Programmierkurs II

Prof. Dr. Wolfgang Effelsberg

Universität Mannheim

Sommersemester 2004

Inhalt (1)

Teil I: Die Programmiersprache C

Inhalt (2)

Teil II: Programmieren in Assembler

Literaturhinweise

Zur Programmiersprache C

Karlheinz Zeiner: "Programmieren lernen mit C“. Carl Hanser Verlag, 1994

Brian W. Kernighan, Dennis M. Ritchie: "Programmieren in C“. Carl Hanser Verlag, 1990

Zur Programmierung in Assembler

Lutz Bierl: Das große MSP 430 Praxisbuch. Franzis-Verlag, Poing, 2004

1. Syntax und Semantik von Programmiersprachen

Syntax:

Die Syntax einer Sprache beschreibt die durch die Regeln einer Grammatik und ein Alphabet bestimmte Struktur der ableitbaren, formal richtigen Sätze der Sprache, ohne auf ihre Bedeutung Bezug zu nehmen. Im Falle einer Programmiersprache werden durch die Syntax die formal richtigen Programme beschrieben.

Semantik:

Die Semantik ist die Lehre von der Beziehung der Zeichen zum gemeinten Gegenstand, die Lehre von den Bedeutungen (linguistisch: Lehre von den Beziehungen zwischen Sprache und Wirklichkeit). Im Falle einer Programmiersprache beschreibt die Semantik, was ein Programm leistet und was die einzelnen Arbeitsvorschriften bedeuten.

Beispiel 1

a, b seien ganze Zahlen

Mathematische Ausdrücke:

a + 3 = b ist syntaktisch korrekt

a + = 3 ist syntaktisch falsch

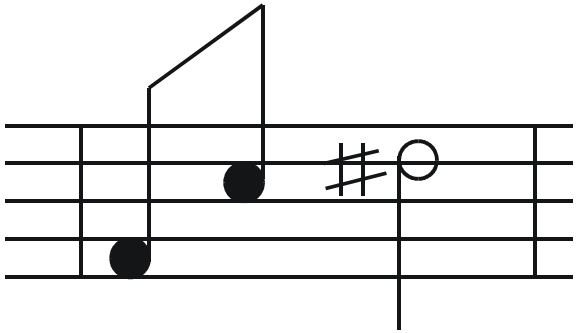
Programmiersprache C:

b == a + 3 ist eine syntaktisch korrekte Bedingung

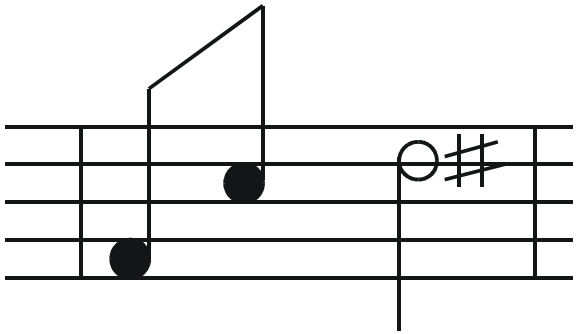
b = a + 3 ist eine syntaktisch korrekte Wertzuweisung

a + 3 = b ist syntaktisch falsch

Beispiel 2



syntaktisch korrekt, Semantik ist jedem Musiker bekannt



syntaktisch falsch

Beispiel 3

Anweisungen:

Voraussetzung: mitteleuropäischer Kalender (gregorianisch)

1. "Schreibe den Namen des ersten Monats im Jahr"
syntaktisch korrekt, semantisch klar

2. "Schreibe den Namen des ersten Mohnatz im Jahr"
syntaktisch falsch

3. "Schreibe den Namen des dreizehnten Monats im Jahr"
syntaktisch korrekt, semantisch falsch

4. "Lies n ein; schreibe den Namen des n-ten Monats im Jahr"
syntaktisch korrekt, semantisch nur korrekt für $1 \leq n \leq 12$

Beschreibungssprachen für Syntax (1)

Eine Programmiersprache muss eine wohldefinierte Syntax und Semantik haben. Deshalb werden oft formale Methoden der Informatik zu ihrer Beschreibung eingesetzt, z. B. Backus-Naur-Form (BNF) oder Syntax-Diagramme.

Zur maschinellen Verarbeitung eines Algorithmus muss dieser in einer wohldefinierten Sprache (frei von Mehrdeutigkeiten und Ungenauigkeiten) ausgedrückt werden. Eine solche Sprache heißt **Programmiersprache**.

