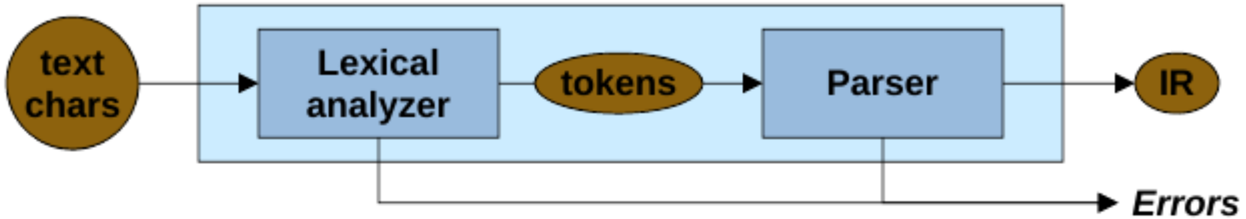
Lexical and Syntactic Analyser Specification

Using JFlex and Java CUP

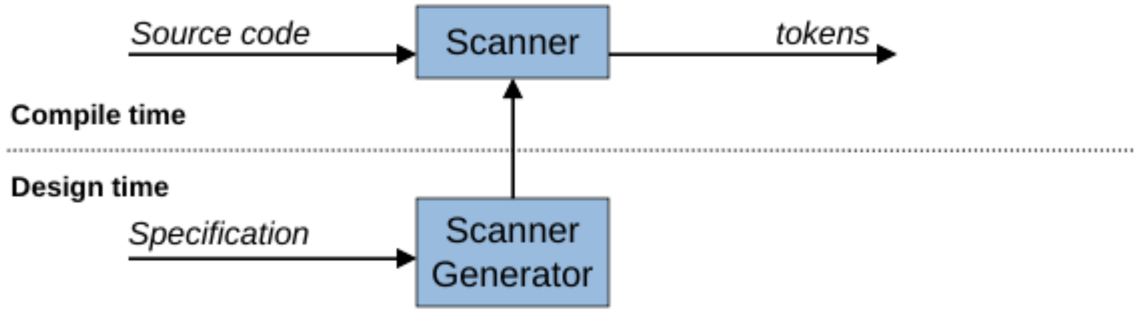
Compiler Front End



- Writing a Scanner (a.k.a. Lexer) by hand is a complicated task.
- Solution: Use scanner generators (lex, Flex, JFlex, ANTLR).

Scanner construction using scanner generators

- Goal: automate process
  - Avoid writing scanners by hand
  - Leverage the underlying theory of languages



Scanner generation using JFlex

- JFlex is designed to work together with the LALR parser generator CUP (more on this later).

- Lexical specification file structure consists of three parts:

```
UserCode
%%
Options and declarations
%%
Lexical rules
```

- C style comments.

Specification file structure

User code

The first part contains user code that is copied verbatim into the beginning of the source file of the generated lexer before the scanner class is declared. This is the place to put package declarations and import statements.

Options and declarations

The second part of the lexical specification contains options to customise your generated lexer, declarations of lexical states and macro definitions for use in the third section ``Lexical rules'' of the lexical specification file.

Lexical rules

The ``lexical rules'' section of a JFlex specification contains a set of regular expressions and actions (Java code) that are executed when the scanner matches the associated regular expression.

Understanding JFlex specification

- Get your hands dirty: work on an example specification.

- Check the JFlex user manual for more “technicalities”.

Java CUP parser generator

CUP is a LALR(1) parser generation for Java. As a parser author, you specify the symbols of your grammar

(`terminal T1,T2;`

`non terminal N1, N2;`),

as well as the productions

(`LHS :== RHS1 | RHS2 ;`).

If you provide each production alternative with action code

(`{: RESULT = myfunction(); :}`)

the parser will call this action code after performing a reduction with the particular production.

Java CUP specification

A CUP specification consists of:

- package and import specifications
- parser class name
- user code components
- symbol (terminal and non-terminal) lists
- precedence declarations
- the grammar