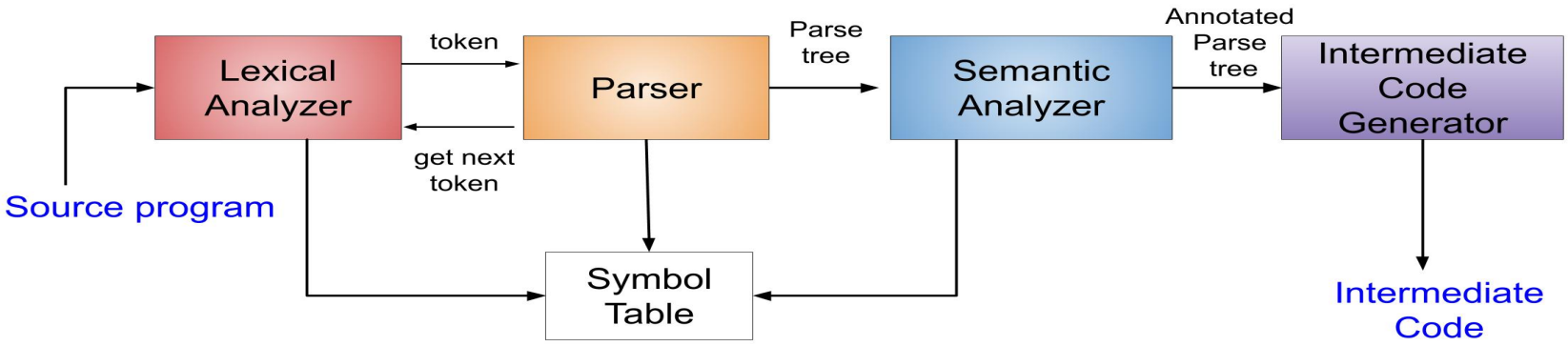
CSEN 1003: Compilers

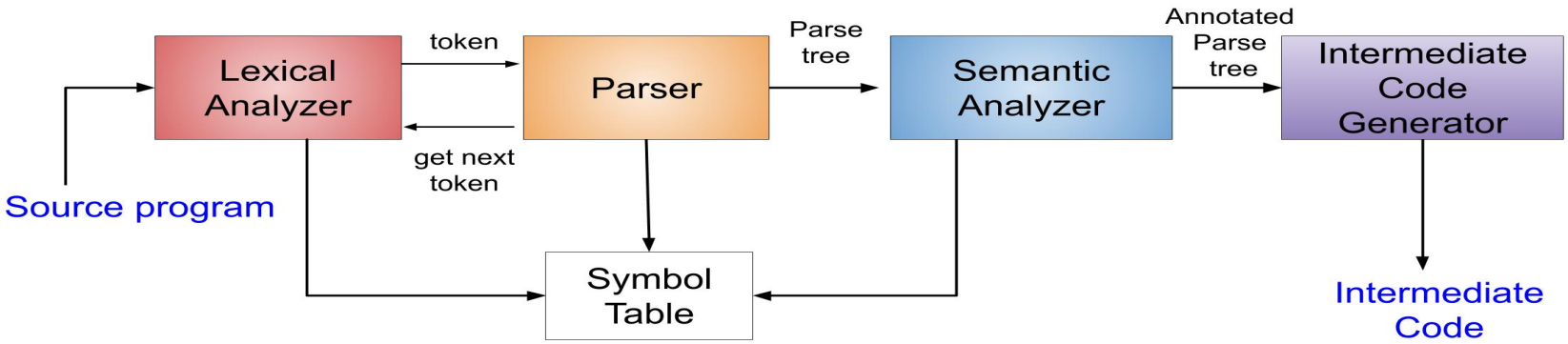
Tutorial 10 - Semantic Analysis II

The Semantic Analyser



- The semantic analyzer deals with types management and any remaining checks after parsing (such as types checking and scope resolution).

The Semantic Analyser



- The semantic analyzer deals with types management and any remaining checks after parsing (such as types checking and scope resolution).
- The application of types can be grouped under two main processes:
  1. Type checking: ensures operand types matches the type expected by an operator.
  2. Translation Applications: from types a compiler can determine the storage needed at compile time.

Today's Plan

1. Type Checking
2. Translation Applications
3. Recap

Type Checking

- Type checking is assigning a type expression to each program component and checking that these types observes the type system of the language.

Type Checking

- Type checking is assigning a type expression to each program component and checking that these types observes the type system of the language.
- A strongly typed language does type checking at compile time.

