Angewandte Mathematik.

Primzahlproblematiken.

Posselt, Christian
Schubert, Jonathan

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Primzahlproblematiken.

Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Angewandte Mathematik.
Selbst gelöste Probleme.

Teilbarkeit & Primzahlen:

382 – Perfection
583 – Prime Factors
10392 – Factoring Large Numbers
10699 – Count the Factors
11417 – GCD
11466 – Largest Prime Devisor

Zahlen, Zahlen, Zahlen:

256 – Quirksome Squares
371 – Ackermann
640 – Self Numbers
10579 – Fibonacci Numbers
10622 – Perfect Pth Power

Angewandte Mathematik.
Selbst gelöste Probleme.

Zeichenketten:

445 – Marvelous Mazes
10260 – Soundex
10282 – Bubblefish
10815 – Andy‘s First Dictonary
11233 – Deli Deli

Verschiedene Formeln:

484 – The Departement of Redundancy Department
579 – Clock Hands
10023 – Square Root
10070 – Leap Year or Not Leap Year and...
10107 – What is the Median?
10108 – Beat the Spread!
11063 – B2-Sequence
11172 – Relational Operator

Angewandte Mathematik.
Selbst gelöste Probleme.

Basisumwandlung:

389 – Basically Speaking
446 – Kibbles "n" Bits "n" Bits "n" Bits
575 – Skew Binary
10038 – Jolly Jumpers
10473 – Simple Base Conversion
11185 – Ternary

Große Zahlen, Modulo, Kombinatorik :

288 – Arithmetic Operations with Large Integers
485 – Pascal Triangle of Death
623 – 500!
11172 – Relational Operator

Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlproblematiken.

Schönste Probleme.

1. Next Same Factored (11099)
2. Bank (Not Quite O.C.R) (433)
3. Clock Hands (579)

Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlproblematiken.
Schönste Programme.

Big Chocolate (10970):	Tschannerl, Leib
Magic Number (471):	Wilfert, Studt
Square Root (10023):	Posselt, Schubert

Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlen.
Begriff.

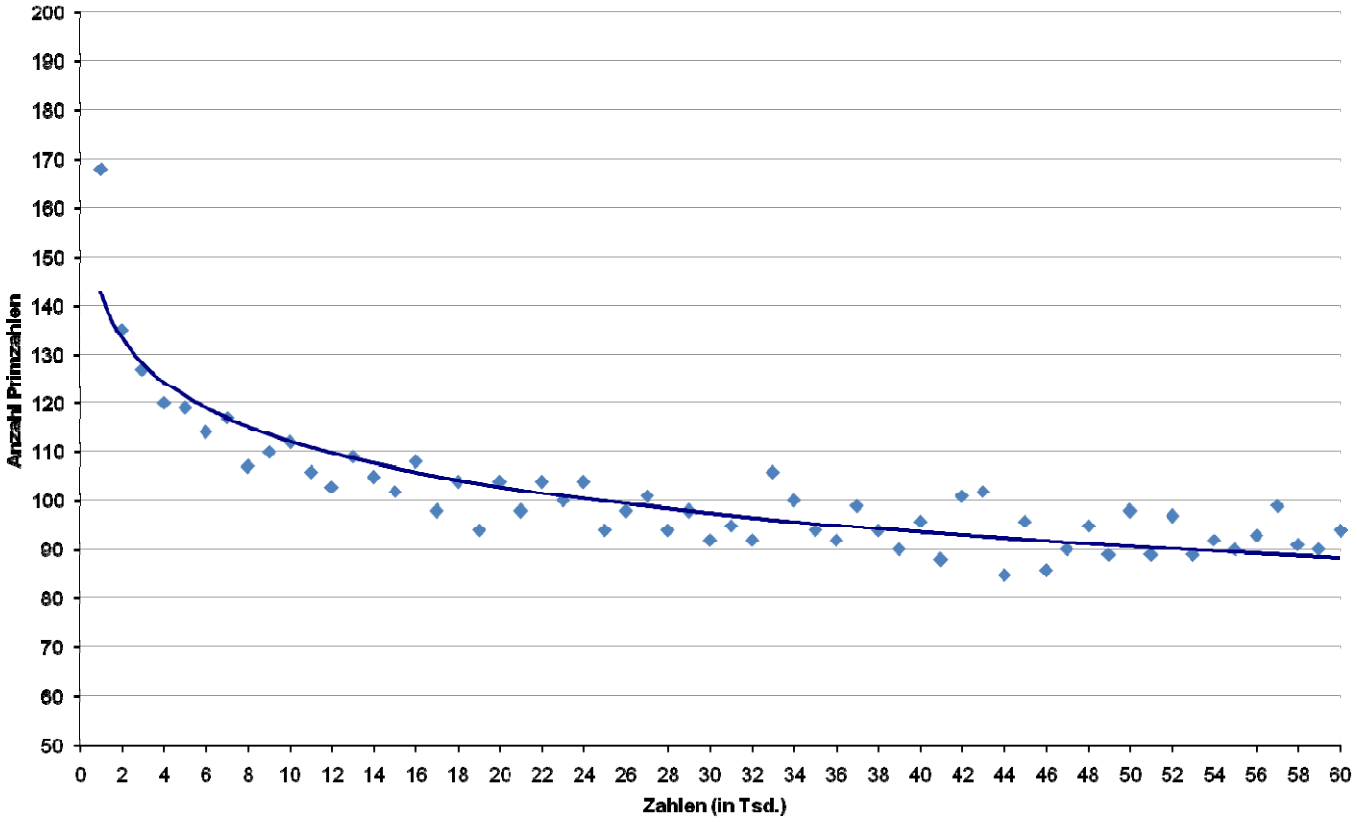
Definition:

Eine Zahl p, die außer den (trivialen) Teilern 1 und p (sich selbst) keine weiteren Teiler hat, heißt Primzahl.

