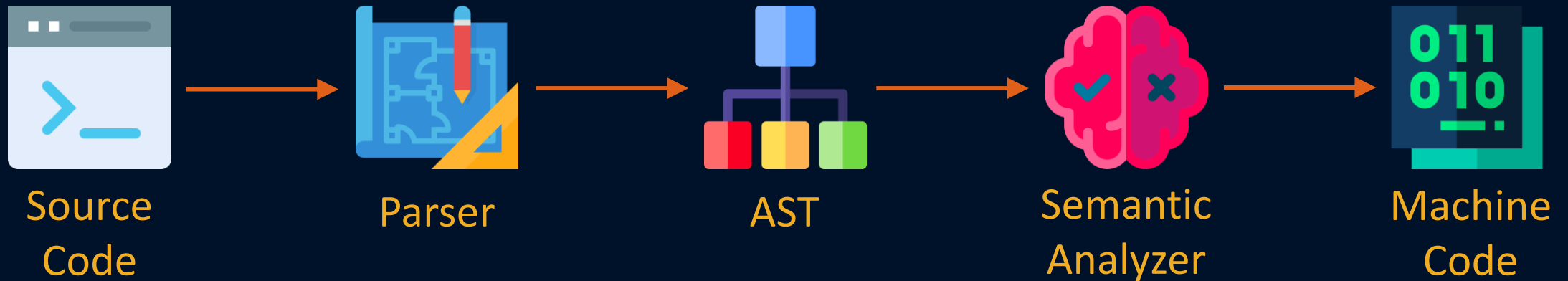


# Abstrakter Syntaxbaum & Semantische Analyse

Vom Code zur bedeutungsvollen Struktur



Qais Latif und Mohammad Osman

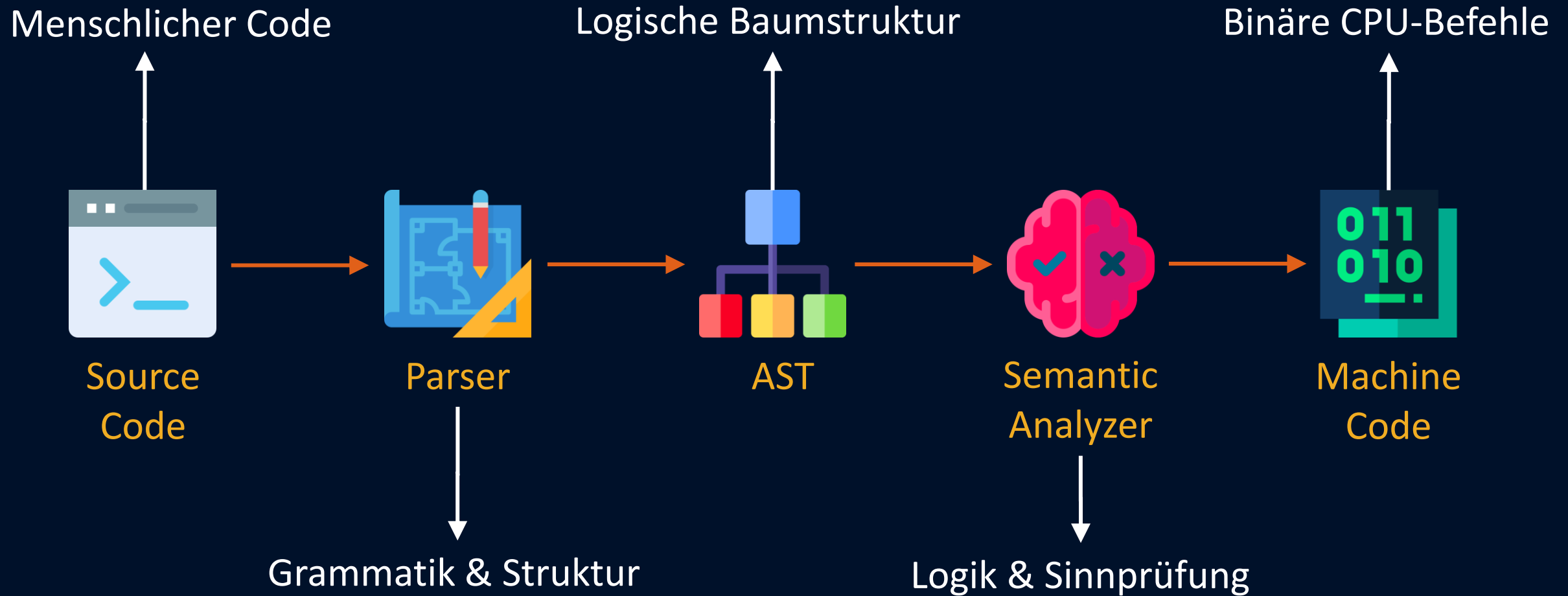
PRO+SEM: Compilerbau

13.Oktober 2025