Zur Ausführung eines Programms, das in einer Programmiersprache vorliegt, muss der Computer in der Lage sein,

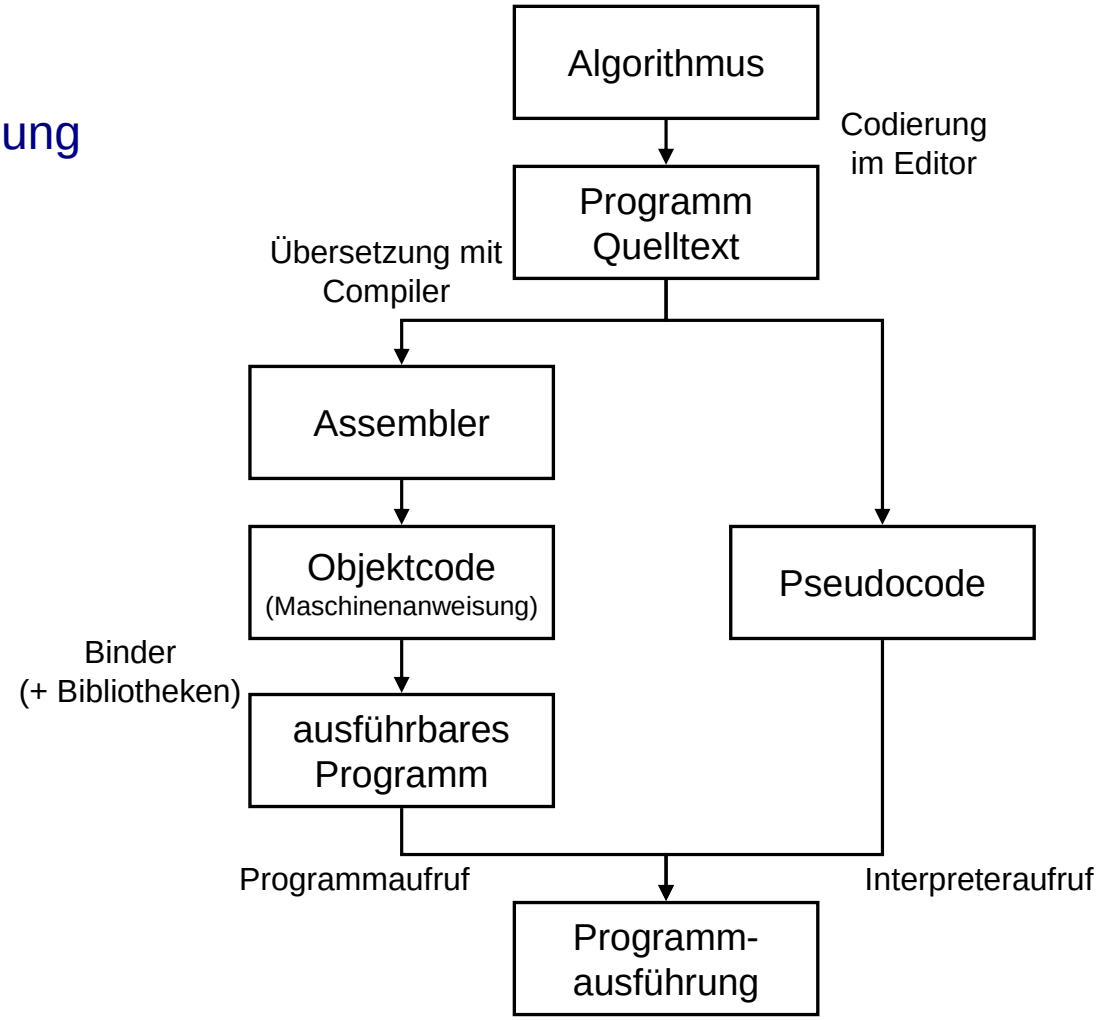
1. die Symbole, in denen der Algorithmusschritt ausgedrückt ist, zu verstehen (Syntax),
2. dem Algorithmusschritt in Form von auszuführenden Operationen eine Bedeutung zuzuordnen (Semantik),
3. die entsprechenden Operationen auszuführen.

Beschreibungssprachen für Syntax (2)

Ein Programm wird von einem Übersetzer(Compiler) oder einem Pseudo-compiler (Teil des Interpretierers) zunächst syntaktisch analysiert. Ist es syntaktisch korrekt, so werden die Anweisungen ausgeführt, entweder sofort (**Interpretierer**) oder durch Übersetzen in Maschinensprache und Ausführung in einem separaten Schritt (**Compiler**).

Programmentwicklung in C

1. Spezifikation
2. Algorithmische Lösung
3. Codierung
4. Test



Klassifikation von Programmierfehlern (1)

a) Syntaktische Fehler

Werden vom Compiler oder Interpreter erkannt und gemeldet. Anweisung wird nicht ausgeführt.

