Aufgabe 1

(a)

Preorder $+(*(:(8,3),7),-(4,*(+(1,5),2)))$

Inorder $(((8:3)*7)+(4-((1+5)*2)))$

Postorder $(((8,3):,7)*,(4,((1,5)+,2)*)-)+$

(b) Preorder und Postorder sind auch ohne Klammerung und Operator-Rangfolge eindeutig

(c) Ja, das ist einfach Postorder rückwärts gelesen

Aufgabe 2

Syntax: (Zahl)

$$\text{<zahl>} ::= [+|-]\{0|1|2|3|4|5|6|7|8|9\}^+$$

Syntax: (Index)

$$\text{<index>} ::= \{_0|_1|_2|_3|_4|_5|_6|_7|_8|_9\}^+$$

Syntax: (Unterterm)

$$\text{<unterterm>} ::= \text{<zahl>}\underline{\text{x}}\text{<index>}$$

Syntax: (Term)

$$\text{<term>} ::= \{\text{unterterm}\}^+$$

Syntax: (Gleichung)

$$\text{<gleichung>} ::= \text{<term>}\underline{=}\text{<zahl>}$$

Syntax: (Gleichungssystem)

$$\text{<gleichungssystem>} ::= \{\text{<gleichung>}\underline{\backslash n}\}^+$$

Nein, in der ersten Zeile des Gleichungssystems können beliebig viele Variablen x_n vorkommen. Es gibt keine Möglichkeit, zu kontrollieren, dass höchstens n Gleichungen darunter vorkommen.

Aufgabe 3

Listing 1: potenz.cc

```
1 #include "fcpp.hh"
2
3 int quadrat (int x)
4 {
5   return x*x;
```

```
6  }
7
8  int potenz(int x, int exp)
9  {
10   return cond (exp == 1, x,
11               cond( exp % 2 == 0, quadrat( potenz(x, exp / 2)), x * potenz(x, exp - 1)));
12 }
13
14 int main ()
15 {
16   return print( potenz(2,4));
17 }
```

applikativ

$$
\begin{aligned}
&\text{potenz}(2,4)\\
&= \text{cond}(4 == 1, 2, \text{cond}(4\%2 == 0, \text{quadrat}(\text{potenz}(2, 4/2)), 2 \cdot \text{potenz}(2, 4-1)))\\
&= \text{cond}(0 == 0, \text{quadrat}(\text{potenz}(2,2)), 2 \cdot \text{potenz}(2,3))\\
&= \text{quadrat}(\text{potenz}(2,2))\\
&= \text{quadrat}(\text{cond}(2 == 1, 2, \text{cond}(2\%2 == 0, \text{quadrat}(\text{potenz}(2, 2/2)), 2 \cdot \text{potenz}(2, 2-1))))\\
&= \text{quadrat}(\text{cond}(0 == 0, \text{quadrat}(\text{potenz}(2,1)), 2 \cdot \text{potenz}(2,1)))\\
&= \text{quadrat}(\text{quadrat}(\text{potenz}(2,1)))\\
&= \text{quadrat}(\text{quadrat}(\text{cond}(1 == 1, 2, \text{cond}(1\%2 == 0, \text{quadrat}(\text{potenz}(2, 1/2)), 2 \cdot \text{potenz}(2, 1-1)))))\\
&= \text{quadrat}(\text{quadrat}(2)) = \text{quadrat}(2*2) = \text{quadrat}(4) = 4*4 = 16
\end{aligned}
$$

normal

$$\begin{aligned}
&\text{potenz}(2,4)\\
&=\text{cond}(4==1,2,\text{cond}(4\%2==0,\text{quadrat}(\text{potenz}(2,4/2)),2\cdot\text{potenz}(2,4-1)))\\
&=\text{cond}(0==0,\text{quadrat}(\text{potenz}(2,(4/2))),2\cdot\text{potenz}(2,4-1))\\
&=\text{quadrat}(\text{potenz}(2,(4/2)))\\
&=\text{potenz}(2,(4/2))*\text{potenz}(2,(4/2))\\
&=\text{cond}((4/2)==1,2,\text{cond}((4/2)\%2==0,\text{quadrat}(\text{potenz}(2,(4/2)/2)),2\cdot\text{potenz}(2,(4/2)-1)))\\
&*\text{cond}((4/2)==1,2,\text{cond}((4/2)\%2==0,\text{quadrat}(\text{potenz}(2,(4/2)/2)),2\cdot\text{potenz}(2,(4/2)-1)))\\
&=\text{cond}(0==0,\text{quadrat}(\text{potenz}(2,(4/2)/2),2\cdot\text{potenz}(2,(4/2)-1)\\
&*\text{cond}(0==0,\text{quadrat}(\text{potenz}(2,(4/2)/2),2\cdot\text{potenz}(2,(4/2)-1)\\
&=\text{quadrat}(\text{potenz}(2,(4/2)/2))*\text{quadrat}(\text{potenz}(2,(4/2)/2))\\
&=\text{potenz}(2,(4/2)/2)*\text{potenz}(2,(4/2)/2)*\text{potenz}(2,(4/2)/2)*\text{potenz}(2,(4/2)/2)\\
&=\text{cond}((4/2)/2)==1,2,\text{cond}((4/2)/2)\%2==0,\text{quadrat}(\text{potenz}(2,(4/2)/2)/2)),2\cdot\text{potenz}(2,(4/2)/2)-1)))\\
&*\text{cond}((4/2)/2)==1,2,\text{cond}((4/2)/2)\%2==0,\text{quadrat}(\text{potenz}(2,(4/2)/2)/2)),2\cdot\text{potenz}(2,(4/2)/2)-1)))\\
&*\text{cond}((4/2)/2)==1,2,\text{cond}((4/2)/2)\%2==0,\text{quadrat}(\text{potenz}(2,(4/2)/2)/2)),2\cdot\text{potenz}(2,(4/2)/2)-1)))\\
&*\text{cond}((4/2)/2)==1,2,\text{cond}((4/2)/2)\%2==0,\text{quadrat}(\text{potenz}(2,(4/2)/2)/2)),2\cdot\text{potenz}(2,(4/2)/2)-1)))\\
&=2*2*2*2=16
\end{aligned}$$

Wird der Code normal ausgewertet, so erhöht sich die Effizienz durch unseren Algorithmus überhaupt nicht. Wird der Code stattdessen applikativ ausgewertet, so erhält man die gewünschte Effizienzsteigerung.

Aufgabe 4

Listing 2: vollkommenezahlen.cc

```
1  #include "fcpp.hh"
2
3  int teiler(int a, int b)
4  {
5      return cond(a % b == 0, b, 0);
6  }
7
8  int teilersumme(int gesamtzahl, int max)
9  {
10     return teiler(gesamtzahl, max) + cond(max > 1, teilersumme(gesamtzahl, max-1), 0);
11 }
12
13 bool vollkommen(int zahl)
14 {
15     return cond(teilersumme(zahl, zahl-1) == zahl, true, false);
16 }
17
18 int main()
19 {
20     return print(vollkommen(497));
21 }
```