

Wie konstruiert man einen BigInteger-Typ (Langzahlarithmetik) ?

Zur Berechnung von sehr großen Ganzzahlen („Big Integers“) kann man „**Register**“ verwenden, die eine bestimmte Anzahl von Ziffern (alle zwischen 0 und 9) aufnehmen. Nimmt man z.B. 9 Ziffern pro Register, so hat man das 1000000000-System, d.h. anstelle der Basis 10 verwenden wir nun die **Basis 1000000000** = 10^9 .

Zum Vergleich:

Der in gängigen Programmiersprachen verwendete Datentyp **Integer** umspannt einen ganzzahligen Zahlenbereich von -2^{31} bis zu $2^{31}-1$. Der Maximalwert ist hier 2147483647, also ca. $2 \cdot 10^9$.

Beispiel für eine lange Zahl mit 3 Registern: 418530014 744010226 159260008

Die verwendete Programmiersprache muss natürlich einen geeigneten Ganzzahl-Datentyp bereitstellen, der in der Lage ist, auch die Multiplikation zweier solcher Register durchführen zu können, ohne dass es zu einem Überlauf innerhalb des Datentyp-Bereiches kommt.

Das größte mögliche Produkt zweier 9-stelligen Zahlen ist :

$$999999999 \cdot 999999999 = 999999998000000001$$

Programmiert man mit **Delphi (bzw. Lazarus)**, so kann man den Datentyp **int64** verwenden.

Bei Verwendung von **Java** eignet sich der Datentyp **long** .

(Anmerkung: Java stellt auch standardmäßig einen Datentyp BigInteger zur Verfügung !)

Beide Datentypen umspannen einen ganzzahligen Zahlenbereich von -2^{63} bis zu $2^{63}-1$.

Der Bereich erstreckt sich also von -9223372036854775808 bis 9223372036854775807 (ca. $9 \cdot 10^{18}$).

Daher eignen sich int64 sowie long als Datentyp für das Rechnen mit der Basis 1000000000 .

Die Ziffern zur Basis 1000000000 werden in einem Array (Feld) x vom Typ Integer abgespeichert.

Besteht die zu verarbeitende Langzahl x also aus $n = 10000$ Stellen, so muss man ein Feld mit $10000 \div 9 + 1 = 1112$ Registern bereitstellen, also $x[0]$ bis $x[1111]$.

Anmerkung: Sollte die Division von n durch 9 ohne Rest aufgehen, so genügen sogar $n \div 9$ Register.

Beispiel: $n=25$ Stellen $x = 8530014744010226159260008$

Es werden $25 \div 9 + 1$ Register, also 3 Register $x[0]$ bis $x[2]$ benötigt. Die Aufteilung ist wie folgt:

x[2]	x[1]	x[0]
8530014	744010226	159260008

Wie man sieht, kann die Anzahl der einzelnen Ziffern eines Registers auch kleiner als 9 sein.

Alternative Basis:

Am Schluss wird auf die Verwendung einer Zweierpotenzbasis eingegangen .

Die Grundrechenarten im 1 000 000 000-System:

1) Summe $c = a + b$:

Maximal kann die Summe zweier Register $999999999 + 999999999 = 1999999998$ sein.
Daher kann man die Addition sogar im Integer-Bereich durchführen (vgl. Rechnung oben) .

Algorithmus:

```
n = Maximum der Längen von a und b
übertrag = 0
für i von 0 bis n-1 wiederhole
  c[i] = a[i] + b[i] + übertrag
  falls c[i] < basis
    dann übertrag = 0
  sonst
    übertrag = 1
    c[i] = c[i] - basis
falls übertrag = 1
  n = n + 1
  c[n-1] = 1
```

Beispiel (2 Register):

	x[1]	x[0]
a	995315926	814210325
b	904707996	538980214

Die Summe $c = a + b$ wird von rechts nach links mit Übertrag (0 oder 1) gebildet.
Ein zusätzliches Register $x[2]$ ist dann erforderlich, wenn das Ergebnis von $x[1]$ überläuft.
Dies ist übrigens beim Beispiel der Fall !

