

Vom Sieben und Nullenschieben

Tutorium Duatalia

Florian Sihler ◦ KW 45



Präsenzaufgabe

1

Ju can not desieve me!

Entwickeln Sie ein Programm, welches als Eingabe eine Zahl N erhält und mit Hilfe des Sieb des Eratosthenes alle Primzahlen zwischen 1 und N bestimmt und ausgibt. Die Idee hier ist, alle Vielfachen der Zahlen, beginnend bei 2 zu markieren, also „herauszusieben“. Danach wird mit der nächsten unmarkierten Zahl fortgefahren. Hier ein grafisches Beispiel mit $N = 16$:



Präsenzaufgabe - Naive Lösung

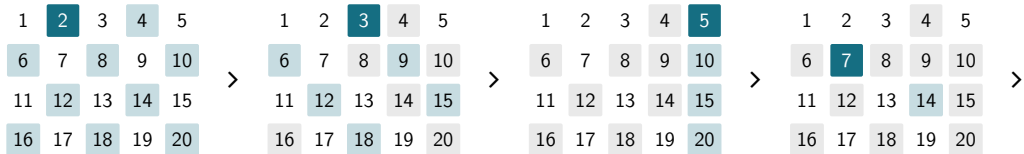
Input : Obergrenze $N \in \mathbb{N}$ (da Primzahlen)

```
1 marker = (marker1, ..., markerN);  
2 for i = 1 to N do markeri = false;  
   // Siebe von 2 bis N  
3 for i = 2 to N step 1 do  
4   | if markeri then continue;  
5   | Gebe aus: „Prim: i“;  
   | // Markiere alle Vielfache  
6   | for j = 2 · i to N step i do markerj = true;  
7 end
```

Algorithmus 1 : Naives Sieben

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

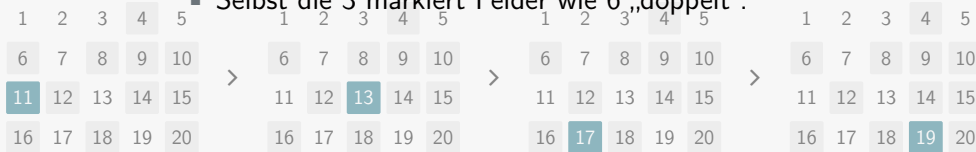
Präsenzaufgabe - Was bemerken wir?



Präsenzaufgabe - Was bemerken wir?



- Selbst die 3 markiert Felder wie 6 „doppelt“.



Präsenzaufgabe - Verbesserung

- Allgemeiner: Wir können beim Markieren bei i^2 anstelle von $2 \cdot i$ beginnen. Alle Vielfachen kleiner i^2 wurden bereits markiert!
- Damit können wir das Markieren bereits bei \sqrt{N} stoppen.
- *Hinweis:* Im folgenden Pseudocode gehen wir davon aus, dass **for** nur ganzzahlige Schritte macht. Für „**for** $i = \sqrt{N} + 1$ “ gehen wir also beispielsweise davon aus, dass der Algorithmus immer abrundet.

Präsenzaufgabe - Effizientere Lösung

Input : Obergrenze $N \in \mathbb{N}$

```
1 marker = (marker1, ..., markerN);  
2 for i = 1 to N do markeri = false;  
3 for i = 2 to  $\sqrt{N}$  step 1 do  
4   |   if markeri then continue;  
5   |   Gebe aus: „Prim: i“;  
6   |   for j = i · i to N step i do markerj = true;  
7 end  
8 for i =  $\sqrt{N} + 1$  to N step 1 do  
9   |   if not markeri then Gebe aus: „Prim: i“;  
10 end
```

