

Computergrundlagen Turingmaschinen und Programmierung

Jens Smiatek und Axel Arnold

Institut für Computerphysik
Universität Stuttgart

Wintersemester 2016/17



Was ist ein Computer?

*Ein Computer (Rechner oder elektronische Datenverarbeitungsanlage) ist ein Gerät, welches mittels **programmierbarer Rechenvorschriften (Algorithmen)** Daten verarbeitet.*

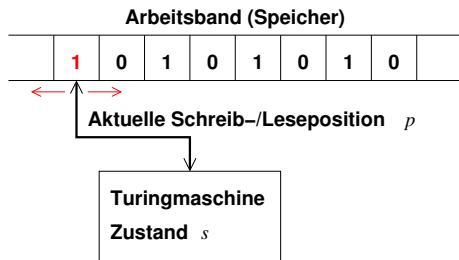
Fragen:

- Was heisst programmierbar?
- Wie sehen Rechenvorschriften (Algorithmen) aus?

Theoretische Informatik (Berechenbarkeit)



A. Turing, 1912 - 1954



- ca 1920, D. Hilbert: Was ist berechenbar, was beweisbar?
- 1931, K. Gödel: Entweder widerspruchsfrei oder vollständig!
- Nicht alles ist berechenbar!
- Wir können nicht mal bestimmen, was nicht beweisbar ist

Implikationen

Entscheidungsproblem (David Hilbert):

Ist eine Formel allgemeingültig?

Ist jede Interpretation einer formal logischen Aussage wahr?

**Turingmaschine (Alan Turing):
Definition des Algorithmus und der
Berechenbarkeit als formale, mathematische
Begriffe**

A. Turing: *On computable numbers, with an application to the Entscheidungsproblem.* (1936)

Turingmaschinen

Eine *Turingmaschine* ist definiert durch

- eine endliche Menge Γ , das Arbeitsalphabet, mit Leerzeichen $\sqcup \in \Gamma$
- eine endliche Menge Z von Zuständen der Maschine
- den Startzustand $s \in Z$ und den Haltezustand $h \in Z$
- eine Überföhrungsfunktion $\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$

Eine *Konfiguration* ist

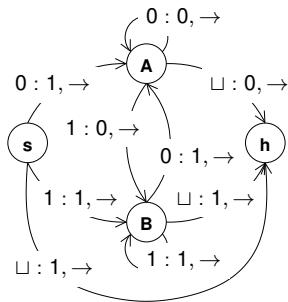
- eine Zeichenkette w über dem Arbeitsalphabet
- eine Position p in w , die aktuelle Arbeitsposition
- der aktuelle Zustand z der Turingmaschine

Eine Konfiguration (w, p, z) führt in eine andere (w', p', z') über, wenn $\delta(z, w_p) = (z', w'_p, d)$, und d p nach p' versetzt.

Startkonfiguration: Eine Eingabe w , p am Anfang von w , Zustand s .

Darstellung von Turingmaschinen

Beispiel: Eine „1“ vorne einfügen



A: „0“ gemerkt

B: „1“ gemerkt

s	0	A	1	→
s	1	B	1	→
s	□	h	1	→
A	0	A	0	→
A	1	B	0	→
A	□	h	0	→
B	0	A	1	→
B	1	B	1	→
B	□	h	1	→

Ablauf: Schreibe Zustand in das Arbeitsband vor die aktuelle Position, \vdash bezeichnet Übergang

$s101 \vdash 1B01 \vdash 11A1 \vdash 110B \square \vdash 1101h \square$

Turingmaschinen II

- Turingmaschinen können kombiniert werden
Beispiel: Hinter jedem Zeichen eine 0 einfügen
- *universelle Turingmaschine*: eine Turingmaschine, die berechnet, was eine Turingmaschine tut
- *Halteproblem*: es gibt keine Turingmaschine, die entscheidet, ob eine gegebene Turingmaschine bei leerer Eingabe anhält
- Turingmaschinen und von Neumann-Computer sind äquivalent
- Church-Turing-These:

Turing-berechenbare Funktionen sind genau die von Menschen intuitiv berechenbaren Funktionen

Was sind Programme?