# Agenda

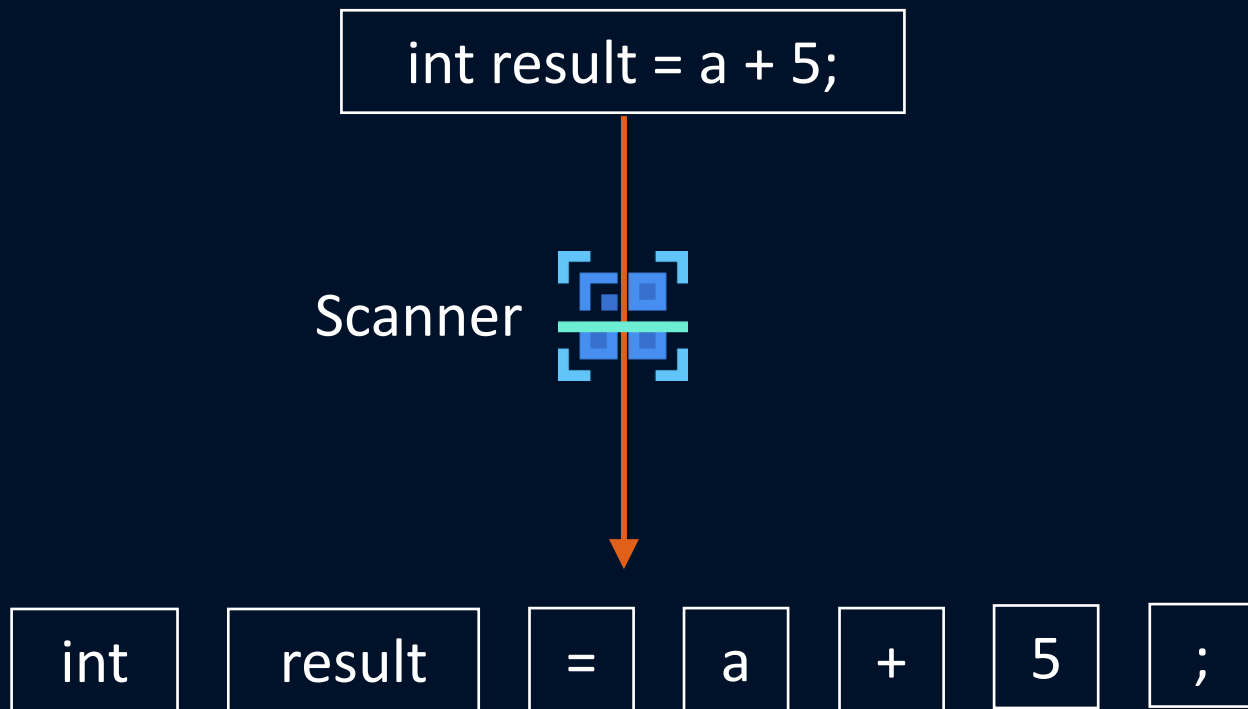
1. Der Weg des Compilers: Wo stehen wir?
2. Konkrete vs. Abstrakte Syntaxbäume
3. Semantische Analyse: Die Prüfung auf Sinnhaftigkeit
4. Werkzeuge: Symboltabelle & Typprüfung
5. Zusammenfassung

