Compiler I

(dt. Übersetzer I)

Prof. Dr. Uwe Kastens

Winter 2001/2002

Lecture Compiler I WS 2001/2002 / Slide 01

In the lecture:

Welcome to the lecture!

Objectives

The participants are taught to

- understand **fundamental techniques** of language implementation,
- use **generating tools and standard solutions**,
- understand compiler construction as a systematic combination of **algorithms, theories** and **software engineering** methods for the solution of a **precisely specified task**,
- apply compiler techniques for languages **other than programming languages**.

Forms of teaching:

Lectures

Tutorials

Homeworks

Exercises

Running project

Lecture Compiler I WS 2001/2002 / Slide 02

Objectives:

Understand the objectives of the course.

In the lecture:

The objectives are explained.

Questions:

- What are your objectives?
- Do they match with these?
- When did you last listen to a talk given in English?

Lectures in English

Some agreements about giving lectures in English:

- I’ll speak English unless someone asks me to explain something in German.
- Stop me or slow me down whenever you get lost.
- I don‘t speak as well as a native speaker; but I’ll do my best ...
- You may ask questions and give answers in English or in German.
- I’ll prepare the slides in English. A German version is available.
- You‘ll have to learn to speak about the material in at least one of the two languages.
- You may vote which language to be used in the tutorials.
- You may chose German or English for the oral exam.

Lecture Compiler I WS 2001/2002 / Slide 03

Objectives:

Clarification about the use of the English language in this course

In the lecture:

The topics on the slide are discussed.

CI-4

Syllabus

Week	Chapter	Topic
1	**Introduction**	Compiler tasks
2		Compiler structure
3	**Lexical analysis**	Scanning, token representation
4	**Syntactic analysis**	Recursive decent parsing
5		LR Parsing
6		Parser generators
7		Grammar design
8	**Semantic analysis**	Attribute grammars
9		Attribute grammar specifications
10		Name analysis
11		Type analysis
12	**Transformation**	Intermediate language, target trees
13		Target texts
14	**Synthesis**	Overview
15	**Summary**	

Lecture Compiler I WS 2001/2002 / Slide 04

Objectives:

Overview over the topics of the course

In the lecture:

Comments on the topics.

Prerequisites

from Lecture	Topic	here needed for
Foundations of Programming Languages:		
	4 levels of language properties	Compiler tasks, compiler structure
	Context-free grammars	Syntactic analysis
	Scope rules	Name analysis
	Data types	Type analysis
	Lifetime, runtime stack	Storage model, code generation
Modeling:		
	Finite automata	Lexical analysis
	Context-free grammars	Syntactic analysis

Lecture Compiler I WS 2001/2002 / Slide 05

Objectives:

Identify concrete topics of other courses

In the lecture:

Point to material to be used for repetition

Suggested reading:

Course material for *Foundations of Programming Languages*

Course material for *Modeling*

Questions:

- Do you have the prerequisites?
- Are you going to learn or to repeat that material?

References

Material for this course **Compiler I**: http://www.uni-paderborn.de/cs/ag-kastens/compi
in German **Übersetzer I** (1999/2000): http://www.uni-paderborn.de/cs/ag-kastens/uebi
in English **Compiler II**: http://www.uni-paderborn.de/cs/ag-kastens/uebii

Modellierung: http://www.uni-paderborn.de/cs/ag-kastens/model
Grundlagen der Programmiersprachen: http://www.uni-paderborn.de/cs/ag-kastens/gdp

U. Kastens: **Übersetzerbau**, Handbuch der Informatik 3.3, Oldenbourg, 1990
(not available on the market anymore, available in the library of the University)

W. M. Waite, L. R. Carter: **An Introduction to Compiler Construction,**
Harper Collins, New York, 1993

W. M. Waite, G. Goos: **Compiler Construction**, Springer-Verlag, 1983

R. Wilhelm, D. Maurer: **Übersetzerbau - Theorie, Konstruktion, Generierung**,
Springer-Verlag, 1992

A. Aho, R. Sethi, J. D. Ullman: **Compilers - Principles, Techniques and Tools**,
Addison-Wesley, 1986

A. W. Appel: **Modern Compiler Implementation in C**, Cambridge University Press, 1997
(available for Java and for ML, too)

Lecture Compiler I WS 2001/2002 / Slide 06

Objectives:

Useful references for the course

In the lecture:

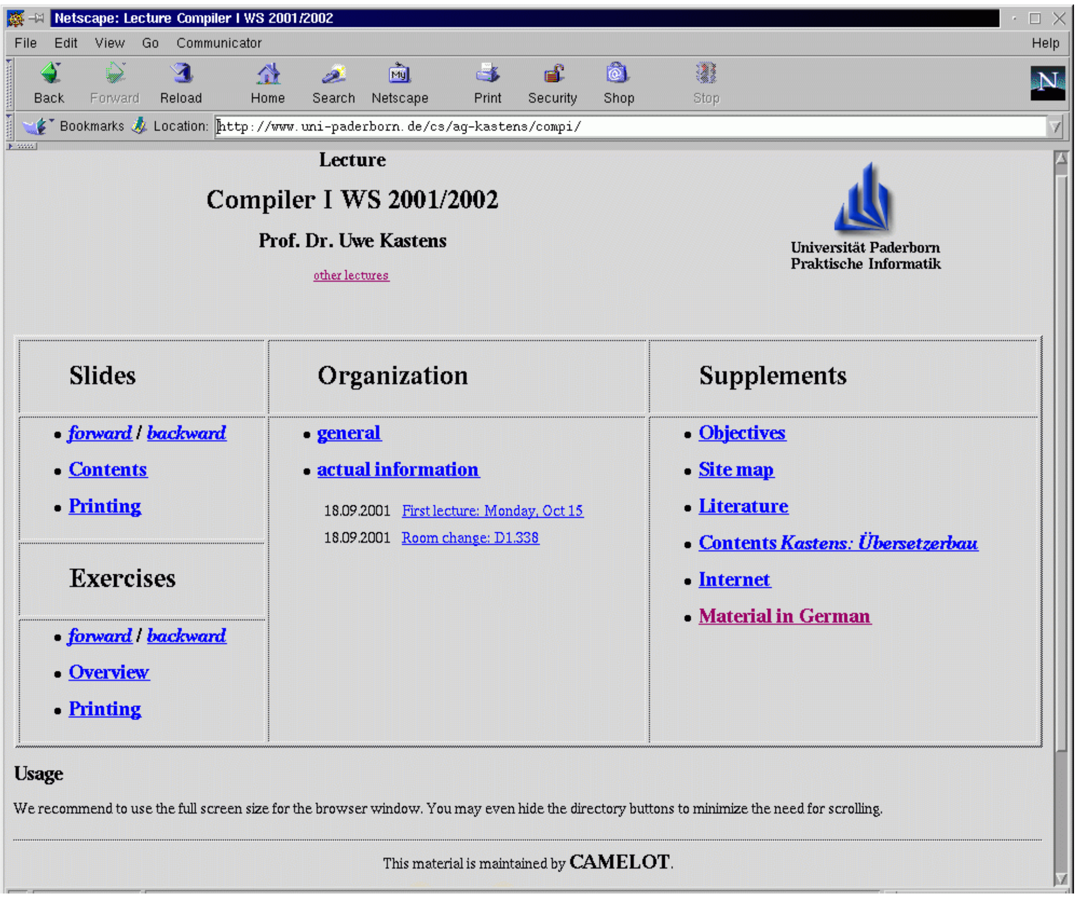
Comments of the course material and books