Algorithmus 2 : Optimiertes Sieben

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

Übungsblatt 2 - Aufgabe 1

- Wenn es heißt „dreistellige Dezimalzahl“, ist das *mit* führenden Nullen oder *ohne*?
- Wir nehmen zuerst an, dass führende Nullen *erlaubt* sind.
- Da Prozeduren als solche noch aus fernen Landen stammen, schreiben wir alles als ein Programm.


```
1 Wähle zufällig  $z_1, z_2, z_3 \in \{0, \dots, 9\}$ ;  
2 for round = 1 to 3 step 1 do  
3     guesses_left = 3;  
4     while true do  
5         Erbitte um Eingabe für Stelle  $\text{round}^1$ , bei  $\text{guesses\_left}^1$  Versuchen;  
6         e = Eingabe von Nutzer mit  $e \in \{0, \dots, 9\}$ ;  
7         if  $e = z_{\text{round}}$  then verlasse while-Schleife;  
8         else  
9             guesses_left = guesses_left - 1;  
10            if guesses_left  $\leq 0$  then  
11                Vermerke scheitern;  
12                return;  
13 Verkünde Gewinn;
```

Algorithmus 3 : Rate mit führenden Nullen

- Wir wählen nun zufällig $z \in \{100, \dots, 999\}$.
- Doch wie erhalten wir z_1 , z_2 und z_3 ?
- Wir können beispielsweise Modulon™:

$$z_1 = \lfloor z/100 \rfloor \bmod 10 \quad z_2 = \lfloor z/10 \rfloor \bmod 10 \quad z_3 = z \bmod 10$$

- Der Rest bleibt unverändert.

Übungsblatt 2 - Aufgabe 1, ein paar Bitten

- Geht mittels verschiedener Simulationen durch den Pseudocode
- Ihr *wollt* beim Überprüfen des Pseudocodes, dass etwas nicht stimmt. (Es ist besser einen Fehler früh zu finden als sich irgendwann im Betrieb zu Fragen warum die (Mars-)Rakete auf die Sonne zu steuert.)
- Pseudocode ist nicht nur was, was in den Augen des Tutors bestehen muss!
- Nutzt Schleifen und vermeidet Redundanzen wo möglich. Bei einer Änderung sollte und darf „Alles Ersetzen“ *niemals* das Mittel der Wahl sein.
- Hardcoding löst das Problem nicht. So wird für die Aufgabenstellung: „Geben Sie von den ganzen Zahlen aus Eins bis Zehn die aus, die gerade sind“ *nicht* durch `System.out.println("2, 4, 6, 8, 10")` gelöst! (Der Compiler kann *loop-unrolling* betreiben, der Code würde dann entsprechend optimiert werden. Dies löst die Aufgabe allerdings nicht richtig!)

Übungsblatt 2 - Aufgabe 2



Die Ente ist knuffig, die bleibt 😊

Übungsblatt 2 - Aufgabe 2

- *Name einer Person*

`String` der Name einer Person bedient sich der Welt der Buchstaben.

- *Akkustand eines Smartphone*

Kleinstmöglich ist `byte`, das genügt für die übliche Prozentdarstellung ohne Nachkommastelle. `float` basierte Darstellungen mögen auch noch in Ordnung sein. `double` ist aber definitiv overkill (und dennoch nicht falsch).

- *Preis eines Produktes*

Wenn wir damit nicht rechnen müssen: `float` oder `double`.

- *Ergebnis unbenoteter Prüfung*

Da wir so nur bestanden und nicht bestanden haben, genügt ein `boolean`.

Übungsblatt 2 - Aufgabe 2

- *Raumtemperatur*

In der Hoffnung, dass wir von Temperaturen sprechen in denen Menschen noch überleben können, sollte ein `float` absolut ausreichen.

- *Satzzeichen*

Alle gängigen (und sogar ungängigen 😊) Satzzeichen sind als ASCII bzw. Unicode kodiert: ein `char` reicht in Java aus.

Übungsblatt 2 - Aufgabe 2, zusätzliche Freuden

- *Identifikationsnummer eines Nutzers*

Möglich: `short` (wenn weniger Nutzer), aber auch `int` oder `long`. `String` ist nicht zu empfehlen, da es das Generieren eindeutiger IDs erschwert sowie im Vergleich aufwändiger ist.

