Aufgabe 3.1 (Ü) Reguläre Ausdrücke

Schreiben Sie reguläre Ausdrücke für die folgenden Mengen:

a) Die Menge der nichtleeren Zeichenketten, die nur aus Nullen und Einsen bestehen.
b) Die Menge aller Zeichenreihen aus a,b,c die mindesten ein a und mindestens ein b enthalten.
c) Die Menge der Zeichenreihen, in denen die Anzahl der vorhandenen Nullen ein Vielfaches von 3 ist.
d) Die Menge der Zeichenreihen aus Nullen und Einsen, die mit einer Null anfangen und eine ungerade Länge haben, oder mit einer Eins anfangen und eine gerade Länge haben.
e) Die Menge der Zeichenreihen aus Nullen und Einsen, die die Folge 110 nicht enthalten.

Lösungsvorschlag 3.1

a) (0|1)(0|1)*
b) (a|b|c)*a(a|b|c)*b(a|b|c)*| (a|b|c)*b(a|b|c)*a(a|b|c)*
c) 1* | (1*01*01*01*)*
d) 0((0|1)(0|1))* | 1(0|1)((0|1)(0|1))*
e) (0|10)*1*

Aufgabe 3.2 (Ü) Syntaxbaum

Gegeben ist das folgende MiniJava-Programm:

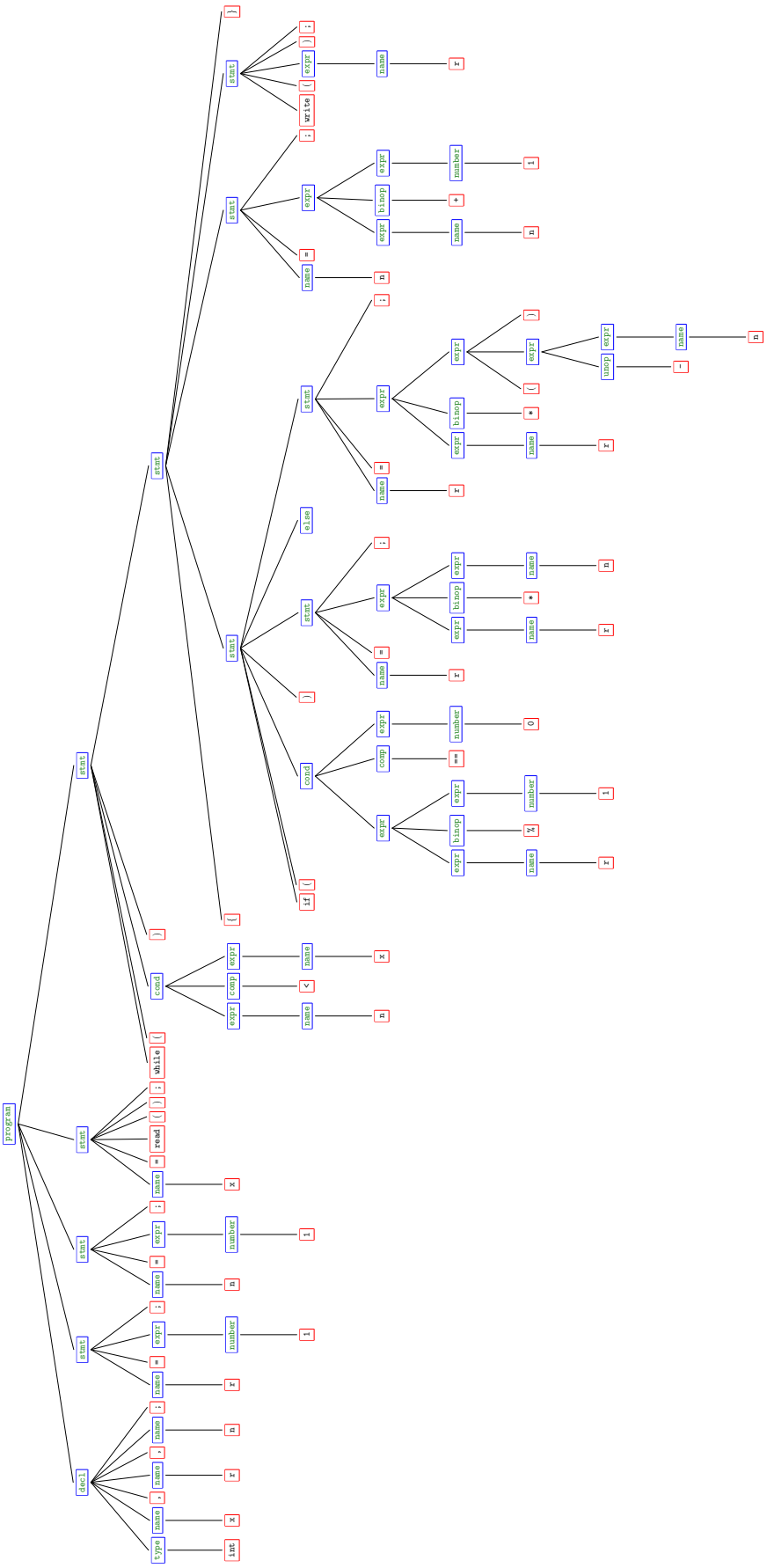
```
int x,r,n;
r = 1;
n = 1;
x = read();
while (n < x) {
    if (r % 1 == 0)
        r = r * n;
    else
        r = r * (-n);
    n = n + 1;
    write (r);
}
```