- The material for this course is being translated from the material of "Übersetzer I (WS 1999/2000)" while the course is given
- The course "Compiler II" will follow next semester.

Questions:

- Find the material in the Web, get used to its structure, place suitable bookmarks.

Course material in the Web



Netscape: Lecture Compiler I WS 2001/2002
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.uni-paderborn.de/cs/ag-kastens/compi/

Lecture

Compiler I WS 2001/2002

Prof. Dr. Uwe Kastens

other lectures

Universität Paderborn
Praktische Informatik

Slides	Organization	Supplements
• *forward* / *backward* • Contents • Printing	• general • actual information 18.09.2001 First lecture: Monday, Oct 15 18.09.2001 Room change: D1.338	• Objectives • Site map • Literature • Contents *Kastens: Übersetzerbau* • Internet • Material in German
Exercises		
• *forward* / *backward* • Overview • Printing		

Usage

We recommend to use the full screen size for the browser window. You may even hide the directory buttons to minimize the need for scrolling.

This material is maintained by CAMELOT.

Lecture Compiler I WS 2001/2002 / Slide 07

Objectives:

The root page of the course material.

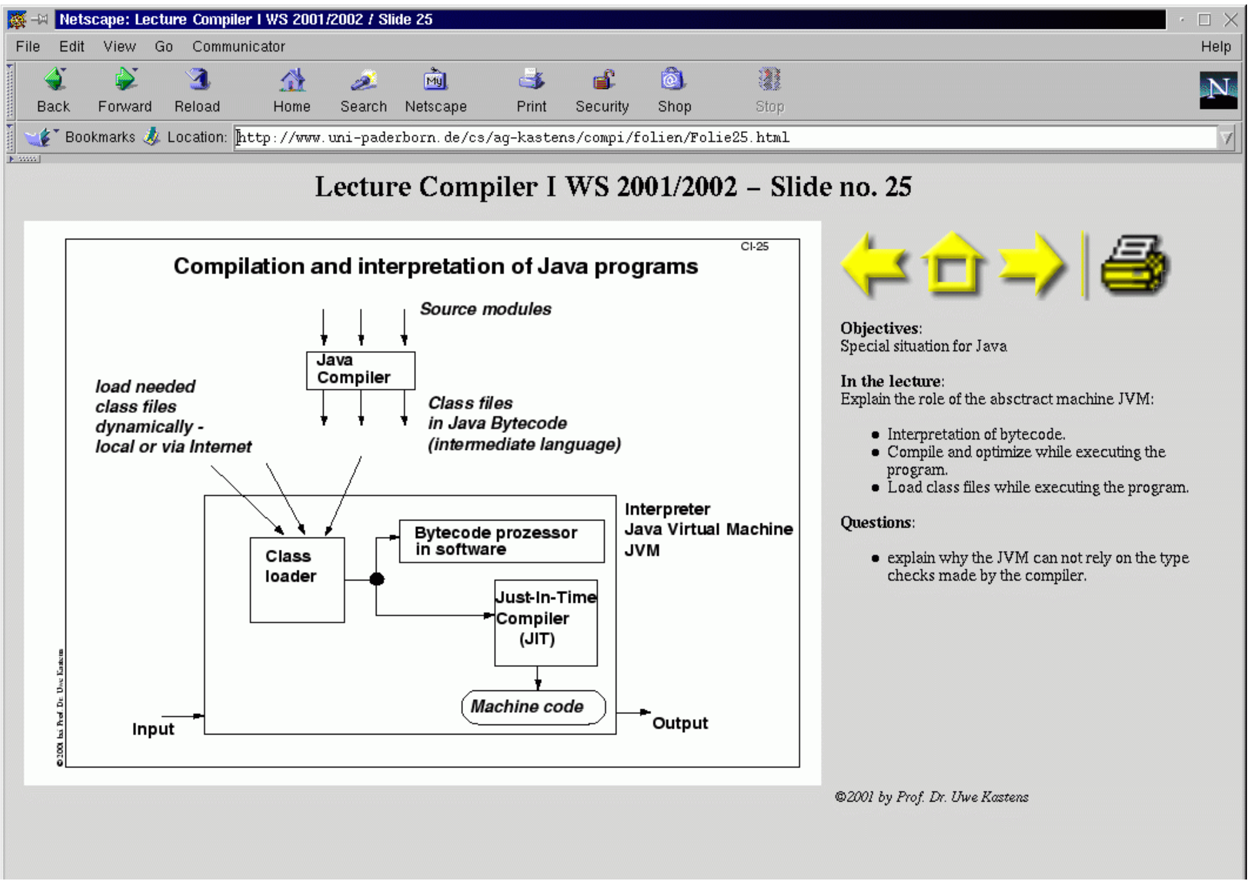
In the lecture:

The navigation structure is explained.

Assignments:

Explore the course material.

Commented slide in the course material


Netscape: Lecture Compiler I WS 2001/2002 / Slide 25
File
Edit
View
Go
Communicator
Help
Back
Forward
Reload
Home
Search
Netscape
Print
Security
Shop
Stop
Bookmarks
Location: http://www.uni-paderborn.de/cs/ag-kastens/compi/folien/Folie25.html
CI-25
Compilation and interpretation of Java programs
Source modules
Java Compiler
Class files in Java Bytecode (intermediate language)
load needed class files dynamically - local or via Internet
Class loader
Bytecode prozessor in software
Just-In-Time Compiler (JIT)
Machine code
Interpreter Java Virtual Machine JVM
Input
Output
©2001 bei Prof. Dr. Uwe Kastens

Lecture Compiler I WS 2001/2002 – Slide no. 25

Objectives:
Special situation for Java

In the lecture:
Explain the role of the absctract machine JVM:

- Interpretation of bytecode.
- Compile and optimize while executing the program.
- Load class files while executing the program.

