C Grundlagen

- Programmierung
- Programmerstellung und -ausführung
- Organisation eines Rechners
  - ◆ Klassischer Universalrechner
  - ◆ Datendarstellung im Rechner
  - ◆ Dateien
  - ◆ Benutzerumgebung
  - ◆ Programmierumgebung
- Begriffe

C.1 Programmierung

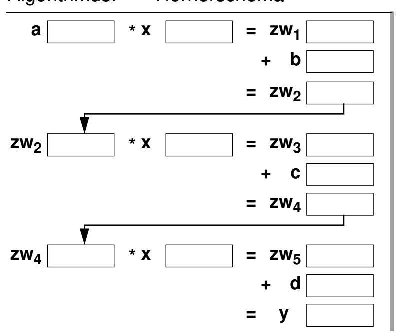
- Einsatz von Rechenanlagen
  - ◆ Informationen speichern, strukturieren, sortieren, zugänglich machen
  - ◆ Rechnen an sich ist eher selten
- Problemlösung mit Rechenanlagen
  - ◆ Programmierung

C.2 Programm

- Daten: Abstrakte Darstellung des problemrelevanten Teils der realen Welt
- Algorithmus: Schrittfolge von Operationen auf den Daten, die als allgemeines Verfahren die Lösung einer Klasse von Problemen zum Ziel hat
  - endliche Beschreibung
  - eindeutige Operationsfolge
    es steht stets fest, welches der nächste Schritt ist
  - Problemklasse: z. B. größter gemeinsamer Teil von zwei beliebigen (!) natürlichen Zahlen

C.3 Beispiel (1) — Polynomberechnung

- Problemklasse: Berechnung von $f(x) = ax^3 + bx^2 + cx + d$
- Daten: Parameter $a, b, c, d, zw_{1,2,3,4} \in \mathbb{R}$
- Algorithmus: Hornerschema



Schritte:

1. $a * x \rightarrow zw_1$
2. $zw_1 + b \rightarrow zw_2$
3. $zw_2 * x \rightarrow zw_3$
4. $zw_3 + c \rightarrow zw_4$
5. $zw_4 * x \rightarrow zw_5$
6. $zw_5 + d \rightarrow y$

C.4 Beispiel (1) — Programm

```
main()
{
  /* Beschreibung der Daten */
  float x;                       /* Argument zu dem Pol. berechnet wird */
  float y;                       /* Polynomwert */

  float a = 11, b = 7, c = 5, d = 3;
  float zw1, zw2, zw3, zw4, zw5;      /* Zwischenergebnisse */

  /* Einlesen des Arguments */
  printf("Argument = ");
  scanf("%f", &x);

  /* Kontrollausgabe der Eingabedaten */
  printf("Der Polynomwert soll fuer das");
  printf(" Argument %f berechnet werden.\n", x);

  /* Berechnung des Polynomwertes mit Hornerschema */
  zw1 = a * x;
  zw2 = zw1 + b;
  zw3 = zw2 * x;
  zw4 = zw3 + c;
  zw5 = zw4 * x;
  y = zw5 + d;

  /* Ausgabe des Ergebnisses */
  printf("Ergebnis: f(%f) = %f\n", x, y);
}
```

