



Compiler

编译器建设

Compilerbau

Michael Leuschel

John Witulski



Compiler

Lehrangebot

Softwaretechnik

Programmiersprachen

Compilerbau

5 LP

Von Nand zu Tetris

5 LP

Einführung in die
Logische
Programmierung
5 LP

Sicherheitskritische
Systeme
5 LP

Dynamische
Programmiersprachen
5 LP

Model Checking
5 LP

Funktionale
Programmierung
5 LP

Vertiefung
Logische
Programmierung
5 LP



Compiler

Organisation - Übungen

Klausurzulassung:

- Übungen und Blätter freiwillig
- Bearbeitung praktischer Übungen freiwillig
- Bearbeitung Compilerprojekt **Pflicht**

Übungen:

- Ab 29.10.2021 in 25.12.02.55* um 14:30
- Material auf ilias.hhu.de



Compiler

Organisation - Benotung

- Klausur 60%-75%
(Entscheidung im November)
- Compilerprojekt 25% - 40%
(Entscheidung im November)
- Bachelor Informatik 5 LP



Compiler

Organisation - Sonstiges

- Kontakt Übungen:

John.Witulski@hhu.de

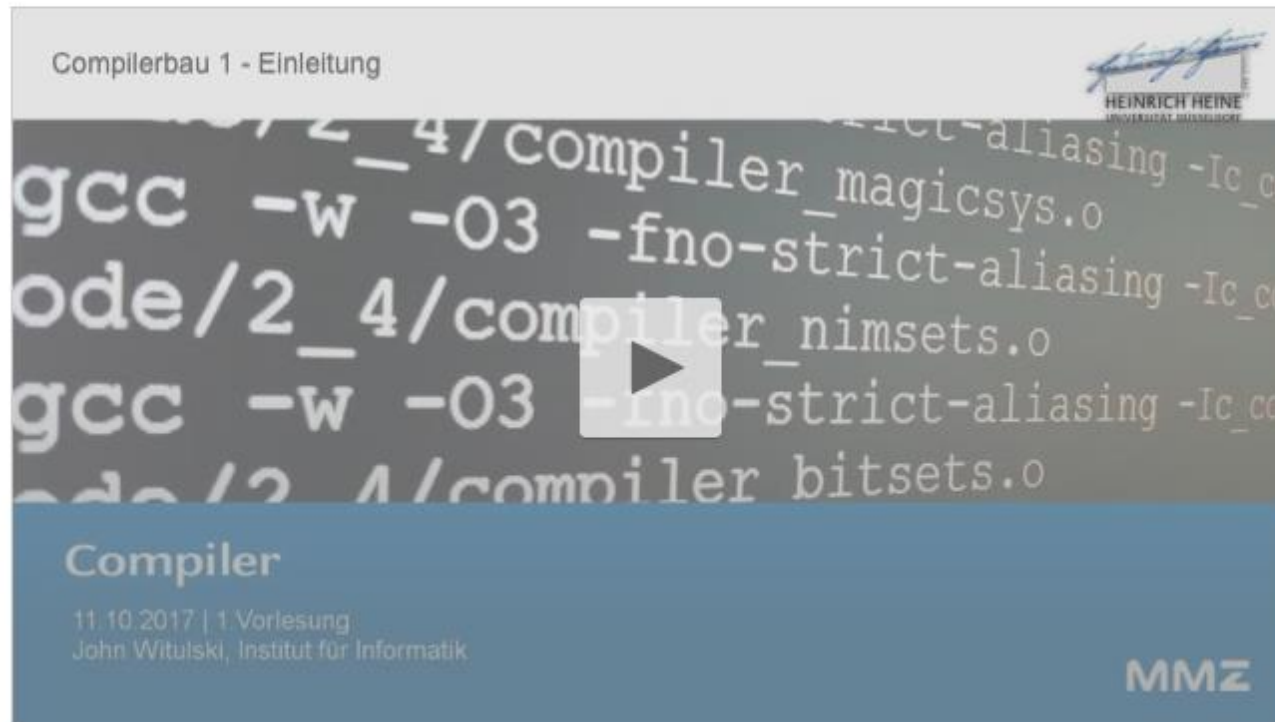
Termine Freitags nach Vereinbarung

- Videoaufzeichnung

Mediathek.hhu.de

Compilerbau 1 - Einleitung

John Witulski



Uploaded by [Witulski](#) at 10/25/2017 - recorded at 10/24/2017

Report

Information

Speaker:

John Witulski

Description:

Vorlesung Compilerbau im WS 2017/2028

Category:

Vorlesungen



Compilerbau



Compilerbau 2 - Lexing

John Witulski
10/18/2017
by Witulski



Compilerbau 3 - Parsing

John Witulski
10/26/2017
by Witulski



Compilerbau 4 - LL Parsi...

John Witulski
11/8/2017
by Witulski



Compilerbau 5 - LR Parsi...

John Witulski
11/15/2017
by Witulski



Compilerbau 6 - Semantik

John Witulski
11/22/2017
by Witulski



Compilerbau 8 - Typechec...