Questions:

- explain why the JVM can not rely on the type checks made by the compiler.

Lecture Compiler I WS 2001/2002 / Slide 07a

Objectives:

A slide of the course material.

In the lecture:

The comments are explained.

Assignments:

Explore the course material.

What does a compiler compile?

A **compiler** transforms correct sentences of its **source language** into sentences of its **target language** such that their **meaning is unchanged.**

Examples:

Source language:	**Target language:**
Programming language	**Machine language**
C++	Sparc code
Programming language	**Abstract machine**
Java	Java Bytecode
Programming language	**Programming language (source-to-source)**
C++	C
Application language	**Application language**
LaTeX	HTML
Data base language (SQL)	Data base system calls

Lecture Compiler I WS 2001/2002 / Slide 08

Objectives:

Variety of compiler applications

In the lecture:

Explain examples for pairs of source and target languages.

Suggested reading:

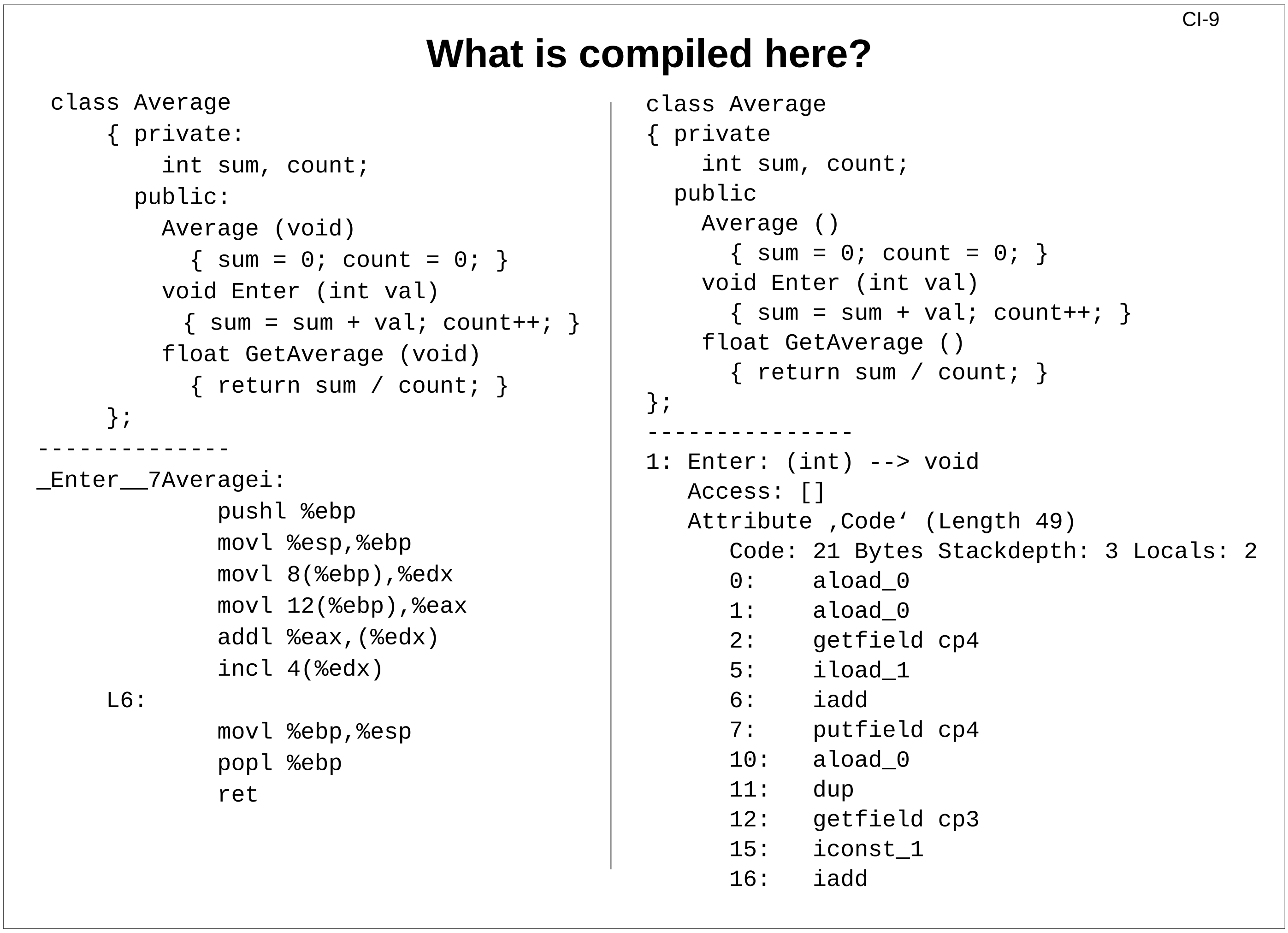
Kastens / Übersetzerbau, Section 1.

Assignments:

- Find more examples for application languages.
- Exercise 3 Recognize patterns in the target programs compiled from simple source programs.

Questions:

What are reasons to compile into other than machine languages?



What is compiled here?

```
 class Average
     { private:
         int sum, count;
       public:
         Average (void)
           { sum = 0; count = 0; }
         void Enter (int val)
          { sum = sum + val; count++; }
         float GetAverage (void)
           { return sum / count; }
     };
--------------
_Enter__7Averagei:
             pushl %ebp
             movl %esp,%ebp
             movl 8(%ebp),%edx
             movl 12(%ebp),%eax
             addl %eax,(%edx)
             incl 4(%edx)
     L6:
             movl %ebp,%esp
             popl %ebp
             ret
```

```
class Average
{ private
    int sum, count;
  public
    Average ()
      { sum = 0; count = 0; }
    void Enter (int val)
      { sum = sum + val; count++; }
    float GetAverage ()
      { return sum / count; }
};
---------------
1: Enter: (int) --> void
   Access: []
   Attribute ‚Code‘ (Length 49)
      Code: 21 Bytes Stackdepth: 3 Locals: 2
      0:     aload_0
      1:     aload_0
      2:     getfield cp4
      5:     iload_1
      6:     iadd
      7:     putfield cp4
      10:    aload_0
      11:    dup
      12:    getfield cp3
      15:    iconst_1
      16:    iadd
```

Lecture Compiler I WS 2001/2002 / Slide 09

Objectives:

Recognize examples for compilations

In the lecture:

Anwer the questions below.