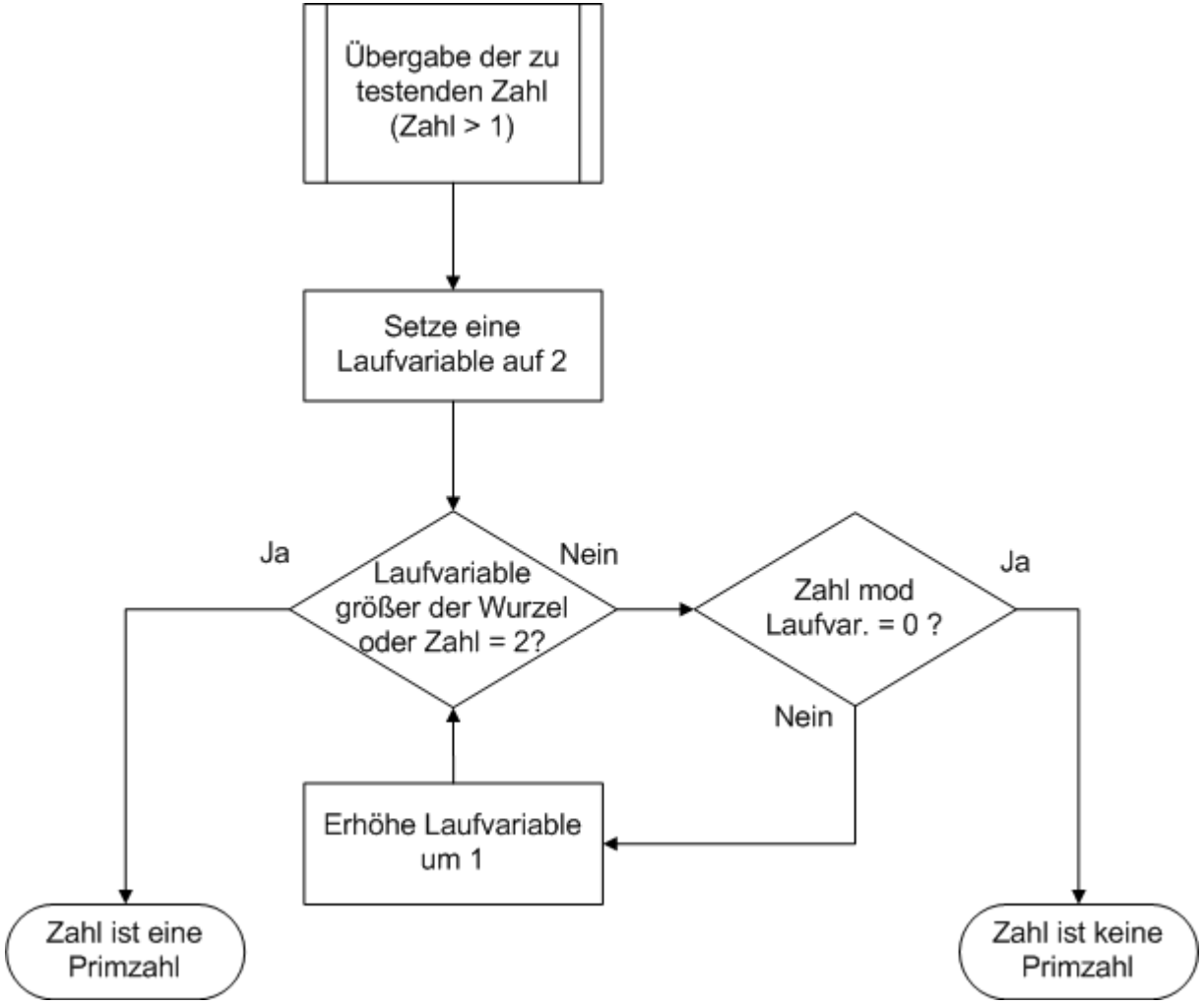
Die ersten Primzahlen sind also 2, 3, 5, 7, 11, 13, 17, 19.

Primzahlen.
Verteilung.

- Es gibt unendlich viele Primzahlen (Satz von Euklid)
- Anzahl der Primzahlen an der Gesamtheit von Zahlen nimmt ab:



Primzahltest.
Implementierung.



Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

ACM Probleme.

Factoring Large Numbers (10392).

Aufgabenstellung:

- Faktorisierung von Zahlen
- Datentyp Long ausreichend
- Abbruch bei einem negativen Input
- Ausgabeformatierung:

  [blank][blank][blank][blank]*Factors(ascending)with[\n]*
  [\n]

ACM Probleme.

Factoring Large Numbers (10392).

