Einführung in die Informatik 1

Algorithmen

- Beschreibt Funktion
- Berechenbar
- | Algorithmen | (abzählbar) < | Funktionen | (Überabzählbar) → Kein Algorithmus zum Vergleichen von Funktionen

Eigenschaften:

- Abstrahierung
- Finitheit (endliche Länge)
- Terminierung (Liefert in endlichen Schritten ein Resultat)
- Determiniertheit
- Deterministischer Ablauf
- Effizienz

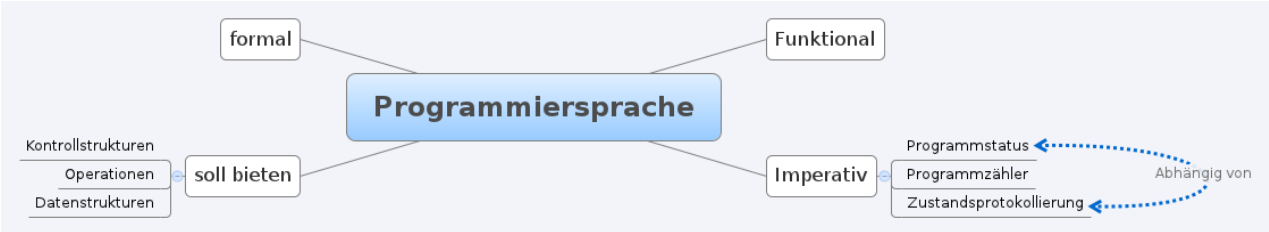
Effiziente Suche: Linear vs. Log_2 (Suchbaum)

Programmiersprache: Leicht zu verstehen vs. Effizient

Von Neumann Architektur (Formales System für Computer): E/A – CPU – Speicher

Bsp: Turing. Kann alles, was moderner Rechner kann (Berechenbarkeitstheorie)

Programmiersprachen



Java:

Datentypen: (Bits: 1 8 16 16 32 32 64 64)