# Das große Ganze: Phasen eines Compilers



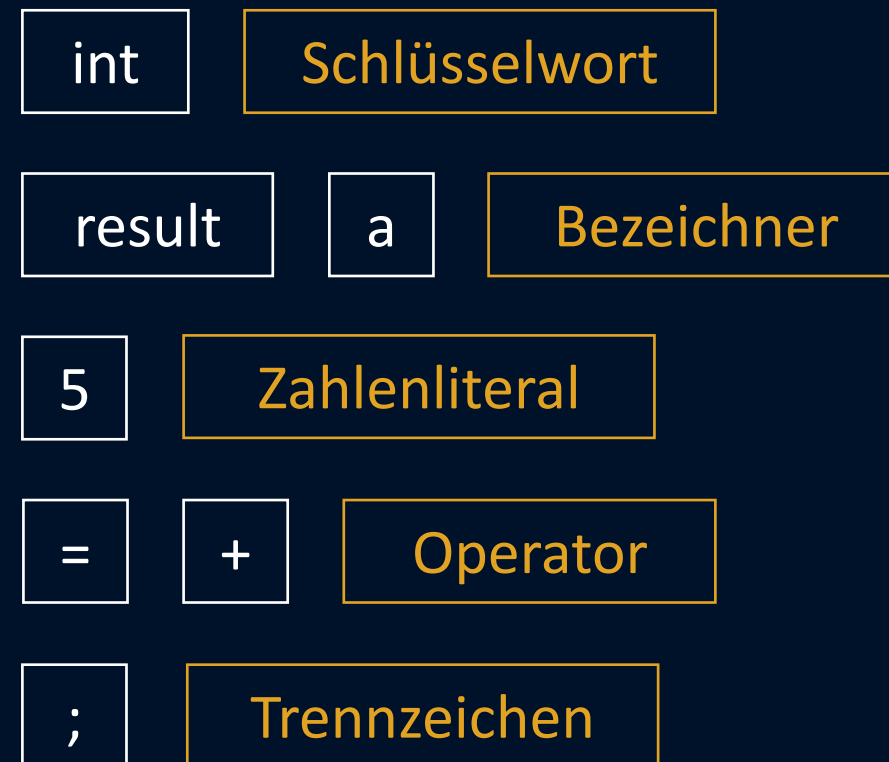
# Der erste Schritt: Lexikalische Analyse

## Der Compiler zerlegt Code in Tokens

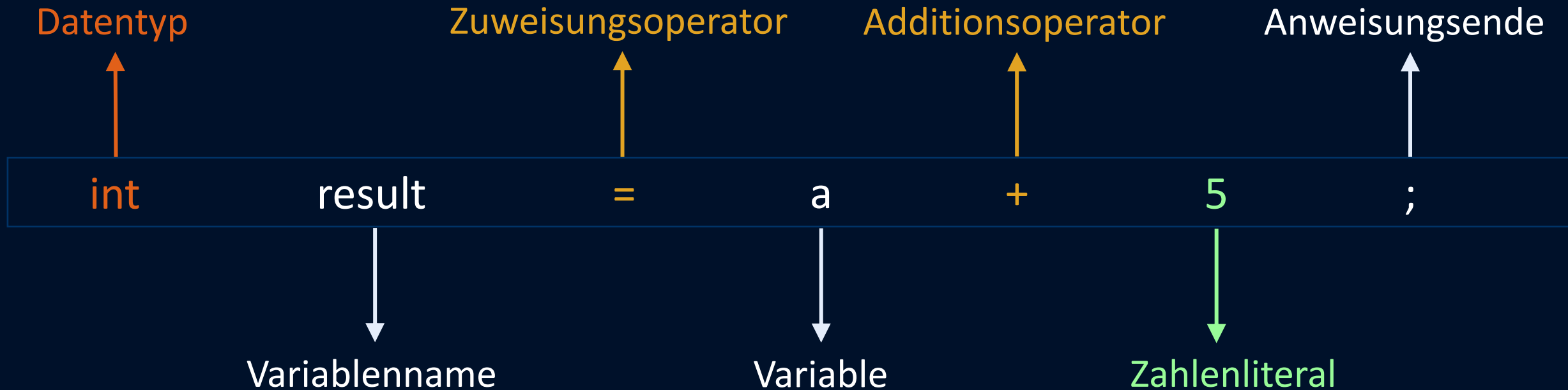


**Ergebnis:** 7 Tokens insgesamt

## Token-Typen



## Unser Beispiel für heute : C++



Wie repräsentiert der Computer diese Anweisung nach dem Parsen?