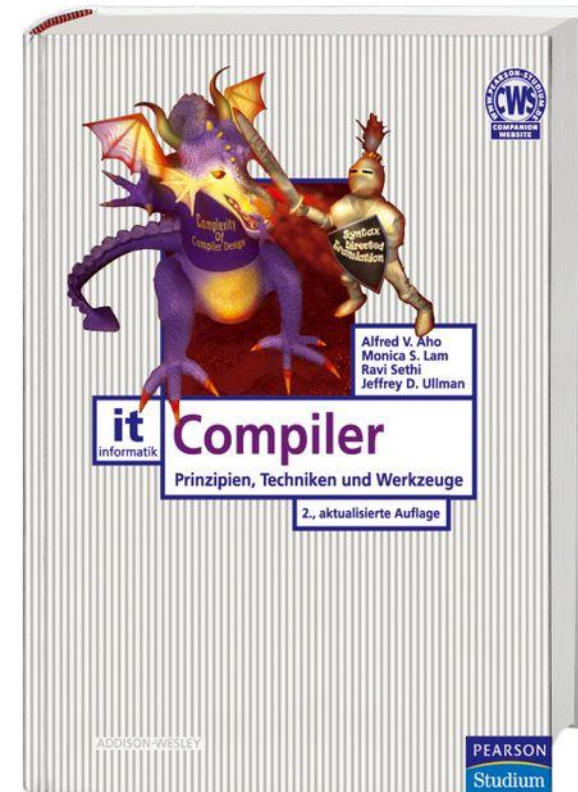
John Witulski
12/6/2017
by Witulski



Compiler

Bücher I

- Aho, Lam, Sethi, Ullman: Compiler, 2008, Pearson
- Errata:
 - <https://www.cs.hhu.de/lehrstuehle-und-arbeitsgruppen/softwaretechnik-und-programmiersprachen/unser-team/team/leuschel/errata.html>





Compiler

Bücher II



- Compilerbau:
 - Andrew W. Appel, *Modern Compiler Implementation in Java (2nd edition)*, Cambridge University Press, 2002.
 - <http://www.cambridge.org/us/catalogue/catalogue.asp?isbn=052182060x>
- Anderes Hintergrundmaterial:
 - Watt & Brown, *Programming Language Processors in Java*, Prentice-Hall, 2000.
 - Cooper, Torczon, *Engineering a Compiler*, Elsevier, 2004.



Compiler

Einleitung

Compiler Kapitel 1

Einleitung

Programmiersprachen

编程语言



- Maschinen- und Assemblersprachen
- 机器和汇编语言
- Höhere Programmiersprachen
- 高级编程语言
 - übernehmen Speicherverwaltung, Typkonsistenz, parallele Ausführung,...接管内存管理、类型一致性、并行执行
 - höhere Produktivität, weniger Fehler提高生产力，减少错误
 - gute Übersetzung wichtig um Hardware effektiv zu nutzen良好的翻译对有效使用硬件很重要



Compiler

Maschinensprache

0014360	105510	137400	000003	000000	110377	000240	000000	044215
0014400	104402	130415	075410	101400	001700	002611	004260	000173
0014420	002611	004262	000173	002707	004260	000173	000000	000000
0014440	105511	044007	000213	002277	000000	177400	060220	000006
0014460	044000	002611	004240	000173	102510	072700	044423	003613
0014500	105510	030400	137377	177777	177777	110377	005640	000000
0014520	002613	004142	000173	106504	177140	144377	104510	134105
0014540	104504	146145	104514	140155	102515	072755	165424	110061
0014560	104504	042366	171211	110377	001440	000000	102515	072355
0014600	044437	042613	044400	042473	071410	044065	044215	044401

B1 80

F0 11

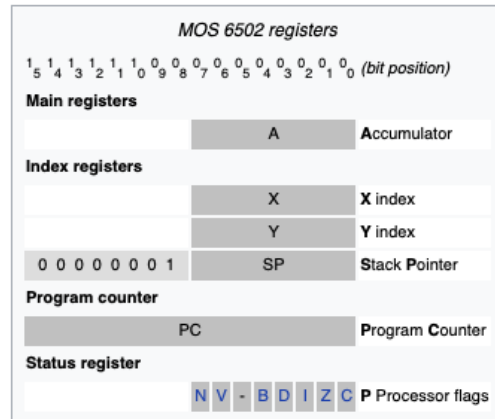
C9 41

90 06



Compiler

Assembler



```

LOOP  LDA (SRC),Y ;get from source string
      BEQ DONE   ;end of string ;
      CMP #'A'    ;if lower than UC alphabet...
      BCC SKIP    ;copy unchanged

      ...
SKIP   ...

      ...
DONE   ...
  
```

B1 80

F0 11

C9 41

90 06

https://commons.wikimedia.org/wiki/File:Apple_iieb.jpg
https://en.wikipedia.org/wiki/MOS_Technology_6502
 CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=91538>



Compiler

Höhere Programmiersprachen

```
LOOP  LDA (SRC),Y ;get from source string
      BEQ DONE  ;end of string ;
      CMP #'A'   ;if lower than UC alphabet...
      BCC SKIP   ;copy unchanged
...
SKIP   ...
...
DONE   ...
```

Was bekommt man im Vergleich zu Assembler?
Was verliert man?

