TECHNISCHE UNIVERSITÄT DORTMUND

Wintersemester 2006/07

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

(alias **Einführung in die Programmierung)**

(Vorlesung)

Prof. Dr. Günter Rudolph

Fachbereich Informatik

Lehrstuhl für Algorithm Engineering

Inhalt

- Funktionen
  - mit / ohne Parameter
  - mit / ohne Rückgabewerte

  } ☑

- Übergabemechanismen
  - Übergabe eines Wertes
  - Übergabe einer Referenz
  - Übergabe eines Zeigers

  } ☑

- Programmieren mit Funktionen → heute

  \+ static / inline / MAKROS

Variation der Aufgabe:

Finde Index des 1. Minimums in einem Array von Typ `double`
Falls Array leer, gebe **-1** zurück.

Entwurf mit Implementierung:

```
int imin(unsigned int const n, double a[]) {
  // leeres Array?
  if (n == 0) return -1;
  // Array hat also mindestens 1 Element!
  int i, imin = 0;
  for(i = 1; i < n; i++)
    if (a[i] < a[imin]) imin = i;
  return imin;
}
```

Neue Aufgabe:

Sortiere Elemente in einem Array vom Typ `double`.
Verändere dabei die Werte im Array.

Bsp:

8	44	14	81	12	
8	44	14	81	12	min{ 8, 44, 14, 81 } = 8 < 12 ?
12	44	14	81	8	ja → tausche 8 und 12
12	44	14	81	8	min{ 12, 44, 14 } = 12 < 81 ?
81	44	14	12	8	ja → tausche 12 und 81
81	44	14	12	8	min{ 81, 44 } = 44 < 14 ?
81	44	14	12	8	nein → keine Vertauschung
81	44	14	12	8	min{ 81} = 81 < 44 ?
81	44	14	12	8	nein → keine Vertauschung
81	44	14	12	8	fertig!

Neue Aufgabe:

Sortiere Elemente in einem Array vom Typ **double**.
Verändere dabei die Werte im Array.

Mögliche Lösung:

```
void sortiere(const unsigned int n, double a[]) {
  int i, k;
  for (k = n – 1; k > 1; k--) {
    i = imin(k - 1, a);
    if (a[i] < a[k]) swap_dbl(a[i], a[k]);
  }
}
```

```
void swap_dbl(double &a, double &b) {
  double h = a; a = b; b = h;
}
```

Wir halten fest:

- Arrays sind statische Datenbehälter: ihre Größe ist nicht veränderbar!
- Die Bereichsgrenzen von Arrays sollten an Funktionen übergeben werden, wenn sie nicht zur Übersetzungszeit bekannt sind.
- Die Programmierung mit Arrays ist unhandlich!
  Ist ein Relikt aus C. In C++ gibt es handlichere Datenstrukturen!
  (Kommt bald … Geduld!)
- Die Aufteilung von komplexen Aufgaben in kleine Teilaufgaben, die dann in parametrisierten Funktionen abgearbeitet werden, erleichtert die Lösung des Gesamtproblems. Beispiel: Sortieren!
- Funktionen für spezielle kleine Aufgaben sind wieder verwertbar und bei anderen Problemstellungen einsetzbar.

  ⇒ Deshalb gibt es viele Funktionsbibliotheken, die die Programmierung erleichtern!

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

`exp()`	Exponentialfunktion e^x
`ldexp()`	Exponent zur Basis 2, also 2^x
`log()`	natürlicher Logarithmus $\log_e x$
`log10()`	Logarithmus zur Basis 10, also $\log_{10} x$
`pow()`	Potenz x^y
`sqrt()`	Quadratwurzel
`ceil()`	nächst größere oder gleiche Ganzzahl
`floor()`	nächst kleinere oder gleiche Ganzzahl
`fabs()`	Betrag einer Fließkommazahl
`modf()`	zerlegt Fließkommazahl in Ganzzahlteil und Bruchteil
`fmod()`	Modulo-Division für Fließkommazahlen

und zahlreiche trigonometrische Funktionen wie `sin`, `cosh`, `atan`

```
#include <stdlib.h>
```

`atof()`	Zeichenkette in Fließkommazahl wandeln
`atoi()`	Zeichenkette in Ganzzahl wandeln (**A**SCII **to** **i**nteger)
`atol()`	Zeichenkette in lange Ganzzahl wandeln
`strtod()`	Zeichenkette in `double` wandeln
`strtol()`	Zeichenkette in `long` wandeln
`rand()`	Liefert eine Zufallszahl
`srand()`	Initialisiert den Zufallszahlengenerator

und viele andere …

Wofür braucht man diese Funktionen?