## Was ist Grammatik?

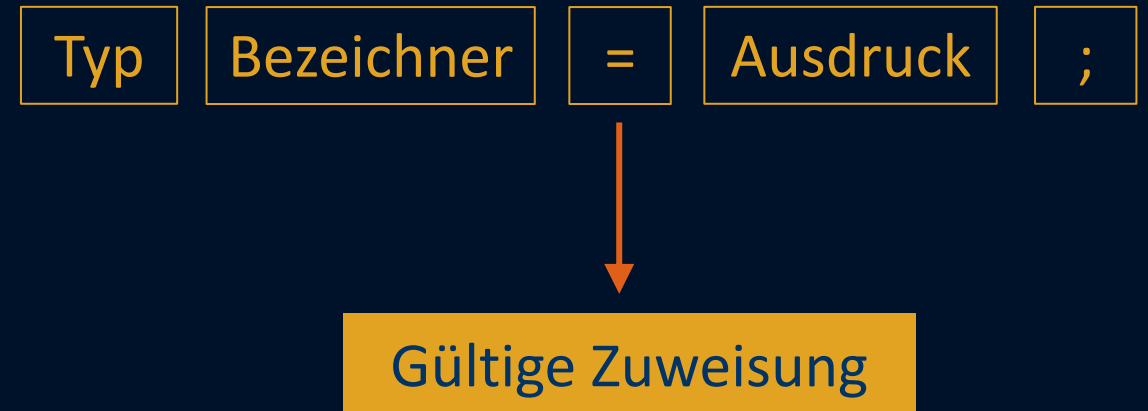
**Grammatik:** Definiert die Syntax-Regeln einer Sprache.

**Parser:** Prüft die Token-Reihenfolge anhand dieser Regeln.

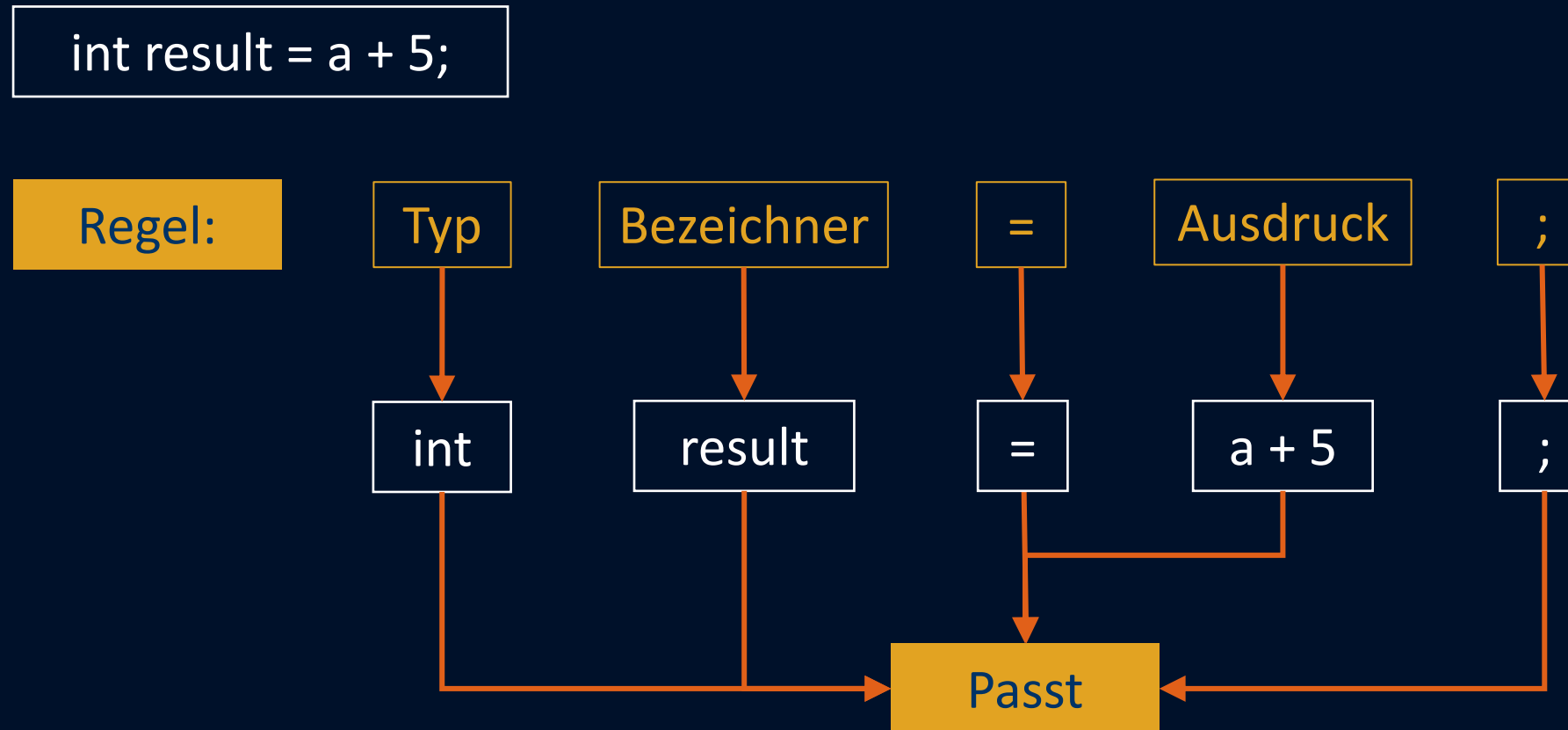
**Regel:** Beschreibt den Aufbau einer Struktur (z.B. Zuweisung).