Type Checking

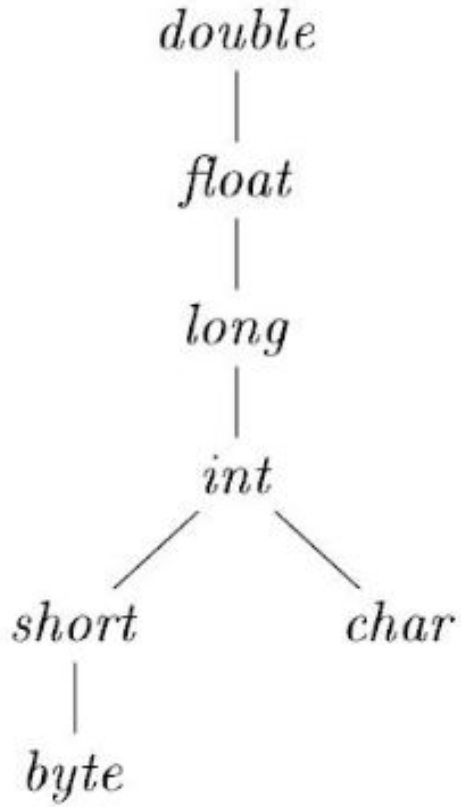
- Type checking is assigning a type expression to each program component and checking that these types observes the type system of the language.
- A strongly typed language does type checking at compile time.
- Type checking can be done by:
  1. Type Synthesis: if f has the type $s \to t$ and x has type s, then $f(x)$ has the type t. Requires the program to specify types of the basic components of the program.
  2. Type Inference: If $f(x)$ is an expression, there must be types α and β such that $f(x)$ has type $\alpha \to \beta$ and x has the type α.

Type Synthesis

- The compiler does implicit type conversions called coercions.
- Coercions are limited to widening operations.

Example

$$2 * 3.14$$
$$t1 = (\mathit{float})\ 2,\ t2 = t1 * 3.14$$

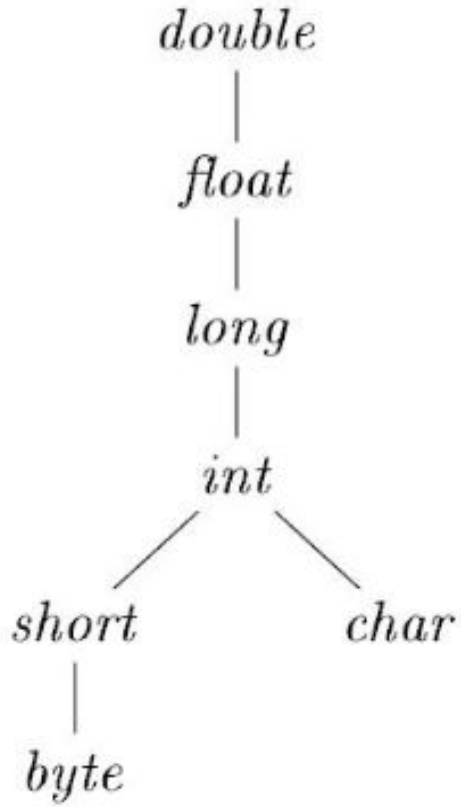


Type Synthesis

- The compiler does implicit type conversions called coercions.
- Coercions are limited to widening operations.

Example

$$2 * 3.14$$
$$t1 = (float)\ 2,\ t2 = t1 * 3.14$$



- If $E \rightarrow E_1 + E_2$, to synthesize a value for E two functions are used:
  1. `max(t1,t2):` max type according to the hierarchy.
  2. `widen(a,t,w):` widen address a from type t to type w.

Type Synthesis

$$
\begin{aligned}
E \rightarrow E_1 + E_2 \quad \{\ & E.type = max(E_1.type, E_2.type); \\
& a_1 = widen(E_1.addr, E_1.type, E.type); \\
& a_2 = widen(E_2.addr, E_2.type, E.type); \\
& E.addr = \textbf{new}\ Temp\,(); \\
& gen(E.addr\ '='\ a_1\ '+'\ a_2);\ \}
\end{aligned}
$$

$$
\begin{aligned}
&Addr\ widen(Addr\ a,\ Type\ t,\ Type\ w) \\
&\quad \textbf{if}\ (\ t = w\ )\ \ \textbf{return}\ a; \\
&\quad \textbf{else if}\ (\ t = integer\ \textbf{and}\ w = float\ )\ \{ \\
&\qquad temp = \textbf{new}\ Temp(); \\
&\qquad gen(temp\ '='\ '(\textbf{float})'\ a); \\
&\qquad \textbf{return}\ temp; \\
&\quad \} \\
&\quad \textbf{else}\ \ \textbf{error}; \\
&\}
\end{aligned}
$$

SDD for Coercion

$$E \rightarrow E_1 = E_2 \quad \{a = widen(E_2.addr, E_2.type, E_1.type \\ gen(E_1.addr' =' a)\}$$

Example

Assuming that function *widen* can handle any of the types in the widening hierarchy, translate the expressions below. Assume that c and d are characters, s and t are short integers, i and j are integers, and x is a float.

