

# Ein- und Ausgabe

## Übersicht

- ◆ Wie werden Schriftzeichen kodiert?
- ◆ Ein- und Ausgabe von ASCII-Zeichen
  - ◆ Eingabe: get/1, get\_code/1
  - ◆ Ausgabe: put\_code/1, nl/0, tab/1
  - ◆ Konvertierung: name/2, atom\_codes/2, number\_codes/2
  - ◆ Zeichenketten
- ◆ Ein- und Ausgabe von Termen
  - ◆ write/1, read/1, write\_canonical/1
- ◆ Umlenken von Ein- und Ausgabe in Dateien
  - ◆ tell/1, telling/1, told/0
  - ◆ see/1, seeing/1, seen/0
- ◆ Dateienden und -Verarbeitung

# Buchstaben als Zahlen: Kodierung

## Buchstaben können als Zahlen angesehen werden.

- ◆ Eine *Kodierung* legt fest, welcher Buchstabe mit welcher Zahl gemeint ist.

| Willkürliche Zuordnung 1 | Willkürliche Zuordnung 2 |
|--------------------------|--------------------------|
| 1 = A                    | 65 = A                   |
| 2 = B                    | 66 = B                   |
| ...                      | ...                      |

- ◆ Die Zahlen selbst wurden 'traditionell' durch 8-stellige 0/1-Folgen (*byte*) dargestellt. D.H. 256 mögliche unterschiedliche Werte

| Willkürliche Zuordnung 1 | Willkürliche Zuordnung 2 |
|--------------------------|--------------------------|
| 00000001 = A             | 01000001 = A             |
| 00000010 = B             | 01000010 = B             |
| ...                      | ...                      |

# Kodierungsstandards

## Verschiedene Standards/Konventionen

- ◆ *American Standard Code for Information Interchange* (ASCII)
  - ◆ anderer Name: *International Alphabet 5* (IA5)
  - ▶ regelt Zeichenkodierung fürs englische Alphabet (A – Z; a – z; 0 – 9) und einige Sonderzeichen wie @, {, /, (, %, [.
- ◆ **ISO 8859-1** erweitert ASCII um Zeichenkodierung für die Schriftzeichen der meisten westeuropäischen Sprachen, z.B. ä, ß, É, Ò
  - ◆ UNIX- und WIN-Systeme verwenden oft ISO-8859-1 (ANSI). MacOS und DOS nicht.
- ◆ Dutzende andere Konventionen

## Probleme

- ◆ Manche Konventionen widersprechen sich (MacOS vs. WIN)
- ◆ Nur ASCII ist wirklich weit verbreitet, umfasst aber wenig Zeichen

# ASCII-Zeichensatz-Tabelle

## Ausschnitt aus den 128 Zeichen der ASCII-Tabelle

|                  |      |      |      |       |       |
|------------------|------|------|------|-------|-------|
| 10/13 neue Zeile | 47 / | 63 ? | 79 O | 95 _  | 111 o |
| 32 Leerschlag    | 48 0 | 64 @ | 80 P | 96 `  | 112 p |
| 33 !             | 49 1 | 65 A | 81 Q | 97 a  | 113 q |
| 34 "             | 50 2 | 66 B | 82 R | 98 b  | 114 r |
| 35 #             | 51 3 | 67 C | 83 S | 99 c  | 115 s |
| 36 \$            | 52 4 | 68 D | 84 T | 100 d | 116 t |
| 37 %             | 53 5 | 69 E | 85 U | 101 e | 117 u |
| 38 &             | 54 6 | 70 F | 86 V | 102 f | 118 v |
| 39 '             | 55 7 | 71 G | 87 W | 103 g | 119 w |
| 40 (             | 56 8 | 72 H | 88 X | 104 h | 120 x |
| 41 )             | 57 9 | 73 I | 89 Y | 105 i | 121 y |
| 42 *             | 58 : | 74 J | 90 Z | 106 j | 122 z |
| 43 +             | 59 ; | 75 K | 91 [ | 107 k | 123 { |
| 44 ,             | 60 < | 76 L | 92 \ | 108 l | 124   |
| 45 -             | 61 = | 77 M | 93 ] | 109 m | 125 } |
| 46 .             | 62 > | 78 N | 94 ^ | 110 n | 126 ~ |

