Die Programmiersprache C

- höhere Programmiersprache (mit einigen assemblerähnlichen Konstrukten)
  - gut verständliche Kommandos
  - muss von Compiler in maschinenlesbaren Code (Binärdatei) übersetzt werden
- universell, weit verbreitet
- relativ kleiner Sprachumfang
- UNIX ist in C geschrieben (Kern und die meisten Systemkommandos)
  - ➔UNIX-Systemprogrammierung in C
- 1970/71 aus dem Vorläufer B entwickelt
- viele moderne Sprachen eng an C angelehnt (z.B. C++, Java, C#)

Programm (Rückschau)

- ein Text (Code), der einen Algorithmus formuliert, so dass er auf einer Rechenanlage ausgeführt werden kann
- Ein Algorithmus ist eine Folge von Anweisungen, die Eingabedaten in Ausgabedaten überführt (intuitiver Algorithmenbegriff).

  Dabei muss bei jeder Eingabe eindeutig sein:

  - Welche Anweisung wird zuerst ausgeführt?
  - Welche Anweisung folgt auf eine gerade ausgeführte Anweisung?
  - In welchen Situationen ist der Algorithmus beendet?

Zwei Schlussfolgerungen

1. Ein C-Programm berechnet eine Funktion.

algorithmisch:	Eingabedaten	->Ausgabedaten
mathematisch:	Argumente	->Funktionswert
in C:	(aktuelle) Parameter	->Rückgabewert

2. Berechnung von Funktionen durch Abarbeitung einer Folge von Anweisungen.

➔C ist eine imperative Programmiersprache

Funktionen und C-Programme

- Funktionen können weitere Funktionen aufrufen, z.B.:
  `f(x) = sin(ln x)` vordefinierte Funktionen in richtiger Folge aufrufen
  `g(x) = √(f(x))` selbstdefinierte und Standardfunktion aufrufen
- Aufruf `g` mit Argument `x` : -> `g:` Aufruf `f` mit Argument `x`
  ➔ `f:` Rückgabewert `f(x)`
  -> `g:` Aufruf √ mit `f(x)`
  ➔ √: Rückgabewert √ `f(x)`
  -> `g:` Rückgabewert `g(x)` = √ `f(x)`
- Den Rückgabewert einer aufgerufenen Funktion erhält die aufrufende Funktion.

Struktur von C-Programmen

- C-Programm: Definition einer oder mehrerer Funktionen
  - vom Programm realisierte Funktion: `main()`
    - ➔ wird stets zuerst aufgerufen
  - ggf. weitere, aufzurufende Funktionen
- Häufig zu benutzende Funktionen (Standardfunktionen) sind in *Bibliotheksdateien* vordefiniert.
  - ➔ können eingebunden und dann aufgerufen werden
- Besonderheit: Den Rückgabewert von `main()` erhält das Programm, das das C-Programm aufruft
  - ➔ kann als Exit-Status interpretiert werden
  - ➔ `main()` gibt ganzzahligen Wert zurück

Ein erstes Programm

```
/* hello.c
*
* Ausgabe einer Zeichenkette auf stdout
*/

#include <stdio.h>     // Bibliotheksdatei einbinden

int main() {
    printf("Hello world!\n");
    return 0;          // Rueckgabewert 0
}
```

Erläuterungen zum ersten Programm

- Zeichen hinter // und zwischen /* und */ sind Kommentar
- `int main()`:
  - `()` zeigen (stets) an, dass es sich um eine Funktion handelt
  - **`int`** zeigt an, dass der Rückgabewert ganzzahlig ist
- `printf()`:
  - Aufruf einer Funktion zur formatierten Ausgabe auf `stdout`
  - ist Standardfunktion aus der Bibliotheksdatei (header-Datei) `stdio.h`
  - Parameter von `printf()` zwischen `():`
    - Anführungszeichen ➔ Zeichenkette;
    - `\n` ➔ Zeilenvorschub (newline)
- Kommandos und Funktionsaufrufe müssen mit `;` abgeschlossen werden