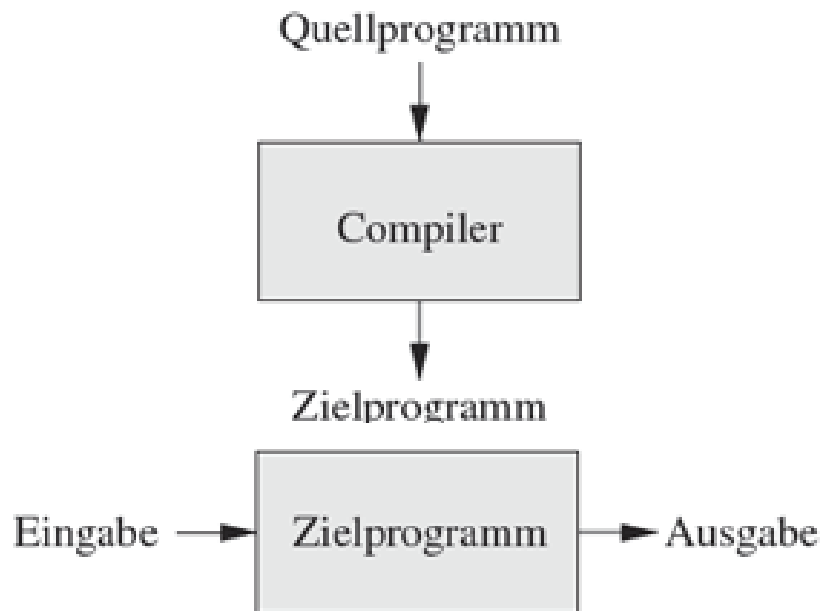


Höhere Programmiersprachen

- Ausdrücke ($2 \cdot x + 1$)
- Datentypen
- Kontrollstrukturen (while,...)
- Deklarationen
- Abstraktion (Methoden mit Namen und Parametern,...)
- Kapselung, Sichtbarkeitsregeln



Ausführung eines Programms 方案的执行





Vorteile eines Compilers

编译器的优点

- (viel) schnellere Ausführung
- 执行速度快得多
- **Gesamter Code wird überprüft**
整个代码被检查
- **Quellcode/Interpreter muss nicht ausgeliefert werden**
不需要交付源代码/解释器



Vorteile eines Interpreters

解释器 的优点

- **Keine Kompilationszeit** 无编译时间
- **Bessere Fehlerdiagnose** 更好的错误诊断 (Debugging)
- **Dynamische Sprachkonstrukte**: 动态语言结构。
 - Introspektion/Reflection 自省/反思
 - Modifikation des Programms zur Laufzeit möglich 在运行时可以对程序进行修改



Compiler

Hybridcompiler混合编译器

Einleitung

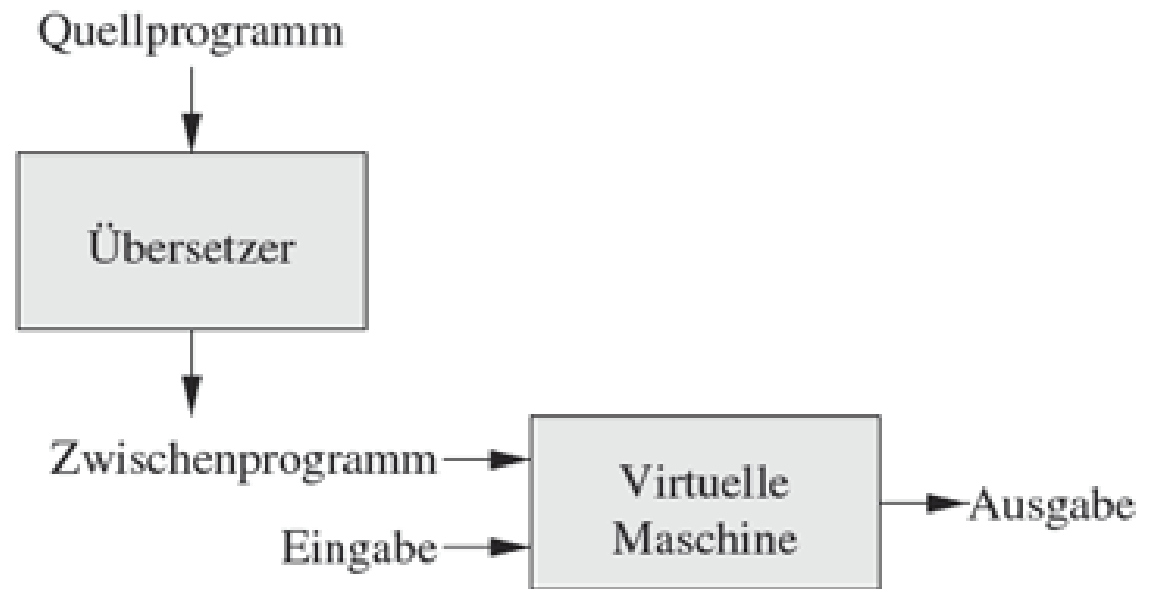


Abbildung 1.4: Ein Hybridcompiler

manchmal:
JIT Compiler



Vorteile eines Hybrid-Compilers 混合编译器的优势

- **kompakter Zwischenkode** 紧凑的中间代码
- **gegenüber Compiler:** 与编译器相比
 - Plattformunabhängig, bessere Fehlerdiagnose, dynamische Konstrukte bedingt möglich 独立于平台，更好的错误诊断，有条件的动态构造
- **gegenüber Interpreter**
 - **Gesamter Code überprüft, schneller** 与解释者相比
 - - 检查整个代码，速度更快