Beispiel:

Input:

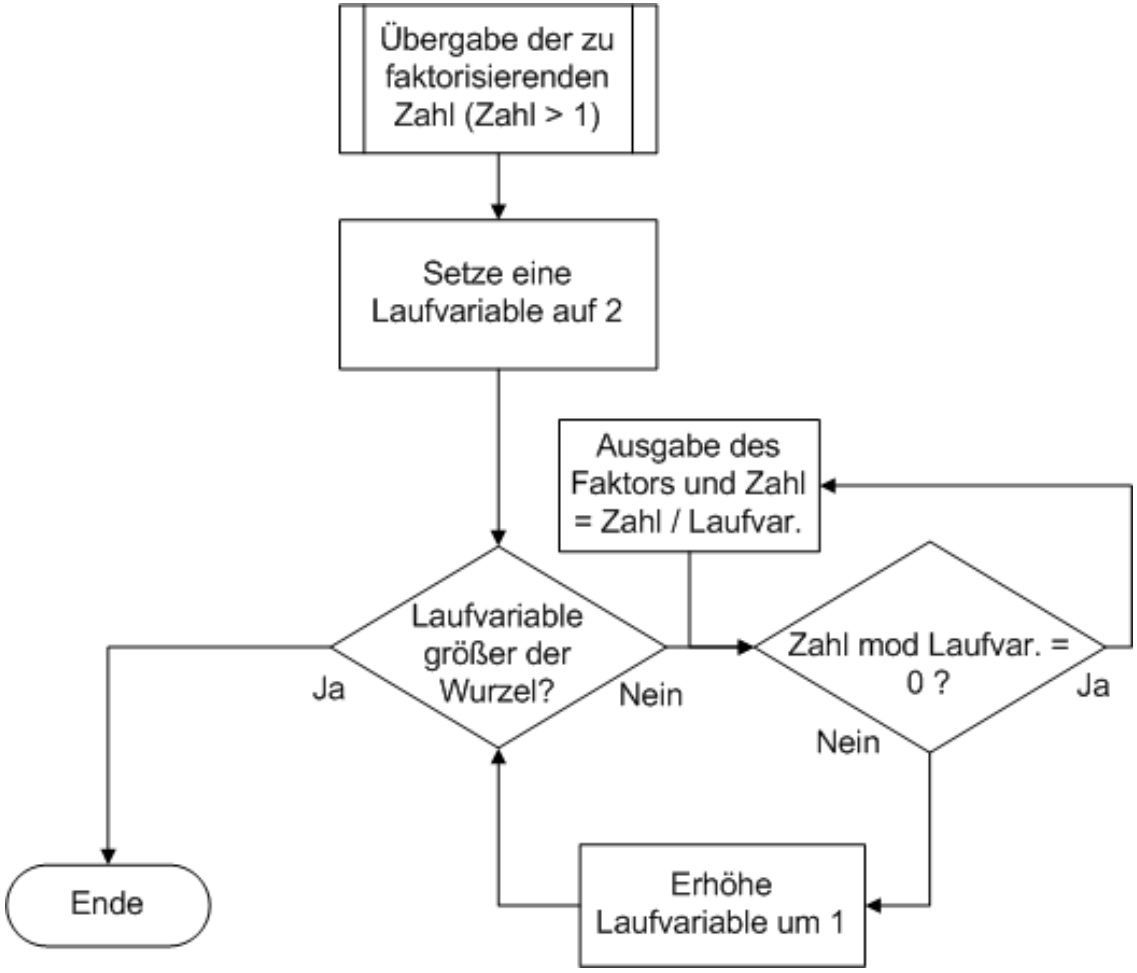
90 120 204 45

Output:

2	2	2	3
3	2	2	3
3	2	61	5
5	3		
	5		

ACM Probleme.

Factoring Large Numbers (10392).



Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

ACM Probleme.
Count the Factors (10699).

Aufgabenstellung:

- Ausgabe der Anzahl von Primfaktoren einer Zahl
- Zahlen kleiner oder gleich 1.000.000
- Ende des Inputs bei einer 0
- Ausgabeformatierung

    Input[blank]:[blank]*Anzahl*

ACM Probleme.

Count the Factors (10699).