1. $x[0]$ von $a + x[0]$ von b = $x[0]$ von $c = 1353190539$
Da $x[0] > \text{Basis}$, wird die Basis subtrahiert und der Übertrag 1 notiert: $x[0] = \mathbf{353190539}$; Übertrag=1
2. $x[1]$ von $a + x[1]$ von $b + \text{Übertrag}$ = $x[1]$ von $c = \mathbf{900023923}$; Übertrag = 1
3. Übertrag bei $x[1]$ entstanden, also $x[2]$ von $c = \text{Übertrag von } x[1] = \mathbf{1}$

Die Zeichenkette $x[2] x[1] x[0]$ stellt dann die Summe c dar. $c = \mathbf{1900023923353190539}$ (19 Stellen)

2) Differenz $c = a - b$:

Günstig ist es, a größer als b vorauszusetzen, da dann jedes Ergebnis-Register positiv ist. Es entsteht dann auch kein zusätzliches Register.

Falls $a < b$, dann vertauscht man a mit b und setzt vor das Ergebnis ein Minuszeichen.
Auch die Differenzbildung kann (wie die Summenbildung) im Integer-Bereich durchgeführt werden.

Algorithmus:

```
n = Maximum der Längen von a und b
übertrag = 0
für i von 0 bis n-1 wiederhole
  c[i] = a[i] - b[i] - übertrag
  falls c[i] < 0
    dann
      übertrag = 1
      c[i] = c[i] + basis
  sonst übertrag = 0
```

Beispiel (2 Register):

	x[1]	x[0]
a	995315926	314210325
b	904707996	538980214

Die Differenz $c = a - b$ wird von rechts nach links mit Übertrag (0 oder 1) gebildet.

1. $x[0]$ von $a - x[0]$ von b = $x[0]$ von $c = -224769889$
Da $x[0] < 0$, wird die Basis addiert und der Übertrag 1 notiert: $x[0] = 775230111$; Übertrag = 1
2. $x[1]$ von $a - x[1]$ von $b - \text{Übertrag}$ = $x[1]$ von $c = 90607929$

Die Zeichenkette $x[1] x[0]$ stellt dann die Differenz c dar. $c = \mathbf{90607929775230111}$ (17 Stellen)

Produkt $c = a \cdot b$:

Das Produkt kann wegen des möglichen Integer-Überlaufs nur im int64-Bereich (long-Bereich) durchgeführt werden (vgl. Einleitung).

A) Produkt eines BigIntegers mit einem Integer. Der einfachere der beiden Fälle:

Algorithmus:

```
n = Länge von a
übertrag = 0
für i von 0 bis n-1 wiederhole
  c[i] = a[i] · b + übertrag
  falls c[i] < basis
    dann übertrag = 0
  sonst
    übertrag = c[i] div basis
    c[i] = c[i] - übertrag · basis
falls übertrag > 0
  n = n + 1
  c[n-1] = übertrag
```

Beispiel (2 Register):

	x[1]	x[0]		
a	995315926	314210325	b	538980214

Das Produkt $c = a \cdot b$ wird von rechts nach links mit Übertrag gebildet.

- $x[0]$ von $a \cdot b$ = $x[0]$ von $c = 169353148209509550$
Da $x[0] > \text{Basis}$, wird ein Übertrag berechnet mittels $x[0] \text{ div Basis}$: Übertrag = 169353148
 $x[0]$ wird ersetzt durch $x[0] \text{ mod Basis}$: $x[0]$ von $c = \mathbf{209509550}$
- $x[1]$ von $a \cdot b + \text{Übertrag}$ = $x[1]$ von $c = 536455590962441312$
Da $x[1] > \text{Basis}$, wird ein Übertrag berechnet mittels $x[1] \text{ div Basis}$: Übertrag = 536455590
 $x[1]$ wird ersetzt durch $x[1] \text{ mod Basis}$: $x[1]$ von $c = \mathbf{962441312}$
- Übertrag = $x[2]$ von $c = \mathbf{536455590}$

Die Zeichenkette $x[2] x[1] x[0]$ stellt dann das Produkt c dar. $c = \mathbf{536455590962441312209509550}$

B) Produkt zweier BigIntegers:

Algorithmus:

```
n1 = Länge von a
n2 = Länge von b
n = n1 + n2 // Länge von c
für i von 0 bis n-1 wiederhole
  c[i] = 0
für i von 0 bis n1-1 wiederhole
  übertrag = 0
  für k von 0 bis n2-1 wiederhole
    tmp = a[i] // tmp vom Datentyp int64 bzw. long !!
    übertrag = übertrag div basis + tmp · b[k] + c[i+k]
    c[i+k] = übertrag mod basis
  c[i+n2] = übertrag div basis
```

Beispiel (2 Register):

	x[1]	x[0]	
a	000023410	074658394	
b	000007631	110240019	

Das Produkt $c = a \cdot b$ wird von rechts nach links mit Übertrag gebildet.