Compiler im Kontext

背景下的编译器

Compiler

Einleitung

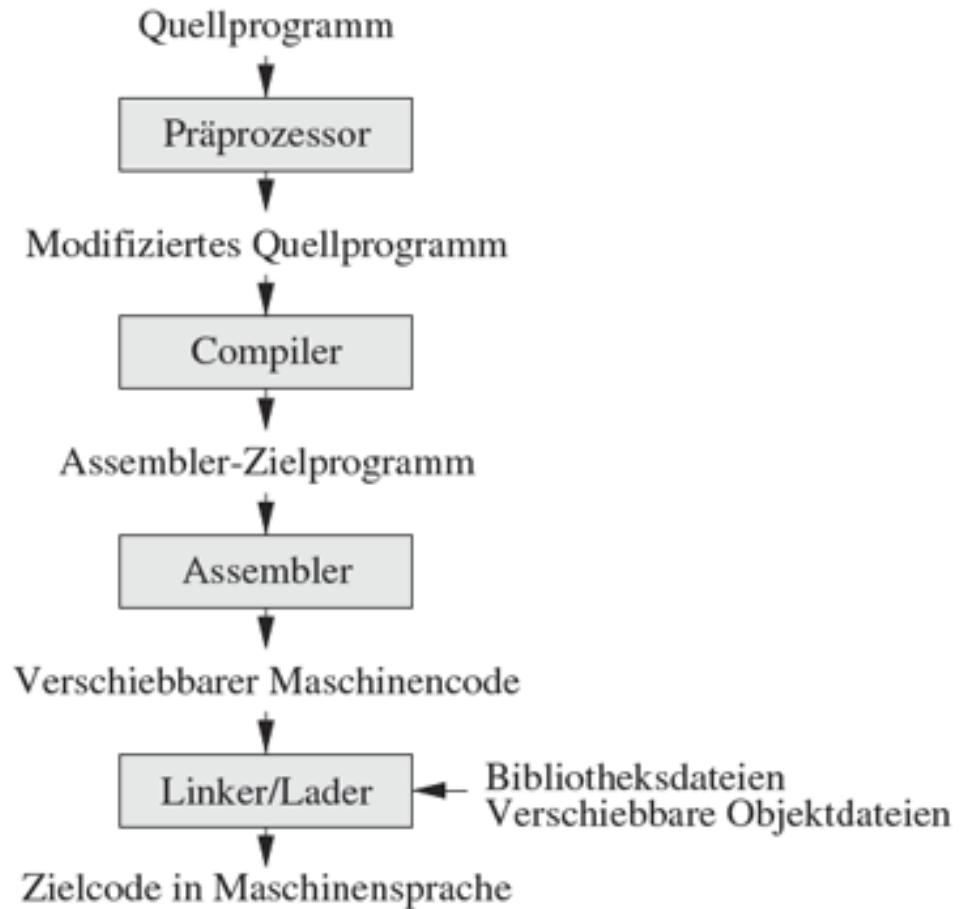


Abbildung 1.5: Ein Sprachverarbeitungssystem



Einige bekannte Compiler一些著名的编译器

Compiler



<https://gcc.gnu.org/>

Einleitung



<http://clang.llvm.org> <http://llvm.org>

javac

Java Hotspot



Microsoft Visual C++



<https://code.google.com/p/v8/>



Warum Compiler studieren ?

为什么要学习编译器？

- **Ein Standardwerkzeug eines Informatikers** 计算机科学家 **Compiler** 的一个标准工具
- **Domain specific languages (DSLs)** 特定领域语言 **Einleitung**
 - Vielleicht müssen Sie einen Compiler schreiben 您可能需要编写一个编译器
- **Studium eines mittelgroßen Softwaresystems** 对一个中等规模的软件系统的研究
- **Interessante Theorie und Algorithmen**
 - Nützlich für viele andere Anwendungen 有趣的理论和算法
 - - 适用于许多其他应用



Interessante Problemen und Lösungen 有趣的问题和解决方案

- **Compiler** Graphalgorithmen (Dead-Code Elimination) 图形算法（消除死代码）
- Suchalgorithmen, Greedy Search 搜索算法，贪婪的搜索 (Registerallokation,...)
- Dynamische Programmierung 动态编程 (Instruction Selection)
- Automaten (Lexing) 自动机
- Grammatiken (Parsing) 语法 (解析)
- Fixpunktalgorithmen (Datenflussanalyse) 固定点算法（数据流分析）



Should I learn compilers?



Compiler

27

Einleitung



If you just want to be a run-of-the-mill coder, and write stuff... you don't need to take compilers.

If you want to learn computer science and appreciate and really become a computer scientist, you *MUST* take compilers.

Compilers is a microcosm of computer science! It contains every single problem, including (but not limited to) AI (greedy algorithms & heuristic search), algorithms, theory (formal languages, automata), systems, architecture, etc.

You get to see a lot of computer science come together in an amazing way. Not only will you understand more about why programming languages work the way that they do, but you will become a better coder for having that understanding. You will learn to understand the low level, which helps at the high level.

As programmers, we very often like to talk about things being a "black box"... but things are a lot smoother when you understand a little bit about what's in the box. Even if you don't build a whole compiler, you will surely learn a lot. You will get to see the formalisms behind parsing (and realize it's not just a bunch of special cases hacked together), and a bunch of NP complete problems. You will see why the theory of computer science is so important to understand for practical things. (After all, compilers are extremely practical... and we wouldn't have the compilers we have today without formalisms).

