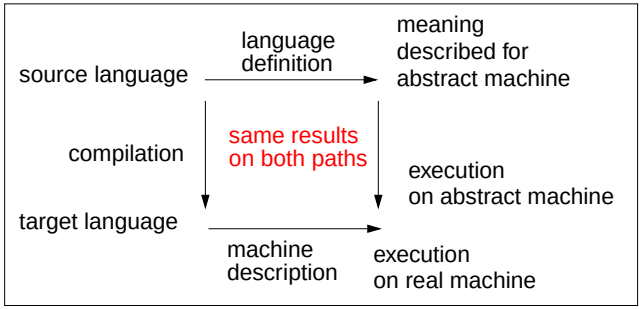
1. Language properties - compiler tasks
Meaning preserving transformation

A **compiler** transforms **any correct sentence** of its **source language** into a sentence of its **target language** such that its **meaning is unchanged**.



A **meaning** is defined only for **all correct** programs => compiler task: error handling

Static language properties are analyzed at **compile time,** e. g. definitions of Variables, types of expressions; => determine the transformation, if the program **compilable**

Dynamic properties of the program are determined and checked at **runtime,** e. g. indexing of arrays => determine the effect, if the program **executable** (However, just-in-time compilation for Java: bytecode is compiled at runtime.)

Lecture Programming Languages and Compilers WS 2010/11 / Slide 101

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.

Levels of language properties - compiler tasks

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

Lecture Programming Languages and Compilers WS 2010/11 / Slide 102

Objectives:

Relate language properties to levels of definitions

In the lecture:

- These are prerequisites of the course "Grundlagen der Programmiersprachen" (see course material GPS-1.16, GPS-1.17).
- Discuss the examples of the following 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:

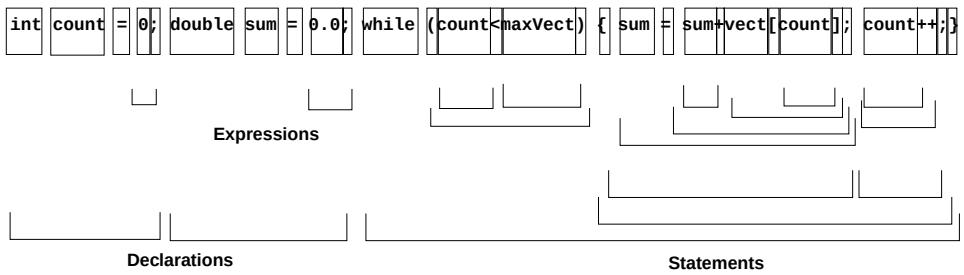
- "Parameters may not be of array type." Syntax or static semantics?
- "The index range of an array may not be empty." Static or dynamic semantics?

Example: Tokens and structure

Character sequence

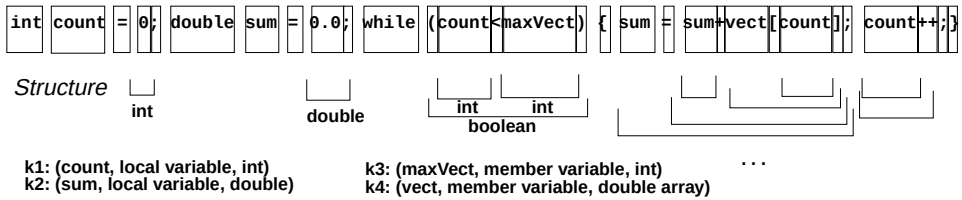
```
int count = 0; double sum = 0.0; while (count<maxVect) { sum = sum+vect[count]; count++;}
```

Tokens



Structure

Example: Names, types, generated code



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

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

Static properties: 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 Programming Languages and Compilers WS 2010/11 / Slide 103

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?

Lecture Programming Languages and Compilers WS 2010/11 / Slide 104

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?

