```
ALP 1 -- WS 02/03 -- Tutor: Till Zoppke
Uebung 1

Do_12
Monika Budde
Emrah Somay

Aufg. 1
// Algorithmus zur Bestimmung der drei groessten Zahlen einer
// Zahlenfolge von 100 bzw. n unterschiedlichen Zahlen (n>0).
// Die gesuchten Zahlen werden in absteigender Reihenfolge extrahiert,
// die Laufzeit ist schlecht (3 Durchlaeufe durch die Zahlenfolge
// noetig), das Zaehlen der Vergleiche einfach.
// Ein 'guter' Algorithmus sucht die drei Zahlen in einem Durchlauf
// und speichert sie etwa in einem -- sortierten -- Array.
// Ausserdem sollte die Eingabe nicht manipuliert werden.

0. Eingabe: Zahlenfolge Zf aus 100 bzw. n unterschiedlichen Zahlen.
1. Suche die groesste Zahl in Zf und speichere sie in Zf_max:
   // wg. n>0 existiert eine solche Zahl immer
   1.a) Setze Zf_max = Zf(1) und die Laufvariable i = 1
        // Initialisiere Zf_max mit dem 1. Folgenglied
   1.b) Wiederhole,
         (i) solange i<100 (bzw. i<n):
        (ii) Falls Zf_max < Zf(i+1), setze Zf_max = Zf(i+1);
       (iii) inkrementiere i (also i=i+1).
        // Das Ende der Folge ist erreicht (i = 100 bzw. i=n).
        // In Zf_max steht die groesste Zahl aus Zf.
   1.c) Streiche Zf_max aus Zf: Zf = Zf ohne Zf_max.
        // Das Ergebnis ist eine Folge mit 99 (bzw. n-1) Gliedern.

2. Suche die zweitgroesste Zahl in Zf ohne Zf_max und speichere sie
   in Zf_2max (falls eine solche Zahl existiert).
   Im allgemeinen Fall:
   Pruefe, ob n>1. Falls n<=1, gib eine geeignete Meldung aus und
   gehe zu (4).
   2.a) Setze Zf_2max = Zf(1) und die Laufvariable i = 1
        // Initialisiere Zf_2max mit dem 1. Folgenglied
   2.b) Wiederhole,
         (i) solange i<99 (bzw. i<n-1):
        (ii) Falls Zf_2max < Zf(i+1), setze Zf_2max = Zf(i+1);
       (iii) inkrementiere i (also i=i+1).
        // Das Ende der Folge ist erreicht (i = 99 bzw. i=n-1).
        // In Zf_2max steht die zweitgroesste Zahl aus Zf.
   2.c) Streiche Zf_2max aus Zf: Zf = Zf ohne Zf_2max.
        // Das Ergebnis ist eine Folge mit 98 (bzw. n-2) Gliedern.

3. Suche die drittgroesste Zahl in Zf ohne Zf_max und Zf_2max,
   speichere diese Zahl in Zf_3max (falls eine solche Zahl existiert).
   Im allgemeinen Fall:
   Pruefe, ob n>2. Falls n<=2, gib eine geeignete Meldung aus und
   gehe zu (4).
   2.a) Setze Zf_3max = Zf(1) und die Laufvariable i = 1
        // Initialisiere Zf_3max mit dem 1. Folgenglied
   2.b) Wiederhole,
         (i) solange i<98 (bzw. i<n-2):
        (ii) Falls Zf_3max < Zf(i+1), setze Zf_3max = Zf(i+1);
       (iii) inkrementiere i (also i=i+1).
        // Das Ende der Folge ist erreicht (i = 98 bzw. i=n-2).
```

```
       // In Zf_3max steht die drittgroesste Zahl aus Zf.

4. Gib die gefundenen Zahlen Zf_max, Zf_2max und Zf_3max aus
   (soweit sie existieren).

------------------
```

