**1. Aussage auswerten**

**2. Ast erstellen**

**3. Aussage von einem Ast ableiten**

**4. Pseudocode erstellen. Also 3 reguläre Ausdrücke bekommen und den jeweiligen parser erstellen**

1. **Hier ist eine theoretische Aufgabe 1 und ich berechne nur den Wert des Beispiels (x \* y + z == 42 ? x : y), wobei wir konkrete Werte für x, y und z einsetzen.**

Beispiel:

Angenommen:

* x = 5
* y = 4
* z = 22

Schritte:

1. Berechne den arithmetischen Ausdruck vor dem Vergleich:  
   x \* y + z  
   → 5 \* 4 + 22  
   → 20 + 22  
   → 42
2. Vergleiche mit == 42:  
   42 == 42  
   → true
3. Wende den Ternär-Operator ? : an:  
   Da die Bedingung true ist, wird x zurückgegeben.

**Ergebnis:  
5**

**Grundlagen  
Präzedenz und Assoziativität beachten. Lerne die Reihenfolge der Operatoren:**

* Vergleich (==, !=) hat niedrigere Priorität als arithmetische Operatoren (+, -, \*, /).
* Klammern () ändern die Reihenfolge.
* Beispiel: x + y \* z wird zuerst als y \* z berechnet, dann das Ergebnis mit x addiert.

1. **Abstrakte Syntaxbäume (AST)  
   zeichne einen AST für gegebene Ausdrücke.**

* Beispiel 1: Für den Ausdruck x + y \* z == 42 ? a : b:  
  SELECT  
  |---COMPARE  
  | |---ADDITION  
  | | |---x  
  | | |---MULTIPLICATION  
  | | |---y  
  | | |---z  
  | |---42  
  |---a  
  |---b
* Beachte: Jeder Operator ist ein Knoten, Operanden (x, y, z, 42, a, b) sind Blätter.

1. **Codeverständnis  
   Schreibe den Code für die Stackmaschine oder Registermaschine basierend auf den ASTs.**

* Beispiel für Stackmaschine:  
  PUSH x  
  PUSH y  
  PUSH z  
  MULT  
  ADD  
  PUSH 42  
  EQUALS  
  JUMP\_IF\_FALSE label1  
  PUSH a  
  JUMP label2  
  label1: PUSH b  
  label2:
* Beispiel für Registermaschine:  
  r1 = x  
  r2 = y \* z  
  r3 = r1 + r2  
  if r3 == 42: r4 = a else: r4 = b

1. Pseudocode schreiben  
   Parser für gegebene Grammatik.

* Beispiel: Eine Grammatik akzeptieren und verarbeiten.  
  rule SelectExpression():  
  if token == SELECT:  
  acceptIt()  
  parse(Expression)  
  if token == "?":  
  acceptIt()  
  parse(Expression)  
  if token == ":":  
  acceptIt()  
  parse(Expression)  
  else:  
  error("Expected SELECT")
* Nutze accept und acceptIt, um Tokens zu verarbeiten. Kein Rückgabewert erforderlich, nur Fehlerprüfung.

1. **Übungsaufgaben:  
   a) Erstelle einen AST für (x \* y + z == 42 ? x : y).  
   b) Schreibe den Stackmaschinen-Code für den obigen Ausdruck.**  
   **c) Schreibe einen Pseudocode-Parser für eine Grammatik, die while-Schleifen akzeptiert.**

Beispiel für while: "while (x < y) do { z = z + 1 }"  
rule WhileLoop():  
if token == WHILE:  
acceptIt()  
parse(Condition)  
if token == "do":  
acceptIt()  
parse(Block)  
else:  
error("Expected WHILE")