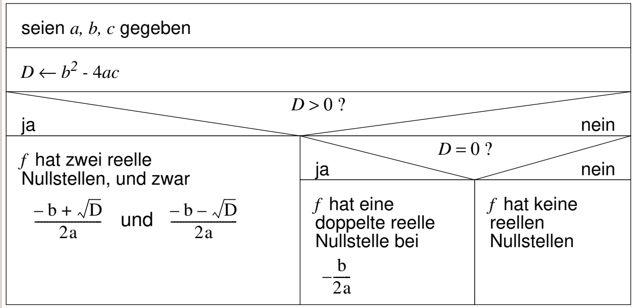

C.5 Beispiel (2) — Nullstellen einer Funktion

- Problemklasse: Was sind die reellen Nullstellen der Funktion f: $\mathbb{R} \rightarrow \mathbb{R}$ mit $f(x) = ax^2 + bx + c$
- Daten: Parameter $a, b, c \in \mathbb{R}$, Diskriminante D
- Algorithmus:
  1. seien a, b, c gegeben
  2. berechne die Diskriminante D zu $b^2 - 4ac$
  3. wenn $D > 0$ weiter mit Schritt 7
  4. wenn $D = 0$ weiter mit Schritt 9
  5. f hat keine reellen Nullstellen
  6. fertig
  7. f hat zwei reelle Nullstellen, und zwar $\frac{-b + \sqrt{D}}{2a}$ und $\frac{-b - \sqrt{D}}{2a}$
  8. fertig
  9. f hat eine doppelte reelle Nullstelle bei $-\frac{b}{2a}$
  10. fertig

C.6 Beispiel (2) — Struktogramm

- einfache Dokumentation des Programmentwurfs

Nullstellen — bestimmt reelle Nullstellen von $f(x) = ax^2 + bx + c$

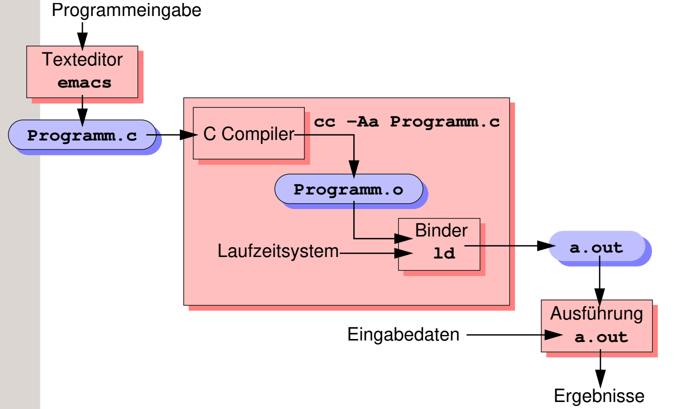


C.7 Beispiel (2) — Programm

```
#include <stdio.h>
#include <math.h>

main ()
{
  float a, b, c;                         /* Parameter in a*x^2+b*x+c */
  float dis;                             /* Diskriminante */

  /* Initialisierung: Einlesen der Parameter */
  printf("Nullstellen von f(x)=a*x^2+b*x+c\n");
  printf("a=");
  scanf("%f", &a);
  printf("b=");
  scanf("%f", &b);
  printf("c=");
  scanf("%f", &c);

  dis = b*b - 4.0 * a * c;               /* Berechnung der Diskriminante */

  /* Berechnung der Nullstellen */
  if ( dis == 0.0 )                      /* doppelte Nullstelle */
    printf("Doppelte Nullstelle bei %f\n", -b/(2.0*a));
  else if ( dis > 0.0 ) {                /* zwei reelle Nullstellen */
    printf("1. Nullstelle bei %f\n", (sqrt(dis)-b)/(2.0*a));
    printf("2. Nullstelle bei %f\n", (-sqrt(dis)-b)/(2.0*a));
  }
  else                                   /* keine reelle Nullstelle */
    printf("Keine reelle Nullstelle\n");
}
```

C.8 Programmerstellung und -ausführung

1 Vorgehensweise



2 einfaches Beispiel

- Programm

```
main()
{
  printf("hello world\n");
}
```

- Übersetzung und Ausführung

```
faui06a% cc hello.c
faui06a% a.out
hello world
faui06a%
```


3 Übersetzung und Ausführung der Beispielprogramme

- Ausführung des Beispiels 'Polynom'

