Programming Languages and Compilers

Introduction

Dr. Peter Pfahler
Based on the lecture by Prof. Dr. Uwe Kastens

Universität Paderborn
Fakultät EIM
Institut für Informatik

Winter 2016/2017

Learning Objectives

At the end of this class, students should be able to:

- identify language elements and characteristics of programming languages and domain-specific languages
- recall the fundamental techniques of language implementation
- develop formal specifications of a language
- apply generating tools and standard solutions to implement compilers

Forms of Teaching

- Lecture
- Discussions
- Reading
- "Check your Knowledge"
- Exercises
- Lab Exercises
- Running Project "SetLan"
- Homework Assignments

Contents

Prerequisites

From the lecture "Modeling" (1st semester)

- Finite automata
  ↪ needed for lexical analysis
- Context-free grammars
  ↪ needed for grammar design, syntactic analysis

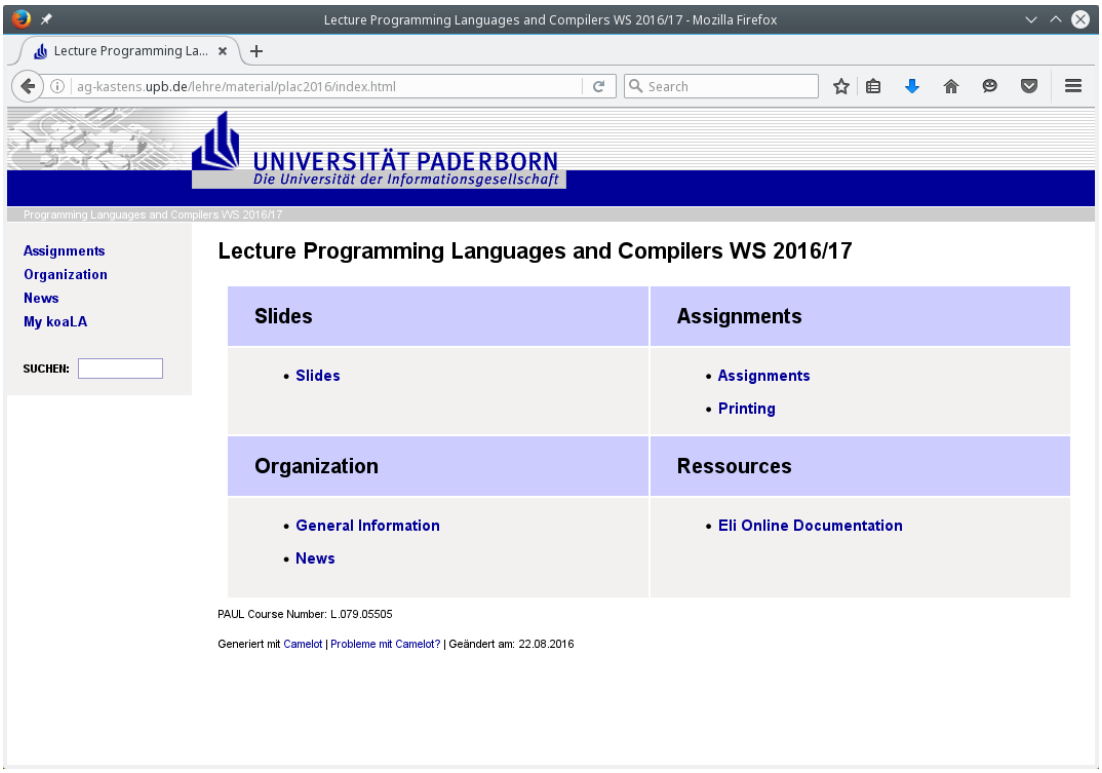
From the lecture "Foundations of Programming Languages" (2nd semester)

- Levels of language properties
  ↪ needed for language specification, compiler tasks
- Context-free grammars
  ↪ needed for grammar design, syntactic analysis
- Scope rules
  ↪ needed for name analysis
- Data types
  ↪ needed for type analysis, type systems