Funktion `main` (→ Hauptprogramm)

wir kennen:

```
int main() {
  // ...
  return 0;
}
```

allgemeiner:

```
int main(int argc, char *argv[]) {
  // ...
  return 0;
}
```

argc: Anzahl der Elemente

argv: Array von Zeichenketten

Programmaufruf in der Kommandozeile:

```
D:\> mein_programm 3.14 hallo 8
```

argv[0] (mein_programm) argv[1] (3.14) argv[2] (hallo) argv[3] (8)

Alle Parameter werden textuell als Zeichenkette aus der Kommandozeile übergeben!

`argc` hat Wert 4

Funktion `main` (→ Hauptprogramm)

Programmaufruf in der Kommandozeile:

```
D:\> mein_programm 3.14 hallo 8
```

Alle Parameter werden **textuell** als Zeichenkette aus der Kommandozeile übergeben!

```
#include <stdlib>

int main(int argc, char *argv[]) {

  if (argc != 4) {
    cerr << argv[0] << ": 3 Argumente erwartet!" << endl;
    return 1;
  }
  double dwert = atof(argv[1]);
  int iwert = atoi(argv[3]);

  // ...

}
```

```
#include <ctype.h>
```

`tolower()`	Umwandlung in Kleinbuchstaben
`toupper()`	Umwandlung in Großbuchstaben
`isalpha()`	Ist das Zeichen ein Buchstabe?
`isdigit()`	Ist das Zeichen eine Ziffer?
`isxdigit()`	Ist das Zeichen eine hexadezimale Ziffer?
`isalnum()`	Ist das Zeichen ein Buchstabe oder eine Ziffer?
`iscntrl()`	Ist das Zeichen ein Steuerzeichen?
`isprint()`	Ist das Zeichen druckbar?
`islower()`	Ist das Zeichen ein Kleinbuchstabe?
`isupper()`	Ist das Zeichen ein Großbuchstabe?
`isspace()`	Ist das Zeichen ein Leerzeichen?

Beispiele für nützliche Hilfsfunktionen:

<u>Aufgabe:</u> Wandle alle Zeichen einer Zeichenkette in Grossbuchstaben!

```
#include <ctype.h>

char *ToUpper(char *s) {
  char *t = s;
  while (*s != 0) *s++ = toupper(*s);
  return t;
}
```

<u>Aufgabe:</u> Ersetze alle nicht druckbaren Zeichen durch ein Leerzeichen.

```
#include <ctype.h>

char *MakePrintable(char *s) {
  char *t = s;
  while (*s != 0) *s++ = isprint(*s) ? *s : ‘ ‘;
  return t;
}
```

```
#include <time.h>
```

`time()`	Liefert aktuelle Zeit in Sekunden seit dem 1.1.1970 UTC
`localtime()`	wandelt UTC-„Sekundenzeit“ in lokale Zeit (`struct`)
`asctime()`	wandelt Zeit in `struct` in lesbare Form als `char[]`

und viele weitere mehr …

```
#include <iostream>
#include <time.h>

int main() {
  time_t jetzt = time(0);
  char *uhrzeit = asctime(localtime(&jetzt));
  std::cout << uhrzeit << std::endl;
  return 0;
}
```

engl. FSM: finite state machine

Der DEA ist zentrales Modellierungswerkzeug in der Informatik.

Definition

Ein <u>deterministischer endlicher Automat</u> ist ein 5-Tupel $(S, \Sigma, \delta, F, s_0)$, wobei

- S eine endliche Menge von Zuständen,
- Σ das endliche Eingabealphabet,
- $\delta: S \times \Sigma \rightarrow S$ die Übergangsfunktion,
- F eine Menge von Finalzuständen mit $F \subseteq S$ und
- s_0 der Startzustand. ■

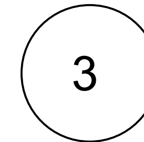
Er startet immer im Zustand s_0, verarbeitet Eingaben und wechselt dabei seinen Zustand. Gelangt er in einen Endzustand aus F, dann terminiert er.