Beispiel: `a = b * + c;`

b) Semantische Fehler

Werden manchmal zur Laufzeit erkannt und gemeldet (Abbruch), manchmal überhaupt nicht.

Beispiel 1: /* Überschreitung von Array-Grenzen */

Vektor a habe 100 Elemente.

```
i = 101;
a[i] = 17;
```

Klassifikation von Programmierfehlern (2)

Beispiel 2: /* Division durch 0 */

```
r = 0.0;
s  = 170/r;
```

c) Logische Fehler

Das Programm beschreibt nicht das , was der Programmierer eigentlich wollte.

Beispiel: /* Berechne Kreisumfang */

```
umfang =  pi * radius;  /* statt 2.0*pi*radius */
```

Die meisten semantischen Fehler und alle logischen Fehler können grundsätzlich durch Testen oder durch formale Verifikation gefunden werden.

Man beachte: **Testen ist kein Beweis für die Korrektheit eines Programms!**

Syntaxdiagramme

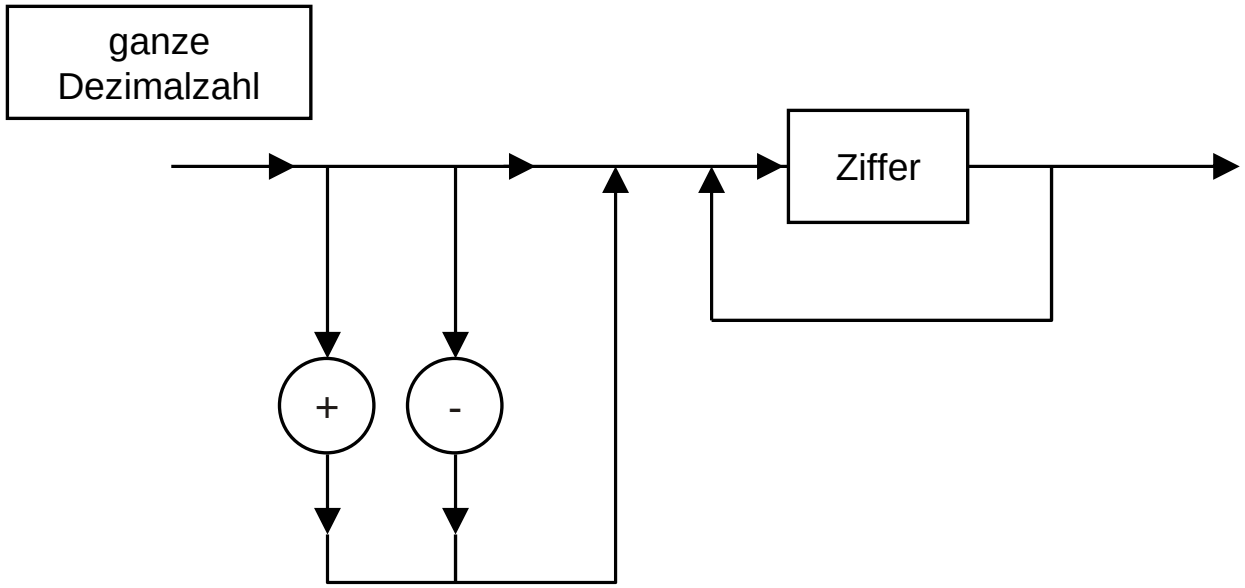
Üblicherweise wird zur formalen Beschreibung von C die Backus-Naur-Notation gewählt (vgl. Zeiner, Kernighan/Ritchie). Wir verwenden Syntaxdiagramme, eine graphische Darstellung.

Syntaxdiagramme beschreiben korrekte syntaktische Ausdrücke. Sie bestehen aus

- Terminalsymbolen (Kreisen)
- Nicht-Terminalsymbolen (Quadraten)
- gerichteten Kanten (Pfeilen).