I really hope you consider learning about them... it will help you get to the next level as a computer scientist :-).

share improve this answer

answered Apr 9 '09 at 7:31

<http://stackoverflow.com/questions/733093/when-should-i-learn-compilers>



Tom

8,455 ● 3 ● 26 ● 39

Compilerbau: 编译器建设。

一项复杂的工作

Ein komplexes Unterfangen



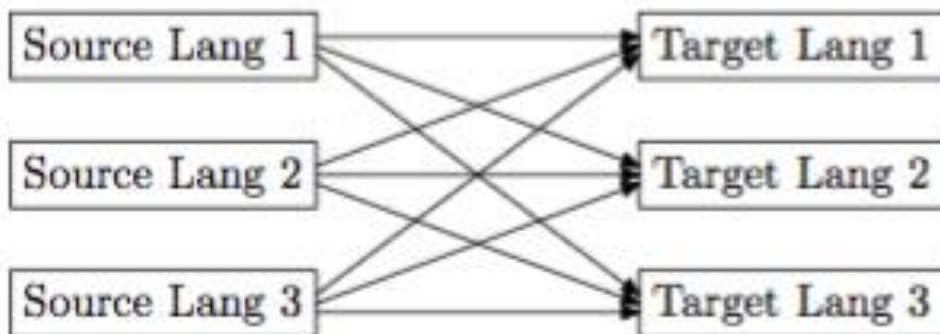
Compiler

```
position = initial + rate*60
```

?

```
LDF  R2,  id3
MULF R2,  R2, #60.0
LDF  R1,  id2
ADDF R1,  R1, R2
STF  id1, R1
```

Einleitung





Compiler

Beispiel

```
#define CUBE(x) (x)*(x)*(x)
int main() {
    int i = 0;
    int x = 2;
    int sum = 0;
    while (i++ < 100) {
        sum += CUBE(x);
    }
    printf("The sum is %d\n", sum);
}
```

https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Compiled_Programs



Compiler

```
#define CUBE(x) (x)*(x)*(x)
int main() {
    int i = 0;
    int x = 2;
    int sum = 0;
    while (i++ < 100) {
        sum += CUBE(x);
    }
    printf("The sum is %d\n", sum);
}
```

# Assembly of x86	# Assembly of ARM
.cstring	_main:
LC0:	@ BB#0:
.ascii "The sum is %d\12\0"	push {r7, lr}
.text	mov r7, sp
.globl _main	sub sp, sp, #16
_main:	mov r1, #2
pushl %ebp	mov r0, #0
movl %esp, %ebp	str r0, [r7, #-4]
subl \$40, %esp	str r0, [sp, #8]
movl \$0, -20(%ebp)	stm sp, {r0, r1}
movl \$2, -16(%ebp)	b LBB0_2
movl \$0, -12(%ebp)	LBB0_1:
jmp L2	ldr r0, [sp, #4]
L3:	r3, [sp]
movl -	r0, r0
imull -	r1, r0, r3
imull -	[sp]
addl %	r0, [sp, #8]
L2:	r1, r0, #1
cmpl \$	#99
setle %	r1, [sp, #8]
addl \$	LBB0_1
testb %	LCPI0_0
jne L	[sp]
movl -	r0, pc, r0
movl %	_printf
movl \$	r0, [r7, #-4]
movl \$	sp, r7
call -	{r7, lr}
leave	mov
ret	pc, lr

```
.cstring
LC0:
.ascii "The sum is %d\12\0"
.text
.globl _main
_main:
    pushl %ebp
    movl %esp, %ebp
    subl $24, %esp
    movl $800, 4(%esp)
    movl $LC0, (%esp)
    call _printf
    leave
    ret
```



Compiler

Simple1.java

Einleitung

```
public class Simple1 {  
    public static void main(String args[])  
    {  
        int x = 3;  
        int y = 5;  
        int res = x + x*y + 2;  
        System.out.print("Result: ");  
        System.out.println(res);  
    }  
}
```