# Ein Ansatz zur Vereinheitlichung

## Unicode Version 4.0.1 (März 2004)

- ◆ Prinzip: Eine eindeutige Zahl für jedes Zeichen!
  - ◆ Ursprünglich Kodierung mit maximal 16-stelligen 0/1-Folgen (Zahlen von 0 bis 65535, d.h. maximal 65536 verschiedene Zeichen)
- ◆ Abdeckung der Kodierung (ca. 96'000 Kodierungen in Version 4)
  - ◆ Gegenwärtig verwendeten Schriftzeichen (*glyphs*) und Symbole von (fast) allen Sprachen der Welt
  - ◆ Zeichen wichtiger ausgestorbener Sprachen
  - ◆ Heute: Gute Unterstützung durch alle modernen Betriebssysteme!
  - ◆ Im Web: Unicode-Kodierung (UTF-8-Format) von HTML-Dateien
- ◆ Kodierungstabellen und Infos unter <http://www.unicode.org>
- ◆ Konform bzw. parallel entwickelt zur ISO/IEC-Normierung 10646
  - ◆ Dieses definiert dieselbe Kodierung als Universal Character Set (UCS).

# Ausschnitte aus Unicode-Tabellen

| iso-8859-1 |   |   |   |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|---|---|---|
| +          | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 160        |   | ı | φ | £ | ¥ | ı | § | " | ø |   |
| 170        | æ | « | ¬ | – | ø | – | ° | ± | ² | ³ |
| 180        | ˆ | µ | ¶ | · | ˆ | ˆ | ˆ | ˆ | ˆ | ˆ |
| 190        | ¸ | ó | ô | õ | ö | ÷ | ø | ù | ú | û |
| 200        | Ë | É | Ê | Ë | Ì | Í | Î | Ï | Ð | Ñ |
| 210        | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û |
| 220        | Ü | Ý | Þ | ß | à | á | â | ã | ä | å |
| 230        | æ | ç | è | é | ê | ë | ì | í | î | ï |
| 240        | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù |
| 250        | ú | û | ü | ý | þ | ÿ |   |   |   |   |

|   | 000 | 001 | 002 | 003 | 004 | 005 | 006 | 007 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | À   | Á   | Â   | Ã   | Ä   | Å   | Æ   | Ç   |
| 1 | È   | É   | Ê   | Ë   | Ì   | Í   | Î   | Ï   |
| 2 | Ò   | Ó   | Ô   | Õ   | Ö   | ×   | Ø   | Ù   |
| 3 | Ü   | Ý   | Þ   | ß   | à   | á   | â   | ã   |
| 4 | ä   | å   | æ   | ç   | è   | é   | ê   | ë   |
| 5 | ì   | í   | î   | ï   | ð   | ñ   | ò   | ó   |
| 6 | ô   | õ   | ö   | ÷   | ø   | ù   | ú   | û   |
| 7 | ü   | ý   | þ   | ÿ   |     |     |     |     |
| 8 |     |     |     |     |     |     |     |     |
| 9 |     |     |     |     |     |     |     |     |
| A |     |     |     |     |     |     |     |     |
| B |     |     |     |     |     |     |     |     |
| C |     |     |     |     |     |     |     |     |
| D |     |     |     |     |     |     |     |     |
| E |     |     |     |     |     |     |     |     |
| F |     |     |     |     |     |     |     |     |
| G |     |     |     |     |     |     |     |     |
| H |     |     |     |     |     |     |     |     |

Unicode enthält ISO-Latin 8859-1 zwischen 0 und 255

Exotische Zeichen

# Zeichen ausgeben: put\_code/1

- **put\_code/1** gibt ein einzelnes Zeichen aus. Das Argument ist der Code des Zeichens.

```
?- put_code(72), put_code(97), put_code(108),
   put_code(108), put_code(111).
Hallo
```

- ◆ Anstelle des ISO-Prolog-Prädikats **put\_code/1** wird noch oft das traditionelle Prädikat **put/1** verwendet.
- ◆ Die WCX-Erweiterung von SICStus Prolog (Wide Character eXtension) erlaubt u.a. das Einlesen und Ausgeben von Unicode-Zeichen im utf-8-Format.