$\Rightarrow$ Beschreibung eines Programms!

Grafische Darstellung

Zustände als Kreise

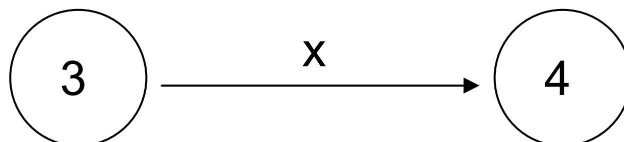
im Kreis der Bezeichner des Zustands (häufig durchnummeriert)



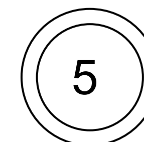
Übergänge von einem Zustand zum anderen ist abhängig von der Eingabe.

Mögliche Übergänge sind durch Pfeile zwischen den Zuständen dargestellt.

Über / unter dem Pfeil steht das Eingabesymbol, das den Übergang auslöst.

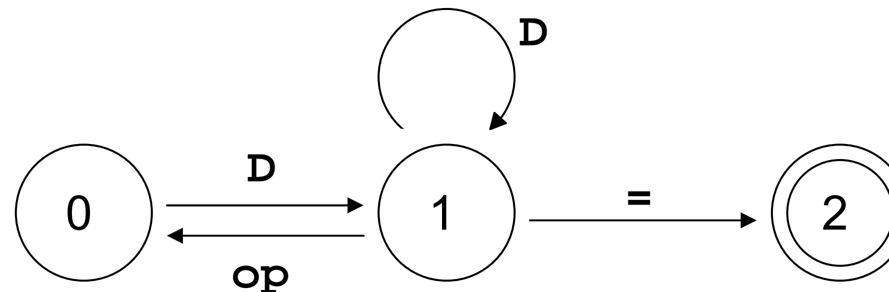


Endzustände werden durch „Doppelkreise“ dargestellt.



Beispiel:

Entwerfe DEA, der arithmetische Ausdrücke ohne Klammern für nichtnegative Ganzzahlen auf Korrektheit prüft.



Zustände S = { 0, 1, 2 }

Startzustand s_0 = 0

Endzustände F = { 2 }

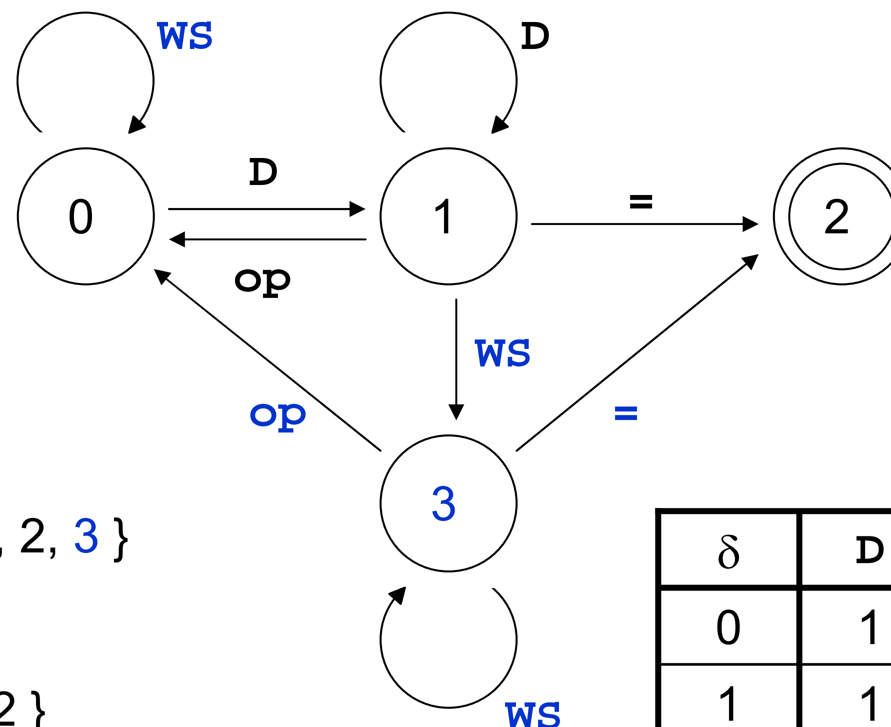
Eingabealphabet Σ = { D, op, = }

δ	D	op	=
0	1	-1	-1
1	1	0	2
2	-	-	-

-1: Fehlerzustand

Beispiel:

Erweiterung: Akzeptiere auch „white space“ zwischen Operanden und Operatoren



Zustände S = { 0, 1, 2, 3 }

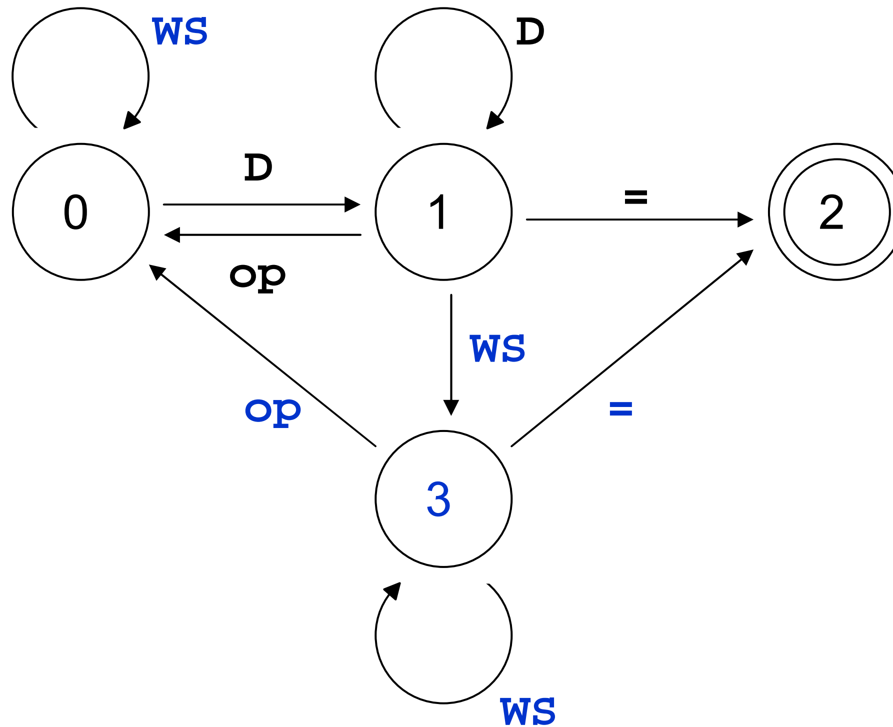
Startzustand $s_0 = 0$

Endzustände F = { 2 }

Eingabealphabet Σ = { D, op, =, WS }

δ	D	op	=	WS
0	1	-1	-1	0
1	1	0	2	3
2	-	-	-	-
3	-1	0	2	3

Exkurs: Deterministische endliche Automaten (DEA)



Eingabe:

```
3+ 4 - 5=
```

Zustand 0, lese D →
Zustand 1, lese op →
Zustand 0, lese WS →
Zustand 0, lese D →
Zustand 1, lese WS →
Zustand 3, lese op →
Zustand 0, lese WS →
Zustand 0, lese D →
Zustand 1, lese = →
Zustand 2 (Endzustand)

Exkurs: Deterministische endliche Automaten (DEA)

Wenn **grafisches Modell** aufgestellt, dann Umsetzung in ein **Programm**:

- Zustände durchnummeriert: 0, 1, 2, 3
- Eingabesymbole: z.B. als `enum { D, OP, IS, WS }` (`IS` für =)
- Übergangsfunktion als Tabelle / Array:

```
int GetState[][4] = {
  {  1, -1, -1,  0 },
  {  1,  0,  2,  3 },
  {  2,  2,  2,  2 },
  { -1,  0,  2,  3 }
};
```

Array enthält die gesamte Steuerung des Automaten!

- Eingabesymbole erkennen u.a. mit: `isdigit(), isspace()`

```
bool isbinop(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}
```

Exkurs: Deterministische endliche Automaten (DEA)

```
enum TokenT { D, OP, IS, WS, ERR };
```

```
bool Akzeptor(char const* input) {
  int state = 0;
  while (*input != '\0' && state != 2 && state != -1) {

    char s = *input++;
    TokenT token = ERR;
    if (isdigit(s)) token = D;
    if (isbinop(s)) token = OP;
    if (s == '=')   token = IS;
    if (isspace(s)  token = WS;
    state = (token == ERR) ? -1 : GetState[state][token];

  }

  return (state == 2);

}
```