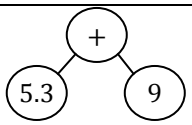
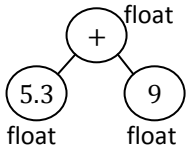
COMPILERBAU

Idee

Programm als Text → Compiler → Code für Zielplattform

Vorgang

	1. Lexikalische Analyse (Scanner)	2. Syntaktische Analyse (Parser)	3. Semantische Analyse (Type Checker)	4. Code Generierung
Folge von Zeichen	➔ Symbol	➔ Syntaxbaum mithilfe von EBNF	➔ attribuierter Syntaxbaum	➔ Folge von Instruktionen
'1' '2' '3' ' '	Zahl 123			Maschinencode

Sprachen

unterscheidet man die Syntax (Struktur) und die Semantik (Bedeutung)

EBNF

Extended Backus Naur Form

Begriffe

Symbol	Bezeichnung	Beispiel
L (S,T,N,P)	Sprache, definiert Satz (ganzes File)	L={ Ø, a, aa, aaa, ...}
S	Startsymbol	S = A B
T	Menge der Terminalsymbole (Token) (in den Blattknoten des Parsetree)	T = {"a", "b", "c", "d"}
N	Menge der Nichtterminalsymbole (im Inneren des Parsetree)	NT = {S, A, B}
P	Menge der Produktion	
Ø	leere Menge	Ø = { }

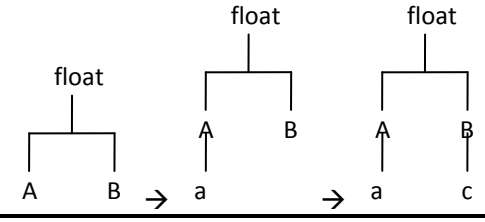
Rekursion

z.B. A="a" A | Ø

Produktion (in EBNF)

S = A B
A= "a" | "b".
B = "c" | "d".

Lösungen

Variante 1	Variante 2: Parsetree
S -> AB -> aB -> ac	

eindeutige Grammatik

wenn aufgrund des aktuellen Tokens die Produktionsregel bestimmt werden kann.
= Deterministische Grammatik, immer nur ein Weg

Chomsky Hierarchie

Reguläre Sprache	1EBNF-Regel, ohne Rekursion
Kontextfreie Sprache	Mehrere EBNF-Regeln, mit Rekursion, linke Seite immer ein NT
Kontextsensitive Sprache	Linke Seite egal/frei
Freie Sprache	

Selection Sets (Selektions-menge)

Frage: Welche Regel soll ich benutzen, wenn ich ein Terminalsymbol bekomme.
Vorgehen: Erstes Element anschauen. Terminalsymbole aufschreiben, NT weitersuchen.
Beachten: Wenn gleiches Terminalsymbol in mehreren gleichgestellten Regeln vorkommt. Nur erstes schreiben.
Wenn leere Menge -> Regel weiter oben beachten und nächste suchen.

Type Inference

int + int = int **double + double = double** **int + double = double**