# Zeichen einlesen: get\_code/1

- **get\_code/1** wartet, bis der Benutzer ein einzelnes Zeichen auf der Tastatur tippt und mit RETURN bestätigt.

- ◆ Danach unifiziert das Argument von **get\_code/1** mit dem Code des eingegebenen Zeichens.

```
?- get_code(X).
|: A ← Benutzer hat Taste A
X = 65 ? und RETURN gedrückt.
yes
```

|    |   |
|----|---|
| 63 | ? |
| 64 | @ |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| 69 | E |
| 70 | F |
| 71 | G |
| 72 | H |

- ▶ Anstelle des ISO-Prolog-Prädikats **get\_code/1** wird noch oft das traditionelle Prädikat **get0/1** verwendet.

## Druckbare Zeichen einlesen: get/1

- Das traditionelle Prädikat **get/1** liest das erste druckbare Zeichen ein, das eingegeben wird.
  - Danach unifiziert das Argument von get/1 mit dem Code des eingegebenen Zeichens.

```
?- get(X).
|:  A ← Benutzer hat die
X = 65 ? Leertaste, Taste A und
yes      RETURN gedrückt.
```

|    |   |
|----|---|
| 63 | ? |
| 64 | @ |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| 69 | E |
| 70 | F |
| 71 | G |
| 72 | H |

Als **nicht-druckbar** gelten die ASCII-Zeichen von 0-32 und 127. Darunter sind Zeilenende (10/13), Tabulator (9) und Leerzeichen (32).

Ein- und Ausgabe – 9

## Zeilenende nl/0 und Leerzeichen tab/1

- Zeilenenden** werden auf unterschiedlichen Betriebssystemen durch unterschiedliche ASCII-Codes repräsentiert.

| Betriebssystem | ASCII   |
|----------------|---------|
| UNIX           | 10      |
| MacOS          | 13      |
| DOS/WIN        | 10 + 13 |

10 = Zeilenvorschub (*linefeed, LF*)  
13 = Wagenrücklauf (*carriage return, CR*)

- nl/0** gibt betriebssystemabhängig die richtigen ASCII-Codes für Zeilenende aus!

- Mehrere Leerzeichen** werden gerne mit **tab/1** ausgegeben:

- Das Argument gibt die Anzahl der auszugebenden Leerzeichen an (keine Tabulatorzeichen!).
- ```
?- tab(1+2), put(33).
!
yes
```

Ein- und Ausgabe – 10

## Wie beweist Prolog Ein- und Ausgabe?

**Eingabepredikate** sind bewiesen, wenn eine entsprechende Eingabe erfolgt ist.

- Interaktive Eingabe** "blockiert" den Prolog-Interpreter bis Input erfolgt.
  - Bei Backtracking werden allfällige Variablenbindungen rückgängig gemacht, aber es erfolgt keine weitere Eingabeaufforderung!

**Ausgabepredikate** sind bewiesen, wenn eine entsprechende Ausgabe als Seiteneffekt erfolgt ist.

- Die Ausgabe selbst hat auf den Beweis keinen Einfluss.**
  - Bei Backtracking bleibt der Seiteneffekt (die Ausgabe) bestehen, kein Backtracking!