Compiler

Tokenized Simple1.java 符号化

Einleitung

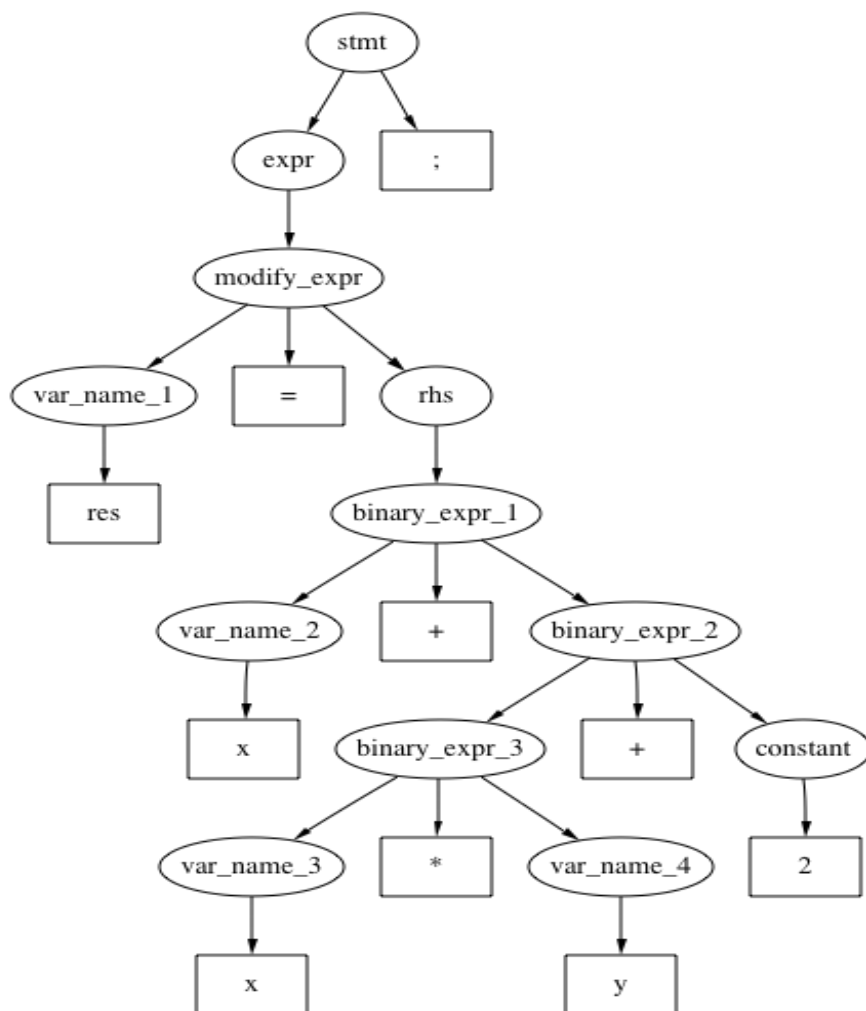
LINE 2	SEMICOLON	IDENTIFIER: out
PUBLIC_SYMBOL	LINE 7	DOT
CLASS_SYMBOL	INT_SYMBOL	IDENTIFIER: print
IDENTIFIER: Simple1	IDENTIFIER: y	LEFT_PARENTHESIS
LEFT_BRACE	EQUALS	STRINGCONST: "Result:"
LINE 4	INTCONST: 5	"
PUBLIC_SYMBOL	SEMICOLON	RIGHT_PARENTHESIS
STATIC_SYMBOL	LINE 8	SEMICOLON
VOID_SYMBOL	INT_SYMBOL	LINE 10
IDENTIFIER: main	IDENTIFIER: res	IDENTIFIER: System
LEFT_PARENTHESIS	EQUALS	DOT
IDENTIFIER: String	IDENTIFIER: x	IDENTIFIER: out
IDENTIFIER: args	PLUS	DOT
LEFT_BRACKET	IDENTIFIER: x	IDENTIFIER: println
RIGHT_BRACKET	STAR	LEFT_PARENTHESIS
RIGHT_PARENTHESIS	IDENTIFIER: y	IDENTIFIER: res
LEFT_BRACE	PLUS	RIGHT_PARENTHESIS
LINE 6	INTCONST: 2	SEMICOLON
INT_SYMBOL	SEMICOLON	LINE 11
IDENTIFIER: x	LINE 9	RIGHT_BRACE
EQUALS	IDENTIFIER: System	LINE 12
INTCONST: 3	DOT	RIGHT_BRACE



Compiler

Einleitung

Syntax tree for Simple1.java (part) 语法树



Hinweis: “+” ist hier rechts-assoziativ

(normalerweise ist es links-assoziativ!)

注: “+”在这里是右联的。
(通常情况下, 它是左联的!)



javap -c Simple1

Bytecode

```
public class Simple1 extends java.lang.Object{  
    public Simple1();
```

Code:

```
    0:    aload_0  
    1:    invokespecial    #1; //Method java/lang/Object."<init>":()V  
    4:    return
```

```
    public static void main(java.lang.String[]);
```

Code:

```
    0:    iconst_3  
    1:    istore_1  
    2:    iconst_5  
    3:    istore_2  
    4:    iload_1  
    5:    iload_1  
    6:    iload_2  
    7:    imul  
    8:    iadd  
    9:    iconst_2  
   10:    iadd  
   11:    istore_3  
   12:    getstatic        #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
   15:    ldc              #3; //String Result:  
   17:    invokevirtual     #4; //Method  
java/io/PrintStream.print:(Ljava/lang/String;)V  
   20:    getstatic        #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
   23:    iload_3  
   24:    invokevirtual     #5; //Method java/io/PrintStream.println:(I)V  
   27:    return  
  
}
```

Compiler

Einleitung



Compiler Architektur 编译器结构

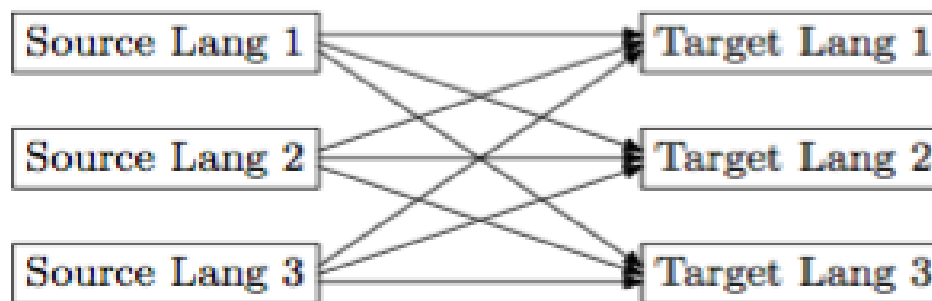
Compiler

```
position = initial + rate*60
```

?

```
LDF  R2,  id3
MULF R2,  R2, #60.0
LDF  R1,  id2
ADDF R1,  R1, R2
STF  id1, R1
```

Einleitung





Compiler

Einleitung

前端(分析)Front End
(Analyse)

Symboltabelle

Back End
(Synthese)