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 , r , n ; r = 1 ; n = 1 ; x = read() ; while ( n < x ) { if ( r % 1 == 0 ) r = r * n ; else r = r * ( - n ) ; n = n + 1 ; write ( r ) ; }
```

Lösungsvorschlag 3.2

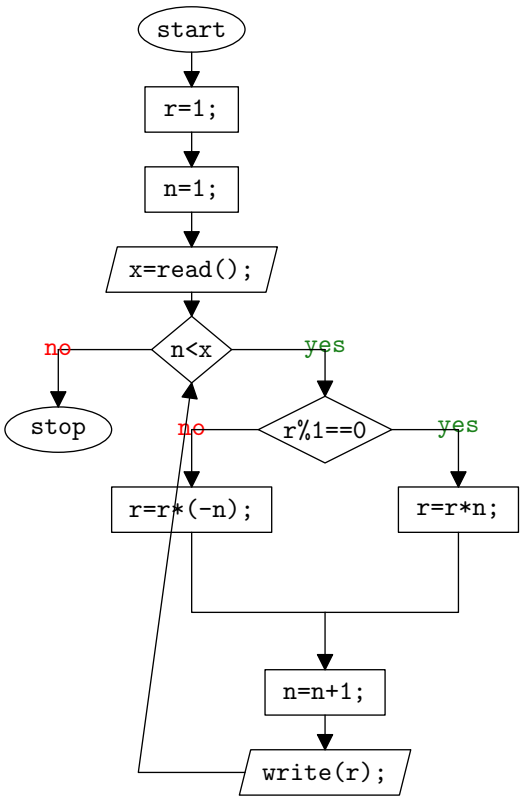


Aufgabe 3.3 (Ü) Kontrollflussdiagramm

Zeichnen Sie für das folgende `MiniJava`-Programm den Kontrollflussgraphen.

```
int x,r,n;
r = 1;
n = 1;
x = read();
while (n < x) {
    if (r % 1 == 0)
        r = r * n;
    else
        r = r * (-n);
    n = n + 1;
    write (r);
}
```

Lösungsvorschlag 3.3



Aufgabe 3.4 (Ü) Suche nach der Wurzel

In dieser Aufgabe wollen wir die Quadratwurzel zu einer Zahl $a \geq 1$ näherungsweise bestimmen. In diesem Bereich gilt $1 \leq \sqrt{a} \leq a$. Da die reelle Wurzelfunktion monoton wächst können wir eine binäre Suche anwenden. Dabei wird der Suchbereich immer wieder halbiert und in der Hälfte weiter gesucht, in der sich die Lösung befinden muss, bis die Lösung gefunden, oder bis auf einen vorgegebenen Fehler ϵ angenähert ist. Sucht man im Intervall $[b_1, b_2]$ nach dem Wert x, so dass $x = \sqrt{a}$, bestimmt man die (zur Wurzel inverse) Funktion $f(x) = x^2$ an der Mitte des Intervalls $m = \frac{b_1+b_2}{2}$. Ist der Wert $(f(m) - a) < \epsilon$, endet die Berechnung. Ist $f(m) > a$ ergibt sich, dass $b_1 < x < m$ sein muss. In diesem Fall wiederholt man die Suche im Intervall $[b_1, m]$. Analog sucht man falls $f(m) < a$ im Intervall $[m, b_2]$ weiter.

a) Schreiben Sie ein MiniJava-Programm `Wurzel.java`, welches für eine einzulesende Zahl a die Quadratwurzel $\sqrt{a}$ nach obigem Algorithmus näherungsweise bestimmt, in dem die Iteration nach einer vom Benutzer festgelegten Anzahl von Durchläufen abgebrochen wird.
b) Ermöglichen sie dem Benutzer die Fehlertoleranz ϵ festzulegen. Ihr Programm soll nun so lange suchen, bis es diese Fehlertoleranz erreicht.

Lösungsvorschlag 3.4