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 Programming Languages and Compilers WS 2010/11 / Slide 105

Objectives:

Language properties lead to decomposed compiler tasks

In the lecture:

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

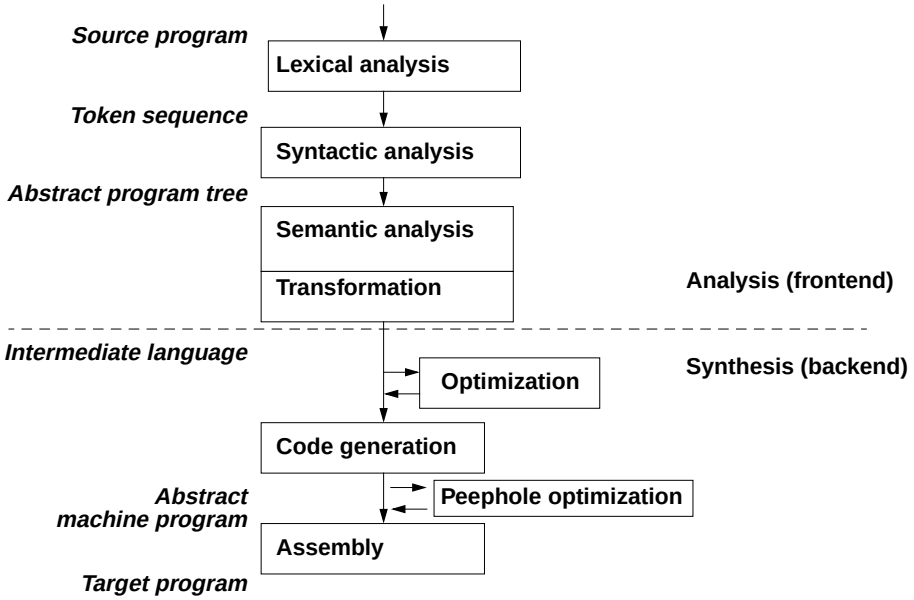
Suggested reading:

Kastens / Übersetzerbau, Section 2.1

Assignments:

Learn the German translations of the technical terms.

Compiler structure and interfaces



Lecture Programming Languages and Compilers WS 2010/11 / Slide 106

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 slides PLaC-1.3, 1.4

Software qualities of the compiler

- **Correctness** Compiler translates correct programs correctly; rejects wrong programs and gives 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** Compiler gives a reasonable reaction on every input; does not break on any program

Lecture Programming Languages and Compilers WS 2010/11 / Slide 107

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 Programming Languages and Compilers WS 2010/11 / Slide 108

Objectives:

Apply software methods for compiler construction

In the lecture:

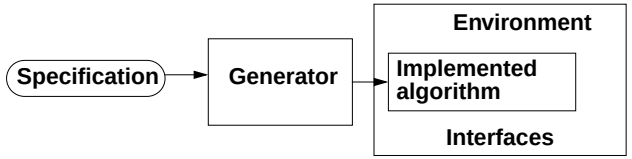
It is explained that effective construction methods exist especially for compilers.

Questions:

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

Generate from specifications

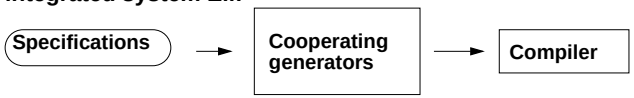
Pattern:



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 Programming Languages and Compilers WS 2010/11 / Slide 109

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.

Compiler Frameworks (Selection)

Amsterdam Compiler Kit: (Uni Amsterdam)
The Amsterdam Compiler Kit is fast, lightweight and retargetable compiler suite and toolchain written by Andrew Tanenbaum and Ceriel Jacobs.
Intermediate language EM, set of frontends and backends

ANTLR: (Terence Parr, Uni San Francisco)
ANother Tool for Language Recognition, (formerly PCCTS) is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing Java, C#, C++, or Python actions

CoCo: (Uni Linz)
Coco/R is a compiler generator, which takes an attributed grammar of a source language and generates a scanner and a parser for this language. The scanner works as a deterministic finite automaton. The parser uses recursive descent.

Eli: (Unis Boulder, Paderborn, Sydney)
Combines a variety of standard tools that implement powerful compiler construction strategies into a domain-specific programming environment called Eli. Using this environment, one can automatically generate complete language implementations from application-oriented specifications.

SUIF: (Uni Stanford)
The SUIF 2 compiler infrastructure project is co-funded by DARPA and NSF.
It is a free infrastructure designed to support collaborative research in optimizing and parallelizing compilers.

Lecture Programming Languages and Compilers WS 2010/11 / Slide 109a

Objectives:

General information on compiler tool kits

In the lecture:

Some characteristics of the systems are explained.

