TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK
Computer Vision Group
Einführung in die Informatik I für Ingenieure (MSE)
Prof. Dr. Daniel Cremers, Mohamed Souiai

WS 2012
Übungsblatt 4
16.11.2012

Aufgabe 4.1 (Ü) Arrays

Diskutieren Sie, was in dem folgenden Programm passiert.

```
1  public class MeineArrays extends MiniJava {
2      public static void main(String[] args) {
3
4          int[] erstesA = {1,2,3,4};
5          write(erstesA[1]);
6
7          int[] zweitesA = new int[7];
8          zweitesA[0] = 3;
9          int i=1;
10         while(i<zweitesA.length) {
11             zweitesA[i] = i+3;
12             i++;
13         }
14
15         for(int j=0;j<zweitesA.length;j++) {
16             write(zweitesA[j]);
17             i++;
18         }
19
20         write(zweitesA[i]);
21
22         zweitesA = erstesA;
23         zweitesA[1]=25;
24         write(erstesA[1]);
25     }
26 }
```

Lösungsvorschlag 4.1

Folgendes passiert in dem Programm:

- 4 Anlegen eines Arrays mit festem Inhalt
- 5 Ausgeben des Wertes an Position 1 $\rightarrow$ 2 (Die Positionen beginnen bei 0.)
- 7 Anlegen eines leeren Arrays der Länge 7
- 8 Füllen der Position 0 mit dem Wert 3
- 11 Füllen der i-ten Position mit $i + 3$

9-13 Das zweite Array ist $[3, 4, 5, 6, 7, 8, 9]$

15 For-Schleife durchläuft das Array

15-18 Schrittweises Ausgeben des Arrays

17 Variable i wird insgesamt um Länge von Array zweitesA hochgezählt

20 Laufzeitfehler! Position i existiert in Array zweitesA nicht. Es wird eine `ArrayIndexOutOfBoundsException` geworfen.

22 Die Referenz auf Array `erstesA` wird in Variable `zweitesA` kopiert.

23 Das Feld 1 des Arrays, auf das sich `zweitesA` bezieht, wird auf 25 gesetzt.

24 Durch Zeile 23 wurde auch diese Ausgabe beeinflusst, da `erstesA` seit Zeile 22 das selbe Array referenziert wie `zweitesA`.

Aufgabe 4.2 (Ü) Minimum und Maximum finden

Schreiben Sie ein `MiniJava`-Programm namens `MinMax.java`, das in einem Array von ganzen Zahlen den kleinsten und den größten Wert findet. Das Programm soll sich wie folgt verhalten:

- Zunächst fragt das Programm ab, wie viele Zahlen in das Array eingegeben werden sollen.
- Dann werden die Zahlen eingegeben und in einem Array gespeichert.
- Anschließend wird das Array durchsucht und in **einem Durchgang** soll sowohl die kleinste als auch die größte Zahl gefunden werden.
- Schließlich sollen die kleinste und die größte Zahl ausgegeben werden.

Lösungsvorschlag 4.2

```
public class MinMax extends MiniJava {

    public static void main(String[] args) {

        // Anzahl der einzugebenden Zahlen abfragen
        int eingaben = readInt("Wie viele Zahlen möchten Sie eingeben?"
            );
        if (eingaben < 1) {
            return;
        }

        // Array anlegen
        int[] zahlen = new int[eingaben];


        // Array mit Zahlen füllen
        int zahl;
        int i = 0;
        while (i < eingaben) {
            zahl = readInt("Bitte geben Sie die "
                    + (i + 1) + ". Zahl ein.");
```

```
            zahlen[i] = zahl;
            i++;
        }

        // min und max berechnen
        int min = zahlen[0];
        int max = zahlen[0];
        i = 1;
        while (i < eingaben) {
            if (zahlen[i] < min) {
                min = zahlen[i];
            }
            if (zahlen[i] > max) {
                max = zahlen[i];
            }
            i++;
        }

        // Ausgeben
        write("Die kleinste Zahl war " + min);
        write("Die größte Zahl war " + max);

    }
}
```

Aufgabe 4.3 (Ü) Sieb Des Eratosthenes (wenn Sie noch Zeit haben)

Folgendes Verfahren zur Primzahlen-Gewinnung ist als **Sieb des Eratosthenes** bekannt:

Aus der Menge der natürlichen Zahlen von 2 bis n werden alle Nicht-Primzahlen gestrichen. Beginne mit der Zahl 2 und streiche alle echten Vielfachen von 2. Streiche nun sukzessiv alle echten Vielfachen der jeweils nächsthöheren noch nicht gestrichenen Zahl. Nach Streichung aller Vielfachen enthält die Restmenge nur noch Primzahlen.

Implementieren Sie das oben beschriebene Verfahren in Java mit Hilfe eines Arrays. Ihr Programm soll nach Festlegung einer Obergrenze durch den Benutzer alle Primzahlen bis zu dieser Grenze bestimmen und ausgeben.

