

# Konzepte von Programmiersprachen

## 1. Einführung

### Ziele der Vorlesung:

- Entwurfsalternativen von Programmiersprachen erkunden
- deklarative Programmierung am Beispiel von Haskell erlernen
- Programmier Techniken und ihre Verbindung zu Sprachkonzepten kennenlernen und verstehen

## Höhere Programmiersprachen

- **Ziele beim Entwurf:**  
**Ausdrucksmächtigkeit, Schlichtheit, Eleganz**
- **Weitere Anforderungen:**
  - **Universalität**
  - **Implementierbarkeit**
  - **Verifizierbarkeit**
  - **Effizienz -> ... bei Ausführung & ... bei Programmierung**

### Konzepte

- **Werte, Speicher, Bindungen**
- **Abstraktion und Kapselung**
- **Typsyste**me
- **Ablaufsteuerung und Nebenläufigkeit**
- ...

### Ziele

- **Erlernen neuer Sprachen**
- **Auswahl geeigneter Sprachen**
- **Entwicklung verlässlicher wohlstrukturierter Programme**
- **Unterscheidung verschiedener Paradigmen**

# Einige Zitate

- **Karl der Große (742-814):**  
„To know another language is to have a second soul.“
- **Ronald Searle (1920- ):**  
„You can never understand one language until you understand at least two.“
- **Ralf Hinze, Uni Bonn: Seit 1950 sind Hunderte von Programmiersprachen entwickelt worden.**  
...  
**Programmiersprachen sind das Medium der Informatik. Ein tiefes Verständnis der zugrundeliegenden Konzepte ist für alles, was mit Informatik zu tun hat, unabdingbar.**



# Fortran (Formula Translator)

- Entwicklung 1954-56 bei IBM unter John Backus
- Standard für numerische Berechnungen (scientific computing)
- Hauptinnovationen:
  - math. Notation für arithmetische Ausdrücke
  - symbolische Namen für Variablen, Unterprogramme, Felder, Dekl.
- Nachfolger heute noch weitverbreitet:  
Fortran 90, HPF (High Performance Fortran), Fortran 2003

```
C Hello World in Fortran  
  
      WRITE (6,7)  
  
7      FORMAT (13HHello, world!)  
  
      STOP  
  
      END
```

# COBOL (Common Business Oriented Language)

- Entwicklung 1960 von Komitee unter Grace Murry Hopper
- Konzeption für Geschäftsanwendungen
- Hauptinnovation: Anlehnung an die natürliche Sprache

```
* Hello World in COBOL.  
IDENTIFICATION DIVISION.  
    Program-Id. Hello-World.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.  
PARA1.  
    DISPLAY „Hello, world!“.  
STOP RUN.
```

E.W. Dijkstra: „The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.“

# LISP (List Processing)

- Entwicklung Ende der 50er Jahre am MIT unter John McCarthy
- Anwendungsgebiete: Künstliche Intelligenz, Symbolische Berechnungen
- Hauptinnovationen: rekursive Funktionen, Funktionen höherer Ordnungen, Programme als Daten, S-Ausdrücke, Speicherverwaltung

```
(PRINT (LIST ,HELLO ,WORLD))
```

```
(write-string „Hello, world!“)
```

Alan Perlis: „A Lisp programmer konws the value of everything, but the cost of nothing.“

E.W. Dijkstra: „Lisp has jokingly been called `the most intelligent way to misuse a computer. I think that description is a great compliment...“