References

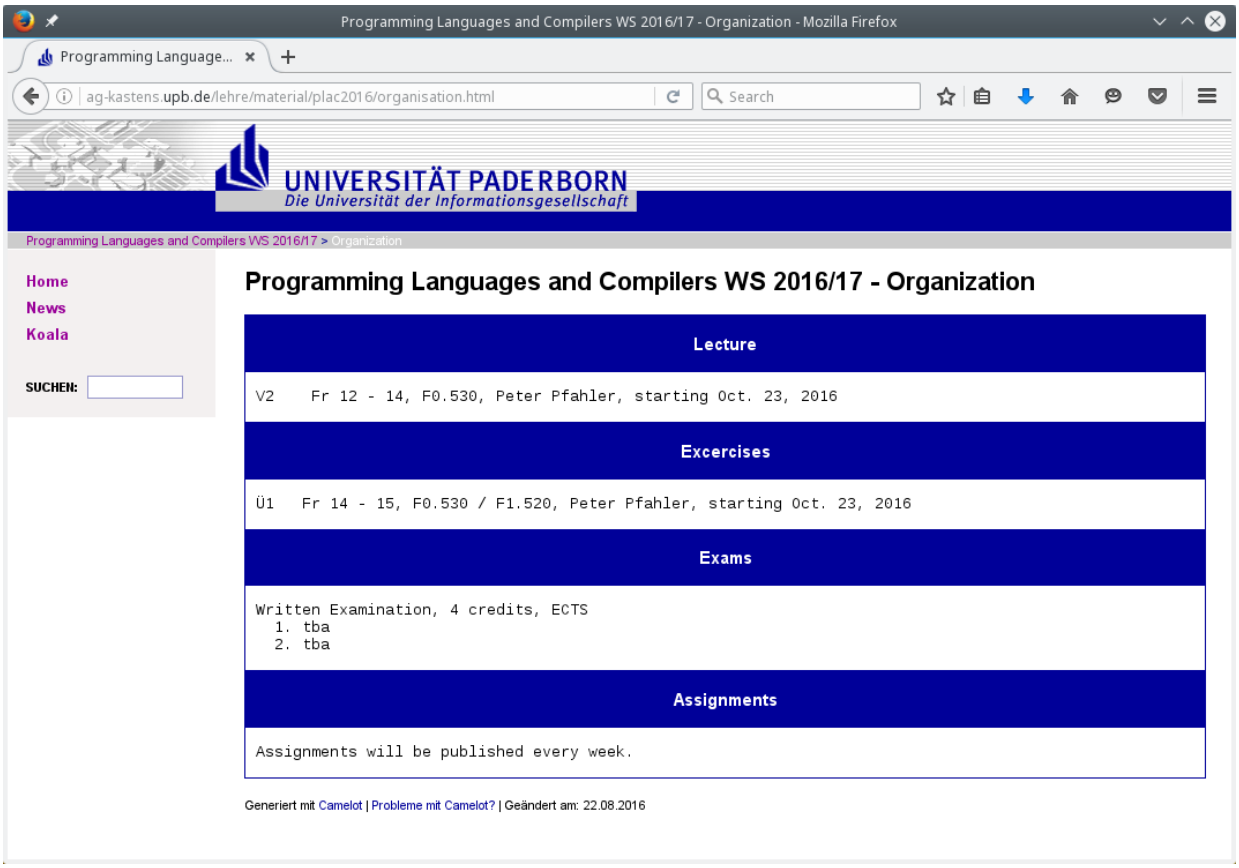
- A. W. Appel: Modern Compiler Implementation in Java
  Cambridge University Press, 2002
- A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman: Compilers: Principles, Techniques and Tools
  Pearson Education, 2006
- T. Mogensen: Basics of Compiler Design
  Free PDF book, University of Copenhagen 2010, `www.diku.dk/~torbenm/Basics`
- U. Kastens: Übersetzerbau, Handbuch der Informatik 3.3 (in German)
  Oldenbourg, 1990 (available in the library)
- W. M. Waite, L. R. Carter: An Introduction to Compiler Construction
  Harper Collins, New York, 1993
- U. Kastens, A. M. Sloane, W. M. Waite: Generating Software from Specifications
  Jones and Bartlett Publishers, 2007
- R.W. Sebesta: Concepts of Programming Languages
  Pearson Education, 2012

PLaC Web Site



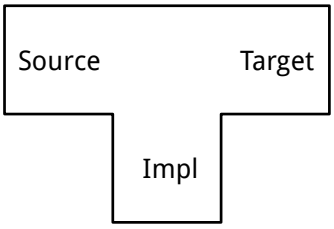
`http://ag-kastens.upb.de/lehre/material/plac2016/`

Organization of the Course



What does a Compiler Compile?

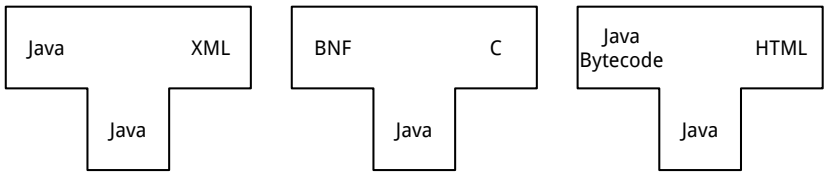
A compiler transforms correct sentences of its source language into sentences of its target language such that their meaning is unchanged.