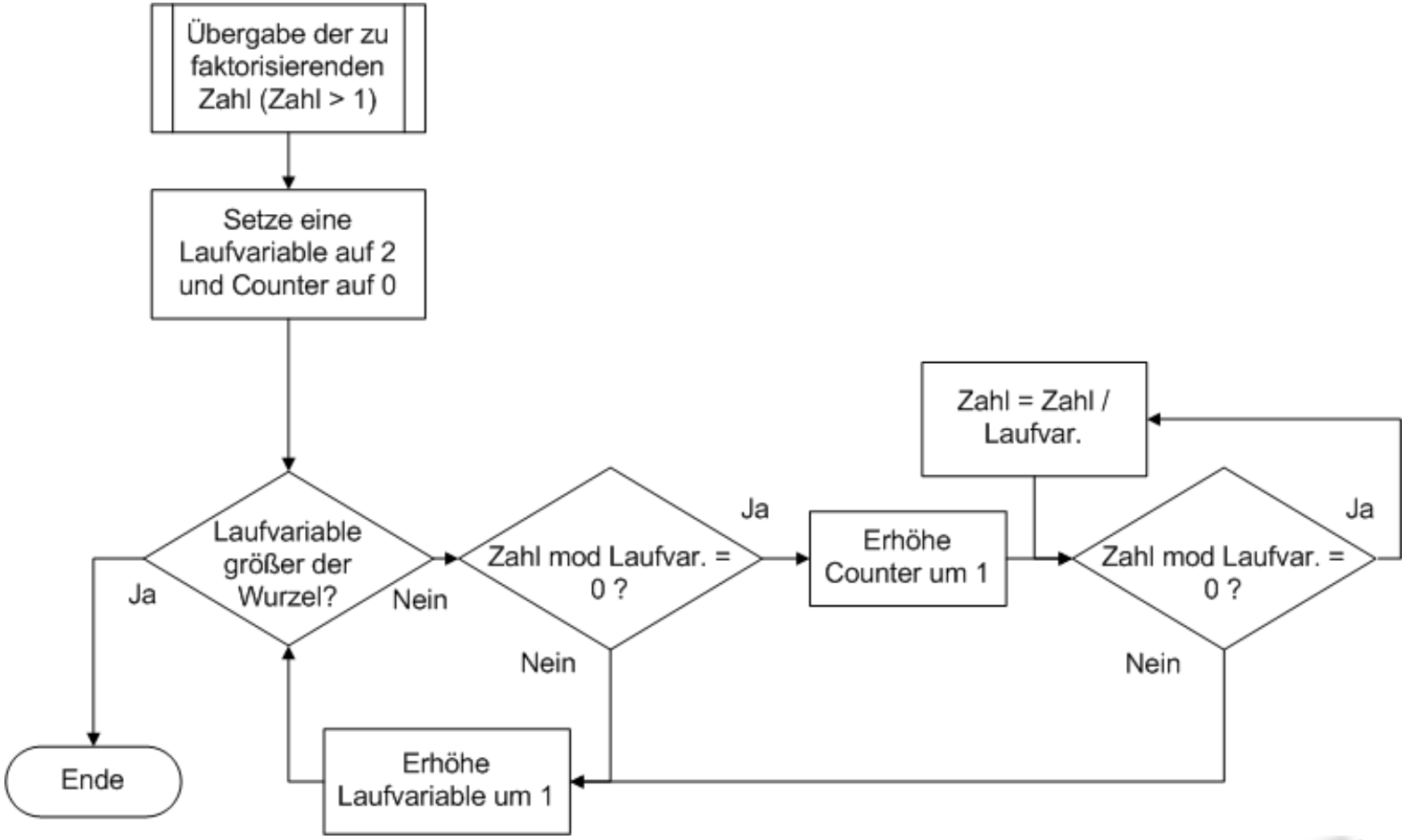
Beispiel:

Input:
90
120
204
45
0

Output:
90 : 3
120 : 3
204 : 2
45 : 2

ACM Probleme.

Count the Factors (10699).



Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

ACM Probleme.
Largest Prime Divisor (11466).

Aufgabenstellung:

- Größter Primfaktor der Zahl (=LPD)
- Zahleninput maximal mit 14 Stellen
- Zahlen mit nur einem Primfaktor haben keinen größten Primfaktor
- Inputende bei 0
- Ausgabeformatierung:

LPD[\n]

ACM Probleme.

Largest Prime Divisor (11466).

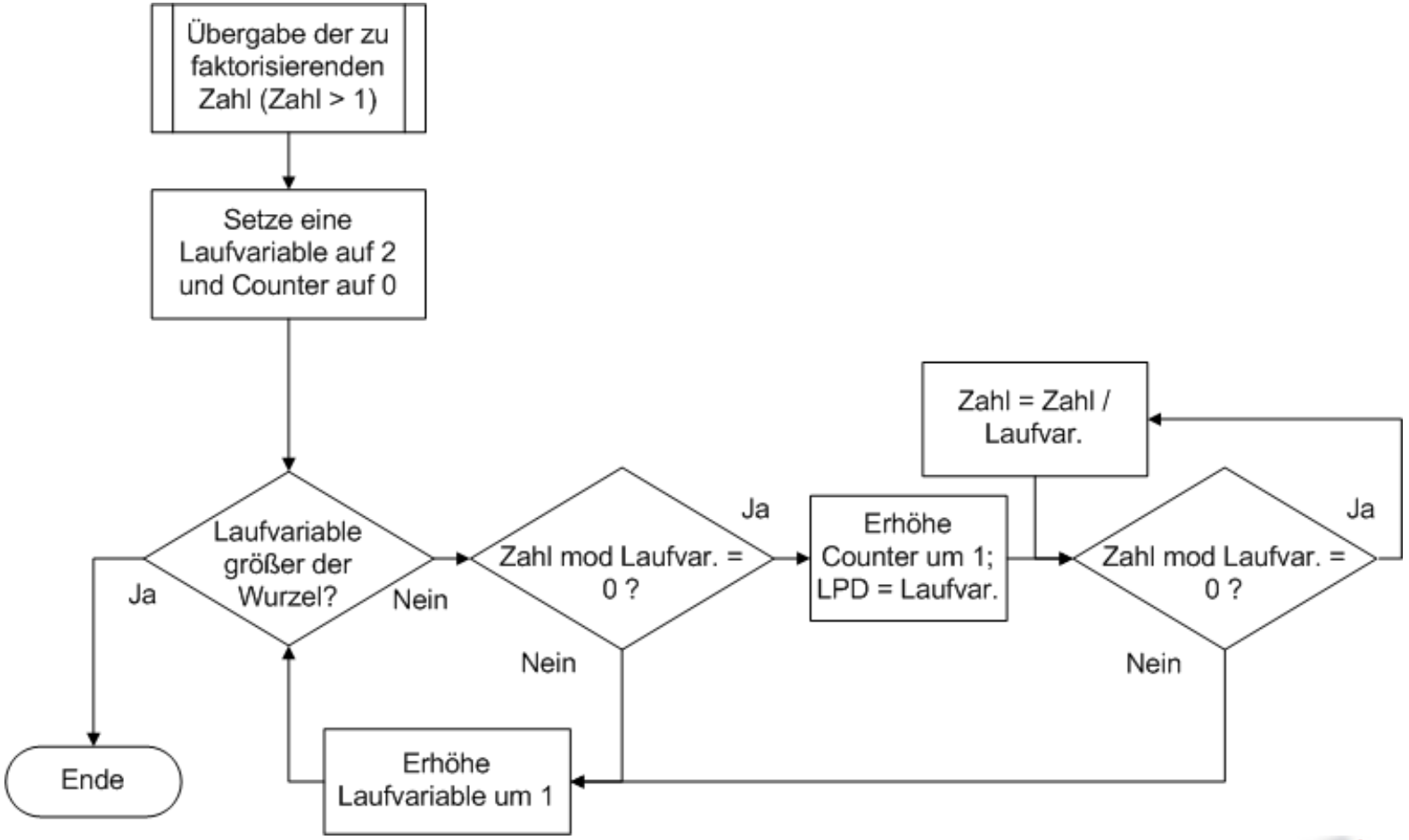
Beispiele:

Input:
90
120
204
45
4
0

Output:
5
5
61
5
-1

ACM Probleme.

Largest Prime Divisor (11466).



Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlproblematiken.

Perfect Pth Powers (10622).

Aufgabenstellung:

- Definition Perfect Powers: $x = b^p, p \to \max$
- Input: $x \geq 2, [32bit]$
- Inputende bei 0
- Ausgabeformatierung:

PPP[\n]

Primzahlproblematiken.

Perfect Pth Powers (10622).

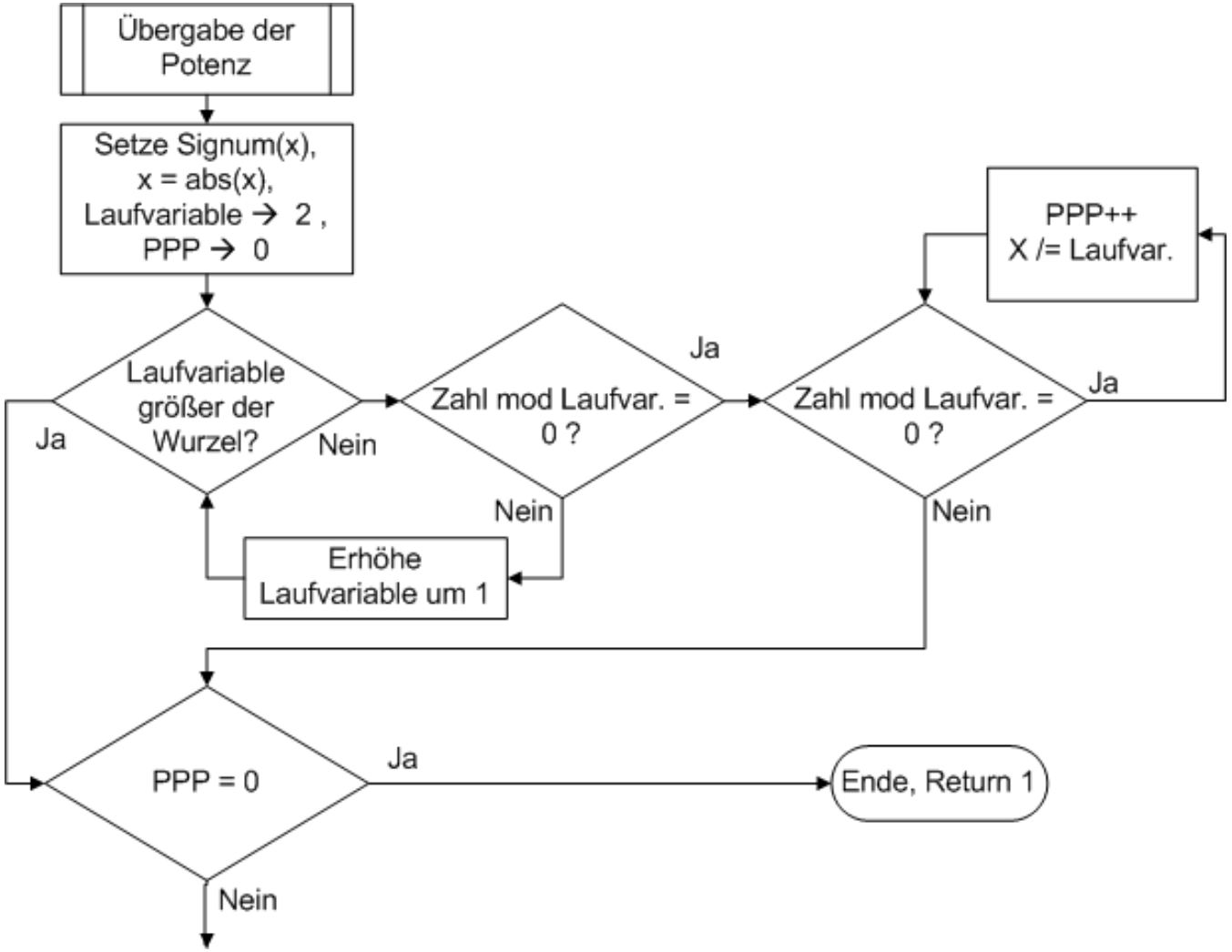
Beispiele:

Input:
17
1073741824
25
0

Output:
1
30
2

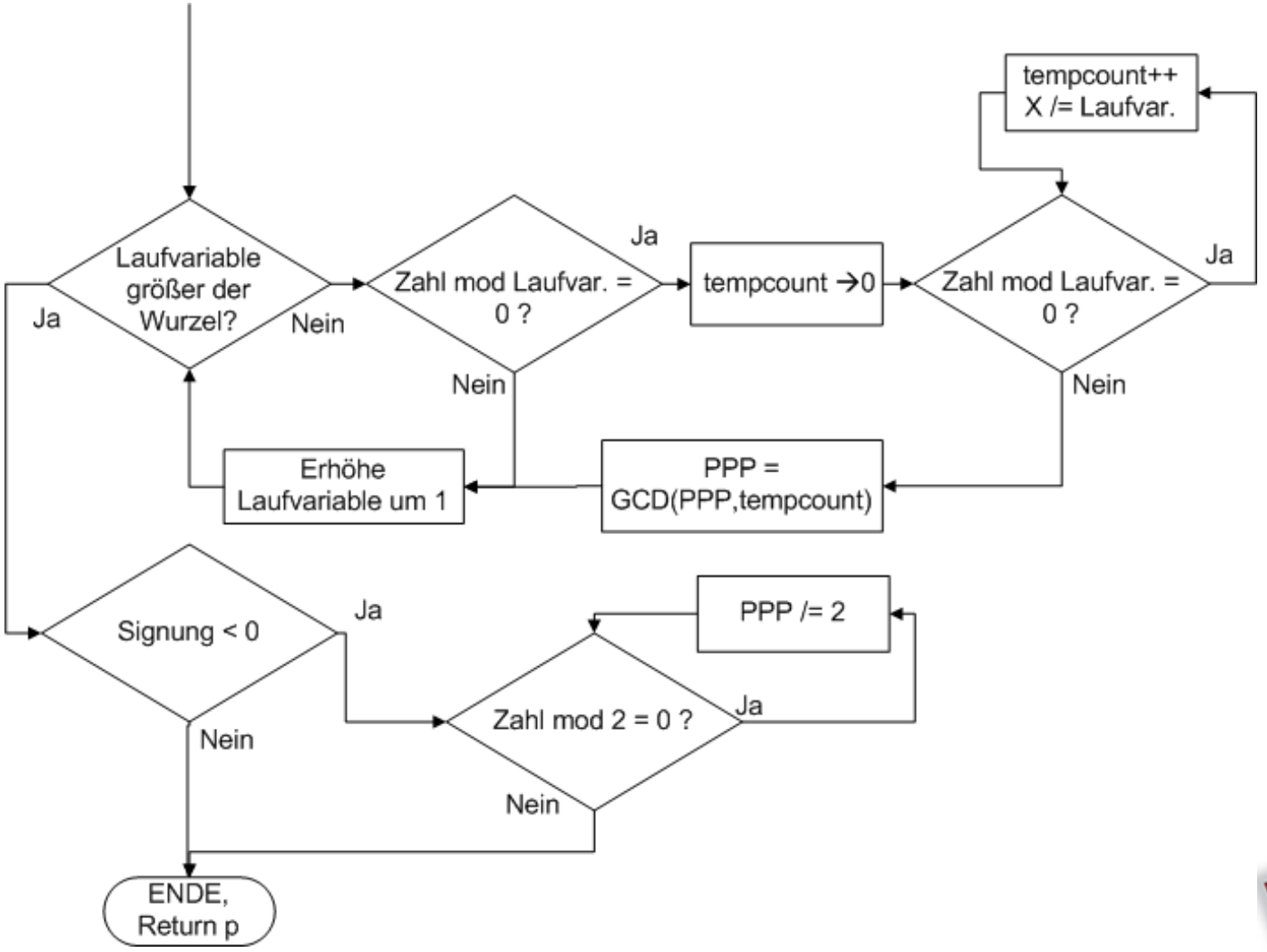
Primzahlproblematiken.

Perfect Pth Powers (10622).



Primzahlproblematiken.

Perfect Pth Powers (10622).



Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlproblematiken.
Factovisors(10139).

Aufgabenstellung:

- $b \mid a!$
- Input: $a, b \mid a \wedge b < 2^{31}$
- Inputende durch EOF
- Ausgabeformatierung:

b[blank]„devides“ / „does not devide“[blank]a“!“[\n]

Primzahlproblematiken.
Factovisors(10139).