Analyse: Anzahl der Vergleiche zwischen Zahlen in der Zahlenfolge

(1.b.ii): Insgesamt werden n-1 Vergleiche dieser Art durchgefuehrt.
Die 1. Zahl wird mit allen nachfolgenden kleineren Zahlen bis zu der ersten groesseren Zahl k_1 bzw. dem Folgen-Ende verglichen sowie ggfs mit k_1 selbst (eine gleichgrosse Zahl gibt es nach Voraussetzung in der Folge nicht). Anschliessend wird ggfs k_1 mit allen nachfolgenden Zahlen, die kleiner als k_1 sind verglichen, bis das Folgenende oder ein k_2 > k_1 erreicht wird. Dabei wird k_1 auch mit k_2 verglichen. Auf diese Weise wird eine Folge von Zahlen k_i mit 0<=i<100 bzw. 0<=i<n-1 gefunden, die die Zahlenfolge in eine Folge von Intervallen einteilt und die jeweils als Vergleichsbasis fuer die Zahlen in dem unmittelbar anschliessenden Intervall dienen.
Die Anzahl der Vergleiche zwischen zwei beliebigen Zahlen haengt daher davon ab, (i) an welcher Stelle der Folge sich die Zahlen befinden und (ii) welcher Art die vorangehenden Zahlen sind: Z.B. werden zwei Zahlen a und b, die beide kleiner als das augenblickliche Zf_max sind, nur mit Zf_max, aber nicht miteinander verglichen. Gilt a < Zf_max < b und kommt a vor b vor, dann werden a und b ebenfalls nicht miteinander verglichen. Gilt hingegen Zf_max < a, wenn a geprueft wird, und folgt b auf a, ohne dass eine Zahl, die groesser als a ist, zwischen a und b vorkommt, dann werden a und b miteinander verglichen.

(2.b.ii) Insgesamt werden entweder ueberhaupt keine Vergleiche zwischen den Folgengliedern durchgefuehrt (falls die Folge nur aus einem Glied besteht), oder es werden n-2 Vergleiche durchgefuehrt, bei n = 100 also 98. Fuer diese gilt Analoges wie bei (1.b.ii).

(3.b.ii) Insgesamt werden entweder ueberhaupt keine Vergleiche zwischen den Folgengliedern durchgefuehrt (falls die Folge weniger als 3 Glieder hatte), oder es werden n-3 Vergleiche durchgefuehrt, bei n=100 also 97. Im uebrigen gilt Analoges wie bei (1.b.ii).

Damit ergeben sich insgesamt 99 + 98 + 97 = 294 Vergleiche.