- In dieser Vorlesung: Python, C, \LaTeX , bash
- Sind das alle Typen von Programmiersprachen?

Wie kann man Programmiersprachen
klassifizieren?

- Kein Prozessor versteht Python-, C- oder Shell-Befehle
- Was muss alles passieren, um ein Programm laufen zu lassen?

Wie wird aus unserem Programm etwas, dass der
Prozessor ausführen kann?

Kommunikation mit Computern

<http://www.icp.uni-stuttgart.de>



Kommunikation zwischen R2-D2 (Computer), C3-PO (Übersetzer) und Luke (Mensch?)

Programmiersprachen

Imperative Sprachen

Beispiele: Python, C, Shell, BASIC...

- Die meisten Sprachen sind imperativ
- Programme erklären, *wie* ein Problem gelöst werden soll
- Umsetzung des von Neumann-Rechners: Befehle und Schleifen
- **prozedural** heißen Sprachen, die Prozeduren kennen

Beispiele: alle außer einfachem BASIC

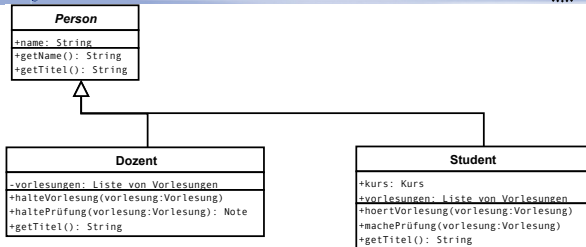
Deklarative Sprachen

- Keine von Neumann-artigen Befehle, kein innerer Zustand
- Rekursion anstatt Schleifen
- **funktional**, basierend auf Funktion im mathematischen Sinn

Beispiele: Haskell, Erlang, Scheme...

- **logisch**, basierend auf Fakten, Axiomen und logischer Ableitung

Beispiele: Prolog

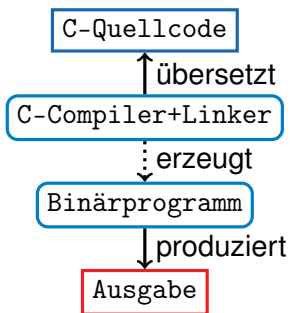
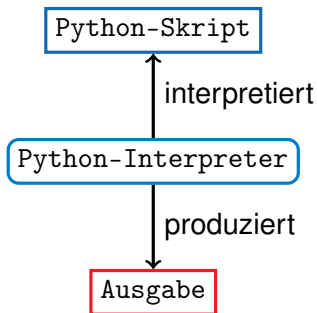


- Objektorientierung (OO) ist ein sprachunabhängiges Programmiermodell
- Ziel ist die bessere Wartbarkeit und Austauschbarkeit von Software durch starke Isolation von Teilproblemen
- Speziell darauf ausgelegt sind z.B. C++, Java, Python,...
- UML (Unified Modelling Language) beschreibt OO-Modelle

Terminologie

- **Klasse:** beschreibt Eigenschaften und Methoden von Objekten
Beispiel: Klassen sind z.B. Dozent, Student, Person
- **Objekt:** eine Instanz einer Klasse. Ein Objekt hat stets eine Klasse, aber es kann viele Instanzen geben
Beispiel: Maria Fyta ist ein Dozent (neutral...), ich auch
- **Eigenschaften:** Datenelemente von Objekten
Beispiel: Dozent hat Name, Titel und zu haltende Vorlesungen
- **Methoden:** Funktionen einer Klasse
Beispiel: Personen können ihren Namen sagen
- **Datenkapselung:** Eigenschaften werden nur durch Funktionen der Klasse verändert
Beispiel: der Name einer Person kann nicht geändert werden
- **Vererbung:** Klassen können von anderen abgeleitet werden, erben dadurch deren Eigenschaften und Funktionen
Beispiel: Jede Person hat einen Namen