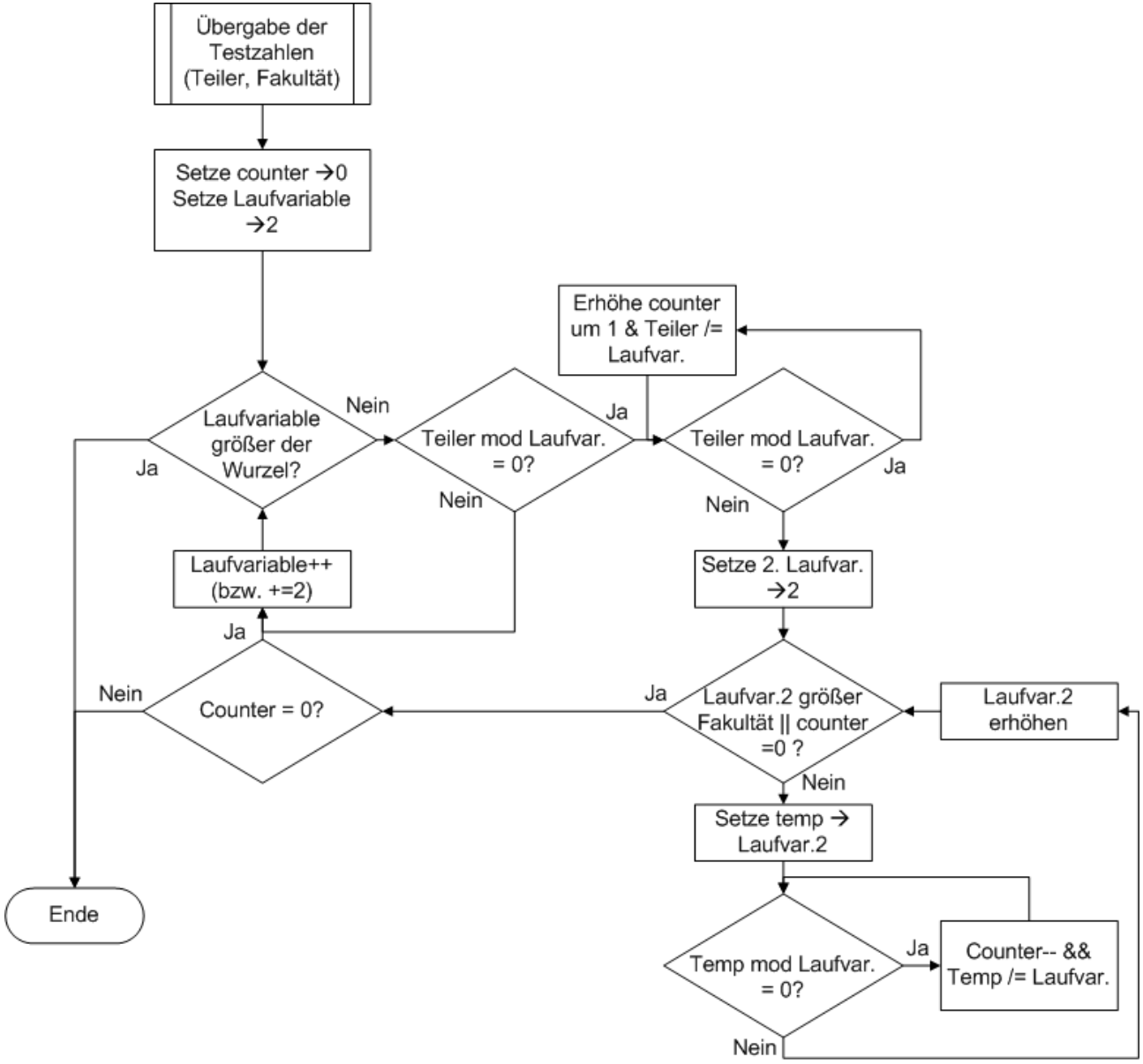
Beispiele:

Input:
6 9
6 27
20 10000
20 100000
1000 1009

Output:
9 divides 6!
27 does not divide 6!
10000 divides 20!
100000 does not divide 20!
1009 does not divide 1000!

Primzahlproblematiken.

Factovisors(10139).



Primzahlproblematiken.
Agenda.

1. Angewandte Mathe
   a) Selbst gelöste Probleme
   b) Schönste Probleme
   c) Schönste Programme

2. Primzahlen: Begriff

3. ACM Probleme
   a) Factoring Large Numbers (10392)
   b) Count the Factors (10699)
   c) Largest Prime Divisor (11466)
   d) Perfect Pth Powers (10622)
   e) Factovisors (10139)
   f) Next Same Factored (11099)

Primzahlproblematiken.

Next Same Factored(11099).

Aufgabenstellung:

- nächst größere Zahl mit den gleichen Primfaktoren finden
- Input: $a, a < 1.000.000$
- Inputende durch EOF
- Ausgabe nur, wenn n < 2.000.000, sonst „Not Exists!“
- Ausgabeformatierung:

  n[\n]

Primzahlproblematiken.

Next Same Factored(11099).

Beispiele:

Input:
4
100
18
7

Output:
8
160
24
49

Primzahlproblematiken.

Next Same Factored(11099).

Lösung: ???

Lösungsansatz:

$$a = p_1^{n_1} * ... * p_k^{n_k}$$

$$a < n$$

$$n = p_1 * ... * p_k * p_1^{x_1} * ... * p_k^{x_k}, x \geq 0$$

$$\Downarrow$$

$$\tilde{a} < p_1^{x_1} * ... * p_k^{x_k}, x \geq 0$$

Beispiel:

$$18 = 2 * 3 * 3$$

$$18 < n$$

$$n = 2 * 3 * 2^x * 3^y$$

$$\Downarrow$$

$$3 < 2^x * 3^y \Rightarrow x = 2 \wedge y = 0$$

$$n = 2 * 3 * 2^2 * 3^0 = 24$$

Vielen Dank für Ihre Aufmerksamkeit.