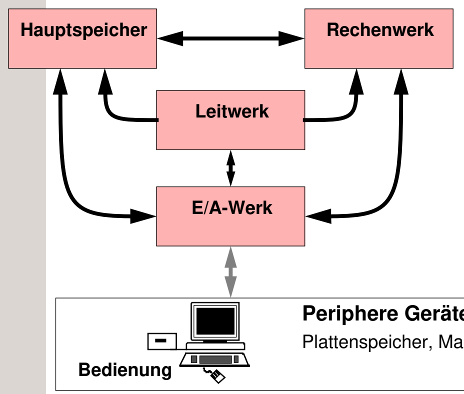
```
faui06a% cc -Aa polynom.c
faui06a% a.out
Argument = 4
Der Polynomwert soll fuer das Argument 4.000000 berechnet
werden.
Ergebnis: f(4.000000) = 839.000000
faui06a%
```

- Ausführung des Beispiels 'Nullstellen'

```
faui06a% cc -Aa Nullstellen.c -lm
faui06a% a.out
Nullstellen von f(x)=a*x^2+b*x+c
a=1
b=2
c=-3
1. Nullstelle bei 1.000000
2. Nullstelle bei -3.00000
faui06a%
```


C.9 Klassischer Universalrechner

- *Burks*, *Goldstine* und *von Neumann* (Princeton 1946/47)



Rechenwerk:
Rechenoperationen und logische Verknüpfungen

Hauptspeicher:
Programm und Daten

Leitwerk:
steuert Programmablauf

E/A-Werk:
Ein-/Ausgabe von Daten und Programmen

Periphere Geräte
Plattenspeicher, Magnetbänder, Drucker, ...

C.10 Datendarstellung im Rechner

- Alle Informationen (Daten, Texte, Programmanweisungen) werden im Speicher durch Bitfolgen dargestellt
  - ◆ Bit = Speicherstelle mit Wert 0 oder 1
  - ◆ Beispiele: Zahl **2** = **10**, Zahl 5 = **101**, Buchstabe **a** = Zahl **97** = **1100 0001**
- 8 Bit bilden ein Byte
- Für ganze Zahlen werden meist 32 Bit vorgesehen (ein Wort)
- Zeichen (Buchstaben) werden durch Zahlen kodiert
- Ein Wort ist normalerweise die Einheit, mit der das Rechenwerk operiert
- Das Rechenwerk führt entsprechend den Anweisungen des Programms arithmetische oder logische Verknüpfungen von Worten durch
- Die Worte auf denen das Rechenwerk operiert, werden in Registern des Rechenwerks zwischengespeichert (für schnelleren Zugriff)

C.11 Dateien und Dateisystem

1 Allgemeines

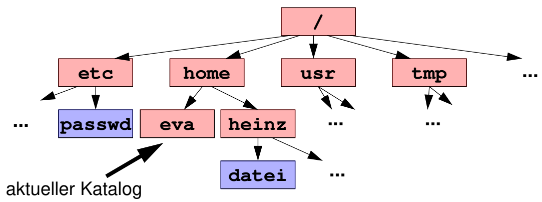
- Dateien sind abstrakte Gebilde zur Speicherung von Daten auf einem Hintergrundspeicher (Festplatte, Diskette)
  - ◆ Verbergen die reale Struktur (Sektoren, Zylinder, etc.)
  - ◆ Haben einen Namen (Zeichenkette)
  - ◆ UNIX verwaltet Zugriffsrechte
    (welcher Benutzer darf lesen, schreiben
    oder ein Programm in der Datei ausführen)

2 Struktur des Dateisystems

- Hierarchische Strukturierung des Dateisystems
  - ◆ reguläre Dateien
    - Können beliebige Daten speichern
      (Texte, Zahlen, Programme, …)
    - Einfache Bedienung (öffnen, lesen, schreiben)
  - ◆ Dateikataloge *(Directories)*
    - Enthalten reguläre Dateien oder weitere Kataloge
  - ➥ Baumförmige Struktur des Dateisystems
  - ◆ Dateien werden durch Aneinanderreihung der Katalognamen und des Dateinamens eindeutig benannt