Questions:

- Which source and target language are shown here?
- How did you recognize them?

What is compiled here?

```
 program Average;
       var sum, count: integer;
           aver: integer;
       procedure Enter (val: integer);
           begin sum := sum + val;
                 count := count + 1;
           end;
     begin
       sum := 0; count := 0;
       Enter (5); Enter (7);
       aver := sum div count;
     end.
-----------
void ENTER_5 (char *slnk , int VAL_4)
     {
     {/* data definitions: */
        /* executable code: */
        {
           SUM_1 = (SUM_1)+(VAL_4);
           COUNT_2 = (COUNT_2)+(1);
           ;
        }
     }}/* ENTER_5 */
```

```
\documentstyle[12pt]{article}
\begin{document}
\section{Introduction}
This is a very short document.
It just shows
\begin{itemize}
\item an item, and
\item another item.
\end{itemize}
\end{document}

-------------

%%Page: 1 1
1 0 bop 164 315 a Fc(1)81
b(In)n(tro)r(duction)
164 425 y Fb(This)16
b(is)g(a)h(v)o(ery)e(short)
i(do)q(cumen)o(t.)j(It)c(just)g
(sho)o(ws)237 527 y Fa(\017)24 b
Fb(an)17 b(item,)
c(and)237 628 y Fa(\017)24 b
Fb(another)17 b(item.)
961 2607 y(1)p
eop
```

Lecture Compiler I WS 2001/2002 / Slide 10

Objectives:

Recognize examples for compilations

In the lecture:

Anwer the questions below.

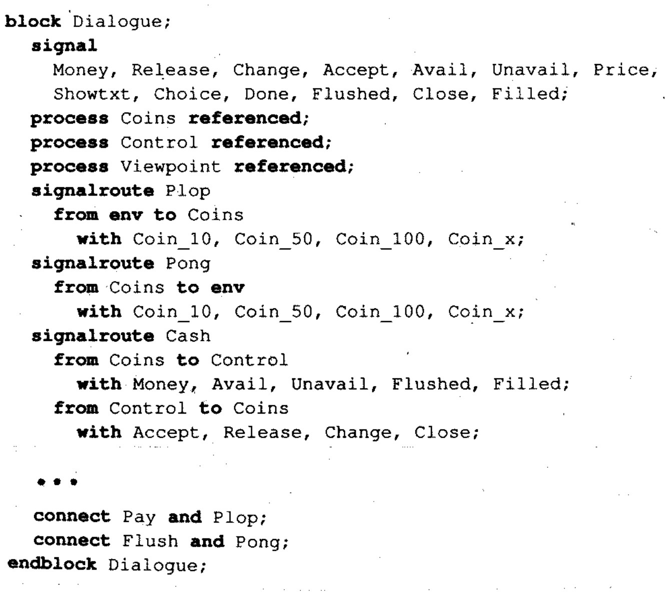
Questions:

- Which source and target language are shown here?
- How did you recognize them?

Languages for specification and modeling

SDL (CCITT)
Specification and Description Language:

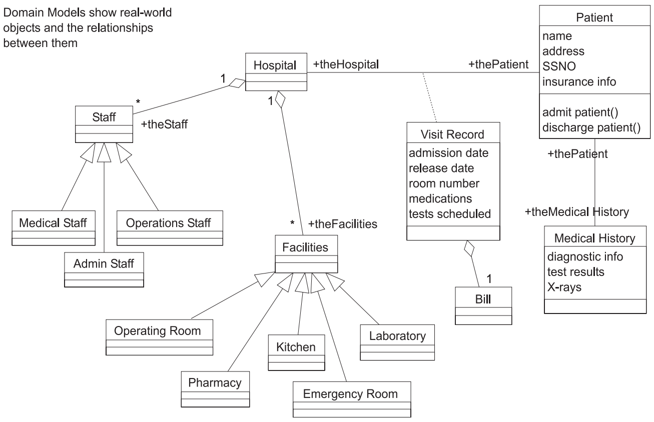
UML
Unified Modeling Language:



```
block Dialogue;
  signal
    Money, Release, Change, Accept, Avail, Unavail, Price,
    Showtxt, Choice, Done, Flushed, Close, Filled;
  process Coins referenced;
  process Control referenced;
  process Viewpoint referenced;
  signalroute Plop
    from env to Coins
      with Coin_10, Coin_50, Coin_100, Coin_x;
  signalroute Pong
    from Coins to env
      with Coin_10, Coin_50, Coin_100, Coin_x;
  signalroute Cash
    from Coins to Control
      with Money, Avail, Unavail, Flushed, Filled;
    from Control to Coins
      with Accept, Release, Change, Close;

  ...

  connect Pay and Plop;
  connect Flush and Pong;
endblock Dialogue;
```



Lecture Compiler I WS 2001/2002 / Slide 11

Objectives:

Be aware of specification languages

In the lecture:

Comments on SDL and UML

Suggested reading:

Text

Questions:

What kind of tools are needed for such specification languages?

Domain Specific Languages (DSL)

A language designed for a **specific application domain.**
Application Generator: Implementation of a DSL by a **program generator**

Examples:

- **Simulation of mechatronic feedback systems**
- **Robot control**
- **Collecting data from instruments**
- **Testing car instruments**
- **Report generator for bibliographies:**

```
string name =   InString "Which author?";
int since =     InInt "Since which year?";
int cnt = 0;

"\nPapers of ", name, " since ", since, ":\n";
[ SELECT name <= Author && since <= Year;
  cnt = cnt + 1;
  Year, "\t", Title, "\n";
]
"\n", name, " published ", cnt, "papers.\n";
```

U. Kastens: Construction of Application Generators Using Eli,
Workshop on Compiler Techniques for Application Domain Languages ...,
Linköping, April 1996

Lecture Compiler I WS 2001/2002 / Slide 12

Objectives:

Understand DSL by examples

In the lecture:

Explain the examples

Suggested reading:

- C.W. Krueger: Software Reuse, ACM Computing Surveys 24, June 1992
- Conference on DSL (USENIX), Santa Babara, Oct. 1997
- ACM SIGPLAN Workshop on DSL (POPL), Paris, Jan 1997

Questions:

Give examples for tools that can be used for such languages.