primitiv: boolean, byte, short, char, int, float, double, long

vordefiniert: String, Buffered Reader

Selbstdefiniert

Ausdruck

besteht aus: Iterale, Operationen,Variablen, Klammern

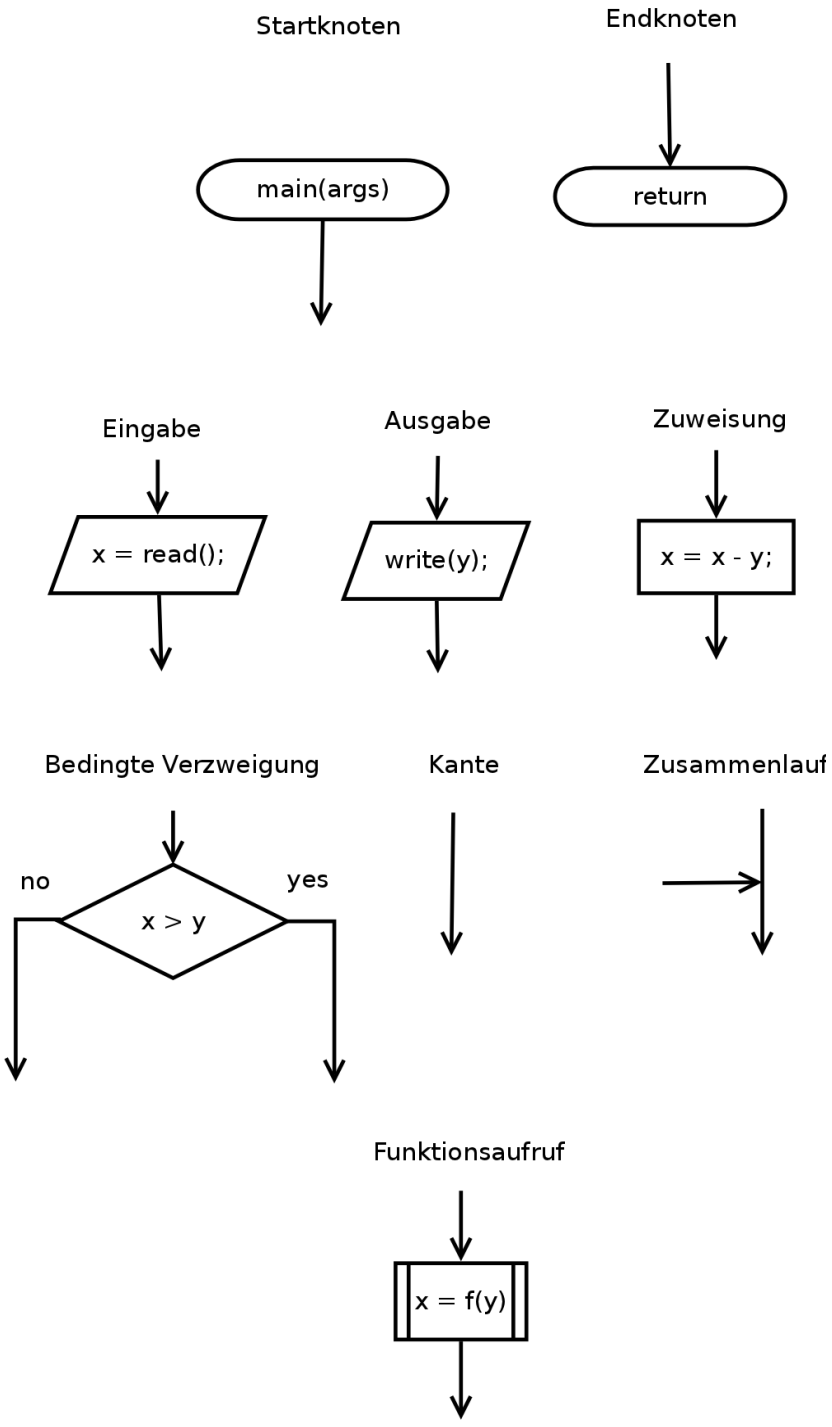
Bsp. Zuweisung

Kontrollstrukturen:

- Selektion: if ( ) { } else { }
- Iteration: while ( ) { }, for ( ) { }
- Sequenz: { }

Berechenbare Funktionen lassen sich damit berechnen

Kontrollflussdiagramme



Lexikalische Analyse

Scanner (ist Automat):

Zerlegung → baut Tokens (beschrieben durch Regex)

Nach gelesenem Token: Lesezeiger -1 (Bsp: if () == if())

Endlicher Automat:

< Q,	Σ,	δ,	q_0,	F >
Zustände	Alphabet,	Übergangsfunktion,	Startzustand,	Endzustände

Akzeptiert Wörter, die den Automaten in einen Endzustand bringen
Darstellbar als Zustandsübergangsdiagram

Reguläre Sprache: Leere Menge, Leeres Zeichen, Elementarzeichen, Vereinigung/Verkettung/Hüllenbildung
→ Endliche Sprache, wird von endlichem Automaten aktzeptiert

Symbole als Teil der Grammatik:	Reguläre Sprache	→ Einfacher, endlicher Automat
Rest:	Kontextfreie Sprache	→ Komplizierterer Automat

Syntax

Wie muss ein Programm aussehen?

	Scanner		Parser	
Source File	→	Tokenfolge	→	Syntaxbaum
	Lexikalesche Analyse:		Grammatikalische Analyse	

- Reservierte Wörter
- Namen
- Konstanten

→ Tokens

Verwerfen von

- Kommentaren
- Leerzeichen

operationell	**Semantik**	denotationell
Wie wird ausgeführt		Was ist der Effekt
Änderung von Zuständen		Funktion im mathematischen Raum

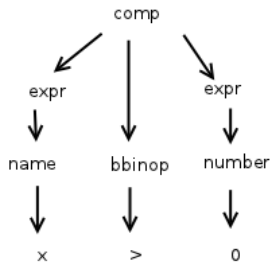
Syntaktisch Korrekt ≠ Semantisch Korrekt

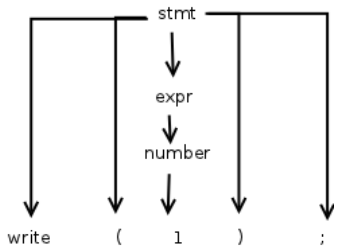
Alphabet:	A, Menge von Zeichen; Nicht leer			
Wort:	Endliche Folge: $w = w_1, w_2, \dots w_n; w_i \in A, n \in N$			
Leeres Wort:	ε			
Alle Worte in A:	$A^* = \{ w \mid w = w_1, w_2, \dots w_n; w_i \in A, n \in N \}$			
Formale Sprache:	$L \subseteq A^*$			
Grammatik:	Beschreibt Sprache, Wort erzeugbar aus Startsymbol			
	< V,	T,	P,	S >
	Variablen	Terminal	$P \subseteq A+ \times A^*$	Startsymbol
	Nicht Termial	Tokens	Ersetzungsregeln	
Reguläre Ausdrücke:	|	Alternative		
	*	Iteration		
		Kombination		

? Option

Struktur von Programmen (hierachisch)

program	:=	decl* stmt*
decl	:=	type name (, name)*;
type	:=	int | double | …
stmt	:=	; | { stmt* } | name = expr; | name = read(); | write(expr); | if (cond) stmt | if (cond) stmt else stmt | while (cond) stmt
expr	:=	number | name | (expr) | unop expr | expr binop expr
unop	:=	-
binob	:=	- | + | * | / | %
cond	:=	true | false | (cond) | expr comp expr | bunop cond | cond binop cond
comp	:=	== | != | > | < | <= | >=
bunop	:=	!
bbinop	:=	|| | &&





Threads

Scheduler verwaltet (Implementierung sollte fair sein, nicht naiv)