- $x[0]$ von $a \cdot x[0]$ von b = $x[0]$ von $c = 8230342773069486$
Da $x[0] > \text{Basis}$, wird ein Übertrag berechnet mittels $x[0] \text{ div Basis}$: Übertrag = 8230342
 $x[0]$ wird ersetzt durch $x[0] \text{ mod Basis}$: $x[0]$ von $c = \mathbf{773069486}$
- $x[0]$ von $a \cdot x[1]$ von $b + \text{Übertrag}$ = $x[1]$ von $c = 569726434956$
Da $x[1] > \text{Basis}$, wird ein Übertrag₀ berechnet mittels $x[1] \text{ div Basis}$: Übertrag₀ = 569
 $x[1]$ wird ersetzt durch $x[1] \text{ mod Basis}$: $x[1]$ von $c = 726434956$
- $x[1]$ von $a \cdot x[0]$ von $b + x[1]$ von c = $x[1]$ von $c = 2581445279746$
Da $x[1] > \text{Basis}$, wird ein Übertrag berechnet mittels $x[1] \text{ div Basis}$: Übertrag = 2581
 $x[1]$ wird ersetzt durch $x[1] \text{ mod Basis}$: $x[1]$ von $c = \mathbf{445279746}$

4. $x[1]$ von $a \cdot x[1]$ von $b + \text{Übertrag} + \text{Übertrag}0 = x[2]$ von $c = 178644860$; $\text{Übertrag} = 0$

Die Zeichenkette $x[2] x[1] x[0]$ stellt dann das Produkt c dar. $c = \mathbf{178644860445279746773069486}$

Quotient $\text{quot} = a \text{ div } b$ (sowie $\text{rest} = a \text{ mod } b$):

Es gibt 2 Fälle:

A) Division eines BigIntegers durch einen Integer. Der einfachere der beiden Fälle:

Algorithmus:

```
n = Länge von a
übertrag = 0
für i von n-1 ab bis 0 wiederhole
  tmp = a[i] + übertrag · basis
  c[i] = tmp div b
  übertrag = tmp mod c[i]
rest = übertrag
quot = c[i]
```

Beispiel (2 Register):

	x[1]	x[0]		
a	995315926	314210325	b	538980214

Der Quotient $\text{quot} = a \text{ div } b$ wird von links nach rechts mit Übertrag gebildet.

1. $x[1]$ von $a \text{ div } b$ = $x[1]$ von $\text{quot} = \mathbf{1}$
Übertrag = $x[1]$ von $a - x[1]$ von $\text{quot} \cdot b$: Übertrag = 456335712
2. $(x[0]$ von $a + \text{Übertrag} \cdot \text{Basis}) \text{ div } b$ = $x[0]$ von $\text{quot} = \mathbf{846665054}$
Übertrag = $(x[0]$ von $a + \text{Übertrag} \cdot \text{Basis}) - x[0]$ von $\text{quot} \cdot b$: Übertrag = 322968769
Rest = Übertrag = 322968769

Es gilt also: $\text{quot} = a \text{ DIV } b = \mathbf{1846665054}$ und $\text{rest} = a \text{ MOD } b = \mathbf{322968769}$

B) Division zweier BigIntegers:

Der (ganzzahlige) Quotient $\text{quot} = a \text{ div } b$ wird bestimmt durch sukzessive Differenzbildung von $a - b$.
Der Algorithmus stoppt, wenn $a - b < 0$, d.h. wenn a kleiner ist als b .
 $a \text{ div } b$ ist dann gleich der Anzahl k der Subtraktionen mit positivem Ergebnis für $a - b$.
Das vorletzte Ergebnis der Subtraktionen ist gleich dem ganzzahligen Rest $a \text{ mod } b$!

Dieser Algorithmus ist sehr langsam (wegen der langsamen Subtraktion) !!
Aus diesem Grund wird hier ein wesentlich schnellerer Algorithmus angegeben:

Besserer (schnellerer) Algorithmus:

```
n = Maximum der Längen von a und b
c = 0
falls a < b
  rest = a
  Abbruch
falls a = b
  rest = c
  c = 1
  Abbruch
tmp = b
wiederhole
  b = 2 · b
bis b ≥ a
wiederhole
  c = 2 · c
  b = b / 2
  falls a ≥ b
    a = a - b
    c = c + 1
bis tmp ≥ b
quot = c
rest = a
```