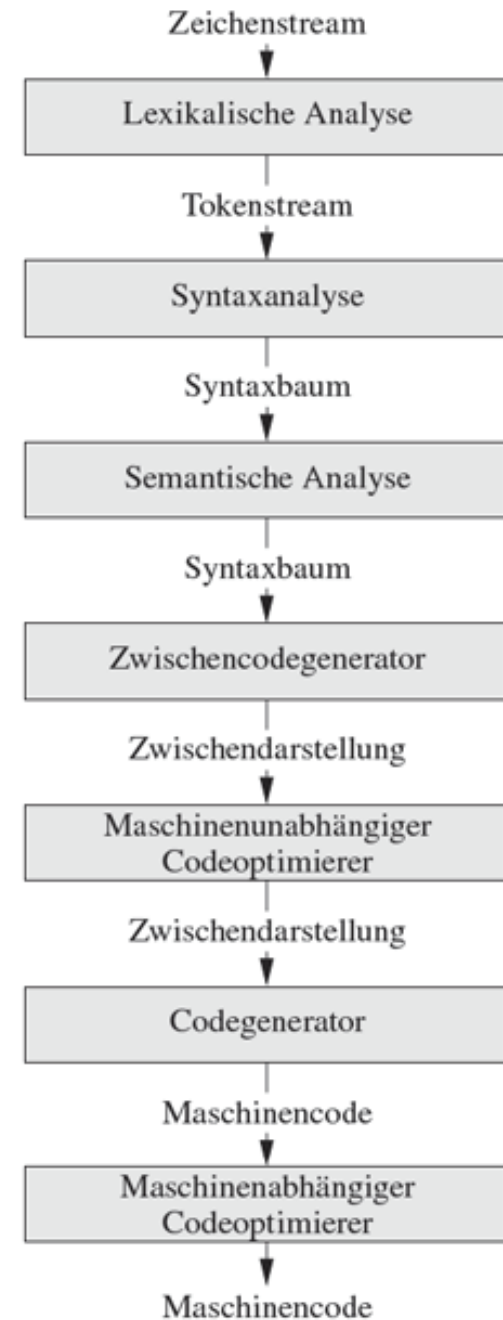


Abbildung 1.6: Phasen eines Compilers



Beispiel

- Übersetzung von:

$$\text{position} = \text{initial} + \text{rate} * 60$$

Compiler

Einleitung



Compiler

Einleitung

1	position	...
2	initial	...
3	rate	...

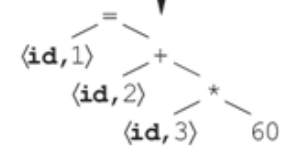
SYMBOLTABELLE

position = initial + rate * 60

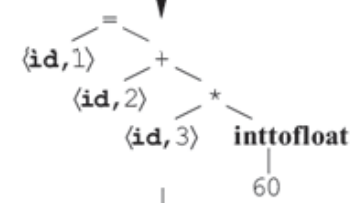
Lexikalischer Analysator

<id,1> <=> <id,2> <+> <id,3> <*> <60>

Syntaktischer Analysator



Semantischer Analysator



Zwischengenerator

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Codeoptimierer

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Codegenerator

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```

Abbildung 1.7: Übersetzung einer Zuweisungsanweisung

Programmiersprachen: Einige Konzepte

(1.6) 编程语言。一些概念



Compiler

Einleitung

- **Statisch**: zur Kompilierzeit 静态 编译阶段
- **Dynamisch**: zur Laufzeit 时间
- **Gültigkeitsbereich** (scope) eines Namens
 - **Statisch** (lexikalisch) oder dynamisch
 - 名称的范围
 - - 静态（词法）或动态
- **Speicherort** eines Namens 名称的存储位置
 - Statisch oder dynamisch bestimmbar
 - **Umgebung** verknüpft Namen mit Speicherort
- **Wert** eines Speicherorts
 - Statisch oder **dynamisch**



Abbildung 1.8: Abbildung von Namen auf Werte in zwei Stufen



Compiler

Gültigkeitsbereich : Blockstruktur (1.6)范围：块 状结构

Einleitung

- Programmiersprachen mit Blockstruktur:
 - erlauben Verschachtelung von Blöcken
具有块状结构的编程语言。
 - - 允许区块嵌套

```
...  
int i;          /* globales i */  
...  
void f(...) {  
    int i;      /* lokales i */  
    ...  
    i = 3;      /* Verwendung des lokalen i */  
    ...  
}  
...  
x = i + 1;     /* Verwendung des globalen i */
```

Abbildung 1.9: Zwei Deklarationen des Namens *i*



Programmiersprachen: Einige Konzepte II

• **Bezeichner**识别器

- Zeichenstring der auf eine Entität verweist (zB Datenobjekt, Prozedur, Klasse oder Type)指向一个实体（如数据对象、程序、类或类型）的字符串。
- Bezeichner sind Namen (Namen sind nicht immer Bezeichner, zB x.y)标识符是名称
- **Variable** eines Speicherorts- 储存地点的变量
 - Verweist auf einen bestimmten Speicherort
 - 指的是一个特定的内存位置



Programmiersprachen: Einige Konzepte III

Compiler

Einleitung

- **Deklaration** 声明
 - Geben Typ an 指定类型
 - `int i`
- **Definition** 定义
 - Geben Wert an 指定值
 - `i = 7`