(Exkurs: Warum? Wenn wir ein String-Objekt erstellen, zum Beispiel durch: `String duck = "Quack"`, speichert Java in `duck` nicht die Zeichenkette selbst, sondern nur, wo sie sich im (Arbeits-)Speicher befindet, also die Speicheradresse. So kann Java alle Variablen an einem Ort Verwalten (dem sogenannten Stack), auch wenn sie eine Variable Größe haben, da dort dann nur ein Verweis auf das eigentliche Datenfeld (im sogenannten Heap) abgelegt wird. Wenn wir zwei Zeichenketten in Java mittels `==` vergleichen, wird deswegen auch *nie* die eigentliche Zeichenkette sondern nur die Adresse an der sie liegt verglichen. Zwei Strings müssen explizit über `duck.equals(b)` (wobei hier `b` ein anderer String ist) verglichen werden.)

Übungsblatt 2 - Aufgabe 2, zusätzliche Freuden

- *Kontostand*

Hier kann man leicht sagen `int`, nur in z.B. Euro-Cent. Aber für die Börse oder Berechnungen empfehlen sich mehr Nachkommastellen. `double` oder `float` sind aber ganz ganz dumme Ideen!

Besser wäre zum Beispiel ein Tupel aus (`int`, `int`).

Übungsblatt 2 - Rundungsproblematik

■ Java-Code:

```
double dmoney = 42; float  
    fmoney = 42f;  
  
for(long n = 0; n < 1000000000  
    ; n++){  
    dmoney += 0.1;  
    fmoney += 0.1f;  
}
```

```
System.out.format("Double: %.9  
    f%n", dmoney);  
System.out.format("Float: %.9  
    f%n", fmoney);
```

Ausgabe für `java Test`:

```
Double: 100000040.745415320  
Float: 2097152.000000000
```

Übungsblatt 2 - Aufgabe 2, zusätzliche Freuden

- Weiterer *wichtiger* Hinweis: Vor allem in Java ist es meistens nicht nötig eine Variable explizit als `byte` zu deklarieren um Speicherplatz zu sparen.

(Die meisten Rechner sind 32 oder 64 Bit Rechner, weswegen Java die Variablen in der Regel in einem „großen“ Feld ablegt. Mehrere `byte` in ein 64-Bit „Wort“ zu packen würde nämlich die Geschwindigkeit beeinflussen (einmal Aufwand für den Zugriff auf das Wort sowie um im Wort den Start zu finden). Das heißt nicht, dass immer `long` verwendet werden muss/soll/darf. Allerdings ist es (da es nicht wie in C/C++: `register` oder `int_fast16_t`) eher eine künstliche/semantische Schranke.)

- Warum sollte man das trotzdem unbedingt machen? Da gibt es *eeetliche* Gründe:
 1. So kann der Compiler/Interpreter entscheiden was besser ist.
 2. Es liefert Programmieren mehr Informationen über die Verwendung einer Variable.
 3. Den passenden Datentyp zu finden, fördert das Verständnis für die Domäne.
 4. Es kann vor Fehlern schützen (z.B.: wenn man dem Blauwert einer RGB-Farbe 35 000 zuordnen will).
 5. Und viele viele Mehr!