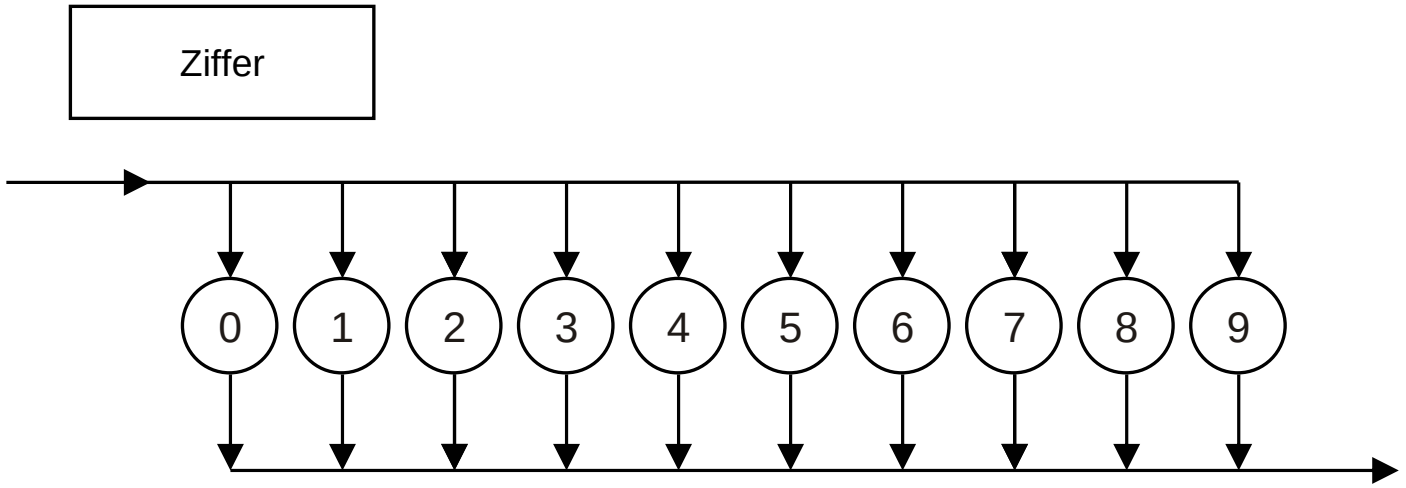
und weisen eine rekursive Struktur auf.

Beispiel für ein Syntaxdiagramm (1)



BNN (Backus-Naur-Notation): ganze Dezimalzahl ::= {+/-}opt {Ziffer}1+

Beispiel für ein Syntaxdiagramm (2)



BNN: Ziffer :: = 0|1|2|3|4|5|6|7|8|9

Alle möglichen Pfade durch das Diagramm ergeben syntaktisch korrekte ganze Zahlen.

Vorteil von Syntaxdiagrammen

Formale Beschreibungen sind klarer als eine umgangsprachliche Definition.

Im Beispiel: Eine ganze Zahl ist eine Folge von Ziffern, der ein Plus- oder Minuszeichen vorangehen kann.

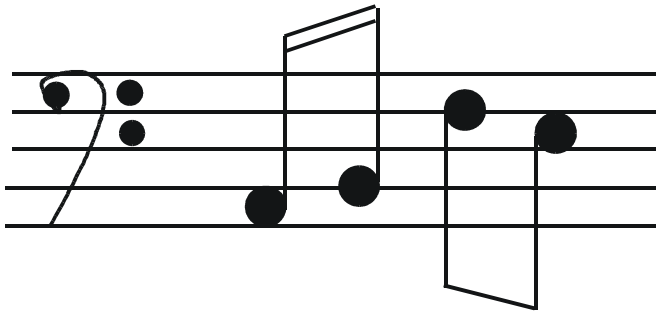
Unklar:
- hex, oktal, binär, dezimal?
  Welche Ziffern (0,1 oder 0-7 oder 0-9...)?
- Kann auch sowohl ein Plus- als auch ein Minuszeichen vorausgehen?
- Kann die Folge auch aus einer einzigen Ziffer bestehen?
- Gibt es eine Höchstzahl von Ziffern?

Man beachte:

- Eine ganze Zahl ohne Vorzeichen wird als positive Zahl interpretiert.
- Die Semantik ist nicht aus dem Syntaxdiagramm zu erkennen!

Beispiele für semantisch unklare Umgangssprache

1) adagio



Die Noten selbst sind syntaktisch und semantisch eindeutig. Die Bezeichnung "adagio" oder "andante" ist syntaktisch korrekt, aber semantisch nicht eindeutig.

2) "Sahne steif schlagen"

Wie steif?

3) "Bei mittlerer Hitze knusprig braun braten"

Was ist mittlere Hitze, was ist knusprig braun?

Beispiele für unklare Semantik in Programmier-sprachen

- Der Wert einer Variablen vor der ersten Wertzuweisung (undefiniert; compilerabhängig)
- Der Wert einer Laufvariablen nach Schleifenende undefiniert; compilerabhängig)
- Der präzise Wert einer Gleitkommazahl (maschinenabhängig (Wortlänge in Bits))
  z. B.

$$r = \sqrt{2}$$

Vorsicht bei Vergleichsoperationen mit Gleitkommazahlen, zum Beispiel beim Abprüfen auf Null!

```
if (2 - r*r == 0) ...
```