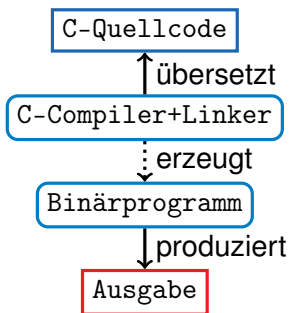
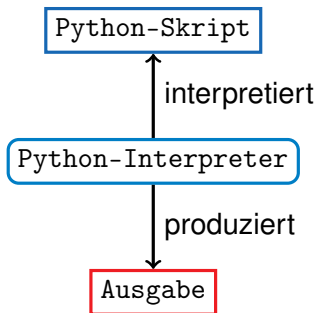
Interpreter- und Compilersprachen



Interpretersprachen

- Z.B. Python, Java, C#, Basic, PHP, Shellsprachen
- Das Programm wird vom Interpreter gelesen und ausgeführt
- Es wird nie in Maschinencode für den Prozessor übersetzt
- Ausnahme: Just-in-Time (JIT) Compiler

Interpreter- und Compilersprachen



Compilersprachen

- Z.B. C/C++, Fortran, Pascal/Delphi
- Compiler übersetzt in maschinenlesbaren Code
- Nicht portabel, erschwerte Fehlersuche, deutlich schneller
- Interpreter selbst müssen kompiliert werden



Maschinensprache

Aber was versteht nun der Prozessor?

Zahlenkolonnen, z.B.

```
AD 34 12 18 69 2A 8D 34
12 AD 35 12 69 00 8d 35
12
```

- Maschinencode für den 6502-Mikroprozessor
- Addiert 42 zur 16-bit-Zahl an Speicherstellen 1234h und 1235h

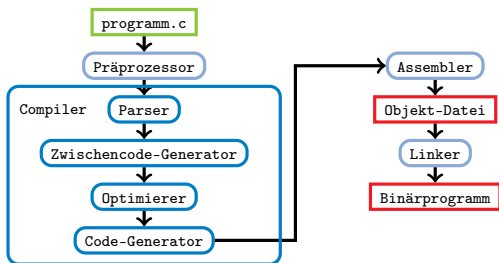
Wie wird aus einem Programm Maschinencode?



Assembler

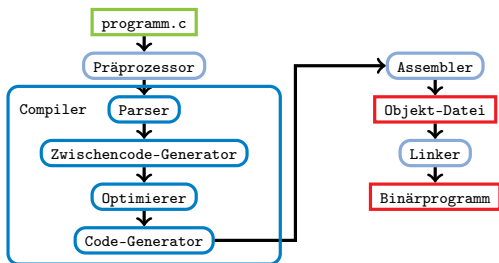
- Assembler ist die menschenlesbare Form der Maschinensprache
- Assembler bezeichnet auch das Programm, das die Befehle in Zahlen übersetzt und Labels an Speicherzellen knüpft
- Zeitkritische Anwendungen werden auch heute noch manchmal in Assembler geschrieben

Vom Sourcecode zum Programm



- Ein Programm durchläuft viele Schritte bis zur fertigen ausführbaren Datei
- **Präprozessor**, **Compiler**, **Assembler** und **Linker** sind meist separate Programme
- meist mehrere Objektdateien aus verschiedenen Quelltextdateien

Vom Sourcecode zum Programm

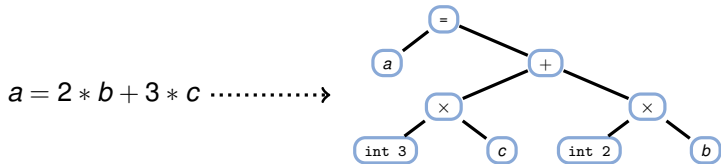


Präprozessor

- Im Prinzip sprachunabhängig, rein textbasiert
- Bindet weitere Quelltextdateien ein
- Erlaubt den Einsatz von *Makros* (Ersetzungen)