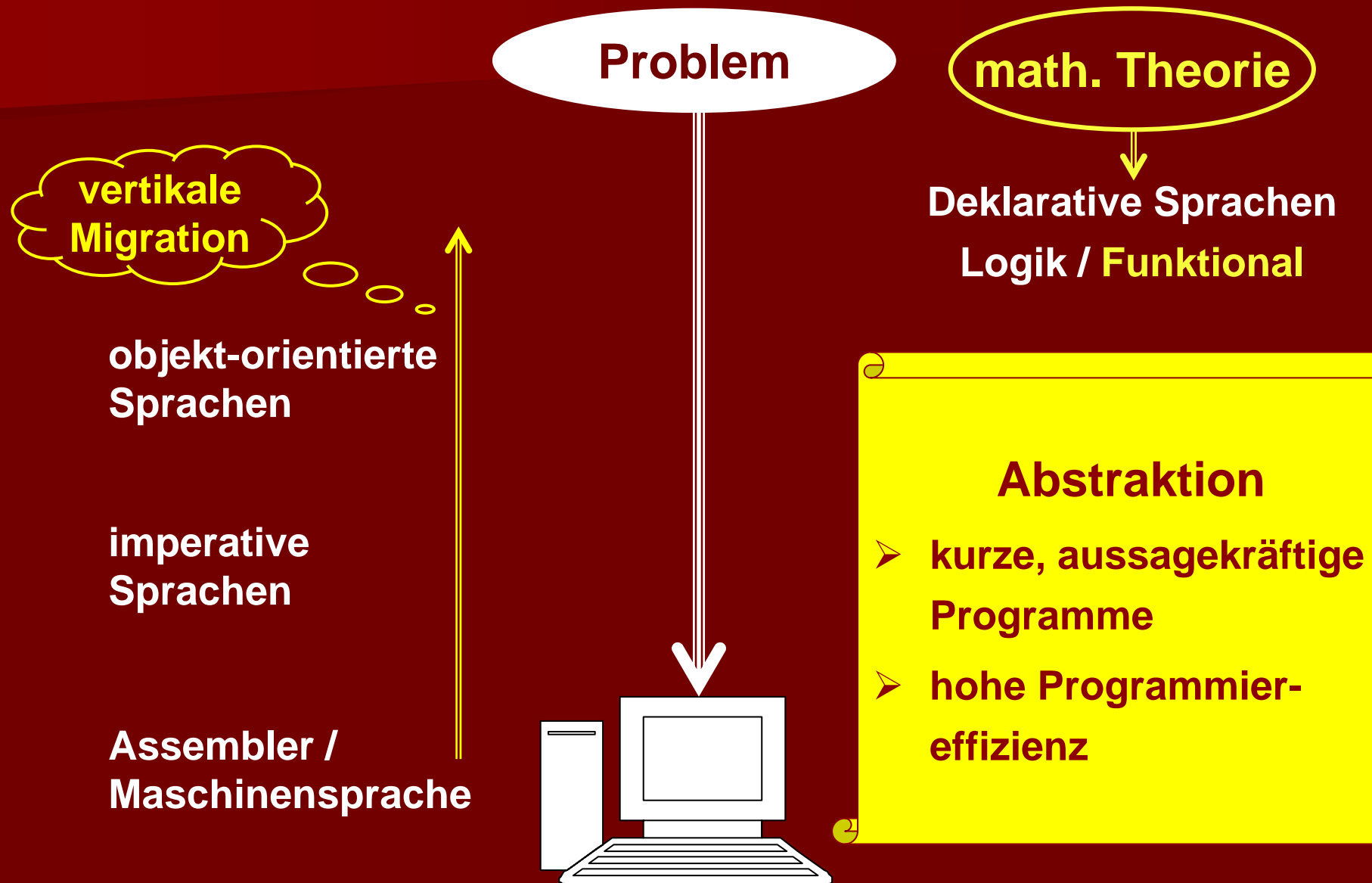
# Algol (Algorithmic Language)

- Entwicklung von 1958-1963 von einem Komitee, dem u.a. John Backus, John McCarthy und Alan Perlis angehörten
- Ausrichtung: „general purpose“ Sprache, vor allem für wissenschaftliche Anwendungen
- Hauptinnovationen:
  - einfache Syntax für Anweisungen (;)
  - Blockstruktur (begin – end)
  - rekursive Funktionen (Stackprinzip)
  - einfaches Typsystem
  - Nebenprodukt: Backus-Naur-Form (BNF)

```
program HELLOWORLD;  
begin  
    print „Hello, world!“  
end
```



