Programmiersprachen – Grundlagen und Konzepte

Basisinformationstechnologie II – Sommersemester 2022 – 22.04.2022

Dr. Jan Wieners

Programmiersprachen und ihre Unterschiede

Differenzierung anhand von C++ und JavaScript:

- Compiler vs. Interpreter
- Lexik, Syntax, Semantik, Pragmatik
- Paradigmen
- Typisierung: Dynamisch vs. statisch
- Variablen: Deklaration vs. Initialisierung
- Hardwarenahe Programmierung: C++ und Zeiger
- Gemeinsamkeiten: Auswahlanweisungen und Kontrollstrukturen
- (Objektorientierung: Klassen, Geheimnisprinzip, Kapselung)

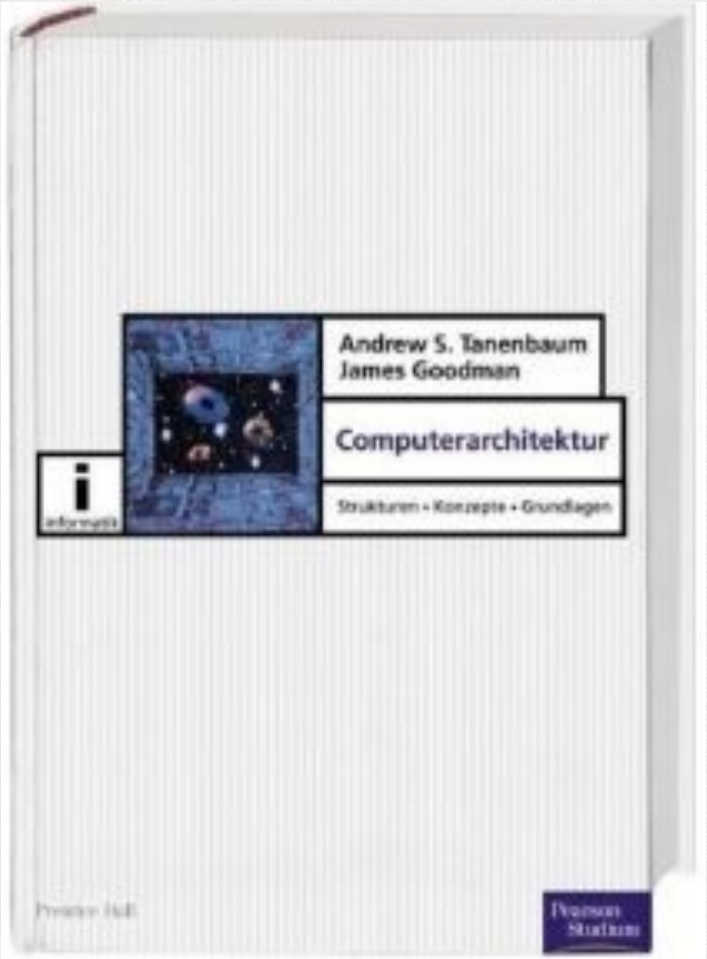
WS15/16: Programmiersprachen und ihre Unterschiede

Differenzierung anhand von C++ und JavaScript:

- Compiler vs. Interpreter
- Typisierung !: Dynamisch vs. statisch
- Variablen: Deklaration vs. Initialisierung
- Paradigmen: funktionale vs. Objektorientierte Programmierung
- Objektorientierung: Klassen, Kapselung, Geheimnisprinzip
- Hardwarenahe Programmierung: C++ und Zeiger
- Gemeinsamkeiten: Auswahlanweisungen und Kontrollstrukturen
- Schichten-Architektur: MVC

Wildschweine
Bitte die Tür
Schließen

Strukturierte Computerorganisation



Ebene 5 — Problemorientierte Sprache

Ebene 4 — Assemblersprache

Ebene 3 — Betriebssystemmaschine

Ebene 2 — Befehlssatzarchitektur (ISA)

Ebene 1 — Mikroarchitektur

Ebene 0 — Digitale Logik

Assembler

Beispiel: „Hello World“ :

```
DATA SEGMENT                    ;- Beginn des Datensegments
Meldung db "Hello World"        ;- Die Zeichenkette "Hello World"
        db "$"                  ;- Endzeichen der Zeichenkette
DATA ENDS                       ;- Ende des Datensegment
CODE SEGMENT                    ;- Beginn des Codesegements
ASSUME CS:CODE,DS:DATA          ;- Dem Assembler die Segmente mitteilen
Anfang:                         ;- Label für den Anfang des Programms
mov ax, DATA                    ;- das Daten...
mov ds, ax                      ;  ...segment festlegen
mov dx, offset Meldung          ;- den Text in das auf DS bezogene Datenregister
   laden
mov ah, 09h                     ;- Die Unterfunktion 9 des Betriebssysteminterrupts
   21h auswählen
int 21h                         ;- den Betriebssysteminterrupt 21h (hier erfolgt
                                      Ausgabe des Texts) aufrufen
mov ax, 4C00h                   ;- Die Unterfunktion 4Ch (Programmbeendigung) des
                                      Betriebssysteminterrupts 21h festlegen
int 21h                         ;- diesen Befehl wiederum ausführen
CODE ENDS                       ;- Ende des Codesegments
END Anfang                      ;- dem Assembler das Ende des Labels Anfang mitteilen
```

