Beispiel aus der Computergrafik – zugleich **Gegenbeispiel**:

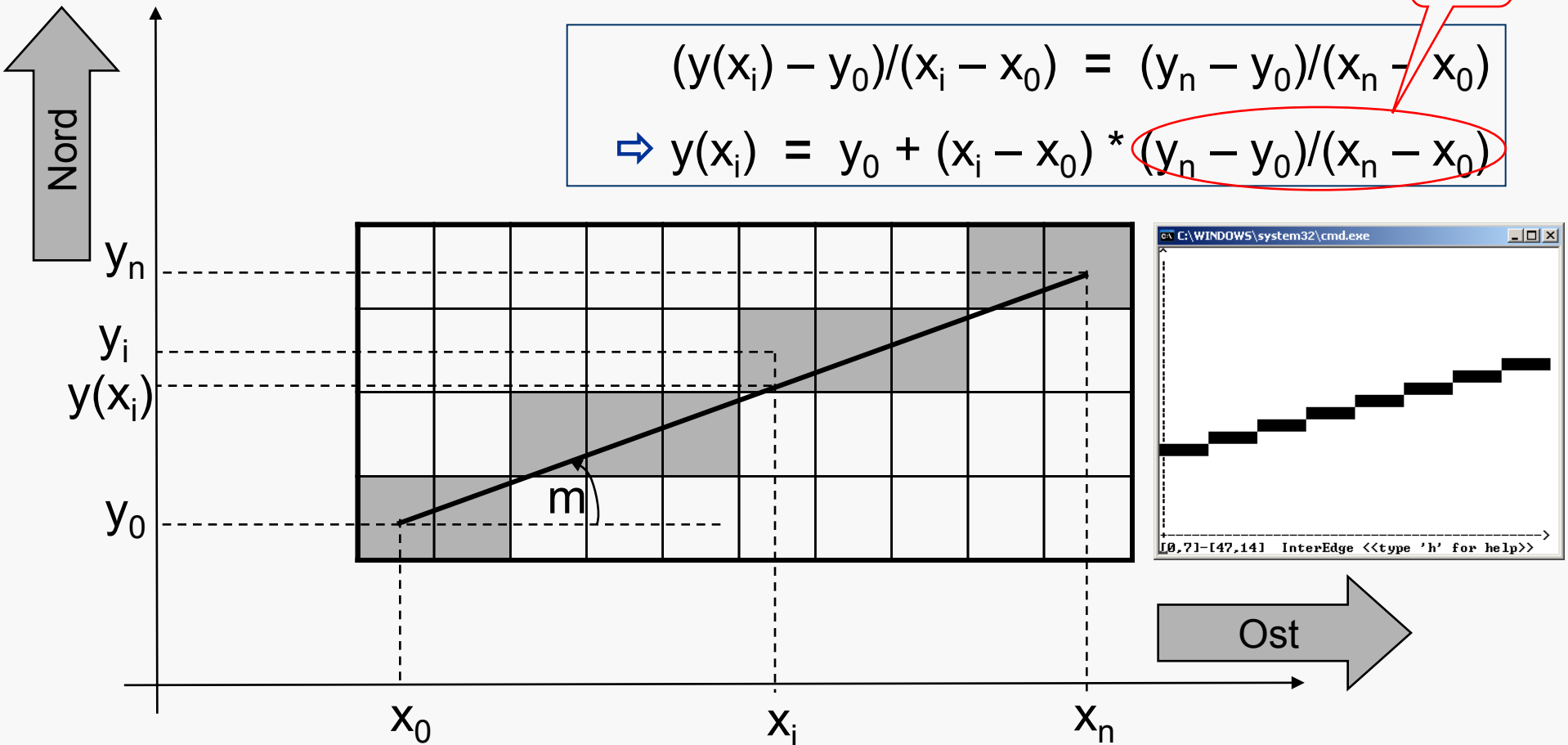
Stationen in der Entwicklung des Linienalgorithmus

Zeichnen gerader Linien mit Steigung ≤ 45° (m ≤ 1)

m

$$(y(x_i) - y_0)/(x_i - x_0) = (y_n - y_0)/(x_n - x_0)$$

$$\Rightarrow y(x_i) = y_0 + (x_i - x_0) * (y_n - y_0)/(x_n - x_0)$$



Algorithmen

O(n)