# Klassifikation von Programmiersprachen



# Programmiersprache

## Syntax

formale  
Beschreibung  
der Kompo-  
nenten eines  
Programms  
mittels  
kontextfreier  
Grammatiken  
bzw. BNF

## Pragmatik

Handhabung  
der Sprache/  
Sprachkon-  
zepte  
  
Implementie-  
rung  
Werkzeuge

## Semantik

Bedeutung der  
Sprache/Progr.  
oft informell  
Formale  
Methoden  
vermeiden  
Unvollständig-  
keit und Mehr-  
deutigkeiten

# Syntaxbeschreibung

kontextfreie Grammatik = < **Nonterminale**, **Terminale**, **Startsymbol**, Regeln >

## % Anweisungen (Statements)

**S** -> **id := E** % Wertzuweisung  
| **begin S ; S end** % Hintereinanderausführung  
| **if B then S** % bedingte Anweisung

## % Ausdrücke (Expressions)

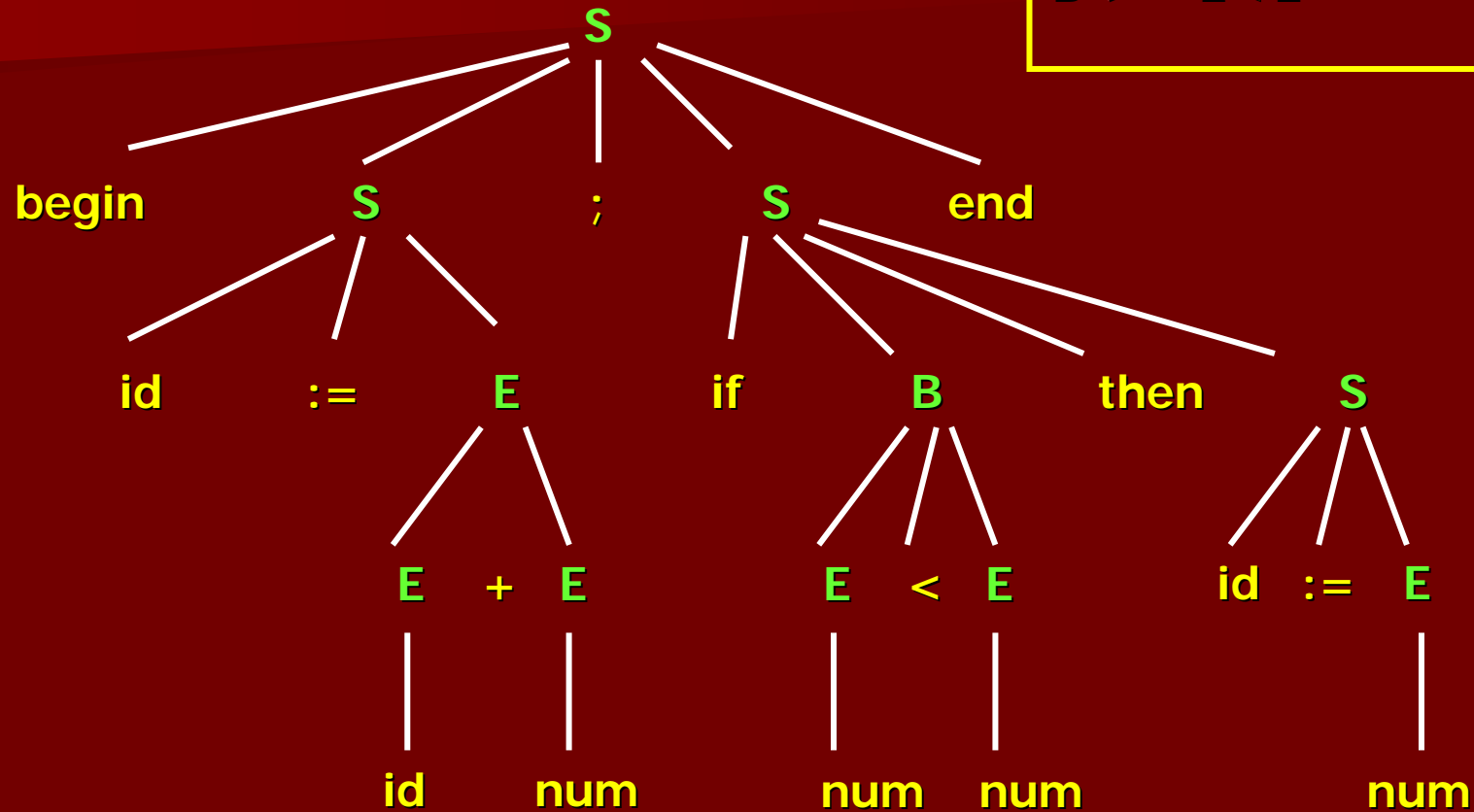
**E** -> **E + E** % Summe  
| **id** % Bezeichner  
| **num** % Zahl