Lösungsvorschlag 4.3

```
public class Eratosthenes extends MiniJava {

    public static void main(String[] args) {
        // Eingabe
        int n = read();

        if (n < 0) {
            // fehlerhafte Eingabe
            write("Nur positive Eingaben erlaubt.");
        } else {
            // korrekte Eingabe; Berechnung kann erfolgen

            // Sieb anlegen
            boolean prime[] = new boolean[n + 1];
            for (int i = 2; i < prime.length; i++)
                prime[i] = true;
```

```
            // Primzahlen berechnen
            for (int i = 2; i < prime.length; i++) {
                if (prime[i]) {
                    // Primzahl ausgeben
                    System.out.println(i);
                    // alle Vielfachen streichen
                    for (int j = i * i; j < prime.length; j = j + i) {
                        prime[j] = false;
                    }
                }
            }
        }
    }
}
```

Mögliche Optimierungen:

- Nur ungerade Zahlen berücksichtigen
- Berechnung bis zur Wurzel der oberen Grenze ist ausreichend

Aufgabe 4.4 [Syntaxbaum Punkte] **(H) Gegeben ist das folgende MiniJava-Programm:**

```
int x, y;
y = read();
x = 1 + (y * y);
while (x % y == 0){
  x = x - 1;
  if (x > 0)
    y = x / y;
}
```

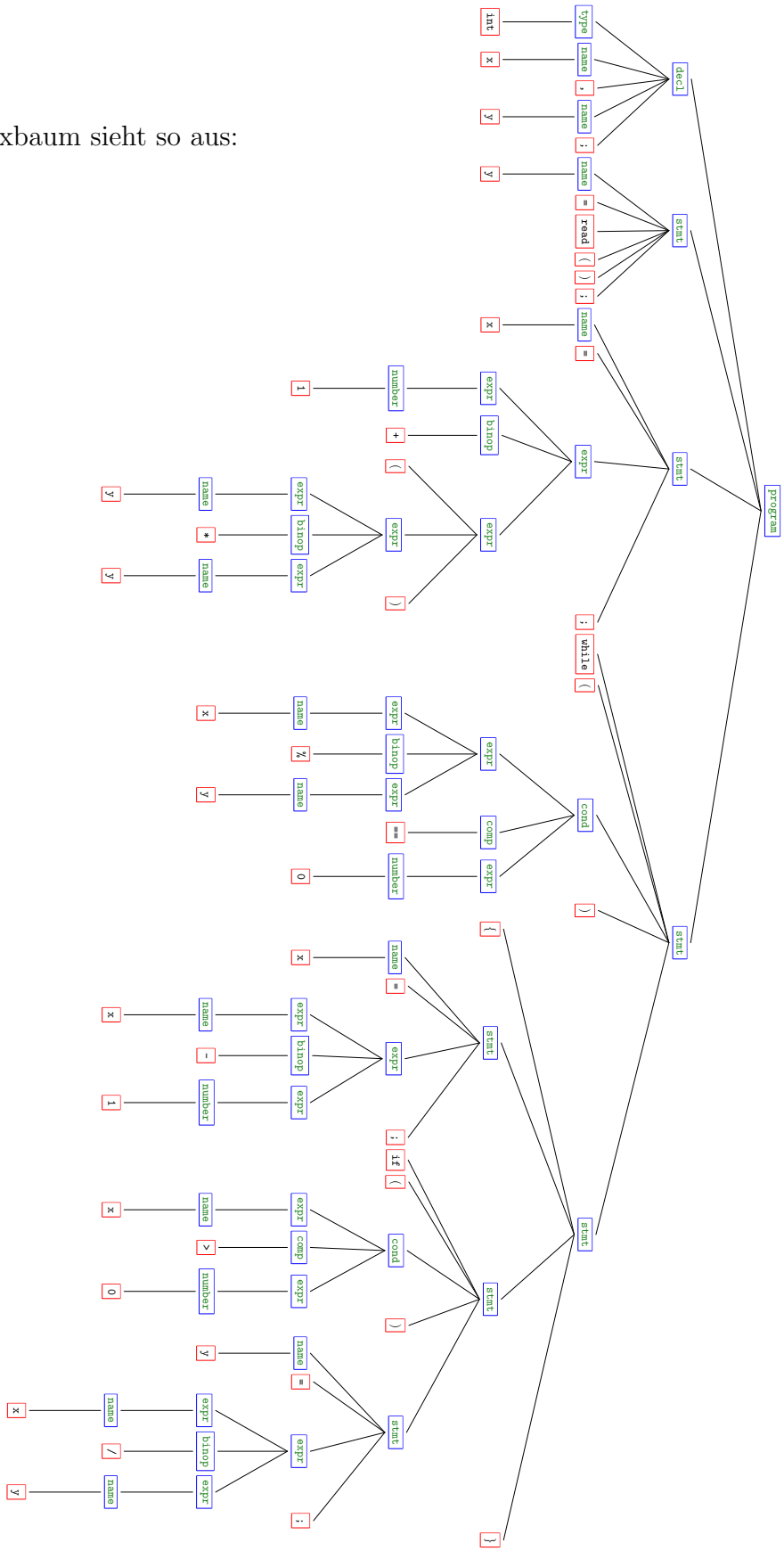
Erstellen Sie den Syntaxbaum dieses Programmes!

Hinweis: Platz für den Syntaxbaum finden Sie auf der nächsten Seite, die MiniJava-Grammatik finden Sie im Anhang!

```
int x , y ; y = read() ; x = 1 + ( y * y ) ; while ( x % y == 0 ) { x = x - 1 ; if ( x > 0 ) y = x / y ; }
```

Lösungsvorschlag 4.3

a) Der Syntaxbaum sieht so aus:



Die Grammatik von MiniJava (aus der Vorlesung):

<program>	::=	<decl>* <stmt>*
<decl>	::=	<type> <*name*> (`,` <*name*>)* `;`
<type>	::=	`int`
<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>	::=	`-`
<binop>	::=	`-` | `+` | `*` | `/` | `%`
<cond>	::=	`true` | `false` | `(` <cond> `)` |
		<expr> <comp> <expr> |
		<bunop> <cond> | <cond> <bbinop> <cond>
<comp>	::=	`==` | `!=` | `<=` | `<` | `>=` | `>`
<bunop>	::=	`!`
<bbinop>	::=	`&&` | `||`

<*name*> und <*number*> für Bezeichner und Zahlen werden nicht weiter verfeinert.