1. x=s+c

SDD for Coercion

$$E \rightarrow E_1 = E_2 \quad \{a = widen(E_2.addr, E_2.type, E_1.type \\ gen(E_1.addr' =' a)\}$$

Example

Assuming that function *widen* can handle any of the types in the widening hierarchy, translate the expressions below. Assume that c and d are characters, s and t are short integers, i and j are integers, and x is a float.

1. x=s+c
   t1 = (int) s
   t2 = (int) c
   t3 = t1 + t2
   t4 = (float) t3
   x = t4

SDD for Coercion

Example

2. x=(s+c) + (t+d)

SDD for Coercion

Example

2. x=(s+c) + (t+d)

   t1 = (int) s
   t2 = (int) c
   t3 = t1 + t2
   t4 = (int) t
   t5 = (int) d
   t6 = t4 + t5
   t7 = t3 + t6
   t8 = (float) t7
   x = t8

Type Inference

- Type inference is useful for statically typed languages that does not require the programmer to declare types.

Type Inference

- Type inference is useful for statically typed languages that does not require the programmer to declare types.
- It ensures that types are used consistently.

Exercise 10-3

Example

Consider the following polymorphic function:

```
fun reverse(x) = if length(x)==1 then x else
append(head(x), reverse(tail(x))
```

What is the type of reverse?

Exercise 10-3

Example

Expression	Type	Unification
`reverse`	$\alpha_1 \rightarrow \beta_1$	
`x`	α_1	
`if`	$bool \times \alpha_2 \times \alpha_2 \rightarrow \alpha_2$	
`length`	$list(\alpha_3) \rightarrow int$	
`length(x)`	int	$\alpha_1 = list(\alpha_3)$
`==`	$\alpha_4 \times \alpha_4 \rightarrow bool$	
`length(x) == 1`	$bool$	$\alpha_4 = int$
`x`	$list(\alpha_3)$	$\alpha_2 = list(\alpha_3)$
`append`	$\alpha_5 \times list(\alpha_5) \rightarrow list(\alpha_5)$	
`append(head(x), reverse (tail(x)))`	$\alpha_2 = list(\alpha_3)$	$\alpha_5 = \alpha_3$
`head`	$list(\alpha_6) \rightarrow \alpha_6$	
`head(x)`	$\alpha_5 = \alpha_3$	$\alpha_6 = \alpha_5 = \alpha_3$
`reverse(tail(x))`	β_1	$\beta_1 = list(\alpha_5) = list(\alpha_3)$
`tail`	$list(\alpha_7) \rightarrow list(\alpha_7)$	
`tail(x)`	$list(\alpha_3)$	$\alpha_7 = \alpha_3$

Today's Plan

1 Type Checking

2 Translation Applications

3 Recap

SDT for Sequence of Variable Declarations

$$
\begin{aligned}
&P \rightarrow \{offset = 0\} \quad D \\
&D \rightarrow T\ id; \quad \{table.put(id.lexeme, T.type, offset) \\
&\qquad offset+ = T.width\} \\
&\qquad D_1 \\
&D \rightarrow \varepsilon \\
&T \rightarrow int \quad \{T.type = int \quad T.width = 4\} \\
&T \rightarrow float\ \{T.type = float \quad T.width = 8\} \\
&T \rightarrow record\{ \quad \{Stack.push(table) \\
&\qquad\qquad\qquad table = newTable() \\
&\qquad\qquad\qquad Stack.push(offset) \\
&\qquad\qquad\qquad offset = 0\} \\
&D \quad \{T.type = record(table) \\
&\qquad T.width = offset \\
&\qquad offset = Stack.pop() \\
&\qquad table = Stack.pop()\}\} \quad \}
\end{aligned}
$$