Vgl.: http://de.wikipedia.org/wiki/Assemblersprache

C++

```
std::cout << “Hello World“;
```

Programmiersprachen

- Programmiersprache: Eine zum Formulieren von **Programmen** geschaffene **künstliche Sprache**
- Anweisungen, die wir dem Computer geben, werden als Text formuliert, z.B. in Python:

```
print "Hello World!"
```

- In C++:

```
[…]
cout << „Hello World“;
[…]
```

Programmiersprachen

- Programmtext ist formuliert nach festen Regeln: Festgelegt durch die **Grammatik** einer Programmiersprache
- Beispiel C++:

```
cout << „Hello World";
```

  - Grammatik schreibt u.a. vor, dass der Ausdruck `cout << „Hello World"` mit einem Semikolon abgeschlossen werden muss

Programmiersprachen

Wer prüft zu welchem Zeitpunkt die Grammatik?

- Programme, die in einer höheren Programmiersprache formuliert sind, können nicht unmittelbar von einem Rechner ausgeführt werden, sondern müssen erst in eine Folge von Maschinenbefehlen übersetzt werden

- Maschinenbefehle == elementare Operationen, z.B.:
  - Daten aus dem Speicher lesen
  - Elementare arithmetische Operationen ausführen
  - Daten in den Speicher schreiben
  - Sprünge: Berechnungen an bestimmter Stelle fortsetzen

Programmiersprachen

Compiler: Computerprogramm, das ein in einer Hochsprache (C++, etc.) formuliertes Programm, das sog. **Quellprogramm** in ein **Zielprogramm**, z.B.:

- Bytecode (Sammlung von Befehlen für eine virtuelle Maschine)
  oder
- Maschinencode (Instruktionen, die der entsprechende Prozessor (Hardware) direkt umsetzen kann)

übersetzt

„kompilieren“: Verb, Anwendung eines Compilers auf ein Quellprogramm

Compiler

Kompilierung zumeist in mehreren Phasen:

- **Analysephase**
  - **Lexikalische** Analyse: Unterteilung des Quelltextes in bedeutungstragende Elemente, z.B. Schlüsselwörter, Zahlen, Operatoren
    → Dieser Teil der Kompilierung wird als **Scanner** oder **Lexer** bezeichnet
  - **Syntaktische** Analyse: Überprüft, ob der Quellcode der Syntax entspricht, d.h. ein wohldefinierter Satz der entsprechenden Programmiersprache ist
    → **Parser**
  - **Semantische** Analyse: Überprüft weiterführende Anforderungen; z.B. muss in einigen Programmiersprachen gewährleistet sein, dass jeder Bezeichner vor seiner Benutzung deklariert worden ist
- **Synthesephase**
  - Zwischencodeerzeugung
  - Programmoptimierung
  - Codegenerierung

Compiler vs. Interpreter

- Ein **Compiler** übersetzt immer einen kompletten Programmtext in eine Folge von Maschinenbefehlen, bevor die erste Programmanweisung ausgeführt wird.
- Ein **Interpreter** dagegen übersetzt immer nur **eine einzige Programmanweisung** in ein kleines **Unterprogramm aus Maschinenbefehlen** und führt dieses sofort aus. Anschließend wird mit der nächsten Anweisung genauso verfahren.

- **Interpreter Pro**: Einfacher zu konstruieren als Compiler
- **Interpreter Contra**: Ein Befehl, der mehrfach ausgeführt wird, muss jedes mal erneut übersetzt werden

Programmiersprachen: Das muss sein

Bei der Definition einer Programmiersprache muss ihre **Lexik, Syntax, Semantik** und **Pragmatik** definiert werden:

- **Lexik:** Definiert die gültigen Zeichen und Wörter, aus denen Programme der Programmiersprache zusammengesetzt sein dürfen.
- **Syntax:** Definiert den korrekten Aufbau der Sätze aus gültigen Zeichen bzw. Wörtern, d.h. sie legt fest, in welcher Reihenfolge lexikalisch korrekte Zeichen bzw. Wörter im Programm auftreten dürfen.
- **Semantik:** Definiert die Bedeutung syntaktisch korrekter Sätze, d.h. sie beschreibt, was passiert, wenn bspw. bestimmte Anweisungen ausgeführt werden.
- **Pragmatik:** Definiert ihren Einsatzbereich, d.h. sie gibt an, für welche Arten von Problemen die Programmiersprache besonders gut geeignet ist.

Quelle: Boles, Dietrich (2015): Programmieren spielend gelernt mit dem Java-Hamster-Modell. http://www.java-hamster-modell.de/eBooks/hamster1.pdf.