Vom Quellcode zum ausfürbaren Code

- Programmierer erstellt den Quellcode: beispiel.c
  - Anweisungsfolge in der Syntax der Sprache C
  - mit Text-Editor, ist „menschenlesbar“
- Erzeugung des maschinell ausführbaren Codes:

  `[g]cc [-Wall]` *beispiel*`.c [-o` *beispiel*`]`

  1. *Präprozessor* bereitet den Quellcode zur Übersetzung vor
     - kopiert Bibliotheksdateien (für den Übersetzungslauf) in den Quellcode,
     - Ersetzt „*Aliasnamen*“ im Quellcode u.ä.
  2. *Compiler* übersetzt in *Objektcode*: Befehlsfolgen für den Prozessor
  3. *Linker* verbindet mehrere Objektcode-Dateien zu einer Datei

     ➔ Funktionen haben überlappungsfreien Speicherbereich

Präprozessor-Anweisungen

- beginnen mit #
- enden nicht mit Semikolon
- Beispiel: `#include` *datei*
  - bindet `datei` für die Arbeit des Compilers (temporär) in den Quellcode ein
  - Funktionen, die in *datei* definiert sind, werden verfügbar
  - *datei* in Anführungszeichen: *datei* aus aktuellem Verzeichnis
  - *datei* in spitzen Klammern: *datei* aus Verzeichnis mit C-Bibliotheken (z.B. /usr/include)

Variablen

- dienen zum Speichern von Werten (Parameter, (Zwischen-)Ergebnisse etc.)
- Werte werden im Arbeitsspeicher abgelegt, können verändert werden
- werden über den Variablennamen angesprochen (gelesen/überschrieben)
- Anweisungen ändern Werte der Variablen,
  z.B. Überschreiben durch Wertzuweisung, z.B. `x = y - x;`
- müssen definiert werden, z.B. `int x;`
  - *Datentyp Variablenname*;
  - Reservierung genügend vieler Speicherzellen im Arbeitsspeicher (abhängig vom Datentyp)
- werden durch die erste Wertzuweisung initialisiert, z.B. `x=3;`

Datentypen

- Datentyp einer Variablen bestimmt
  - Darstellung (Repräsentation) der Werte im Arbeitsspeicher
    - Anzahl der Speicherzellen (Bytes) ➔ Wertebereich/Genauigkeit
    - Bedeutung der einzelnen Bits
  - erlaubte Operationen und deren Wirkung
- **einfache/elementare Datentypen**:
  - Ganzzahltypen `char`, `int`, `short`, `long`, `long long` und deren `unsigned` Typen (z.B. `unsigned int`)
  - Gleitpunktypen `float`, `double`, `long double`
- **abgeleitete Datentypen**: setzen sich aus anderen Datentypen zusammen
- sind als *Standardtypen* vordefiniert oder *selbst definierte Typen*

Elementare Ganzzahltypen

Datentyp	Bytes (z.B.)	Wertebereich (dezimal)
`char`	1	-128 ... +127 oder 0 ... 255
`signed char`	1	-128 ... +127
`unsigned char`	1	0 ... 255 (erweiterter ASCII-Satz)
`[signed] short [int]`	2	-32.768 ... +32.767
`unsigned short [int]`	2	0 ... +65.535
`[signed] int`	4	-2.147.483.648 ... +2.147.483.647
`unsigned int`	4	0 ... +4.294.967.295
`[signed] long [int]`	4	-2.147.483.648 ... +2.147.483.647
`unsigned long [int]`	4	0 ... +4.294.967.295
`[signed] long long`	8	-2^{63} ... $+2^{63} - 1$
`unsigned long long`	8	0 ... $+2^{64} - 1$

|char| < 2 ≤ |short| ≤ |int| ≤ 4 ≤ |long| ≤ |long long|

Elementare Gleitpunktypen

Datentyp	Bytes (z.B.)	Wertebereich (dezimal)
`float`	4	$-3,4 \cdot 10^{38}$ ... $+3.4 \cdot 10^{38}$
`double`	8	$-1.7 \cdot 10^{308}$ ... $+1.7 \cdot 10^{308}$
`long double`	10	$-1.1 \cdot 10^{4932}$ ... $+1.1 \cdot 10^{4932}$