****

Aufg. 2
// Primzahl-Pruefung
// Um eine gegebene Zahl p daraufhin zu pruefen, ob sie prim ist, wuerde es genuegen zu pruefen, ob sie durch alle Primzahlen q mit 2<=q<=Quadratwurzel(p)=:sqrt(p) -- in aufsteigender Reihenfolge -- teilbar ist: Ist sie durch ein Vielfaches einer solchen Primzahl teilbar, dann auch durch die Primzahl selbst. Diese wurde aber bereits ueberprueft. Und hat sie einen Teiler t mit t>sqrt(p), dann muss es ein s<sqrt(p) geben mit p=s*t, und s wurde bereits ueberprueft. Da dies auf einen rekursiven Algorithmus mit erheblichem Platzbedarf fuehrt, bei dem sich die Anzahl der Elementaroperationen zudem nur aufwendig berechnen laesst, beschreiben wir einen ueberschaubareren iterativen Algorithmus, der zudem einige theoretisch ueberfluessige Pruefungen macht: es werden alle ungeraden Zahlen <=sqrt(p) untersucht, nicht nur die Primzahlen.

```
0. Eingabe: eine natuerliche Zahl p>1.
   // Werden weitere ganze Zahlen als Eingabe zugelassen, waere vorab
   // p>1 zu pruefen, was die Gesamtzahl der Elementaroperationen
```

```
   // fuer jede zulaessige Eingabe um 1 erhoeht.

1. Wenn p<4, gib "p ist Primzahl" aus.
2. Andernfalls:
   a. Setze q=2.
   b.  (i) Wenn p mod q == 0 ist, dann ist p eine gerade Zahl >2. Gib
           daher "p ist keine Primzahl" aus.
      (ii) Andernfalls ist p eine ungerade Zahl. Setze daher q=3 und
           wiederhole:
           aa. wenn q*q > p, gib "p ist Primzahl" aus.
                // Die Pruefung kann beendet werden, da groessere
                // Teiler nicht mehr untersucht werden muessen.
           bb. Andernfalls prüfe, ob q Teiler von p ist:
               bb1. falls p mod q != 0,
                    setze q=q+2 und gehe zu (aa) (Schleifenanfang);
                    // "!=" bedeutet "ungleich";
               bb2. andernfalls gib "p ist keine Primzahl" aus.
                    // q ist ein Teiler von p.

-----------------
Analyse: Anzahl der Elementaroperationen in Abhaengigkeit von p

Eingabe | Anzahl der Elementaroperationen
-------------------------------------------------------------------------------
p = 2,  |  1: 1 in (1.) [Vergleich]
p = 3   |
-------------------------------------------------------------------------------
p = 4...|  3: 1 in (1.), 2 in (2.b.i) [Division und Vergleich]
        |     (so bei allen geraden Zahlen)
p = 5   |  5: 1 in (1.), 2 in (2.b.i),
        |     2 in (2.b.ii.aa) [Multiplikation und Vergleich]
p = 7   | 10: 1 in (1.), 2 in (2.b.i), 2 in (2.b.ii.aa),
        |     2+1 in (2.b.ii.bb.bb1) [Division u. Vergleich, Addition]
        |     2 in (2.b.ii.aa).
p = 9   |  7: 1 in (1.), 2 in (2.b.i), 2 in (2.b.ii.aa),
        |     2 in (2.b.ii.bb.bb1).
...

Die Anzahl der Elementaroperationen haengt nicht nur von der Groesse der
eingegebenen Zahl ab, sondern auch von ihrem kleinsten Primteiler: sobald dieser
bei einer Nicht-Primzahl gefunden ist, bricht der Algorithmus ab (vgl. in der
Tabelle z.B. den Fall p=9). Wir berechnen nur die maximale Anzahl von
Elementaroperationen fuer eine (relativ grosse) Eingabe p. Diese wird nur
erreicht, wenn p eine Primzahl ist: Alle kleineren Primzahlen haben die Schleife
mindestens einmal weniger durchlaufen, fuer alle kleineren Nicht-Primzahlen
fuehrt spaetestens der Vergleich in (2.b.ii.bb.bb1) zu dem Fall (bb2) statt
zurueck zum Schleifenanfang. Fuer kleinere Nicht-Primzahlen werden daher
mindestens 3 Elementaroperationen weniger bis zum Ergebnis gebraucht: die
Addition beim Inkrementieren sowie die Multiplikation und der Vergleich am
Schleifenanfang.

Fuer eine relativ grosse Primzahl p werden die folgenden Elementaroperationen
durchgefuehrt:
1 in (1.), 2 in (2.b.i): Anfangspruefungen
Schleife mit q = 3 beim 1. Durchlauf:
  2 in (2.b.ii.aa)
  2+1 in (2.b.ii.bb.bb1) [Division u. Vergleich,
                          Inkrementieren der Schleifenvariablen]
2 in (2.b.ii.aa): Schleifenabbruch bei q*q > p
```

Anzahl n der Schleifendurchlaeufe:
Es werden nur ungerade Zahlen q<sqrt(p), beginnend bei 3, geprueft, wobei q <= |_sqrt(p)_| am Schleifenanfang gilt, wenn die Schleife vollstaendig durchlaufen wird. Die Anzahl der zu pruefenden ungeraden Zahlen q ist also

(*) $n = \left\lfloor \left\lfloor \sqrt{p} \right\rfloor - 2 \right) / 2 \right\rfloor$

(Wenn |_sqrt(p)_| - 2) ungerade ist, etwas 5-2=3, dann ist nach dem Halbieren nach oben zu runden: im Beispiel erfolgt ein Schleifendurchlauf sowohl mit q=3 als auch mit q=5, also 2 Durchlaeufe. 3/2 muss also nach oben gerundet werden.)