**Syntaxfehler:** Wird bei einem Regelverstoß gemeldet.

## Beispiel: Regel für eine Zuweisung



# Die Grammatik: Das Regelwerk des Compilers



**Ergebnis:** Die Anweisung ist syntaktisch korrekt.

## Parse-Baum: Diagramm



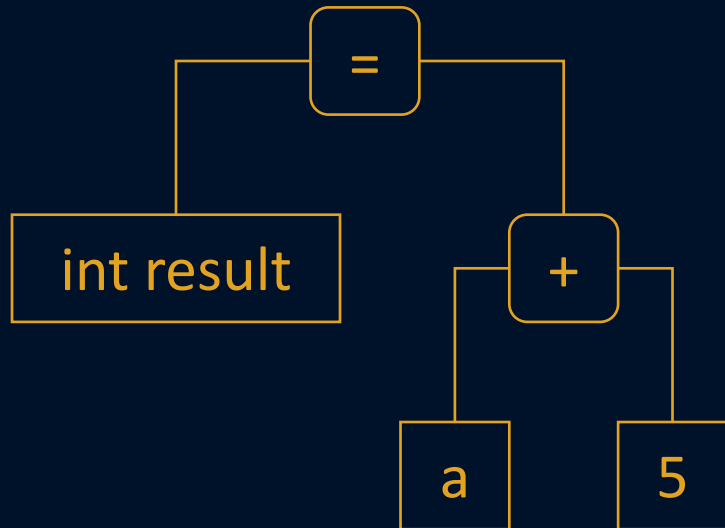
## Schlüsselkonzepte

1:1 Repräsentation der Grammatik

- Auch "Parse-Baum" genannt.
- Enthält **ALLES** (Keywords, Symbole, ;)
- Sehr detailliert & "geschwätzig"

# Der Abstrakte Syntaxbaum (AST)

## Das AST: Diagramm







## Schlüsselkonzepte

Eine komprimierte, abstrakte Darstellung der Code-Struktur.

- Fokus auf der **Bedeutung** (Semantik), nicht der reinen Syntax.
- Unnötige Details (z.B. Semikolons) werden entfernt.
- Operatoren (wie `+` und `=`) werden zu den inneren Knoten des Baumes.

# CST vs. AST Comparison Table

Eigenschaft	Konkreter Syntaxbaum (CST)	Abstrakter Syntaxbaum (AST)
 Zweck	Validierung der Syntax	Repräsentation der Semantik
 Struktur	Folgt strikt der Grammatik	Abstrahiert die Struktur
 Inhalt	Alle Tokens (Keywords, ;, ())	Nur semantisch relevante Knoten
 Verwendung	Meist eine Zwischenstufe	Grundlagen Semantik

## Überprüfung der Bedeutung und Konsistenz des Codes



**Typprüfung** (Type Checking)



**Namensauflösung** (Name Resolution)



**Eindeutigkeit** (Uniqueness Checks)

## Definition & Zweck

Eine Datenstruktur, die Informationen über alle Bezeichner (Variablen, Funktionen etc.) speichert.

Name

result

a

Datentyp

int

Gültigkeitsbereich

lokal & global

## Beispiel-Tabelle

Name	Typ	Scope
a	int	global
result	int	lokal

## Was ist Namensauflösung?

**Ziel:** Jede verwendete Variable muss ihrer Deklaration zugeordnet werden.

**Werkzeug:** Die **Symboltabelle** wird durchsucht, um die Deklaration zu finden.

**Prozess:** Die Suche startet im **lokalen Scope** und geht dann nach außen.

**Fehlerfall:** Ohne Fund wird ein **"Undeclared Variable"-Fehler** gemeldet.

## Anwendung in der Praxis

```
int result = a + 5;
```

**Compiler-Frage:** Welches 'a' ist gemeint?

```
int a = 100
```

```
int a = 5
```

"**Lokaler Scope**" vs. "**Globaler Scope**"

Das lokale a wird gefunden, da die Suche immer im innersten Scope beginnt.