## % Boolesche Ausdrücke (Boolean expressions)

**B** -> **E < E** % Vergleichsausdruck

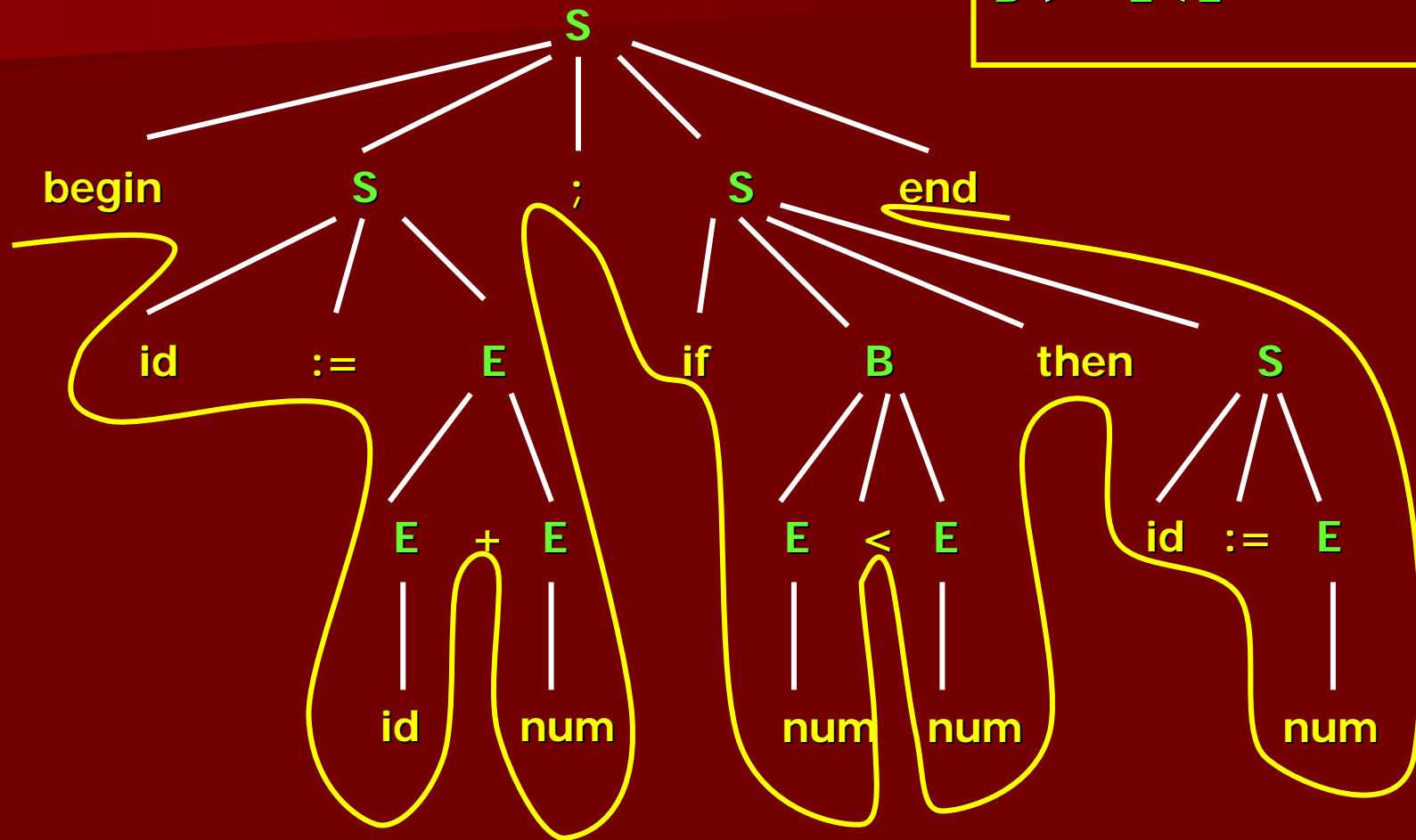
# Syntaxbaum

$S \rightarrow id := E$   
 $S \rightarrow begin S ; S end$   
 $S \rightarrow if B then S$   
 $E \rightarrow E + E \mid id \mid num$   
 $B \rightarrow E < E$



# Syntaxbaum - Front

$S \rightarrow$  id := E  
| begin S ; S end  
| if B then S  
 $E \rightarrow$  E + E | id | num  
 $B \rightarrow$  E < E