Der Compiler

- **Parser** übersetzt den Quellcode in einen Syntaxbaum

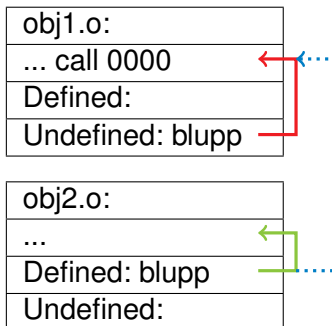
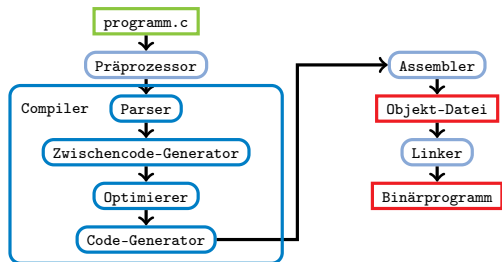


- **Zwischencode-Generator** erzeugt daraus Pseudocode, etwa:

$r1 = b$	$r2 = r1 \times 2$	$r3 = c$	$r4 = r3 \times 3$	$r5 = r2 + r4$	$a = r5$
----------	--------------------	----------	--------------------	----------------	----------

- **Optimierer** versucht, diesen zu verbessern, z.B. durch
 - Prozessorregisterzuweisung
 - Einfügen kurzer Funktionen, Entrollen von Schleifen
 - Suche nach gemeinsamen Termen, ...
- **Code-Generator** erzeugt Assembler aus dem Zwischencode

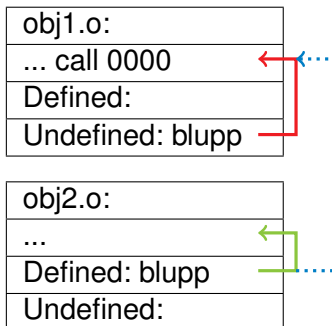
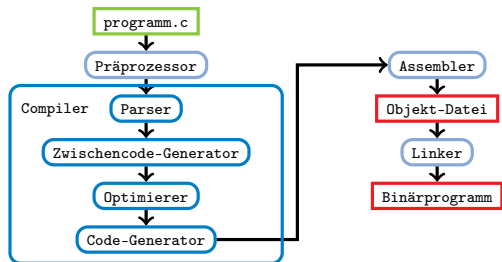
Der Assembler



Der **Assembler** erzeugt eine Objektdatei mit

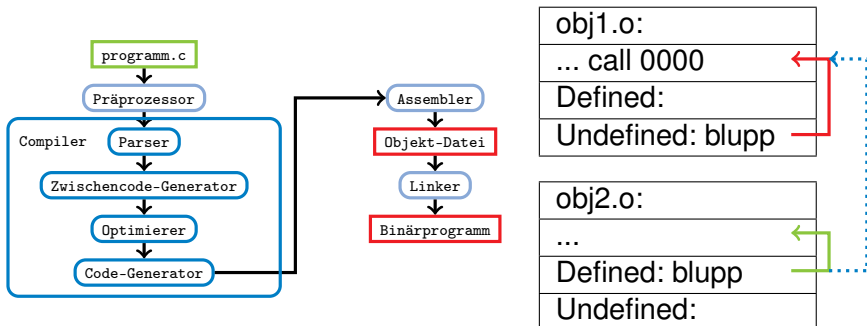
- Maschinencode
- den Speicherpositionen von globalen Variablen und Funktionen
- Stellen, an denen Stellen Information über solche Positionen aus andere Objektdateien benötigt wird

Der Linker



- Verbindet mehrere solcher Objektdateien zu einer ausführbaren Datei (Binary) oder einer **Bibliothek**
- Bibliotheken sind Sammlungen von Objektdateien
- Der Linker verbindet Zugriffe zwischen Objektdateien
- Nur wenn alle Zugriffe aufgelöst wurden, entsteht ein Binary

Dynamisches Linken



- Beim **dynamischen Linken** wird das Linken gegen die Bibliotheken erst beim Starten des Programms erledigt.
- Dadurch können Teile des Programms geupdated werden
- Und mehrfach benutzte Bibliotheken werden nur einmal geladen
- Erfordert spezielle dynamische Bibliotheken (`.so`, `.dylib` oder `.dll`)