Gültigkeitsbereich 有效性范围

- **Compiler** Der **Gültigkeitsbereich (scope)** einer **Deklaration** von x ist der Kontext, in dem Verwendungen von x auf diese Deklaration verweisen. Eine Sprache verwendet einen **statischen** oder **lexikalischen Gültigkeitsbereich**, wenn der Gültigkeitsbereich einer Deklaration aus dem Programmtext abzulesen ist. **Einleitung** Anderenfalls nutzt die Sprache einen **dynamischen Gültigkeitsbereich**
- x 的声明的范围是指 x 的使用指向这个声明的上下文。如果声明的范围可以从程序文本中读出，那么一种语言就会使用静态或词法范围。否则，该语言会使用一个动态的有效性范围
- Lebensdauer:
 - muss mindestens den Gültigkeitsbereich abdecken 必须至少涵盖范围
 - - 但可以更长（例如静态变量）。
 - kann aber länger sein (zB static Variablen)



Statischer Gültigkeitsbereich Blockstruktur

静态范围 块状结构

```
main() {  
    int a = 1;  $B_1$   
    int b = 1;  
    {  
        int b = 2;  $B_2$   
        {  
            int a = 3;  $B_3$   
            cout << a << b;  
        }  
        {  
            int b = 4;  $B_4$   
            cout << a << b;  
        }  
        cout << a << b;  
    }  
    cout << a << b;  
}
```

Abbildung 1.10: Blöcke in einem C++-Programm

Deklaration	Gültigkeitsbereich
int a = 1;	$B_1 - B_3$
int b = 1;	$B_1 - B_2$
int b = 2;	$B_2 - B_4$
int a = 3;	B_3
int b = 4;	B_4

Compiler

Einleitung



Dynamischer Gültigkeitsbereich

Compiler

Einleitung

- **Objekt-Orientierte Programmierung:**
Vererbung 面向对象的编程：继承性
 - Klasse C mit methode m()
 - Unterklasse D von C mit eigenem m()
 - Objekt x der Klasse C, Aufruf x.m()
- **Dynamische Programmiersprachen**
 - Tcl, Python, Ruby 动态编程语言



Übung

Compiler

Einleitung

w=13
x=11
y=13
z=11

```
int w, x, y, z;  
int i = 4; int j = 5;  
{  
    int j = 7;  
    i = 6;  
    w = i + j;  
}  
x = i + j;  
{  
    int i = 8;  
    y = i + j;  
}  
z = i + j;
```

a Code für Übung 1.6.1

```
int w, x, y, z;  
int i = 3; int j = 4;  
{  
    int i = 5;  
    w = i + j;  
}  
x = i + j;  
{  
    int j = 6;  
    i = 7;  
    y = i + j;  
}  
z = i + j;
```

b Code für Übung 1.6.2

w=9
x=7
y=13
z=11

Abbildung 1.13: Code in Blockstruktur

Werte für w,x,y,z ?



Übung

Compiler

int w: B1 - B3 - B4

x: B1-B2-B4

y: B1-B5

z: B1-B2-B5

Einleitung

int x: B2-B3

z: B2

int w,x: B3

int w,x: B4

int y,z: B5

```
{  int w, x, y, z;      /* Block B1 */
  {  int x, z;          /* Block B2 */
    {  int w, x;        /* Block B3 */    }
  }
  {  int w, x;          /* Block B4 */
    {  int y, z;        /* Block B5 */    }
  }
}
```

Abbildung 1.14: Code in Blockstruktur für Übung 1.6.3

Gültigkeitsbereiche der zwölf Deklarationen?



Compiler

Einleitung

Andere Themen in Drachenbuch 1.6 龙之书1.6的其他主题

- Zugriffskontrolle
 - public
 - private
 - protected
- Parameterübergabe
 - call by value
 - call by reference
 - call by name
 - Aliasing
 - Selber durchlesen

Zusammenfassung

- Compiler, Interpreter, Hybridcompiler

编译器, 解释器, 混合编译器

- Compilerphasen 编译阶段

- Compiler: Abfolge von Phasen 阶段的顺序

- Trennung: Back-End, Front-End 分离: 后端、前端

- Theorie und Praxis: Modellierung im Compilerdesign (Automaten, Grammatiken, reguläre Ausdrücke, Bäume) 理论与实践: 编译器设计中的建模

- Einige Programmiersprachenkonzepte

- Statisch/dynamisch, “scope”, ...
 - 一些编程语言的概念



Compiler

Einleitung



Compiler

Rest vom Kurs

- Lexing
- Parsing
- Semantische Analyse
- Zwischencode
- Codegenerierung

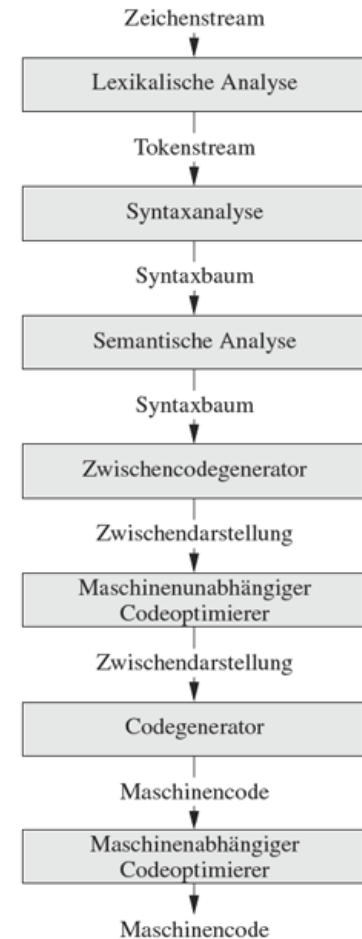
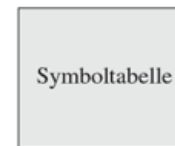


Abbildung 1.6: Phasen eines Compilers