**Ein- und Ausgabe** gelingen höchstens 1-Mal!

Ein- und Ausgabe – 11

## ASCII-Codes und atomare Terme

- Das eingebaute Prädikat name/2**

- gibt den Namen eines nicht-variablen atomaren Terms als Liste von ASCII-Codes zurück

```
?- name(bla, L).           ?- name(27, L).
L = [98,108,97]          L = [50,55]
```

Modus: name(+Atomar, ?Liste)

- oder erzeugt umgekehrt einen atomaren Term aus einer Liste von ASCII-Codes

```
?- name(A, [98,108,97]).
A = bla
```

Modus: name(?Atomar, +Liste)

Ein- und Ausgabe – 12

## Das Problem mit name/2...

### Zahl oder Atom?

- Falls die ASCII-Code-Liste eine Prolog-Zahl beschreibt, wird sie **immer nur** als Zahl instantiiert.

```
?- name(N, [50,55]).  
N = 27 ;  
no
```

↔

```
?- name('27', [50,55]).  
yes
```

Anstelle von name/2 sollten die konsistenten ISO-Prolog-Prädikate **atom\_codes/2** und **number\_codes/2** verwendet werden, die eine ASCII-Liste konsequent in Atome bzw. Zahlen umsetzen.

- atom\_codes/2 und number\_codes/2 erzeugen Exceptions, falls ihre Argumente nicht Atome bzw. ASCII-Kodes für Zahlen sind. number\_codes/2 ist kein wahr/falsch-Testprädikat, ob eine ASCII-Liste eine Zahl darstellt.

Ein- und Ausgabe – 13

## ASCII-Codes als Zeichenketten

Eine beliebige Zeichenkette (*string*), die zwischen zwei " (doppelte Hochkommata) eingeschlossen ist, wird als Liste der ASCII-Codes dargestellt.

```
?- Kette = "Hallo Du".  
Kette = [72,97,108,108,111,32,68,117]
```

|    |            |     |   |
|----|------------|-----|---|
| 32 | Leerschlag | 108 | l |
| 68 | D          | 111 | o |
| 72 | H          | 117 | u |
| 97 | a          |     |   |

Ein- und Ausgabe – 14

## Ein-/Ausgabe von Prolog-Termen

### Prolog hat vordefinierte Ein-/Ausgabe-Prädikate für Prolog-Terme

- Vorteil: Komplexe Ausdrücke müssen nicht als Einzelzeichen eingelesen und mühsam zusammengesetzt werden
- Nachteil: Die Prolog-Term-Syntax muss beachtet werden
  - Jeder Term muss bei der Eingabe mit einem Punkt beendet werden!

Interaktion kann mit der Aussenwelt nach einem einfachen Muster erfolgen.

```
interaktion :-  
    read(Input),  
    verarbeite(Input, Output),  
    write(Output).
```

Ein- und Ausgabe – 15

## Terme einlesen: read/1

- read/1 liest einen Term ein.

```
?- read(Eingabe), write(Eingabe).  
|: bla. ← Eingabe des Benutzers
```

Ausgabe von Prolog → bla

```
Eingabe = bla ? ;  
no
```

- Beachte:** Der Punkt beendet den Term, gehört aber selbst nicht dazu!

```
?- read(Eingabe), write(Eingabe).  
|:bla bla.  
{SYNTAX ERROR...}
```

- Beachte:** Die Syntaxregeln für Prolog-Terme müssen beachtet werden!

Ein- und Ausgabe – 16

## Termausgabe: write/1, write\_canonical/1

- **write/1** gibt einen einzelnen Term aus.

```
?- write(1 + 2 == 3 -0).  
1+2==3-0  
?- write([bla,bli,blu]).  
[bla,bli,blu]
```

- ▶ write/1 respektiert die beim Aufruf definierten Operatoren und Spezialsyntax.

- **write\_canonical/1** ignoriert Spezialsyntax und Operatoren

```
?- write_canonical([bla,bli,blu] = 2).  
=('.'(bla,'.'(bli,'.'(blu,[])),2)
```

Ein- und Ausgabe – 17

## Schreiben in eine Datei

- **tell/1** leitet die Ausgabe der Prädikate

- ◆ put/1
- ◆ nl/0, tab/1
- ◆ write/1, write\_canonical/1

```
?- tell(hans),  
write(hallo), nl,  
write(du), nl,  
told.
```

- in eine Datei um.