```
public class Wurzel extends MiniJava {
    public static void main(String[] args){
        double a, eps, l,r,m,d;
        a = readDouble("Zahl");
        if(a>=1.0)
        {
            eps = readDouble("Fehler");
            l=1.0; //sqrt(a) in [1,a]
            r=a;

            m=(l+r)/2.0;//find middle of interval
            d=m*m-a;//initial error
            while (d*d > eps*eps){//is error small enough?
                if(m*m<a) //left or right interval?
                    l=m;
                else //m*m=a not possible
                    r=m;

                m=(l+r)/2.0; //find middle of cur interval
                d=m*m-a; //calc error
            }
            write("Die Wurzel von "+a+" ist "+m+", mit Ungenauigkeit "
               +d + " gefordert " + eps);
        }
        else
            write("Fehler, Zahl muss groesser gleich 1.0 sein");
    }
}
```

Aufgabe 3.5 (H) Suche nach der Wurzel - Teil II

Das Heron-Verfahren (auch Babylonisches Wurzelziehen genannt) ist ein weiterer iterativer Algorithmus zur Bestimmung einer rationalen Näherung der Quadratwurzel $\sqrt{a}$ einer reellen Zahl $a \geq 0$. Die Iterationsvorschrift lautet:

$$x_{n+1} \quad = \quad \frac{1}{2} \cdot (x_n + \frac{a}{x_n})$$

Hierbei steht a für die Zahl, deren Quadratwurzel bestimmt werden soll. Der Startwert x_0 der Iteration kann, solange er grösser Null ist, beliebig festgesetzt werden.

a) Schreiben Sie ein MiniJava-Programm `Heron.java`, welches für eine einzulesende Zahl a und den Startwert $x_0 = a$ die Quadratwurzel $\sqrt{a}$ näherungsweise, bis auf einen vom Benutzer abgefragten maximalen Fehler ϵ berechnet und ausgibt.

b) In diesem Teil wollen wir die beiden Wurzelverfahren vergleichen. Modifizieren Sie `Heron.java` und `Wurzel.java` aus Tutoraufgabe 2.2 jeweils so, dass beide Methoden Ergebnisse mit gleichem Fehler $\epsilon = 0.0000001$ produzieren. Fügen Sie zusätzlichen Code ein, um die Anzahl der benötigten Schleifendurchläufe zu zählen und um zu testen, welches der Verfahren zur Bestimmung der Wuzerl der Zahlen von 2 bis 10 (a aus {2;3...;10}) mit $x_0 = a$ als jeweiligem Startwert weniger Schleifendurchläufe braucht.

Lösungsvorschlag 3.5

```
public class Heron extends MiniJava {
    public static void main(String[] args){
        double x, a, eps;
        a = readDouble("Bitte␣positive␣Zahl␣zum␣Wurzel␣ziehen␣eingeben"
            );
        while (a<0){
            a=readDouble("Bitte␣---positive---␣Zahl␣zum␣Wurzel␣ziehen␣
                eingeben");
        }
        //eps=0.0000001;
        eps = readDouble("Bitte␣tolerierbaren␣maximalen␣Fehler␣angeben"
            );
        x = a;
        while ((x*x-a) > eps || (x*x-a) < -eps){
            x=(x+a/x)/2.0;
        }
        write("Die␣Wurzel␣von␣"+a+"␣ist␣"+x+"␣,␣mit␣Ungenauigkeit␣"+(
            Math.sqrt(a)-Math.abs(x)));

    }
}

public class Vergleichstest extends MiniJava {
    public static void main(String[] args){

        double heronx;
        double binaerx;
        double x, a, eps, l,r,m,d;
        int heron,binaer;

        eps=0.0000001;
        a=2;
        while (a<11) {
            // Heron
            heron=0;
            x = a;
            while ((x*x-a) > eps || (x*x-a) < -eps){
                x=(x+a/x)/2.0;
                heron=heron+1;
            }
            heronx=x;

            // Binaere Suche
            binaer=0;
            l=1.0; //sqrt(a) in [1,a]
            r=a;
            m=(l+r)/2.0;//find middle of interval
            d=m*m-a;//initial error
            while (d*d > eps*eps){//is error small enough?
                if(m*m<a) //left or right interval?
                    l=m;
                else //m*m=a not possible
                    r=m;

                m=(l+r)/2.0; //find middle of cur interval
                d=m*m-a; //calc error
                binaer=binaer+1;
```

```
            }
            binaerx = m;

            // Vergleich:
            System.out.println(a+": Heron: "+(heronx*heronx)+"/#"+heron
               +" Bin. Suche: "+(binaerx*binaerx)+"/#"+binaer);
            a=a+1;
        }
    }
}
```

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.