T-Diagram of a Compiler: A source language text is translated to a target language text. The Compiler is written in the implementation language.
Example Languages: Java, C, C++, Haskell, Lisp, Prolog, JavaScript, PHP, PostScript, LaTeX, XML, HTML, SQL, BNF, x86, Java Bytecode, ...

Questions

- Create some reasonable T-Diagrams.
- Find some T-Diagrams that make no sense.
- Our C Compiler is written in C, how was it compiled?
- What are the following "compilers" good for?

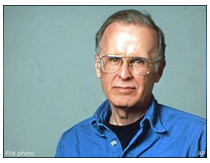


History of Compiler Construction

Research has led to a clear understanding of compiler construction, a large body of theory, and systematic compiler tools.

The first compiler was written in 1952 by Grace Hopper for the A-0 system language (*Arithmetic Language Version 0*).

The FORTRAN team led by John W. Backus at IBM introduced the first complete compiler in 1957. It took 18 person-years to create.



The first ALGOL 58 compiler was completed by the end of 1958 by Friedrich L. Bauer, Hermann Bottenbruch, Heinz Rutishauser, and Klaus Samelson for the Z22 computer (vacuum tube computer built by Konrad Zuse).

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
```

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
```


Languages for Specification and Modeling

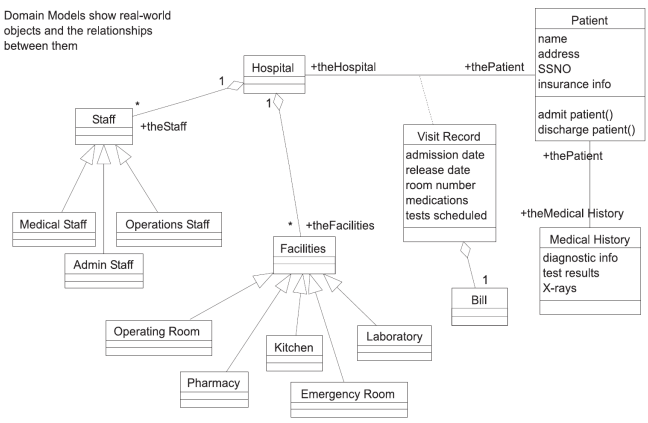
SDL (CCITT)
Specification and Description Language:

```
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;
```

UML
Unified Modeling Language:

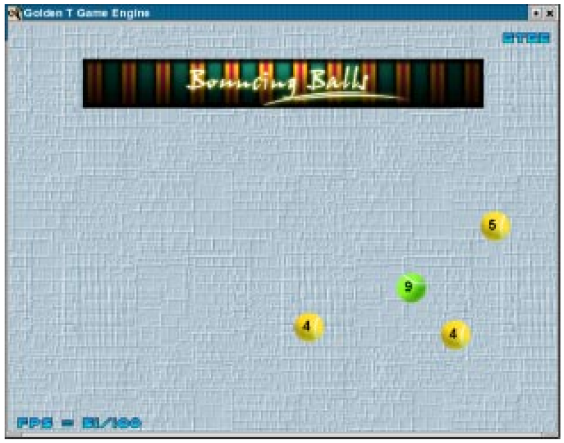


Domain Specific Languages (DSL)

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**
- **Game description language:**




```
game BBall
{  size 640 480;
   background "pics/backgroundbb.png";
   Ball einball; int ballsize;

   initial {
      ballsize=36;
   }

  events {
      pressed SPACE:
      { einball = new Ball(<100,540>, <100,380>);
```


Programming Languages as Source or Target Languages

Programming languages as source languages

- Software development
  compilers, interpreters
- Program analysis
  call graphs, control-flow graph, data dependencies
- Recognition of structures and patterns
  e. g. for reengineering

Programming languages as target languages

- Generating software from specifications
  - graphic modeling of structures (UML, SDL)
  - Domain specific languages, application generators
- Program transformations
- Language extensions
- Source-to-source compilation

Semester Project as Running Example

SetLan: A Language for Set Computations

SetLan is a simple imperative language for programming with sets. The language includes set types and variables and expressions of those types. Specific loop constructs allow to iterate through sets.
A source-to-source translator translates SetLan programs into Java programs.

The SetLan compiler is implemented using the methods and tools introduced in this course. The participants get an implementation of a sub-language of SetLan as a starting point. This language is extended during the practical exercises.

SetLan example

```
{ set a, b; int i;
  i = 1;
  a = [i, 3, 5];
  b = [3, 6, 8];
  print a+b; printLn;
  print a*b <= b;
  printLn;
}
```