Exercise 10-1

Example

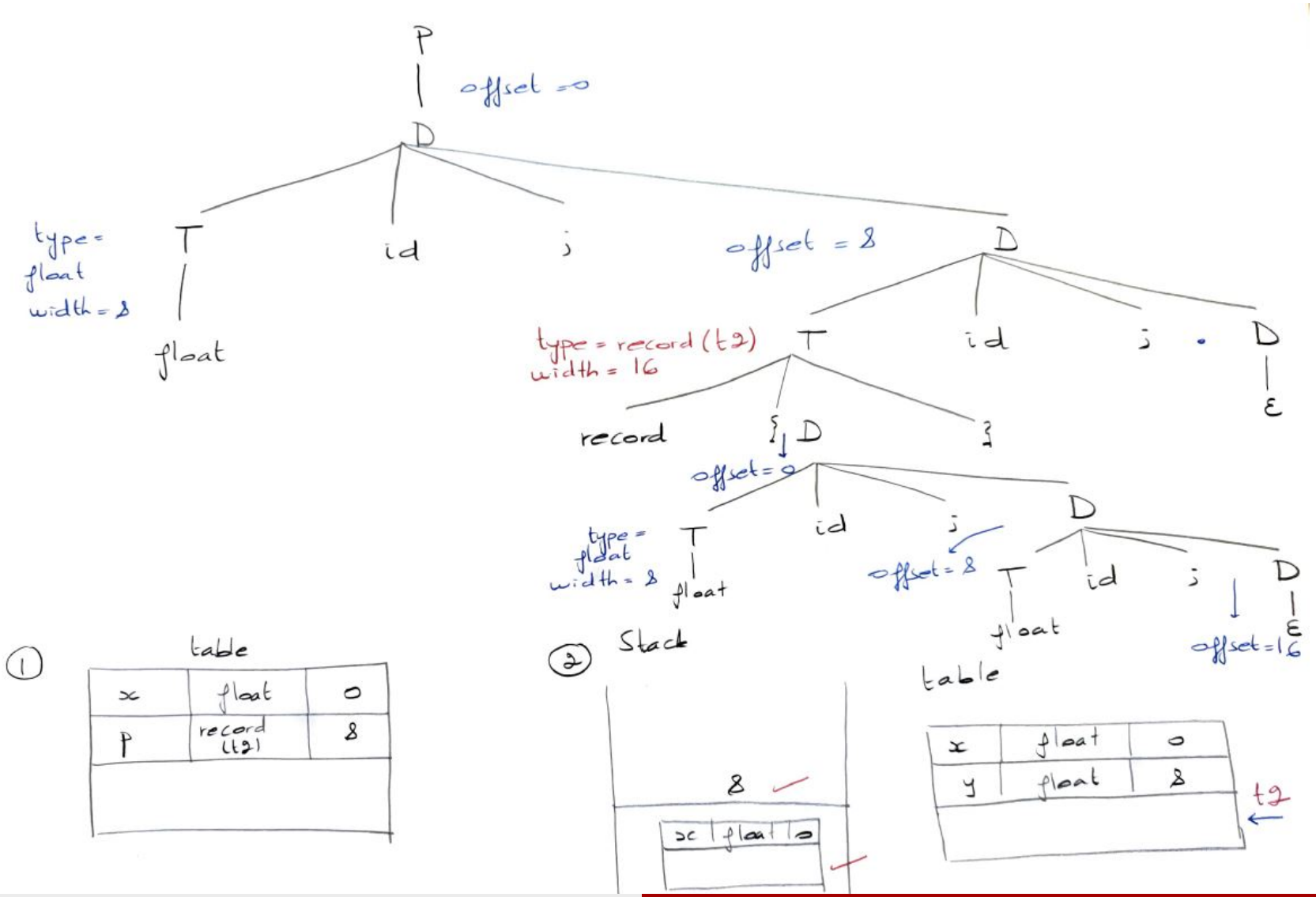
Determine the types and relative addresses for the identifiers in the following sequence of declarations:

float x;

record {float x; float y;} p;

record {int tag; float x; float y;} q;

Exercise 10-1



Exercise 10-1

Example

ID	Address	Type
x	0	*float*
p.x	8	*float*
p.y	16	*float*
p	8	$record([\langle x, 0, float\rangle; \langle y, 8, float\rangle])$
q.tag	24	*integer*
q.x	28	*float*
q.y	36	*float*
q	24	$record([\langle tag, 0, integer\rangle; \langle x, 4, float\rangle; \langle y, 12, float\rangle])$

Today's Plan

1. Type Checking
2. Translation Applications
3. Recap

Covered Topics

1. Type Checking.
2. Memory Management at Compile Time.

Next Week: Intermediate Code Generation!