3 Pfade im Dateisystem

- Baumstruktur & Pfade im Dateibaum



  - ◆ z. B. **/home/heinz/datei** **/tmp** **../heinz/datei**
  - ◆ **/** ist Trennsymbol *(Slash)* zwischen Pfadkomponenten
    Pfade, die mit **/** beginnen, starten im Wurzelkatalog *(root)*
    sonst Beginn implizit im aktuellem Katalog
  - ◆ jeder Katalog enthält Verweise auf sich selbst (**.**) und auf den darüberliegenden Katalog (**..**) — in Pfadnamen verwendbar!
  - ◆ aktueller Katalog kann mit Kommando **cd** gewechselt werden
    z. B. **cd ../heinz**

C.12 Benutzerumgebung

- Das Betriebssystem (UNIX) und eine Reihe von Dienstprogrammen *(Tools)* stellen eine Umgebung bereit, die von der eigentlichen Rechnerarchitektur abstrahiert
- Jedem Benutzer ist ein eigener Dateikatalog zugeordnet *(Home-Directory)*
- Nach dem *Login* wird automatisch eine Fensteroberfläche mit mehreren Fenstern gestartet *(X-Windows)*
  - ◆ Weitere Fenster sind über Menüs mit der Maus zu starten
- Jedes Fenster entspricht einem virtuellen Terminal
  - ◆ Kommandointerpreter (Shell) zur Eingabe von Kommandos (C-Compiler, Dateikatalog auflisten, …)
  - ◆ Editor (Emacs) zur Eingabe von Programmen und Texten
  - ◆ …

C.12 Benutzerumgebung (2)

- Beschreibung über die Benutzerumgebung und alle Kommandos sind online auf dem Rechner abrufbar
  - ◆ Integriert in World-Wide-Web (WWW)
  - ◆ Tool: *Mosaic* (über Menü starten, dann selbsterklärend)

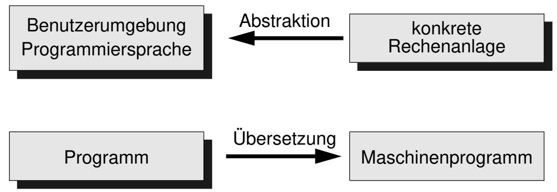
C.13 Programmierumgebung

- Programmiersprache bietet Sicht auf eine abstrakte Rechenanlage
  - ◆ Statt Bits, Bytes und Worten:
    Ganze und reelle Zahlen, Zeichen
  - ◆ Statt einfachen Abfragen und direkten Sprüngen an Adressen:
    if - then - else, Schleifen, …

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.13 Programmierumgebung (2)

- Abbildung Programmiersprache → Maschinensprache
  - ◆ Interpreter: Ein Dienstprogramm interpretiert das Programm des Anwenders
  - ◆ Compiler: Ein Dienstprogramm übersetzt das Anwenderprogramm in Maschinensprache



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.14 Begriffe

Programm:
Eine in einer Programmiersprache abgefaßte Verarbeitungsvorschrift mit einer (evtl. impliziten) Vereinbarung der Daten

Maschinensprache:
Programmiersprache, die nur Anweisungen zuläßt, die von einer bestimmten Rechenanlage (unmittelbar) ausgeführt werden können

Maschinenprogramm:
Ein in Maschinensprache abgefaßtes Programm

Maschinenorientierte Programmiersprache:
Programmiersprache, deren Anweisungen ähnliche/gleiche Struktur haben wie die der Maschinensprache einer bestimmten Rechenanlage

Maschinenorientiertes Programm:
Ein in maschinenorientierter Programmiersprache abgefaßtes Programm

Höhere / problemorientierte Programmiersprache:
Programmiersprache, die dazu dient, Programme eines bestimmten Anwendungsbereiches unabhängig von einer bestimmten Rechenanlage abzufassen, und die dem Anwendungsbereich besonders angemessen ist (FORTRAN, PASCAL, COBOL, C, C++, ...).

C - Programm:
Ein in der Programmiersprache C abgefaßtes Programm

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.