- **told/0**

- ◆ beendet die Umleitung (schliesst die Datei!)
- ◆ sorgt dafür, dass zukünftige Ausgaben wieder auf dem Bildschirm erscheinen.



Ein- und Ausgabe – 18

## Schreiben in eine Datei

- **telling/1** gibt an, in welche Datei die Ausgabe zur Zeit gerade geleitet wird.

- ▶ user steht für den Bildschirm, der als abstrakte Datei betrachtet wird.

```
?- telling(Zuerst),  
tell(hans), write(hallo),  
telling(Mitte),  
told,  
telling(Zuletzt).  
  
Zuerst = user,  
Mitte = hans,  
Zuletzt = user
```

Ein- und Ausgabe – 19

## Lesen aus einer Datei

- **see/1** nimmt die Eingabe für die Prädikate

- ◆ get/1, get0/1
- ◆ read/1

- aus einer Datei. (öffnet die Datei!)



- **seen/0**

- ◆ beendet die Umleitung (schliesst die Datei!)
- ◆ sorgt dafür, dass zukünftige Eingaben wieder vom Benutzer abgefragt werden.

```
?- see(hans),  
get0(_),  
get0(Y), put(Y), nl,  
seen.
```

a

Ein- und Ausgabe – 20

## Lesen aus einer Datei

- `seeing/1` gibt an, aus welcher Datei die Eingabe zur Zeit gerade genommen wird.
  - ▶ `user` steht für die Tastatur, die als abstrakte Datei betrachtet wird.

```
?- seeing(Zuerst),
   see(hans),
   seeing(Mitte),
   seen,
   seeing(Zuletzt).
Zuerst = user,
Mitte = hans,
Zuletzt = user
```

Ein- und Ausgabe – 21

## Das Dateieinde

- ◆ Wie kann beim Einlesen das Erreichen des Dateieendes erkannt werden?  
**Dateieinde wird als spezielles Element repräsentiert:**
  - **Eingabe mit ASCII-Codes**
    - ◆ `get_code/1` liefert die Zahl -1 zurück.
  - **Eingabe mit Termen**
    - ◆ `read/1` liefert das Atom `end_of_file` zurück.
- ◆ **Nachfrage: Wie wird beim Ausgeben das Dateieinde herausgeschrieben?**
  - ◆ Prolog macht das automatisch beim Beweis von `told/0`.

Ein- und Ausgabe – 22

## Verarbeitung eines Dateiinhalts

### Schema für Verarbeiten einer Datei mit Prolog:

```
process_file(F) :-
  see(F),           % Oeffne Datei
  repeat,
  read(T),          % Term einlesen
  process_term(T), % Term verarbeiten
  T = end_of_file, % Fertig, falls
  !,               % Dateieinde
  seen.            % Schliesse Datei
```

- ◆ `repeat/0` ist ein eingebautes Prädikat, das beliebig oft gelingt.

```
repeat.
repeat :- repeat.
```

Ein- und Ausgabe – 23

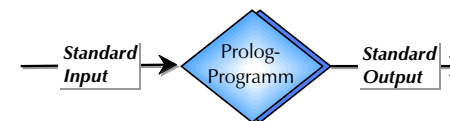
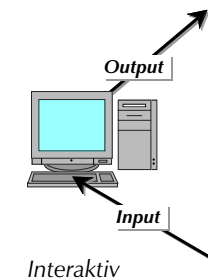
## Wer ist der user?

### Die abstrakten Dateien "user"

- ◆ liefern Input von der Tastatur
- ◆ schreiben Output auf den Bildschirm

### im interaktiven Betrieb!

*Aber:* Der Input kann auch von einem andern Programm kommen und an ein anderes ausgegeben werden!



Prozesskommunikation

Ein- und Ausgabe – 24