begin id := id + num ; if num < num then id := num end

# Abstrakte Syntax

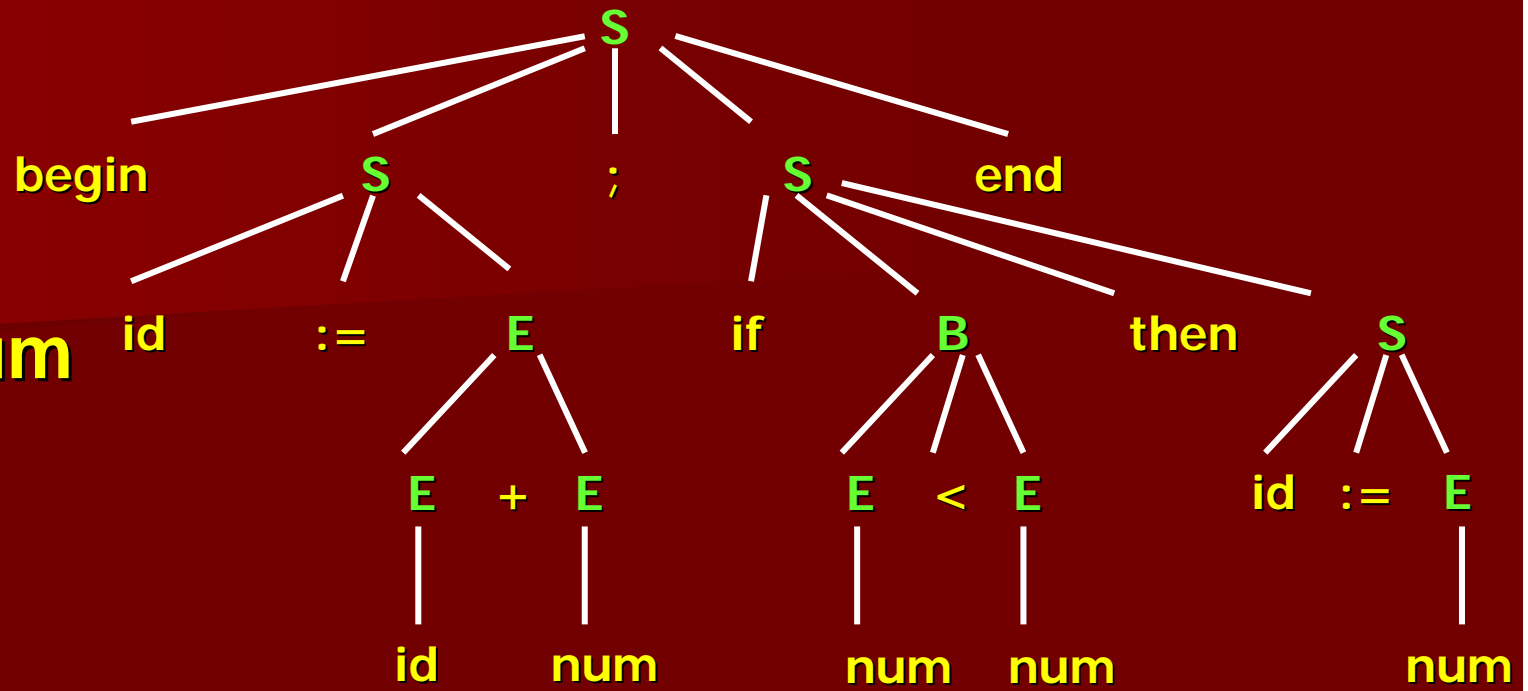
Regeln

$\Rightarrow$

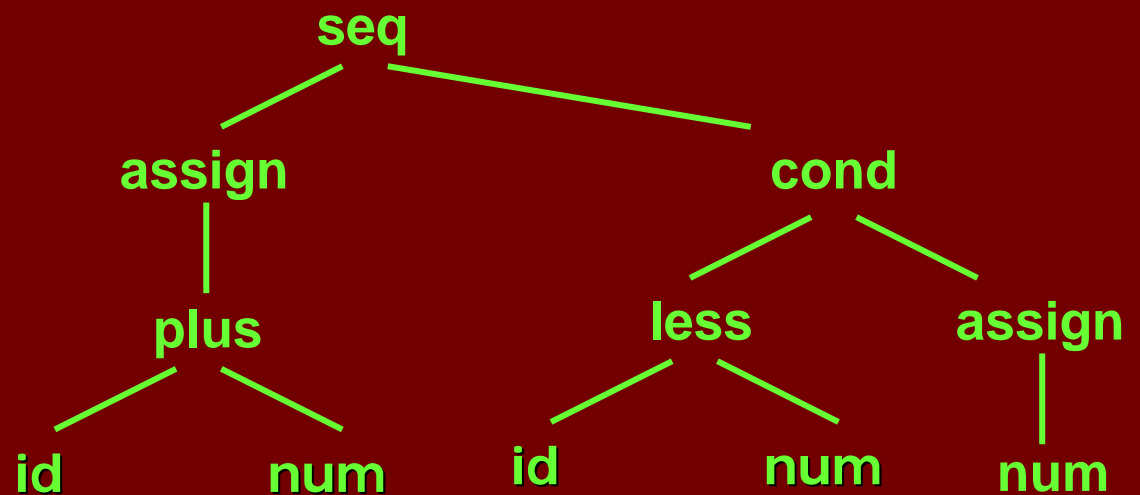
Operationssymbole

<b>S</b> $\rightarrow$	<b>id := E</b>	$\Rightarrow$	<b>assign : E</b>	$\rightarrow$ <b>S</b>
	<b>begin S ; S end</b>	$\Rightarrow$	<b>seq : S x S</b>	$\rightarrow$ <b>S</b>
	<b>if B then S</b>	$\Rightarrow$	<b>cond : B x S</b>	$\rightarrow$ <b>S</b>
<b>E</b> $\rightarrow$	<b>E + E</b>	$\Rightarrow$	<b>plus : E x E</b>	$\rightarrow$ <b>E</b>
	<b>id</b>	$\Rightarrow$	<b>id :</b>	$\rightarrow$ <b>E</b>
	<b>num</b>	$\Rightarrow$	<b>num :</b>	$\rightarrow$ <b>E</b>
<b>B</b> $\rightarrow$	<b>E &lt; E</b>	$\Rightarrow$	<b>less : E x E</b>	$\rightarrow$ <b>B</b>