# Erweiterung: Methoden in der Symboltabelle

## Beispiel: Eine Methode

```
int add(int a, int b)
{
    int summe = a + b;
    return summe;
}
```

## Darstellung in der Symboltabelle

Name	Typ	Scope	Attribute
add	Funktion	global	(a,b: int) -> int
a	int	add	Parameter
b	int	add	Parameter
summe	int	add	Lokal Variable

# Erweiterung: Klassen in der Symboltabelle

## Beispiel: Eine Klasse

```
class Zaehler
{
public:
    int wert;

    Zaehler(int startWert) {
        wert = startWert;
    }
};
```

## Darstellung in der Symboltabelle

Name	Typ	Scope	Attribute
Zaehler	Klasse	global	
wert	int	Zaehler	Member-Var
Zaehler	Konstruktor	Zaehler	(startWert: int)
startWert	int	Z::Zaehler	Parameter

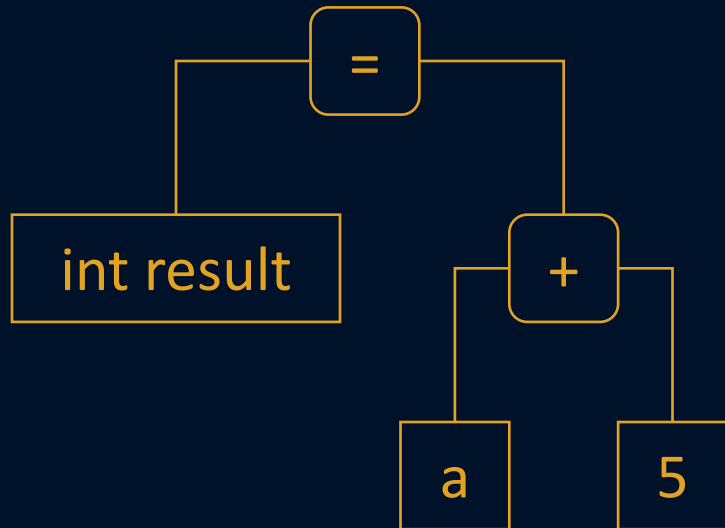
## Code-Beispiel: Shadowing

```
int a = 5    // • Globaler Scope  
void scopeBeispiel() {  
    int a = 100; // • Lokaler Scope  
    int result = a + 1;  
  
    // Welches 'a' wird  
    hier verwendet?  
  
}    => (result wird 101)
```

## Hierarchische Suche

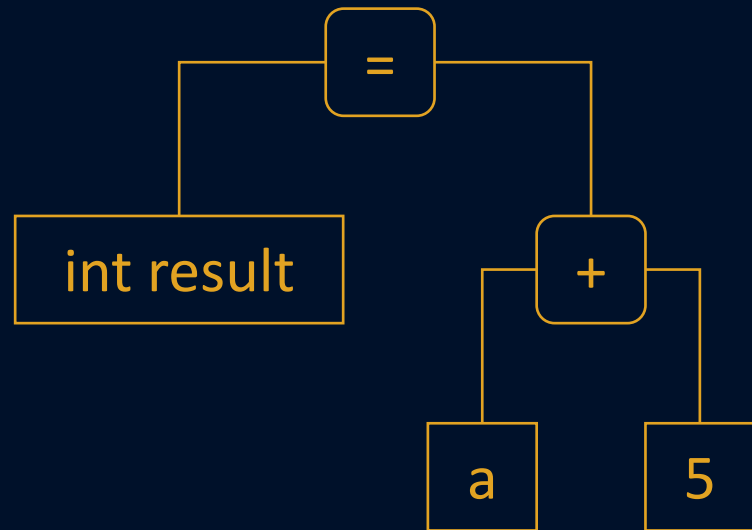
- Der Compiler sucht immer zuerst im **innersten (lokalen)** Scope.
- Er findet  $a = 100$  und verwendet diesen Wert für die Addition.
- Der globale  $a = 5$  wird "**überschattet**" (shadowed).