Insgesamt ergibt sich also fuer p die Anzahl der Elementaroperationen nach der folgenden Formel:

(1+2)+ n*(2+2+1)+2 = 5+5n = 5(n+1),

wobei n wie in (*) bestimmt ist.

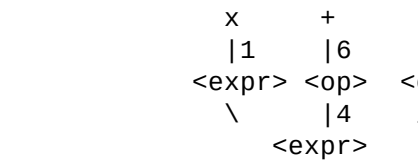
****

Aufg. 3
Regeln
(durchnummeriert fürs Zitieren; "expr" = "Ausdruck" aus Platzgründen):

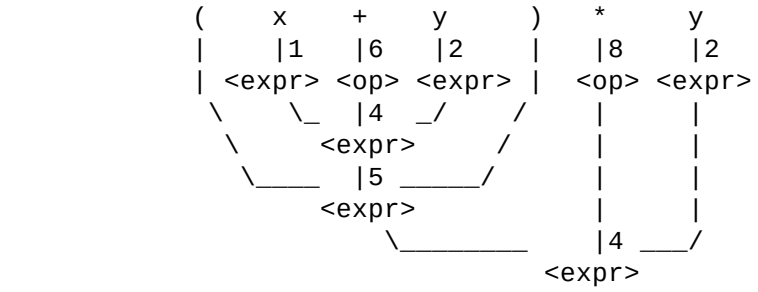
```
          (1) (2) (3)        (4)             (5)
<expr> ->  x | y | z | <expr><op><expr> | (<expr>)

          (6) (7) (8)
<op>   ->  + | - | *
```

Korrektheit bzw. Nichtkorrektheit wird mithilfe von Syntaxbäumen begründet. Aus graphischen Gründen wird ggfs zusätzlicher Leerraum hinzugefügt.

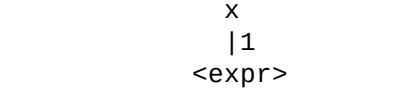
x+y: syntaktisch korrekt



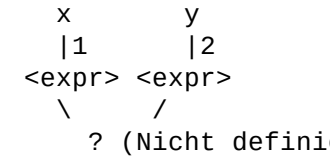
(x+y)*y: syntaktisch korrekt



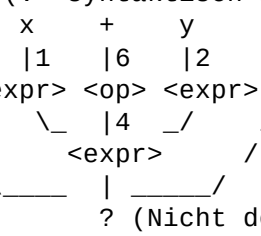
x: syntaktisch korrekt



xy: syntaktisch nicht korrekt, d.h.: keine der Regeln liefert diese Zeichenfolge:



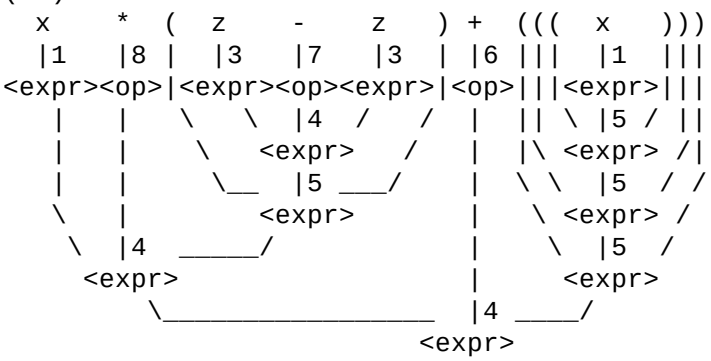
)x+y(: syntaktisch nicht korrekt:



(((x))+y+z+x*(z-z)+(((x)))): syntaktisch korrekt; wir gehen schrittweise vor:

(i) ((x)): <expr> nach (1) und zweimaliger Anwendung von (5)

(ii)



(iii)