Programming languages as source or target languages

Programming languages as source languages:

- **Program analysis**
  call graphs, control-flow graph, data dependencies, e. g. for the year 2000 problem
- **Recognition of structures and patterns**
  e. g. for Reengineering

Program languages as target languages:

- **Specifications (SDL, OMT, UML)**
- **graphic modeling of structures**
- **DSL, Application generator**

=> Compiler task: Source-to-source compilation

Lecture Compiler I WS 2001/2002 / Slide 13

Objectives:

Understand programming languages in different roles

In the lecture:

- Comments on the examples
- Role of program analysis in software engineering
- Role of Source-to-source compilation in software engineering

Questions:

Give examples for the use of program analysis in software engineering.

Semester project as running example

A Structure Generator

We are going to develop a tool that implements **record structures**. In particular, the structure generator takes a set of **record descriptions**. Each specifies a **set of named and typed fields**. For each record a **Java class** declaration is to be generated. It contains a constructor method and access methods for the specified record fields.

The tool will be used in an environment where field description are created by other tools, which for example analyze texts for the occurrence of certain phrases. Hence, the descriptions of fields may occur in arbitrary order, and the same field may be described more than once. The structure generator **accumulates the field descriptions** such that for each record a single class declaration is generated which has all fields of that record.

Design a **domain specific language**.

Implement an **application generator** for it.

Apply all **techniques of the course** that are useful for the task.

Lecture Compiler I WS 2001/2002 / Slide 14

Objectives:

Get an idea of the task

In the lecture:

- Comment the task description.
- Explain the role of the running example.

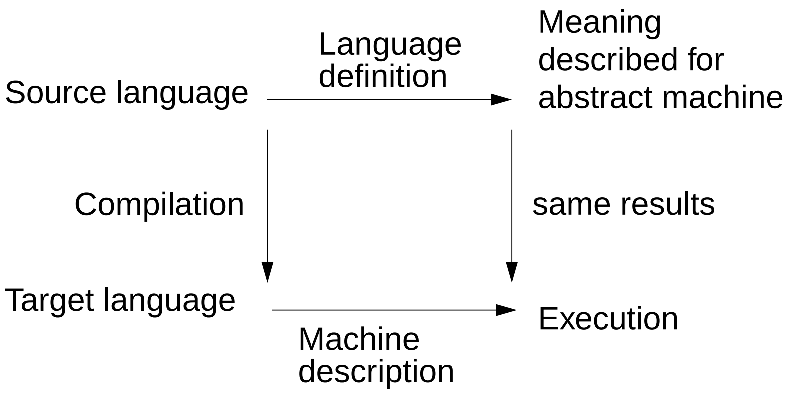
Assignments:

In the tutorial

- Discuss the task description.
- Explain the purpose of such a generator.
- Give examples for its input and output.
- What are the consequences of the second paragraph of the task description?
- Discuss variants of the input.

Meaning preserving transformation

A **compiler** transforms correct sentences of its **source language** into sentences of its **target language** such that their **meaning is unchanged**.



Source language

Language definition

Meaning described for abstract machine

Compilation

same results

Target language

Machine description

Execution

A **meaning** is defined only for **correct** programs. Compiler task: Error handling

The compiler analyses **static** properties of the program at **compile time,**
e. g. definitions of Variables, types of expressions. Decides: Is the program **compilable?**

Dynamic properties of the program are checked at **runtime,**
e. g. indexing of arrays. Decides: Is the program **executable?**

But in Java: Compilation of bytecode at runtime, just in time compilation (JIT)

Lecture Compiler I WS 2001/2002 / Slide 15

Objectives:

Understand fundamental notions of compilation

In the lecture:

The topics on the slide are explained. Examples are given.

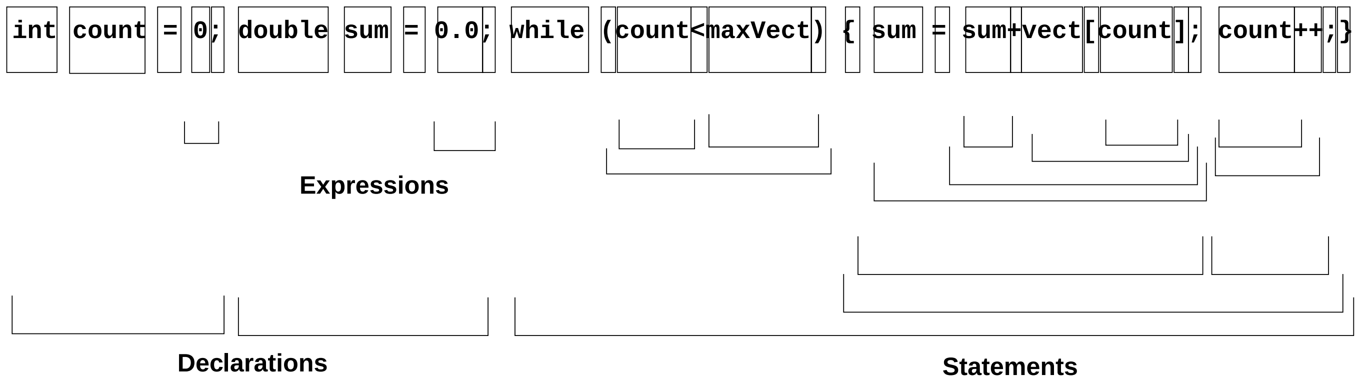
- Explain the role of the arcs in the commuting diagram.
- Distinguish compile time and run-time concepts.
- Discuss examples.