## Die Struktur (AST)



## Das Gedächtnis (Symboltabelle)

Name	Typ	Scope
a	int	global
result	int	lokal



Prüfung erfolgreich!

## 1. Prüfung des '+' Knotens

→ links: 'a' → Typ: int

→ rechts: '5' → Typ: int

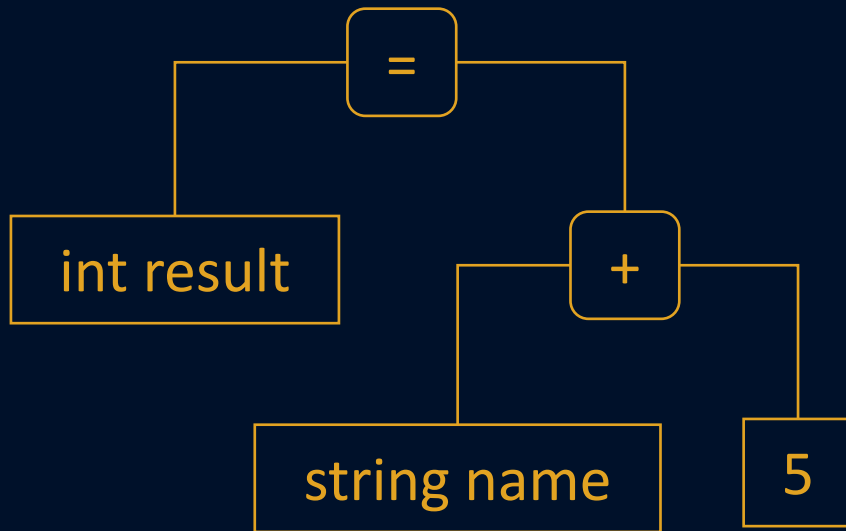
=> int + int  **gültig**. Ergebnis ist int.

## 2. Prüfung des '=' Knotens

→ rechts: Ergebnis von '+' → Typ: int

→ links: 'result' → Typ: int

=> int = int  **gültig**.



## 1. Prüfung des '+' Knotens

→ links: 'name' → Typ: string

→ rechts: '5' → Typ: int

=> string + int ❌ **ungültig!**

Der '=' Knoten wird nicht mehr erreicht.

❌ Prüfung nicht bestanden!

### 1. Nicht deklarierte Variable

```
x = 10;
```

**Fehler:** 'x' nicht deklariert

### 3. Falsche Anzahl an Argumenten

```
myFunction(arg1, arg2);
```

**Fehler:** Falsche Argumentanzahl

### 2. Doppelte Deklaration

```
int a = 5;
```

```
string a = "hello";
```

**Fehler:** 'a' doppelt deklariert

### 4. Zuweisung an Konstante

```
const int a = 10; a = 20;
```

**Fehler:** Zuweisung an Konstante



## 1. Zwischencode-Erzeugung

**Prozess:** AST  $\rightarrow$  plattformunabhängiger IR

**Analogie:** Standard-Bauanleitung

## 2. Optimierung

**Prozess:** IR wird schneller und kleiner gemacht.

**Analogie:** Bauanleitung effizienter machen

## 3. Zielcode-Erzeugung

**Prozess:** IR  $\rightarrow$  finaler Maschinencode.

**Analogie:** Anleitung in Maschinensprache übersetzen

- Der Compiler übersetzt Code in logischen Phasen.
- Der **AST** ist die essenzielle, saubere Struktur für die Analyse.
- Die **Semantische Analyse** prüft die logische Korrektheit des Codes.
- Die **Symboltabelle** dient als "Gedächtnis" für Variablen und Funktionen.
- Kernaufgaben der Semantischen Analyse: **Typprüfung & Scope-Analyse**.



Vielen Dank für Ihre Aufmerksamkeit!



Frage ~

## Primärquelle

- Fachbereich Elektrotechnik und Informationstechnik, Campus Minden. (2025). *CB-Vorlesung-Bachelor-W25*. GitHub Repository. Abgerufen am 9. Oktober 2025, von <https://github.com/Compiler-CampusMinden/CB-Vorlesung-Bachelor-W25>

## Werkzeuge & Medien

- **Unterstützung bei Inhalt & Design:** Künstliche Intelligenz (KI)
- **Icons:** Flaticon.com