Naive Codierung der Geradengleichung:

```
void BruteForceLine
(int x0, int y0,
 int xn, int yn)


{ int   x=x0;
  float y=(float)y0;
  if (xn == x0) return;
  for (x=x0; x <= xn; x++)
 { y = y0
  +(x-x0)*(yn-y0)/(xn-x0);
   SetPixel(x,(int)y);
 }
 return;
}
```

O(n)

Digital Differential Analyzer:

```
void dda
(int x0, int y0,
 int xn, int yn)


{ int   x=x0;
  float y=(float)y0, m;
 m=(float)(yn-y0)/(xn-x0);
  for(x=x0;x<=xn;x++,y+=m)
 { SetPixel(x,(int)y);
 } return;
}
```

t nsec / floatOp * *f* floatOps / Umrißpixel
* *p* Umrißpixel / Dreieck * *d* Dreiecke / Szene

ergeben, falls: *t*=10; *f*=1; *p*=100; *d*=100.000

eine (unnötige) Wartezeit von 0,1 sec / Szene **!**

Datenstrukturen

Bisher erwähnt (s.o.):

Abhängigkeit der Wirksamkeit und der Laufzeit von Algorithmen u.a. von der Struktur der Daten

Allgemein:

Struktur heißt die Anordnung der Teile eines Ganzen zueinander.

Der Begriff steht sowohl für die Form (das Muster) der Zusammenstellung, als auch für das dabei entstehende Gebilde.

Beispiele:

Web-, Vereins-, Gesellschafts-, Denkstruktur

Datenstrukturen

Abstrakt:

- Struktur ist ein Objekt,

  (d.h.: unterscheidbares Ganzes, ein „Etwas“ mit eigener Identität, z.B.: Strickmuster, Zuschnitt -nicht: „Gegenstand“-)
- aufgebaut aus Komponenten

  (z.B.: aus Gegenständen, Größen, Ideen),
- auf eine geordnete, charakteristische Weise.

  (d.h.: mit wiedererkennbaren Gesetzmäßigkeiten)

Beispiele:

- Soldaten-Ansammlung vs. Militärparade
- Haar- / Hefezopf, Versmaß, Reim
- Hierarchie als Organisations-, Zentralismus als Verwaltungsprinzip

Datenstrukturen

Informell:

Eine Datenstruktur ist eine Struktur, deren Komponenten Datenobjekte sind.

In der Informatik – genauer:

Eine **Datenstruktur** ist

- eine Sammlung von Daten,
- die Beziehung unter ihnen sowie
- die Funktionen oder Operationen, die auf sie angewandt werden können.

Bei Abwesenheit (oder Unklarheit bzgl.) eines dieser drei Charakteristika handelt es sich bei der untersuchten Struktur nicht um eine Datenstruktur.

[E.D.Reilly (Ed.) in „Encyclopedia of Computer Science“, IEEE Press 1993]

Beispiel:

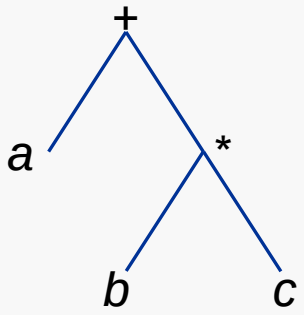
Die Zeichenfolge „*a* + *b* * *c*“ kann gemeint sein:

- (…) als Zeichenkette (String), bestehend aus ASCII-Zeichen, deren „Beziehung“ darin besteht, in dieser Reihenfolge Teil eines Textes zu sein. Eine auf diese Datenobjekte zulässige Operation ist z.B. die Verkettung (engl. *concatenation*), d.h. das Aneinanderhängen solcher Komponenten zu einer längeren (Text-)Struktur
- (…) als Teil eines mathematischen Ausdrucks, dessen alphanumerische Zeichen (a-z, 0-9) in einem arithmetischen Verhältnis zueinander stehen und die Sonderzeichen ‚+‘ und ‚*‘ als mathematische Operatoren für die Operationen Addition und Multiplikation zu verstehen sind. Eine solche Operationsfolge läßt sich wie ein Binärbaum darstellen.

