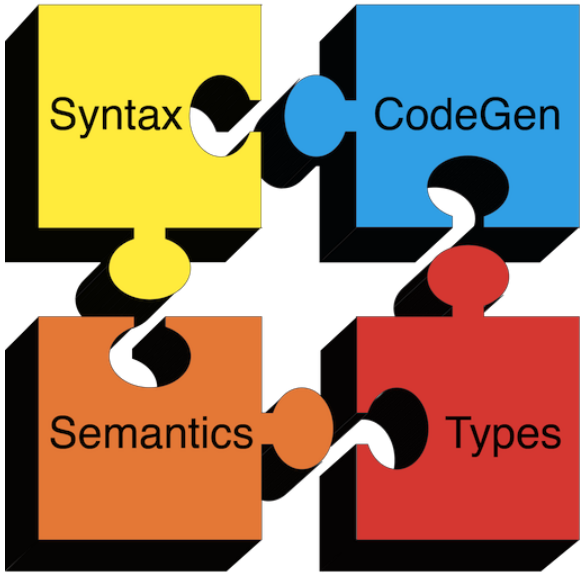

Compilers

Error Handling

Error Handling

- Purpose of the compiler is
  - To detect non-valid programs
  - To translate the valid ones

- Many kinds of possible errors (e.g. in C)

Error kind	Example	Detected by …
Lexical	… $ …	Lexer
Syntax	… x *% …	Parser
Semantic	… int x; y = x(3); …	Type checker
Correctness	your favorite program	Tester/User

Error Handling

- Error handler should
  - Report errors accurately and clearly
  - Recover from an error quickly
  - Not slow down compilation of valid code

Error Handling

- Panic mode
- Error productions
- Automatic local or global correction

Error Handling

- Panic mode is simplest, most popular method
- When an error is detected:
  - Discard tokens until one with a clear role is found
  - Continue from there
- Looking for synchronizing tokens
  - Typically the statement or expression terminators

Error Handling

- Consider the erroneous expression

  (1 + + 2) + 3

- Panic-mode recovery:
  - Skip ahead to next integer and then continue

- Bison: use the special terminal error to describe how much input to skip

  E → int | E + E | (E) | error int | (error)

Error Handling

- Error productions
  - specify known common mistakes in the grammar
- Example:
  - Write 5 x instead of 5 * x
  - Add the production E $\rightarrow$ … | E E
- Disadvantage
  - Complicates the grammar

Error Handling

- Idea: find a correct “nearby” program
  - Try token insertions and deletions
  - Exhaustive search

- Disadvantages:
  - Hard to implement
  - Slows down parsing of correct programs
  - “Nearby” is not necessarily “the intended” program

Error Handling

- Past
  - Slow recompilation cycle (even once a day)
  - Find as many errors in one cycle as possible

- Present
  - Quick recompilation cycle
  - Users tend to correct one error/cycle
  - Complex error recovery is less compelling