Übungsblatt 2 - Aufgabe 3 a)

I $(22 \geq 21) \ \&\& \ (42 > 21) \implies$
 $true \ \&\& \ true \implies true.$

II $c \wedge d \implies true \wedge false \implies true.$

III $((42 - 21) == 21) \ \&\& \ !true \implies$
 $true \ \&\& \ false \implies false.$

IV $(42 == 2 * 21) \ \&\& \ (!true \ || \ !false)$
 $\implies true \ \&\& \ true \implies true$

```
int a = 42; int b = 21;
boolean c = true;
boolean d = false;

if((22 >= b) && (a > b)) { // I
    if((c || d) && !(c && d)) { // II
        if((a - b) == 21) && !c) { // III
            if((a == 2*b) && (!c || !d)) { // IV
                System.out.println("A");
            }
        } else {
            System.out.println("B");
        }
    } else {
        System.out.println("C");
    }
} else {
    System.out.println("D");
}
```

Übungsblatt 2 - Aufgabe 3 b)

- Werfen wir einen Blick auf die Ausgabe `java WeirdBoolean`:

```
B
```

- Warum? Nun, mit *true*, *true*, *false*, *true* verlassen wir bereits bei der dritten Unterscheidung das Konstrukt mit B.

Ein Bonus am Rande: Von Präzedenzregeln

- Wir erinnern uns an die Präzedenzregeln von Java (stark nach schwach, eine Auswahl): (<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>)

$$\begin{aligned} [a++, a-] &\rightarrow [!a, -a, ++a, --a] \rightarrow [*, /, \%] \rightarrow [a + b, a - b] \\ &\rightarrow [==, >=, <, \dots] \rightarrow [\wedge] \rightarrow [\&\&] \rightarrow [||] \end{aligned}$$

- Lets have an example:

```
(A && !A) || !(5 != 6 ^ (1 > 42) == 23 < 23)
false    || !( true  ^  false  == false )
false    || !( true  ^           true  )
false    ||      !(false)
false    ||      true
true
```

Was Flo so macht



Was Flo so machen muss

Input : Obergrenze $N \in \mathbb{N}$

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

Annahmen:

- Wir bleiben heute bei Text. Kein TikZ.
- Genutzt werden die Pakete `pgfmath`, `pgffor` und `etoolbox`. (Die sind zwar alle nicht notwendig, aber erlauben einfachere Notationen.)
- Wir nutzen `xcolor` für Farben.

Übersicht

```
1 \documentclass{article}
2 \usepackage{pgffor,pgfmath}
3 \usepackage{etoolbox}
4 \usepackage{xcolor}
5
6 % Präambel
7
8 \begin{document}
9 SCHDUBBS!
10 \end{document}
```

SCHDUBBS!

Übersicht

```
\documentclass{article}  
\usepackage{pgffor,pgfmath}  
\usepackage{etoolbox}  
\usepackage{xcolor}
```

% Präambel

```
\begin{document}  
SCHDUBBS!  
\end{document}
```

- Abseits aller Erklärungen, gestattet mir bitte ein Magic-Definition:

```
\def\calc#1=#2;{\pgfmathsetmacro#1{int(#2)}\xdef#1{#1}}
```

- Was macht das? Nicht wichtig. Was wir machen können:

```
1 Wir machen Mathe.
2 \calc \mynumber = sqrt(3 + 2 * 3);
3 Jetzt hält mynumber: \mynumber.
```

Wir machen Mathe. Jetzt
hält mynumber: 3.

- Wir können also einfach (mit Ganzzahlen) rechnen und Gruppen ignorieren.

Kommandos definieren

- \LaTeX liefert uns `\newcommand`:

`\newcommand{<cmd>}[<arg-count>]{<body>}`

- Die einzelnen Argumente werden von #1 an durchnummeriert.
- `<arg-count>` wird einfach weggelassen, wenn der Befehl keine Argumente hat!
- \LaTeX ersetzt dann einen Aufruf der Form `\<cmd>\{<#1>\}\{<#2>\}` mit dem `<body>`.
- Es geht um einiges eleganter, das reicht uns aber.

- Betrachten wir einmal:

```
1 \newcommand{\hallo}[2]  
2   {#2: Hallo \textbf{#1}}  
3  
4 Wir schreiben: \hallo{Sonne}{Text Vor}
```

Wir schreiben: Text Vor:
Hallo **Sonne**

- Kommandos können andere Kommandos definieren:

```
1 \newcommand{\M}[2]{\csgdef{M#1}{#2}}  
2 Speichern: \M{eins}{1}.  
3 Benutzen: \Meins.
```

Speichern: . Benutzen: 1.