Beispiel (Forts.)

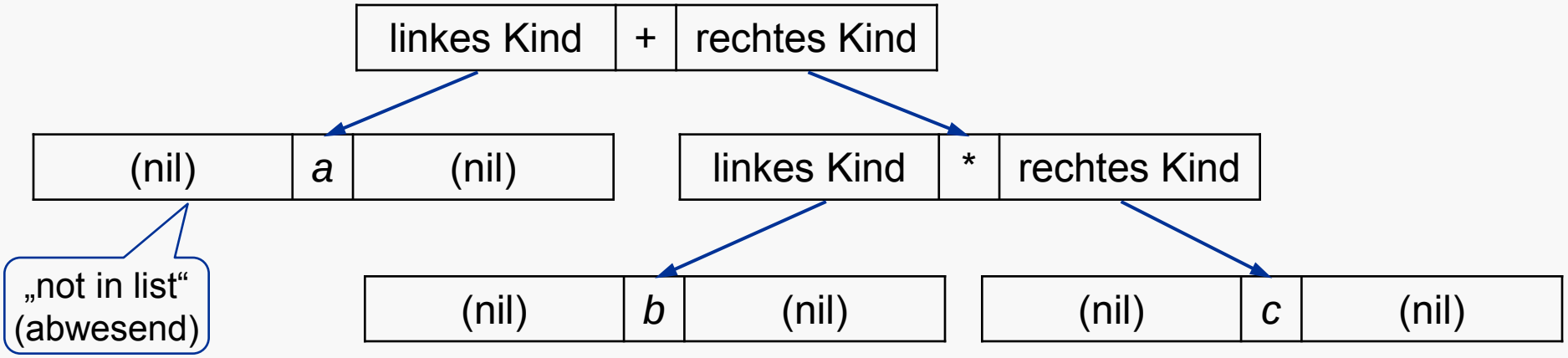
- Darstellung mathematischer Ausdrücke (z.B. *a* + *b* * *c*) als Binärbaum mit den Operatoren (+, *) als Eltern- und den Operanden (*a*, *b*, *c*) als Kindknoten:

  (Visualisierung der logischen Beziehungen)



- Baumstruktur der Speicherung am Rechner:

  5 Speicherzellen für je 3 Komponenten
  (1 Operator, 2 Operanden):



(Operatoren u. Operanden werden auch atomare o. elementare Objekte genannt.)

Datenstrukturen

Am häufigsten verwendete Datenstrukturen:

- **Feld** (*array*): mehrere indizierte Größen desselben Typs, die unter einheitlichem Namen und einem/mehreren individuellen Index/Indizes abrufbar u. im Arbeitsspeicher alle hintereinander abgelegt sind.

  (Mehrdimensionale Felder haben in allen Dimensionen gleich viele Elemente: sie sind „kartesisch“, „rechtwinklig“)

  Anwendung z.B.: Vektoren, Matrizen

- **Datensatz** (*record*): Dateneinheit aus mehreren Elementen mit jeweils eigenem Bezeichner und evtl. von unterschiedlichem Datentyp; Zugriff auf die einzelnen Elemente über ihre Bezeichner.

  Anwendung z.B.: Mitgliedslisten mit Personen- und Straßennamen, Haus- und Tel.-Nr., div. Kennungen;

  Avatar mit 3D-Eckpunkten, Flächen- / Farbcodierung, …

Häufigste Datenstrukturen (Forts.):

- **Menge** (*set*): Größen, die durch Zugehörigkeit zu begrifflichen Mengen gekennzeichnet werden; sie werden durch Operationen wie „hinzufügen“ oder „entfernen“, Bildung von „Vereinigung“, „Durchschnitt“ etc. verknüpft.

  Anwendung z.B.: Verwaltung von Buchsammlungen; Technik nur in „sehr hohen“ Hochsprachen verfügbar

- **Verkettete Liste** (*linked list*): (ggf. unterschiedliche) Datenelemente, die in eine Reihe gesetzt werden, indem jedes von ihnen die Zugriffsinformation (Adresse) des nächsten als Datenelement enthält. **Einfache** (doppelte) Verkettung liegt vor, wenn Listenelemente die Adressen ihrer Nachfolger (und Vorgänger) enthalten; sie heißt **zyklisch**, wenn das letzte Element auf das erste „zeigt“.