Example: Tokens and structure

Character sequence

```
int count = 0; double sum = 0.0; while (count<maxVect) { sum = sum+vect[count]; count++;}
```

Tokens



int count = 0 ; double sum = 0.0 ; while (count < maxVect) { sum = sum + vect [count] ; count ++ ; }

Expressions

Declarations

Statements

Structure

Lecture Compiler I WS 2001/2002 / Slide 16

Objectives:

Get an idea of the structuring task

In the lecture:

Some requirements for recognizing tokens and deriving the program structure are discussed along the example:

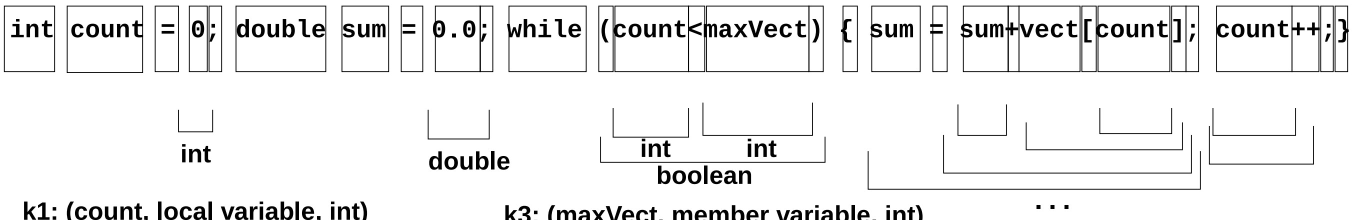
- kinds of tokens,
- characters between tokens,
- nested structure

Questions:

Where do you find the exact requirements for the structuring tasks?

Example: Names, types, generated code

Tokens



k1: (count, local variable, int)
k2: (sum, local variable, double)

k3: (maxVect, member variable, int)
k4: (vect, member variable, double array)

Names and types

generated Bytecode

```
0 iconst_0
1 istore_1
2 dconst_0
3 dstore_2
4 goto 19
7 dload_2
8 getstatic #5 <vect[]>
11 iload_1
12 faload
13 f2d
14 dadd
15 dstore_2
16 iinc 1 1
19 iload_1
20 getstatic #4 <maxVect>
23 if_icmplt 7
```

Lecture Compiler I WS 2001/2002 / Slide 17

Objectives:

Get an idea of the name analysis and transformation task

In the lecture:

Some requirements for these tasks are discussed along the example:

- program objects and their properties,
- program constructs and their types
- target program

Questions:

- Why is the name (e.g. count) a property of a program object (e.g. k1)?
- Can you impose some structure on the target code?

Language definition - Compiler task

• **Notation of tokens** keywords, identifiers, literals formal definition: regular expressions	**lexical analysis**
• **Syntactic structure** formal definition: context-free grammar	**syntactic analysis**
• **Static semantics** binding names to program objects, typing rules usually defined by informal texts	**semantic analysis, transformation**
• **Dynamic semantics** semantics, effect of the execution of constructs usually defined by informal texts in terms of an abstract machine	**transformation, code generation**
• **Definition of the target language (machine)**	**transformation, code generation** **assembly**

Lecture Compiler I WS 2001/2002 / Slide 18

Objectives:

Relate language properties to levels of definitions

In the lecture:

- These are prerequisites of the course "Grundlagen der Programmiersprachen" (see course material GdP-13, GdP13a).
- Discuss the examples of the preceding slides under these categories.

Suggested reading:

Kastens / Übersetzerbau, Section 1.2

Assignments:

- Exercise 1 Let the compiler produce error messages for each level.
- Exercise 2 Relate concrete language properties to these levels.

Questions:

Some language properties can be defined on different levels. Discuss the following for hypothetical languages:

- "Parameters may not be of array type." Syntax or static semantics?
- "The index range of an array may not be empty." Static or dynamic semantics?

Compiler tasks

Structuring	Lexical analysis	Scanning Conversion
	Syntactic analysis	Parsing Tree construction
Translation	Semantic analysis	Name analysis Type analysis
	Transformation	Data mapping Action mapping
Encoding	Code generation	Execution-order Register allocation Instruction selection
	Assembly	Instruction encoding Internal Addressing External Addressing

Lecture Compiler I WS 2001/2002 / Slide 19

Objectives:

Task decomposition leads to compiler structure

In the lecture:

- Explain tasks of the rightmost column.
- Relate the tasks to chapters of the course.

Suggested reading:

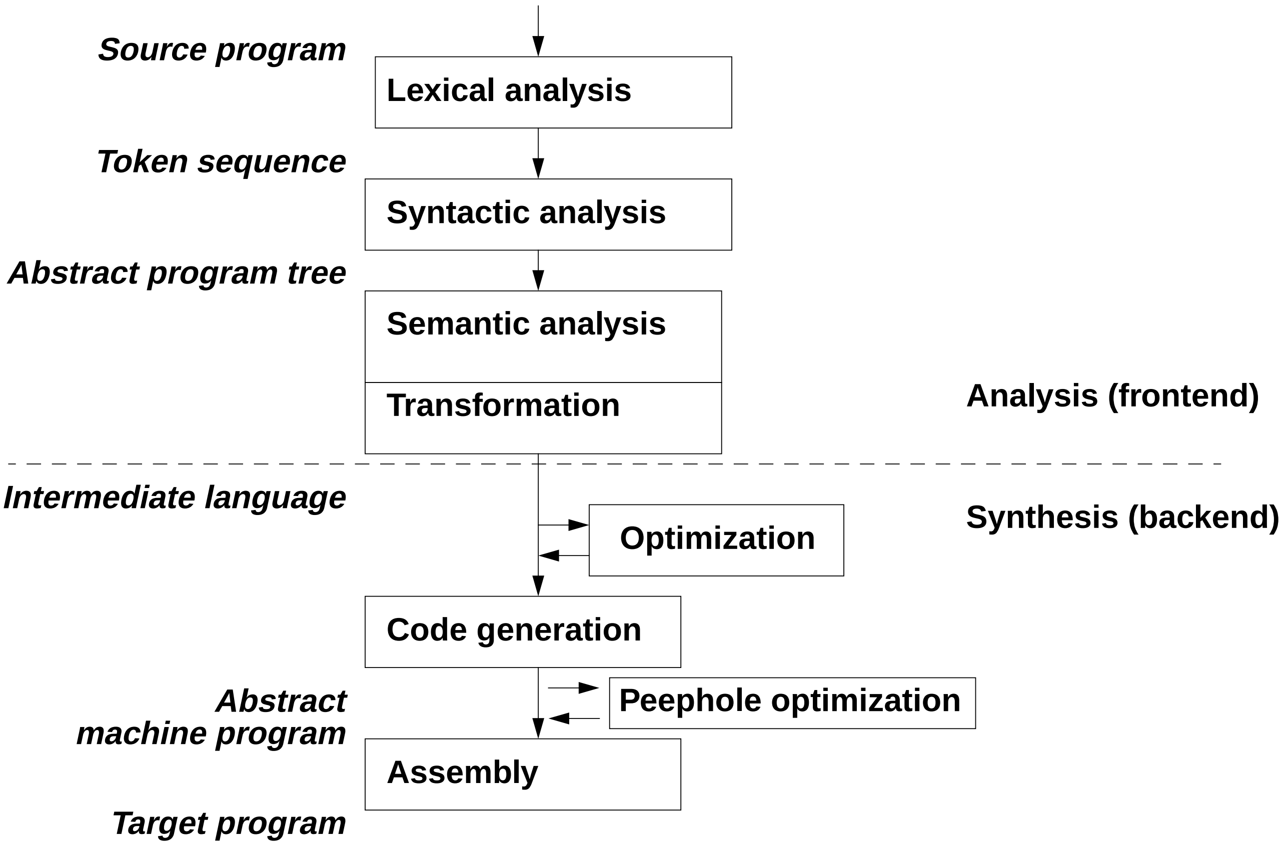
Kastens / Übersetzerbau, Section 2.1

Assignments:

Learn the German translations of the technical terms.