Backup.

Primzahlen.

Satz von Euklid.

Euklidischer Beweis zum Satz: Es gibt keine größte Primzahl.

Angenommen, es gäbe eine größte Primzahl. Sie sei p. Dann bilde man das Produkt aller Primzahlen und addiere dazu 1. Die so entstandene Zahl ist durch keine der bis dahin bekannten Primzahlen teilbar, denn sie lässt bei Division stets den Rest 1. Also ist sie entweder selbst eine Primzahl (größer als p) oder sie hat Primteiler, die notwendig größer als p sind. Das ist ein Widerspruch zu der Annahme, und diese muss demnach falsch sein.

Primzahlen.
Verteilung.

Primzahlen		
Zahl	Anzahl Primen	%-Anteil
10	4	40
100	25	25
1'000	169	16.9
10'000	1'229	12.29
100'000	9'592	9.59
1'000'000	78'498	7.85
10'000'000	664'579	6.65
100'000'000	5'761'455	5.76
1'000'000'000	50'847'534	5.08
2'147'483'646	105'097'564	4.89

Schönste Programme.
Big Chocolate.

```
/**
* Angewandte Mathematik, SS09, IFB 2C
* ACM Problem #10970 Big Chocolate
*
* @author David Leib
* @author Julius Tschannerl
* @version 1.0, 05/08/2009
*
* Status : Accepted
* Runtime: 1.048
*/

import java.util.*;
import java.io.*;

public class Main {

	public static void main(String[] args) throws IOException{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		String line = br.readLine();
		while(line != null){
			StringTokenizer st = new StringTokenizer(line);
			int i = Integer.parseInt(st.nextToken());
			int j = Integer.parseInt(st.nextToken());
			System.out.println(i*j-1);
			line = br.readLine();
		}
	}
}
```

Schönste Programme.
Magic Numbers.

```
/* Angewandte Mathematik, SS09, IFB 2C
* ACM Problem #471 (Magic Numbers)
* @author Dennis Wilfert
* @author Johann Studt
* @version 1.0, 03/29/2009
* Status : Accepted
* Runtime: 0.450
*/
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main {

		//Feststellen ob eine Zahl mehrfach vorkommt
		public static boolean Valid(long s){

			int[] s1valid = new int[10];

			for(int j = 0; j < 10; ++j) {
				int value = (int)(s % 10L);
				s1valid[value] += 1;
				if(s1valid[value] > 1)return false;

				s /= 10;
				if(s<1)return true;
			}
		return true;

		}
```

Schönste Programme.
Magic Numbers.

```
public static void main(String[] args) throws IOException {

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        // Erste Zeile lesen um zu sehen wieviele Werte folgen
        int counter = Integer.parseInt(reader.readLine().trim());

        while(counter > 0){
                        // Leere Zeile Lesen
                reader.readLine().trim().isEmpty();
                        // Wert der Zeile als long-wert speichern
                long number = Long.parseLong(reader.readLine().trim());
                        // Wert number mit i multiplizieren
                for(long i = 1; i<=9876543210L; i++){

                        // Prüfen ob in i ein Wert mehrmals vorkommt
                        if(Valid(i)){
                                long s1 = number*i;
                                if(s1 > 9876543210L){
                                        if (counter>1)System.out.println();
                                        break;
                                }
                        // Kommt im Ergebnis kein Wert mehrfach vor, dann Ausgeben
                        if(Valid(s1)) System.out.println(s1 + " / " + i + " = " + number);
                        }
                }
                counter--;
        }
    }
}
```

Schönste Programme.

Square Root.

```
/**
* Angewandte Mathematik, SS09, IFB 2C
* ACM Problem #10023 (SquareRoot)
* @author Christian Posselt
* @author Jonathan Schubert
* @version 1.0, 03/25/2009
* Status : Accepted
* Runtime: 0.430
*/
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
class Main
{
		public static void main(String[] args) throws IOException
		{
				BufferedReader Reader = new BufferedReader(new InputStreamReader(System.in));
				StringBuilder bu = new StringBuilder();
				int amount = Integer.parseInt(Reader.readLine());
				for(;amount > 0; amount --)
				{
						Reader.readLine();
						BigInteger digit = new BigInteger(Reader.readLine());
						bu.append(sqrt(digit));
						bu.append("\n\n");
				}
				bu.deleteCharAt(bu.length()-1);
				System.out.print(bu.toString());
		}
```

Schönste Programme.

Square Root.

```
public static BigInteger sqrt(BigInteger N)
{
        //Special case zero
        if(N.equals(BigInteger.ZERO))
        return BigInteger.ZERO;

        BigInteger closer = BigInteger.ZERO;

        //first closer value
        closer = N.shiftRight(N.bitLength()/2);
        BigInteger last = BigInteger.ZERO;

        //(n+1) = (a/n + n)/2 Algorithms Herons
        while(!closer.equals(last))
        {
                last = BigInteger.ZERO;
                last = last.add(closer);
                BigInteger part = BigInteger.ZERO;
                part = part.add(N);
                part = part.divide(closer);
                part = part.add(closer);
                part = part.shiftRight(1);
                closer = BigInteger.ZERO;
                closer = closer.add(part);
        }
        return closer;
}
}
```