  Anwendung z.B.: Codierung von Schnitzeljagd-Etappen

  (**Graph**: „Liste der Listen“, mit zyklischen Verbindungen)

Häufigste Datenstrukturen (Forts.):

- **Baum** (*tree*): Verkettete Liste ohne zyklischen Teil, mit genau einem Element, der „Wurzel“ (*root*), ohne Vorgänger („Eltern“, „Vater“). Keine zwei Elemente haben denselben Nachfolger („Kind“, „Sohn“). Elemente ohne Nachfolger heißen „Blätter“ (*leaves*). Die Abwesenheit von Zyklen erleichtert Einfügen und Entfernen von Teilen.

  Anwendung z.B.: Codierung nach Fano und Huffman

- **Stapel, Kellerspeicher** (*stack*): LIFO- (auch: FILO-) Speicher (Last In, First Out): Ablege- („Push-“) u. Abfrage- („Pop-“) Operatoren ermöglichen die Bedienung der damit realisierten linearen Konzepte. Erreichen jedes Elementes nur nach Eliminieren aller darüber liegenden.

  Anwendung z.B.: Abarbeitung verschachtelter Aufrufe (vgl. math. Ausdrücke mit Klammern); Entlassungslisten

Häufigste Datenstrukturen (Forts.):

- **Warteschlange** (*queue*): Speicherung und Bearbeitung nach dem linearen FIFO-Konzept (First In, First Out).

  Drei Operatoren/Operationen: Prüfen, ob weitere Wartende aufgenommen werden können, anstellen (*enqueue*), bedienen/entfernen (*dequeue*). Sonderformen:

  Prioritätsliste (*priority queue*) erlaubt auch Eingriffe (Hinzufügen, Löschen) im Schlangen-Inneren

  „**Deque**“ (*Double-ended queue*) läßt Änderungen an beiden Enden zu.

  Kreisliste (*ring queue, circular q.*), hat hinter der Adresse des letzten Speicherplatzes die Adresse des ersten.

  Anwendung z.B.: Transportbänder, Münzautomaten Rolltreppen, Kofferkarussell (Flughafen)

Datenstrukturen – etc.

Datenstrukturen sind eine der Analyse-Möglichkeiten für rechner-interne Strukturen neben:

- **mathem. Strukturen**, in denen, noch abstrakter, Objekte und Operatoren definiert werden, die Objekte ineinander transformieren;
- **Speicherstrukturen**, die, vor allem durch Verwendung von Speicherzellen für Speicheradressen, eine leichtere und effizientere Umsetzung von Operatoren ermöglichen;
- **Hardwarestrukturen**, d. Operationen u.Transformationen von Speicherungsstrukturen auf Hw-Ebene beschreiben.

Beispiel:

Eine Datenbank ist als Datenstruktur ein Graph, als mathem. Struktur eine Relation, auf Speicherebene ein Optimierungs-problem der Speicherkonfiguration, auf Hw-Ebene die Kon-struktionsaufgabe geeigneter Mikroprogramme u. Hw-Teile.

Datentypen

- Ein **Datentyp** ist eine Interpretation, die auf eine Bitfolge angewandt wird.

  [Bisher kennengelernt: Unterschiedliche Interpretationen der Bitfolgen zur Darstellung ganzer und gebrochener Dualzahlen.]

  Alle Programmiersprachen ermöglichen die Verwendung mehrerer Datentypen, meist nach Vereinbarung (*declaration*) von Speicherzellen (Variablen), deren Inhalt zur Laufzeit in einer gegebenen Weise interpretiert wird.

  Es gibt zwei Klassen von Datentypen.

Einfache („skalare“) Datentypen sind im wesentlichen:

- Der Typ **Integer** (auch: cardinal) ist die Darstellung ganzer Zahlen, intern als Dualzahlen ohne gebrochenen Teil („Festkomma“: man stelle sich eine Nachkomma-Null fest verbunden hinter dem Ende der Zahl vor.)