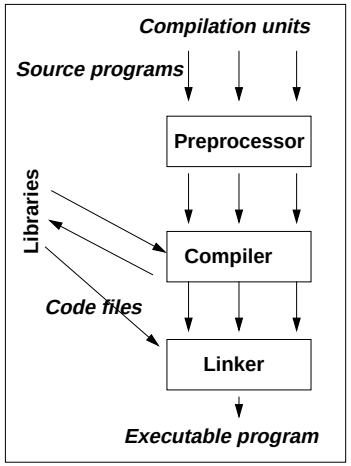
Suggested reading:

Kastens / Übersetzerbau, Section 2.5

Assignments:

- Find more information on the system in the Web

Environment of compilers



Preprocessor cpp substitutes text macros in source programs, e.g.

```
#include <stdio.h>
#include "module.h"

#define SIZE 32
#define SEL(ptr,fld) ((ptr)->fld)
```

Separate compilation of compilation units

- with interface specification, consistency checks, and language specific linker: Modula, Ada, Java
- without ...; checks deferred to system linker: C, C++

Lecture Programming Languages and Compilers WS 2010/11 / Slide 110

Objectives:

Understand the cooperation between compilers and other language tools

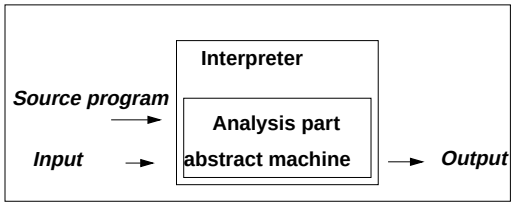
In the lecture:

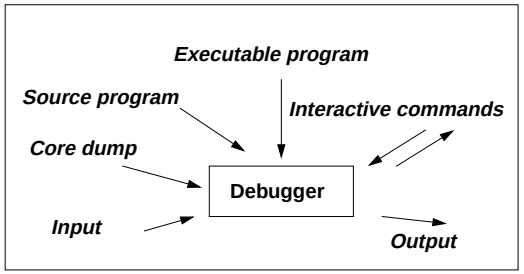
- Explain the roles of language tools
- Explain the flow of information

Suggested reading:

Kastens / Übersetzerbau, Section 2.4

Interpreter and Debugger





Lecture Programming Languages and Compilers WS 2010/11 / Slide 110a

Objectives:

Understand the cooperation between compilers and other language tools

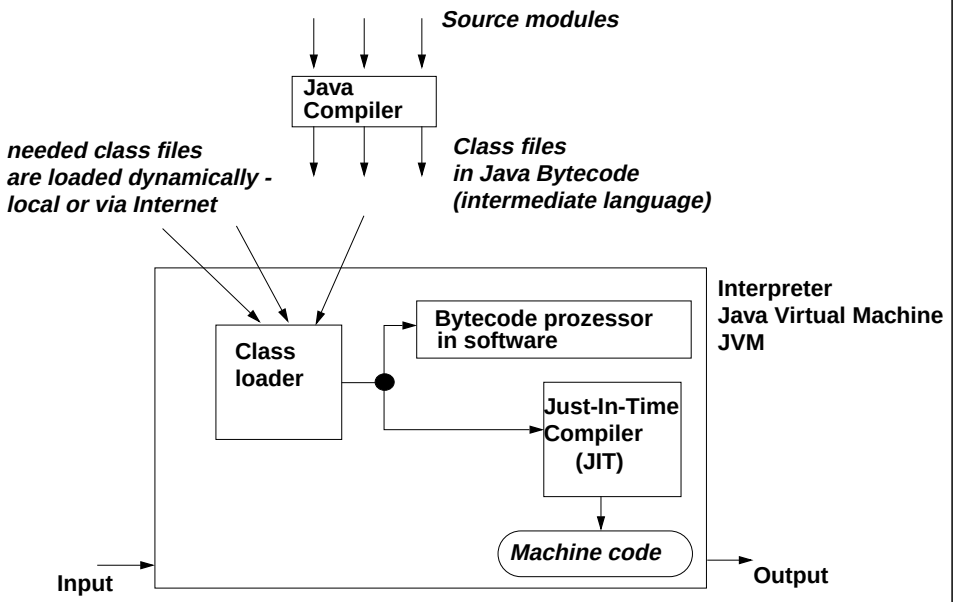
In the lecture:

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

Suggested reading:

Kastens / Übersetzerbau, Section 2.4

Compilation and interpretation of Java programs



Lecture Programming Languages and Compilers WS 2010/11 / Slide 111

Objectives:

Special situation for Java

In the lecture:

Explain the role of the absctract machine JVM:

- Interpretation of bytecode.
- JIT: Compiles and optimizes while executing the program.
- JVM: Loads class files while executing the program.

Questions:

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