Compilers: Introduction and Scanners

a topic in

DM565 – Formal Languages and Data Processing

Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk

September, 2019

Compilers

SDU

Typically, transforming high level constructs to low level constructs.

Ex: Compiling Java to Java bytecode or C to X86 Assembly.

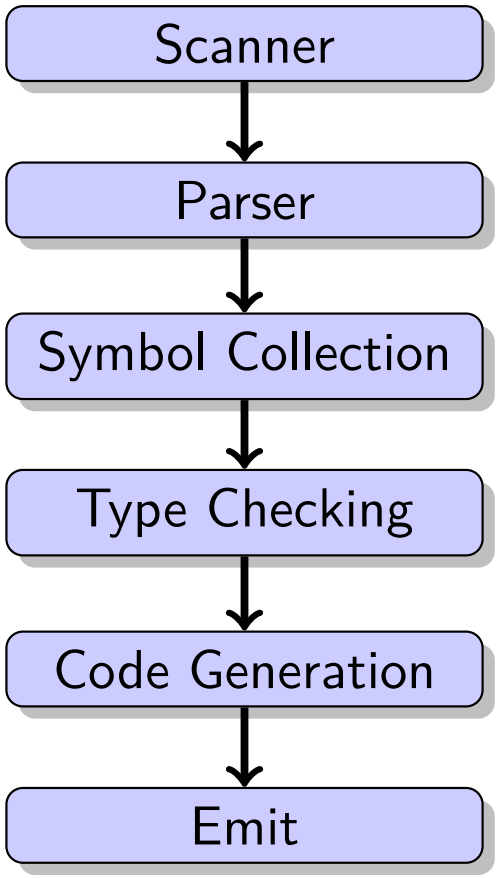
There are many high-level languages, and more keep coming.

Many domain-specific languages require compiler technology, such as LaTeX, lex (flex), yacc (bison), html expansions, etc.

Many companies maintain their own collection of "compilers" for screen control, dbms interfaces, etc.

Compiler Phases

The Minimum

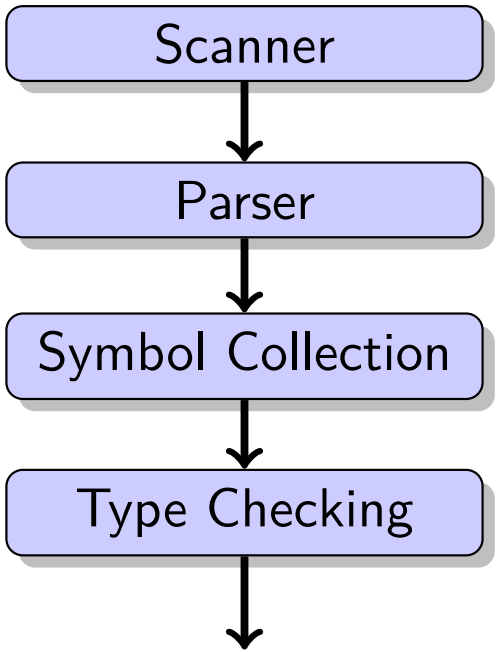


Compiler Phases

SDU

Front End

Analysis: "Ensuring that the input program is correct"



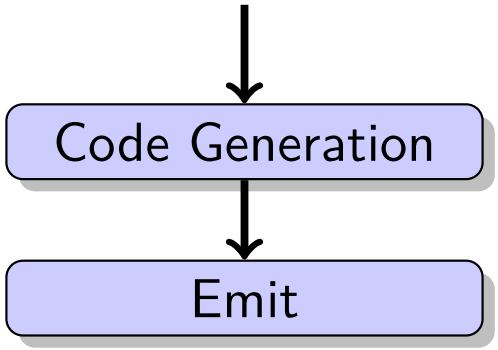
Weed phases can be inserted where required. They are for tasks *not* covered by the above, and therefore separate for modularity.

Compiler Phases

SDU

Back End

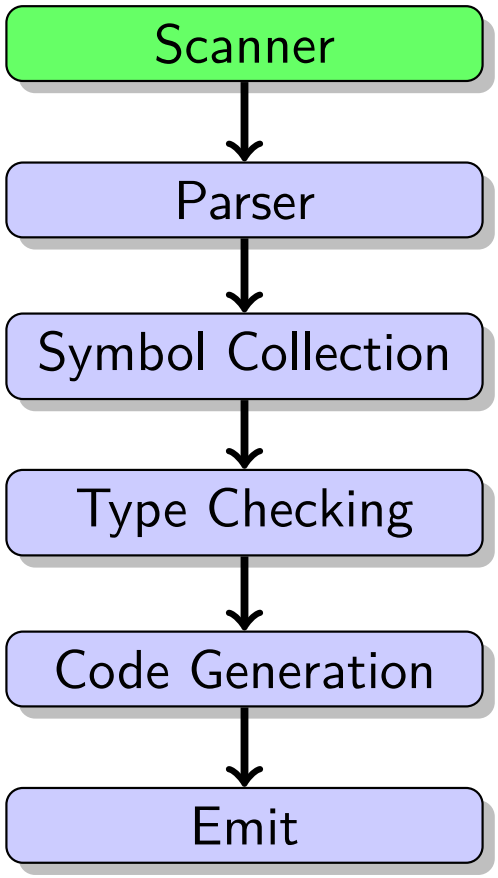
Synthesis: "Generating code for the correct input program"



Optimization phases can be inserted before code generation or after; important options include

- liveness analysis and register allocation
- peep-hole optimization
- garbage collection

Lexical Analysis: scanners



Lexical Analysis: scanners

SDU

Input to phase

A stream of characters (the user program).

Output from phase

A stream of lexical units.

Ex: **function**, **identifier** ("Fibonacci"), **(**, **identifier** ("n"), **:**, **int**, ..., **LEQ**, **num** ("42"), ...

Typically, want to ignore comments and whitespace (used as delimiter, but not output to next phase).

Overview of Lecture

1. How do we make software for this phase?
2. How do we use existing software for this (`flex`)?
3. How is it done in `scil`?