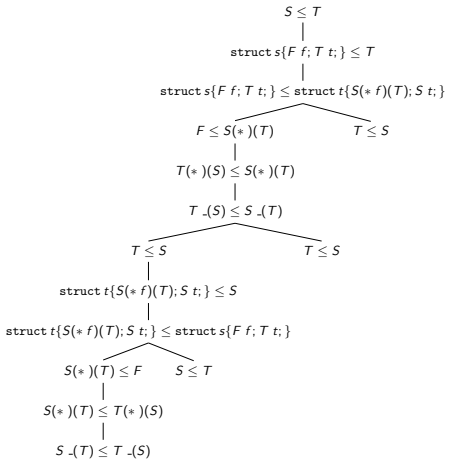
$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$



Structural Subtyping

https://bbb.in.tum.de/mic-yho-pdh-xua

$$\texttt{typedef struct } s\ S;$$

$$\texttt{typedef struct } t\ T;$$

$$\texttt{typedef } T(*F)(S);$$

$$\texttt{struct } s\{F\ f;\ T\ t;\};$$

$$\texttt{struct } t\{S(*f)(T);\ S\ t;\};$$

Prove or disprove: $S \leq T$