CI-20

Compiler structure and interfaces



Source program

Lexical analysis

Token sequence

Syntactic analysis

Abstract program tree

Semantic analysis

Transformation

Analysis (frontend)

Intermediate language

Optimization

Synthesis (backend)

Code generation

Abstract machine program

Peephole optimization

Assembly

Target program

Lecture Compiler I WS 2001/2002 / Slide 20

Objectives:

Derive compiler modules from tasks

In the lecture:

In this course we focus on the analysis phase (frontend).

Suggested reading:

Kastens / Übersetzerbau, Section 2.1

Assignments:

Compare this slide with U-08 and learn the translations of the technical terms used here.

Questions:

Use this information to explain the example on slide CI-16

Software qualities of the compiler

- **Correctness** Translate correct programs correctly.
Reject wrong programs and give error messages
- **Efficiency** Storage and time used by the compiler
- **Code efficiency** Storage and time used by the generated code
Compiler task: Optimization
- **User support** Compiler task: Error handling
(recognition, message, recovery)
- **Robustness** Give a reasonable reaction on every input

Lecture Compiler I WS 2001/2002 / Slide 21

Objectives:

Consider compiler as a software product

In the lecture:

Give examples for the qualities.

Questions:

Explain: For a compiler the requirements are specified much more precisely than for other software products.

Strategies for compiler construction

- Obey exactly to the language definition
- Use generating tools
- Use standard components
- Apply standard methods
- Validate the compiler against a test suite
- Verify components of the compiler

Lecture Compiler I WS 2001/2002 / Slide 22

Objectives:

Apply software methods for compiler construction

In the lecture:

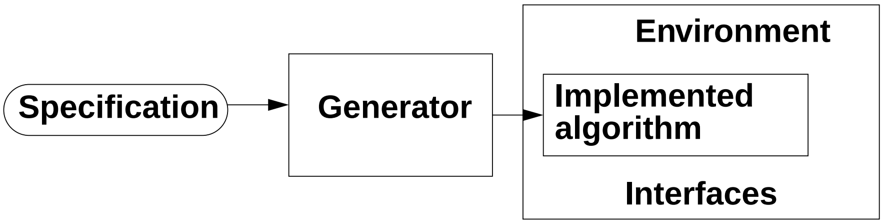
It is explained that effective construction methods exists especially for compilers.

Questions:

What do the specifications of the compiler tasks contribute to more systematic compiler construction?

Generators

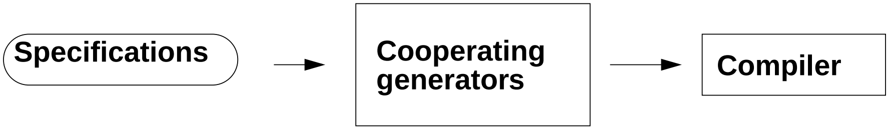
Pattern:



Typical compiler tasks solved by generators:

Regular expressions	**Scanner generator**	Finite automaton
Context-free grammar	**Parser generator**	Stack automaton
Attribute grammar	**Attribute evaluator generator**	Tree walking algorithm
Code patterns	**Code selection generator**	Pattern matching

integrated system Eli:



Lecture Compiler I WS 2001/2002 / Slide 23

Objectives:

Usage of generators in compiler construction

In the lecture:

The topics on the slide are explained. Examples are given.

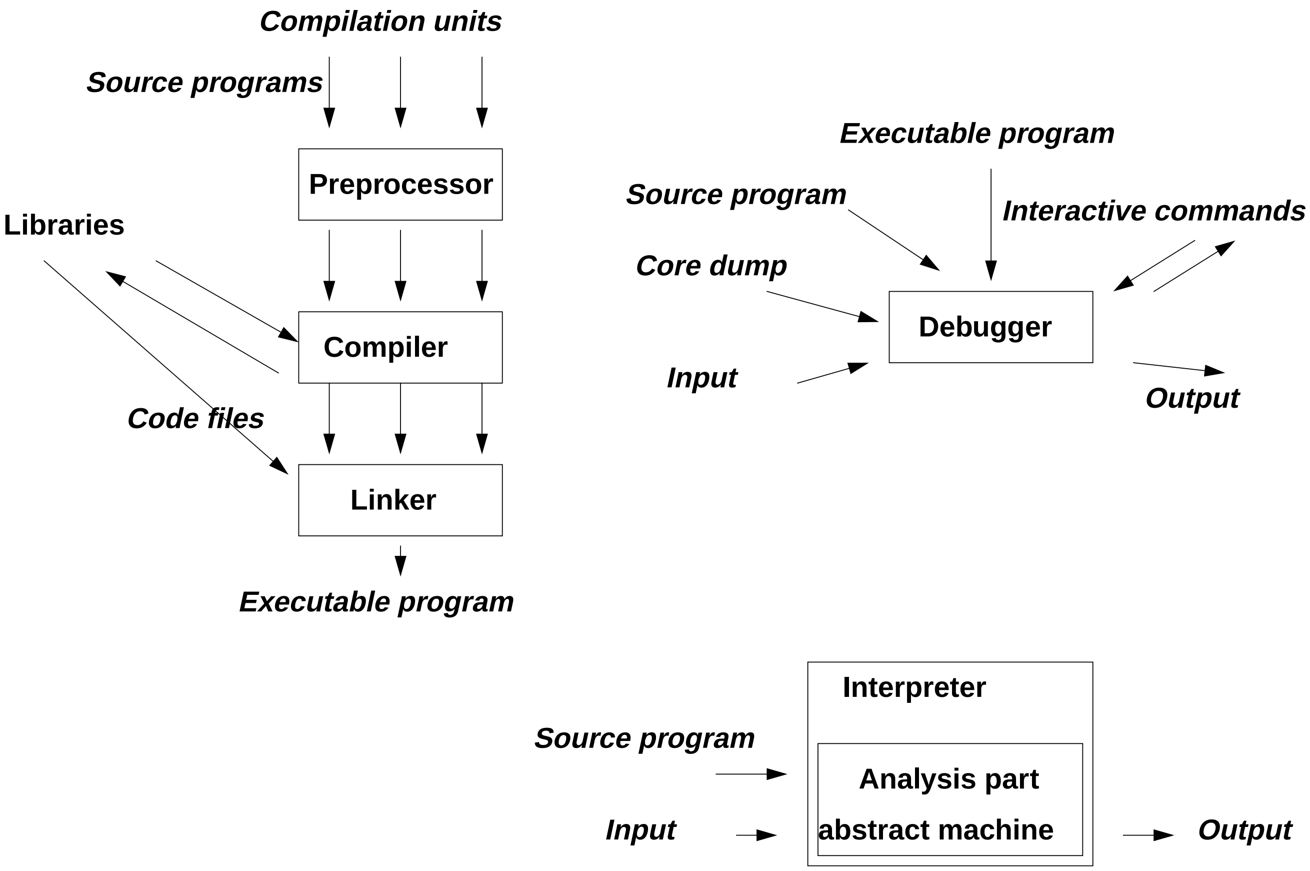
Suggested reading:

Kastens / Übersetzerbau, Section 2.5

Assignments:

- Exercise 5: Find as many generators as possible in the Eli system.

Environment of compilers



Compilation units

Source programs

Preprocessor

Libraries

Compiler

Code files

Linker

Executable program

Executable program

Source program

Interactive commands

Core dump

Debugger

Input

Output

Interpreter

Source program

Analysis part

abstract machine

Input

Output

Lecture Compiler I WS 2001/2002 / Slide 24

Objectives:

Understand the cooperation between compilers and other language tools

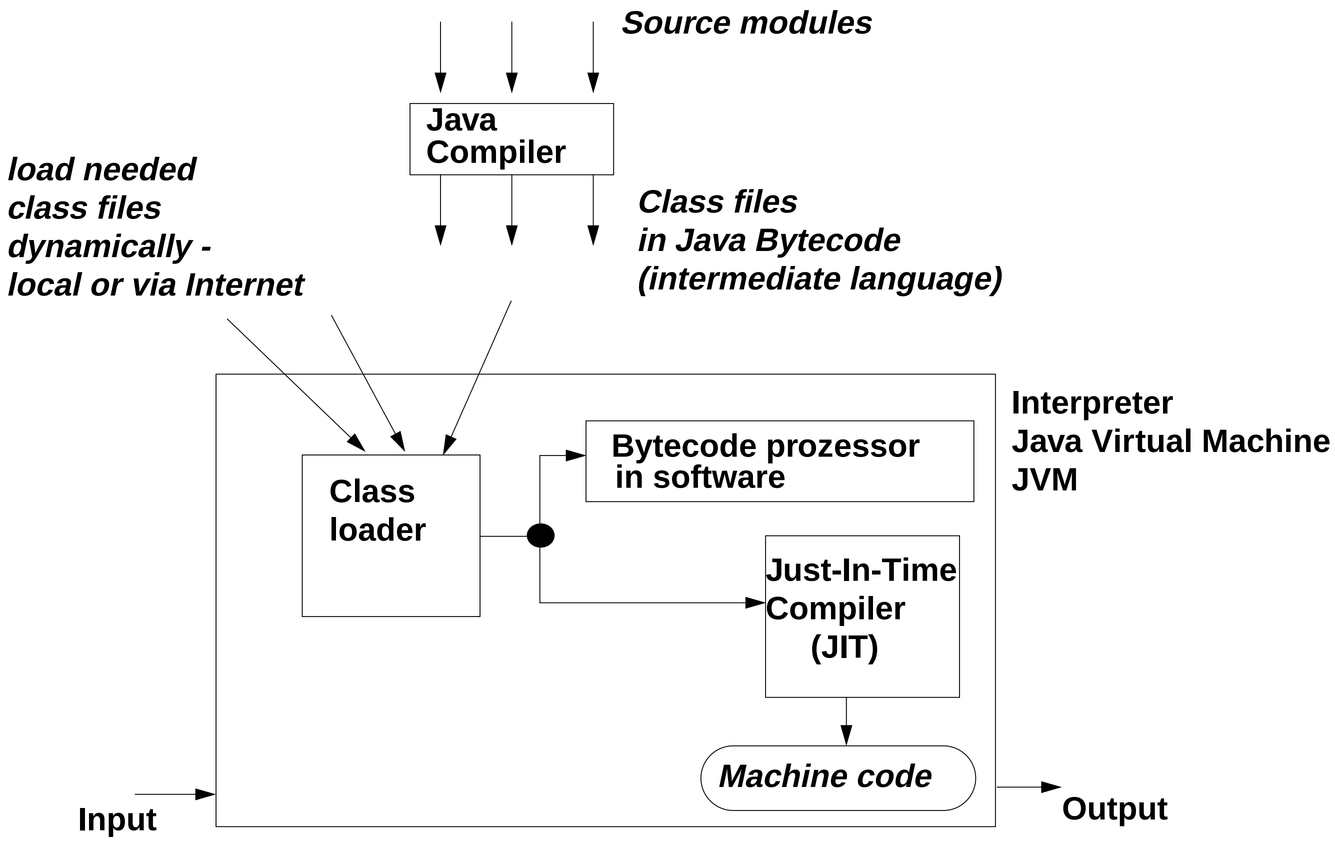
In the lecture:

- Explain the roles of language tools
- Explain the flow of information

Suggested reading:

Kastens / Übersetzerbau, Section 2.4

Compilation and interpretation of Java programs


Source modules
Java Compiler
Class files in Java Bytecode (intermediate language)
load needed class files dynamically - local or via Internet
Class loader
Bytecode prozessor in software
Just-In-Time Compiler (JIT)
Machine code
Interpreter Java Virtual Machine JVM
Input
Output

Lecture Compiler I WS 2001/2002 / Slide 25

Objectives:

Special situation for Java

In the lecture:

Explain the role of the absctract machine JVM:

- Interpretation of bytecode.
- Compile and optimize while executing the program.
- Load class files while executing the program.

Questions:

- explain why the JVM can not rely on the type checks made by the compiler.