Type handling

- Data types support abstraction for the developer (integer vs real)
- Type-checking can support both compile-time and run-time detection of errors, as well as implicit type conversion
- Type might also have a close relationship to implementation (e.g. the range values for a signed integer or a character set)
- Must consider named typing vs structural typing
  - e.g. if Temperature is a named type implemented as a real number, and Speed is a named type implemented as a real number, should they be type-compatible?

CFG forces order of ops

- CFG determines precedence and associativity
- evaluating parse tree “bottom up”: can see types of tokens, then successively check types as we compose expressions
  - e.g. X = 10 + B * 0.5 may become X = (10 + (B * 0.5))
  - types of X,B can be looked up in symbol table, types of 10, 0.5 can be determined from token
  - check if B, 0.5 ok for *, determine type of resulting expression
  - check if that type and 10 are ok for +, determine type of result
  - check if that type and X are ok for =

Expression types & optimization

- Detailed knowledge of expression types can aid in optimizing generated code
  - e.g. X = 3 * 5 + 17 ;
  - After 3*5 realizes can use 15 instead of expression, then after 15+17 realizes can use 32 instead of expression

Compile-time vs run time checking

- Some types can be checked at compile time (i.e. where types have been previously declared)
- Where types cannot be checked at compile time, compiler must either insert suitable run-time checks into the code or accept possibility of run-time crashes

Name vs structural equivalence

- Named equivalence means two types are equivalent iff they have the same type name: provides greater compile-time enforcement of abstract types, easier type checking
- Structural equivalence means the two types are equivalent if they have the same structure (i.e. the same composition of items that are also structurally equivalent), provides greater flexibility w.r.t. implicit type conversions

Inference rules

- For each operator, need a set of rules that specifies the data type of the result based on the data type(s) of the operand(s)
- Rules also need to cover how/when implicit type conversion takes place
  - e.g. int x = 5 * 3.5;
  - * might dictate convert 5 to 5.0, resulting type is real
  - = then dictates conversion from real to integer for storage in x