Datentypen

Einfache Datentypen (Forts.) :

- **Real** dient der Abbildung reeller Zahlen mit Vorzeichen, Mantisse und Exponent.
- **Double precision** ist die Erweiterung (Bit-Verdoppelung) von real, mit größerem Wertebereich für Mantissen und Exponenten.
- **Complex** ist die Verbindung zweier real-Werte zu einer komplexen Zahl ($a + b \cdot i$; $a, b \in \mathbb{R}$, $i = \sqrt{-1}$)
- **Logical** (auch: boolean) sind Daten, die nur zwei Werte annehmen können: **`true`** oder **`false`**.
- **Character** u./o. **string** ist der Datentyp, in dem druckbare Schriftzeichen dargestellt werden; jedes Zeichen wird mit 6, 7, 8 oder 16 Bit codiert.

Datentypen

Einfache Datentypen (Forts.) :

- **Label** (Marke) ist ein Datentyp, der eine Stelle im Programm-Code kennzeichnet.
- **Pointer** (Zeiger) erhält als Wert die Adresse beliebiger Daten.

Für diese Datentypen sind (soweit sinnvoll) in den Programmiersprachen Operatoren definiert, darunter die vier Grundrechenarten und die Exponentiation.

Ebenso sind Vergleichsoperationen definiert: gleich, ungleich, größer, kleiner etc..

Logische Größen können mit logischen Operatoren und Operationen verknüpft werden, darunter AND, OR, NOT.

Datentypen

Einfache Datentypen (Forts.) :

Bei Verknüpfung unterschiedlicher Datentypen werden die Variablen zuvor in einen gemeinsamen Typ konvertiert. Dies geschieht **explizit** (durch den Programmierer) oder **implizit** (durch den Compiler).

Bei Typgleichheit wird die jeweilige Variante einer Operation durchgeführt. Einer der klassischen Anfängerfehler ist dabei (sofern nicht beabsichtigt) die Integer-Division.

Beispiel:

Der Ausdruck **`x=3/4*4`** weist i.d.R. der Variablen **x** den Wert 0 zu; denn die Operanden (Literale) weisen auf Integer-Operationen hin, darunter auf Division ohne Dezimalteil.

Datentypen

Zusammengesetzte (strukturierte) Datentypen sind Zusammenstellungen von Elementen desselben oder mehrerer Typen:

- **Array** (Feld) ist die technische Realisierung der gleichnamigen Datenstruktur
- **Record** (Datensatz) und **structure** (Struktur) sind, je nach Programmiersprache, zusammengefaßte Größen (meist) unterschiedlicher Datentypen.
- **File** (Datei) ist die Vereinigung von Daten verschiedener Datentypen, die Ort und Art nicht-flüchtig gespeicherter Daten kennzeichnen.

Außer einfachen und zusammengesetzten gibt es auch **abstrakte Datentypen**, wie die Implementierung der Datenstruktur Menge. Sie werden hier nicht näher behandelt.

Turing-Maschine

Das Aufkommen erster Rechenmaschinen stellte schon Anfang des 20. Jh. Fachkreise vor die Frage nach den „Grenzen der Berechenbarkeit“ und nach der Konstruktion von Maschinen, die Algorithmen ausführen können.

Alonzo Church (US-Prof., 1903-1995) postulierte:

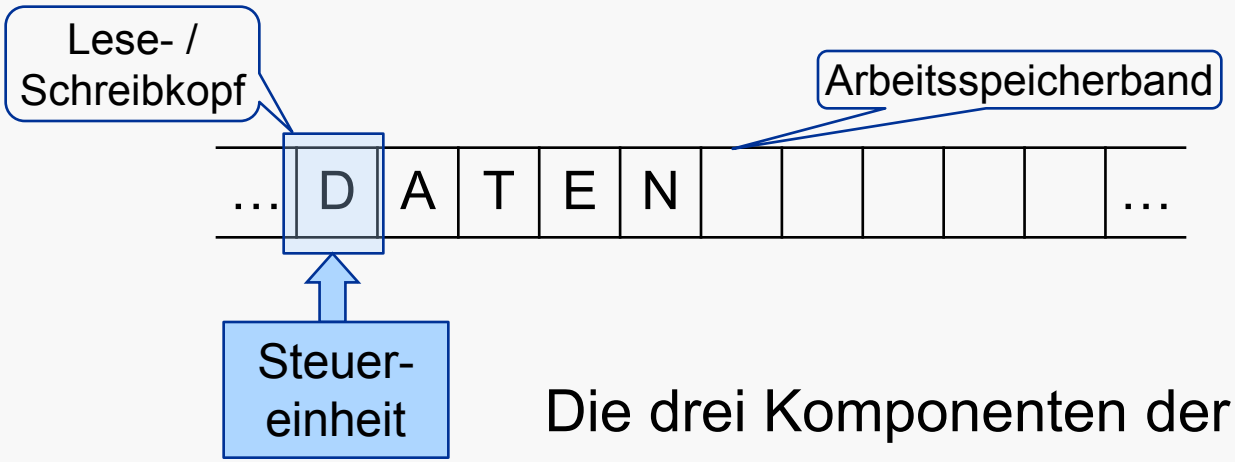
„Die Menge berechenbarer Funktionen ist gegeben durch die Menge der Programme, die man formulieren kann.“

Bahnbrechender Beitrag von Alan Mathison Turing (Student von Church, 1936):

Jede Maschine, die nach fest vorgegebenem Programm arbeitet und logische Prozesse behandelt, läßt sich anhand eines universellen Automatenmodells beschreiben.

⇨ **Turing-Maschine**

Turing-Maschine



Die drei Komponenten der Turing-Maschine:

- **Arbeitsband**: Träger der Daten (z.B. Binär- oder Schriftzeichen); aufgeteilt in Felder, die einzeln gelesen bzw. beschrieben werden durch schrittweisen Transport in beide Richtungen; je nach Aufgabenstellung beliebig (faktisch unendlich, mindestens rechtsseitig) erweiterbar.

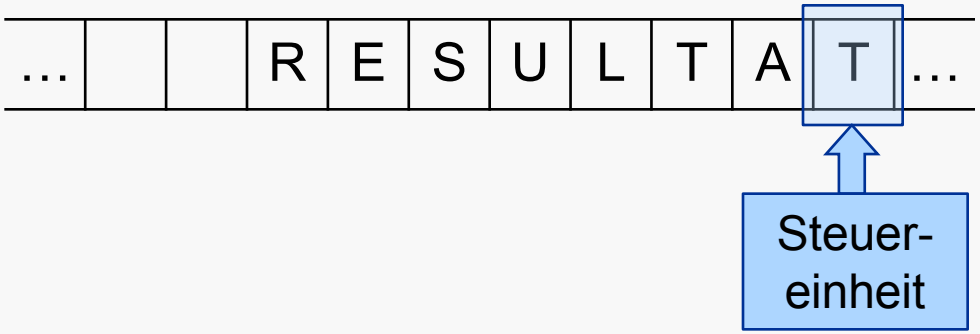
Pro-zessor

- **Lese-/Schreibkopf**: Liest aus oder beschreibt jeweils ein Feld (in ursprüngl. Beschreibung: die bewegliche Einheit).
- **Steuereinheit**: Kontrolleinrichtung für den Transport und die Aktivität des Lese-/Schreibkopfs nach Maßgabe einer Zustandsänderungstabelle (Transitionstabelle).

Turing-Maschine

Arbeitsweise der Turing-Maschine:

- Codierung und Eingabe der Daten über das Band
- Anfangsstellung des Lese-/Schreibkopfs am linken Ende der codierten Daten
- Bewegung und Betätigung des Lese- /Schreibkopfs (vor- / rückwärts, lesend / schreibend) durch die Steuereinheit anhand einer Automatentafel (Zustandstabelle)
- Beendigung (Halt) des Prozesses, wenn das Band nur d. gewünschte Ergebnis und sonst nur leere Felder enthält



Turing-Maschine

Voraussetzungen für die Funktion der Turing-Maschine:

- Endliches Alphabet **A** von Eingabezeichen d. L.-/S.-Kopfs oft: **A** = { a_0=_(Leerzeichen), a_1=0, a_2=1 }
- Endliches Alphabet **T** von Bandzeichen ($\supseteq$ **A**) oft: **T** = **A**
- Ausführbare Anweisungen **E** der Steuereinheit an den Lese-/Schreibkopf:

  E = {nach links / nach rechts rücken, nicht rücken, halten }

  { L / R / N / H }
- Eine endliche Menge v. Zuständen („Handlungsmustern“)

  Q={ q_0:Start-/Anfangszustand, …, q_n:End-/Haltezustand },

  welche die Maschine aufgrund zuvor geschriebener / gelesener Zeichen u. erfolgter Aktionen annehmen kann

  Die Zustände q_i werden durch Quintupel (*) codiert.

  (*) Kennzeichnung durch 5 Angaben in festgelegter Reihenfolge

Turing-Maschine

Quintupel des Zustands einer Turing-Maschine:

(q_i , a_j , a_k , E_l , q_m) , mit:

q_i : aktueller Zustand der Turing-Maschine

a_j : Zeichen, das am aktuellen Band-Feld gelesen wird

a_k : Zeichen, das in das aktuelle Feld geschrieben wird

E_l : als nächstes auszuführende Anweisung

q_m : als nächstes anzunehmender Zustand

Arbeitsweise:

- Je nach Deinem aktuellen Zustand q_i
- lies das Zeichen a_j auf dem aktuellen Band-Feld
- schreibe in dasselbe (aktuelle) Feld das Zeichen a_k
- bewege Dich [nicht] weiter gemäß Vorgabe E_l und
- nimm an dem dann aktuellen Feld den Zustand q_m an.

Turing-Maschine

Beispiel:

Erhöhen einer vierstelligen Dualzahl (Nibble) um eins in elf Schritten – hier: $1011_2 + 1_2 = 1100_2$ (d.h.: $11_{10} + 1_{10} = 12_{10}$)

Zustandsänderungstabelle

	q_i	a_j	a_k	E_l	q_m
Lesen	0	**0**	0	R	0
	0	**1**	1	R	0
	0	**_**	_	L	1
Über-trag	1	**0**	1	L	2
	1	**1**	0	L	1
	1	**_**	1	N	H
Kopie	2	**0**	0	L	2
	2	**1**	1	L	2
	2	**_**	_	R	H

Bandbeschriftung

q_0 :	…	_	**1**	0	1	1	_ …
q_0 :	…	_	1	**0**	1	1	_ …
q_0 :	…	_	1	0	**1**	1	_ …
q_0 :	…	_	1	0	1	**1**	_ …
q_0 :	…	_	1	0	1	1	**_** …
q_1 :	…	_	1	0	1	**1**	_ …
q_1 :	…	_	1	0	**1**	0	_ …
q_1 :	…	_	1	**0**	0	0	_ …
q_2 :	…	_	**1**	1	0	0	_ …
q_2 :	…	**_**	1	1	0	0	_ …
q_2 :	…	_	**1**	1	0	0	_ …

Turing-Maschine

Erkenntnisse:

- Eine Turing-Maschine (1936) kann alles, was Computer können.
- Es ist nicht vorweg entscheidbar, ob die Turing-Maschine am Ende einer konkreten Rechnung hält („Halteproblem“)

  – d.h. entsprechend: Endlosschleifen (nie endende Ausführungen) sind nicht prinzipiell auszuschließen.

Vorteile der Turing-Maschine:

- Jeder Algorithmus läßt sich auf einer T.-M. nachbilden
- Primitive Einzelschritte, die sich immer mechanisieren lassen.
- Einfache, anschauliche Bauweise.

Turing-Maschine

Kritik an der Turing-Maschine:

- Bandlänge läßt sich nicht grundsätzlich begrenzen.
- Primitive Einzelschritte machen Programmierung schwer.
- Programme lassen Rechenweg kaum erkennen, haben kaum Ähnlichkeit zu anderen Programmiersprachen.
- Das Laufzeitverhalten ist mit „reellen“ Rechnern kaum vergleichbar.

Die Turing-Maschine bleibt in der Komplexitätstheorie eine Bezugsgröße.