- Analog zu `\csgdef{<cs>}{<body>}` gibt es `\csuse{<cs>}`.
- Dies erlaubt es Befehle durch Argumente zu basteln:

```
1 \newcommand{ \StoreA}{Super A}  
2 \newcommand{ \Load}[1]{\csuse{Store#1}}  
3 Laden: \Load{A}.
```

Laden: Super A.

Schleifen

- Ich zeige hier nur eine Form der Schleife. Die, die wir brauchen:

```
\foreach \i in {<start>,<step-off>,...,<end>} {  
    <body>  
}
```

- Als Beispiel:

```
1 Hallo: \foreach \i in {1,3,...,12} {  
2     Sup-\i-du  
3 }.
```

Hallo: Sup-1-du Sup-3-
du Sup-5-du Sup-7-du
Sup-9-du Sup-11-du .

Die Idee

Input : Obergrenze N

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

Ideen:

- Schleifen mit `\foreach`.
- Marker sind Makros der Form `\marker@<number>`.
- Berechnungen mit `\calc`.
- Fallunterscheidungen zeige ich noch.

Hilfsmakros – Reset

Input : Obergrenze N

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

- Wir definieren N global: `\newcommand{\N}{25}`
- `\SieveReset` soll die Marker mit `\csgundef` wieder löschen:

```
\newcommand{\SieveReset}{
  \foreach \i in {1,...,\N}{
    \csgundef{marker@\i}
  }
}
```


Hilfsmakros – Step

Input : Obergrenze N

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

- Mit `\SieveStep{<step>}` möchten wir den Schritt <step> durchführen:

```
\newcommand{\SieveStep}[1]{
  \calc \start = #1 * #1;
  \calc \nx = \start + #1;
  \foreach \i in {\start, \nx, ..., \N}{
    % store the step that locked it
    \csgdef{marker@\i}{#1}
  }
}
```

Hilfsmakros – Print

Input : Obergrenze N

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

- Mit `\SievePrint` möchten wir alles ausgeben.
Für dieses Beispiel limitieren wir die Breite auf 5:

```
\newcommand{\SievePrint}{
  \foreach \cell in {1,...,\N}{
    \SievePrintCell{\cell}
    \calc\check = mod(\cell, 5);
    \ifnum\check=0 \newline \fi
  }
}
```

Hilfsmakros – Print

Input : Obergrenze N

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

- Mit `\SievePrintCell{<cell>}` möchten wir eine Zelle ausgeben:

```
\newcommand{\SievePrintCell}[1]{
  \makebox[2em]{
    \ifcsundef{marker@#1}{% undefined
      #1
    }
    {\textcolor{gray}{#1}}
  }
}
```

Hauptschleife

Input : Obergrenze N

```
1 marker = (marker1, ..., markerN);
2 for i = 1 to N do markeri = false;
3 for i = 2 to  $\sqrt{N}$  step 1 do
4   |   if markeri then continue;
5   |   Gebe aus: „Prim: i“;
6   |   for j = i · i to N step i do markerj = true;
7 end
8 for i =  $\sqrt{N} + 1$  to N step 1 do
9   |   if not markeri then Gebe aus: „Prim: i“;
10 end
```

- Mit `\Sieve` nun das Sieben:

```
\newcommand{\Sieve}{
  \SieveReset
  \calc \max = sqrt(\N);
  \foreach \step in {2,...,\max}{
    \ifcsundef{marker@\step}{
      \SieveStep{\step}
    }{}
  } \SievePrint
}
```

Das Ergebnis

- Wir geben das Ganze zwar erst am Ende aus, das Endergebnis ist aber gleich
(`sieve-final.tex`) `\Sieve`:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