# Konkreter Syntaxbaum



# Abstrakter Syntaxbaum



# Beispiel: Konzept Zählschleife

**Abstrakte Syntax: for(index, anfang, ende, rumpf)**

- **FORTRAN**

```
do 1 i=1,10  
1 a(i) = 0
```

- **Pascal**

```
for i:=1 to 10 do  
a(i) := 0;
```

- **C**

```
for (i=1; i<=10; i++)  
a(i)=0;
```



# Beispiel: Konzept Zählschleife

Abstrakte Syntax: `for(index, anfang, ende, rumpf)`

- **FORTRAN**

```
do 1 i=1,10  
1 a(i) = 0
```



- **Pascal**

```
for i:=1 to 10 do  
a(i) := 0;
```



- **C**

```
for (i=1; i<=10; i++)  
a(i)=0;
```



# Programmierparadigmen

## ■ imperative Programmiersprachen

- lat. imperare = befehlen
- Programme = Abfolge von Befehlen
- Abstraktion von Maschinensprachen
- **prozedurale Sprachen wie FORTRAN, PASCAL, MODULA, ADA, C, OCCAM etc.**
- **objektorientierte Sprachen wie SMALLTALK, Eiffel, C++, Java etc.**

## ■ deklarative Programmiersprachen

- lat. declarare = erklären
- Programme = Problemspezifikationen
- Basis: Mathematische Theorie
- **funktionale Sprachen wie LISP, ML, Miranda, Haskell**
- **Logik-Sprachen wie Prolog**

# Grundidee deklarativer Sprachen



# Referentielle Transparenz

Der Wert eines Ausdrucks hängt nur von seiner Umgebung und nicht vom Zeitpunkt seiner Auswertung ab. Deshalb kann ein Ausdruck immer durch einen anderen Ausdruck mit gleichem Wert ersetzt werden (Substitutionsprinzip).



**Seiteneffektfreiheit**



**Gleichheitsprinzip, Substitutionsprinzip**

Ein Ausdruck kann immer durch einen anderen Ausdruck mit gleichem Wert ersetzt werden.

-> equational reasoning

# Beispiel: Seiteneffekte

```
program example;  
  var flag : boolean;  
  
  function f (n : integer) : integer;  
  begin  
    if flag then f:= n  else f:= n+1;  
    flag := not flag;  
  end;  
  
  begin  
    flag := true;  
    if f(2) = f(2) then writeln("ok")  
      else writeln("nicht ok");  
  end.  
end.
```

# Imperatives vs funktionales Programm

```
var a : array [1..n] of integer;
procedure quicksort (l,r: integer);
var x,i,j,tmp : integer;
begin
  if r>l then
    begin
      x:=a[l]; i:=l; j:=r+1;
      repeat
        repeat i:=i+1 until a[i]>=x;
        repeat j:=j-1 until a[j]<=x;
        tmp:=a[j];a[j]:=a[i];a[i]:=tmp;
      until j<=i;
      a[i]:=a[j]; a[j]:=a[l]; a[l]:=tmp;
      quicksort(l,j-1); quicksort(j+1,r);
    end
  end
end
```

*Pascal*

```
quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (x:xs)
  = quicksort (filter (< x) xs)
    ++ [x] ++
    quicksort (filter (>=x) xs)
```

*Haskell*