Programmiersprachen: Klassifizierung

Grobunterscheidung:

- niedere (maschinennahe) Programmiersprachen
- Höhere Programmiersprachen

Programmiersprachen: Klassifizierung

Höhere Programmiersprachen:

- Weitaus einfacher zu handhaben als z.B. Assembler
- Problemorientiert

Unterscheidung höherer Programmiersprachen in fünf Kategorien / Programmierparadigmen:

- **Imperative** Programmiersprachen:
  Programme bestehen aus Folgen von Befehlen (BASIC, PASCAL, MODULA-2).
- **Funktionale** Programmiersprachen:
  Programme werden als mathematische Funktionen betrachtet (LISP, MIRANDA).
- **Prädikative/deklarative** Programmiersprachen:
  Programme bestehen aus Fakten und Regeln, die beschreiben, wie aus gegebenen Fakten neue Fakten hergeleitet werden können (PROLOG).
- **Regelbasierte** Programmiersprachen:
  Programme bestehen aus "wenn-dann-Regeln"; wenn eine angegebene Bedingung gültig ist, dann wird eine angegebene Aktion ausgeführt (OPS5).
- **Objektorientierte** Programmiersprachen:
  Programme bestehen aus Objekten, die bestimmte (Teil-)Probleme lösen und zum Lösen eines Gesamtproblems mit anderen Objekten über Nachrichten kommunizieren können (SMALLTALK).

Quelle: Boles, Dietrich (2015): Programmieren spielend gelernt mit dem Java-Hamster-Modell. http://www.java-hamster-modell.de/eBooks/hamster1.pdf.

Objektorientierte Programmiersprachen: C++

C++:

- Ermöglicht **maschinennahe** Programmierung (Stichw. „Zeiger“), als auch abstrakte Programmierung (i.e. **Objektorientierung**)
- Kompilierung über g++ Compiler, Microsoft Visual C++ Compiler, etc.

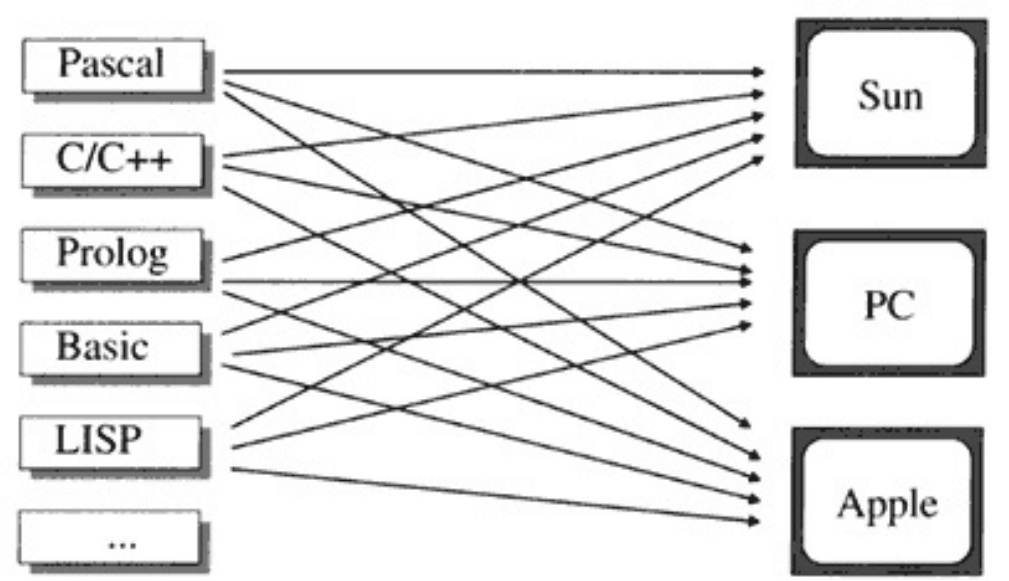


Abb.: n ×m viele Compiler

Objektorientierte Programmiersprachen: Java

Java (Java ist NICHT JavaScript!)

- Besonderheit: Java-Programme werden in Bytecode übersetzt, anschließend in einer Java-Laufzeitumgebung ausgeführt → **Virtuelle Maschine** (VM)

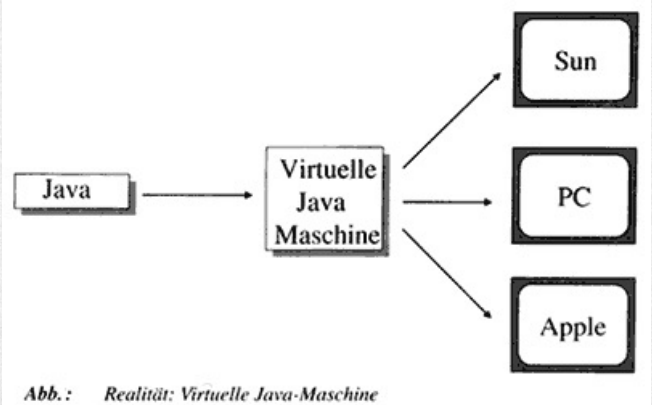


Abb.: Realität: Virtuelle Java-Maschine

- Vorteil: Plattformunabhängigkeit: Java-Programme laufen zumeist ohne weitere Anpassungen auf unterschiedlichen Computer- und Betriebssystemen, für die eine Java-VM existiert

WAT

/