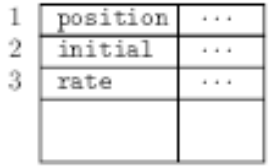
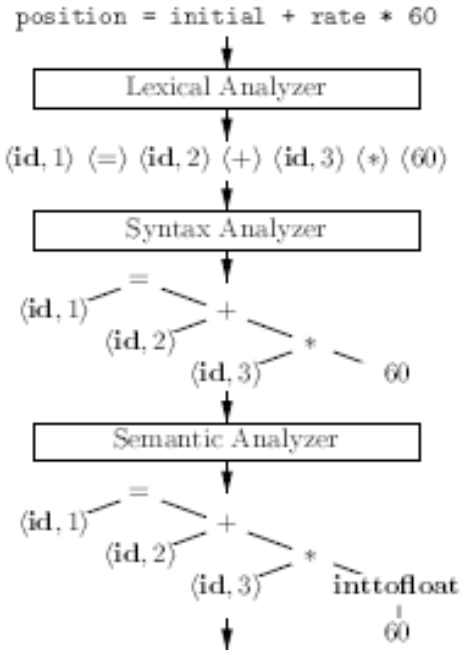
Semantic Analysis

- 3rd phase in compilation
- Parse tree & symbol table
  - Consistency of code
- ***Type Checking***
- Semantic consistency
- Semantic Errors
  - Type mismatch
  - Undeclared variables
  - Reserved keywords
  - Accessing out of scope variable
  - Actual and formal parameter mismatch



SYMBOL TABLE



Type Systems

- Type system
  - Rules that assign types to program constructs (constraints added to check the validity of program)
  - Language specific
- Type Checking (verifying the type)
- Static check (Compilation)
  - Type check, Type conversion, Indexing checks, Function call check, Flow of Control check
- Dynamic Check (Execution)
- Error Recovery
  - Location

Type System

- Type Expression
- Language Construct – part of program formed with lexical tokens. EX: if, else,…python →print (language construct)
- ***Type Expression*** – Associate each language construct with an expression describing its type
- Type expression → Basic Type & Type name
- ***Basic Type*** → int, char, boolean, void, float,…
- ***Type Name /(Type Constructor)***
  - Array, Function, Pointer
  - Ex: Array → T is a Type Expression
  - →Array(I,T)is a Type Expression
  - →Array([1…10],int)

Type System

- Type Equivalence
  - Name Equivalence
    - Ex: char a='A'; char b=a;
  - Structure Equivalence
- Implementation of Type System is called Type Checker

Specification of a Simple Type Checker

- Example:

$P \rightarrow D ; E$

$D \rightarrow D ; D \mid id : T$

$T \rightarrow char \mid integer \mid array [num] of T \mid \uparrow T$

$E \rightarrow literal \mid num \mid id \mid E \bmod E \mid E [E] \mid E \uparrow$

- Translation Scheme

$P \rightarrow D ; E$

$D \rightarrow D ; D$

$D \rightarrow id : T \{ addtype (id.entry , T.type) \}$

$T \rightarrow char \{ T.type := char \}$

$T \rightarrow integer \{ T.type := integer \}$

$T \rightarrow \uparrow T1 \{ T.type := pointer(T1.type) \}$

$T \rightarrow array [num] of T1 \{ T.type := array (1\ldots num.val , T1.type) \}$

Specification of a Simple Type Checker

- ***Type Checking of Expression***
  - E → literal { E.type : = char } E→num { E.type : = integer }
  - E → id { E.type : = lookup (id.entry) }
  - E → E1 mod E2 { E.type : = if E1. type = integer and E2. type = integer then integer else type_error }

- ***Type checking of statements***

- ***Type checking of Functions***

1. Assignment statement: S→id: = E

S→id: = E { S.type : = if id.type = E.type then void else type_error }

2. Conditional statement: S→if E then S1

S→if E then S1 { S.type : = if E.type = boolean then S1.type else type_error }

3. While statement:

S → while E do S1

S → while E do S1 { S.type : = if E.type = boolean then S1.type else type_error }