- Wertebereiche: Gleitpunkt-Eigenschaft beachten!
- interne Darstellung: *Mantisse* $*$ 2 ^{Exponent} (nach IEEE 754)
- |float| ≤ |double| ≤ |long double|

Operatoren

- arithmetische und Vergleichsoperatoren wie in der C-Shell
  (s. Grundlagen von Betriebssystemen)
- Ausdrücke haben einen Rückgabewert
  ➔ können Teil eines Ausdrucks sein
- Inkrement und Dekrement in Präfix- und Suffixnotation:
  - **Präxnotation**: ++A bzw. --A
    ➔ Rückgabewert ist Inkrement bzw. Dekrement von A
    Nebeneffekt: Wert von A ist in- bzw. dekrementiert
  - **Postfixnotation**: A++ bzw. A--
    ➔Rückgabewert ist der Wert von A
    Nebeneffekt: Wert von A ist in- bzw. dekrementiert

  zum Vergleich:

  A + B gibt die Summe der Werte von A und B zurück; keine Nebeneffekte

Operatoren

- Zuweisungsoperatoren `+=, -=, *=, /=, %=` geben den Wert der Operation zurück und weisen als Nebeneffekt diesen Wert dem linken Ausdruck zu, z.B.:
  `x += 8` realisiert `x = x + 8`
- bitweise Operatoren, sonstige Operatoren, Assoziativitäten und Prioritäten

s. Literatur, z.B.:

M. Dausmann, U. Bröckl, J. Goll: C als erste Programmiersprache.
Teubner Verlag/GWV Fachverlage, Wiesbaden, 2008.

Formatiere Ausgabe mit `printf()`

- variable Anzahl von Parametern (Argumenten)
- erstes Argument wird ausgegeben (s. `printf("Hello world!\n")`)
- Argumente durch *Komma* voneinander getrennt
- erstes Argument kann auf Werte der weiteren Argumente zugreifen, z.B.:

```
int x = 42;
printf("%d\t%d\n", 1, x);   ➔ 1  42
```

  - `%d` ist ein Formatelement:
    der nächste, noch nicht verwendete Parameter wird an Stelle des %d als dezimale ganze Zahl ausgegeben
  - `\t` Tabulatorschritt

Formatelemente von `printf()`

`%d`	dezimale ganze Zahl
`%`*m*`d`	dezimale ganze Zahl, mindestens *m* Zeichen breit
`%f`	Gleitpunktzahl
`%`*m*`f`	Gleitpunktzahl, mindestens *m* Zeichen breit
`%:`*n*`f`	Gleitpunktzahl, n Nachkommastellen
`%`*m*`:`*n*`f`	Gleitpunktzahl, mind. *m* Zeichen inkl. *n* Nachkommastellen
`%o`	oktale ganze Zahl
`%x`	hexadezimale ganze Zahl
`%c`	einzelnes Zeichen (Datentyp `char`)

(mehr auf der Manpage `man printf`)

Formatelemente von `printf()`

`%d`	dezimale ganze Zahl
`%`*m*`d`	dezimale ganze Zahl, mindestens *m* Zeichen breit
`%f`	Gleitpunktzahl
`%`*m*`f`	Gleitpunktzahl, mindestens *m* Zeichen breit
`%:`*n*`f`	Gleitpunktzahl, n Nachkommastellen
`%`*m*`:`*n*`f`	Gleitpunktzahl, mind. *m* Zeichen inkl. *n* Nachkommastellen
`%o`	oktale ganze Zahl
`%x`	hexadezimale ganze Zahl
`%c`	einzelnes Zeichen (Datentyp `char`)

(mehr auf der Manpage `man printf`)