Beispiel (2 Register):

	x[1]	x[0]
a	000001589	993310697
b	000000470	799680214

1. $b = 2b$ bis $b \geq a$

2. $c = 2c$ $b = b/2$

3. $a \geq b$, also $a = a - b$ $c = c + 1$

4. $c = 2c$ $b = b/2$

5. $a \geq b$, also $a = a - b$ $c = c + 1$

6. $tmp \geq b$, also Abbruch der Schleife

8. $rest = a$

$b = 1883198720856$

$c = 0$

$c = 1$

$c = 2$

$c = 3$

$b = 941599360428$

$a = 648393950269$

$b = 470799680214$

$a = 177594270055$

$rest = 177594270055$

Es gilt also: $quot = a \text{ DIV } b = 3$ und $rest = a \text{ MOD } b = 177594270055$

Weitere Funktionen im 1 000 000 000-System:

1) Fakultät fak = n! :

Es ist $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$

Dies ist eine sehr schnell wachsende Funktion. Daher sollte man n aus dem Integer-Bereich wählen. Zum Beispiel ist $5! = 120$. Aber bereits $100! \approx 9,3 \cdot 10^{157}$ übertrifft die Kapazität gewöhnlicher TRs. Der Wert von $1000! \approx 4 \cdot 10^{2567}$ zeigt den rasanten Anstieg der Funktion.

Algorithmus:

```
fak = 1
für i von 2 bis n wiederhole
    fak = fak · i ;
```

Benötigt wird hier die Multiplikation eines BigIntegers mit einem Integer.

2) Binomialkoeffizient („n über k“): $binom = \binom{n}{k}$; $n \geq k$

Es gilt:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} = \frac{n(n-1)(n-2) \cdot \dots \cdot (n-(k-2)) \cdot (n-(k-1))}{1 \cdot 2 \cdot \dots \cdot (k-2) \cdot (k-1) \cdot k}$$
$$\frac{(n-k+1) \cdot (n-k+2) \cdot \dots \cdot (n-k+(k-1)) \cdot (n-k+k)}{1 \cdot 2 \cdot \dots \cdot (k-2) \cdot (k-1) \cdot k} = \prod_{i=1}^k \frac{n-k+i}{i}$$

Auch hier wachsen die Werte sowie auch die Rechenzeiten schnell an. Daher sollte man n und k aus dem Integer-Bereich wählen.

Zum Beispiel ist $\binom{10}{5} = 252$. Aber bereits $\binom{400}{200} \approx 1,03 \cdot 10^{119}$ übertrifft die Kapazität gewöhnlicher TRs.

Weitere Ergebnisse: $\binom{10000}{6000} \approx 5,8 \cdot 10^{2920}$ sowie $\binom{200000}{100000} \approx 1,8 \cdot 10^{60203}$.

Algorithmus(optimiert):

```
falls n < k
    bino = 0
    Abbruch
falls n < k + k
    k = n - k
falls k = 0
    bino = 1
    Abbruch
falls k = 1
    bino = n
    Abbruch
bino = n-k+i // der Startwert
für i von 2 bis k wiederhole
    bino = bino · (n-k+i) div i
```

Benötigt werden einfache Multiplikation und einfache Division von BigInteger und Integer !

Beispiel: $n = 10$ $k = 5$

$binom = 10-5+1 = 6$

$binom = 6 * (10-5+2) \text{ div } 2 = 42 \text{ div } 2 = 21$

$binom = 21 * (10-5+3) \text{ div } 3 = 168 \text{ div } 3 = 56$

$binom = 56 * (10-5+4) \text{ div } 4 = 504 \text{ div } 4 = 126$

$binom = 126 * (10-5+5) \text{ div } 5 = 1260 \text{ div } 5 = \underline{\underline{252}}$

3) $kHn = k$ hoch n (Potenzierung):

k ist hier eine BigInteger-Zahl und n ein Integer. k wird $(n-1)$ -mal mit k multipliziert

Algorithmus(optimiert):

```
kHn = 1;
solange n > 0
  falls n mod 2 = 1 ; n ungerade
    kHn = kHn·k
  k = k·k
  n = n div 2
```

Beispiel: $k = 8$ $n = 5$

$kHn = 1$

$kHn = 1 * 8 = 8$

$k = 8 * 8 = 64$

$n = 5 \text{ div } 2 = 2$

$k = 64 * 64 = 4096$

$n = 2 \text{ div } 2 = 1$

$kHn = 8 * 4096 = \mathbf{32768}$

$k = 4096 * 4096 = 16777216$

$n = 1 \text{ div } 2 = 0$

4) $pMod = (basis \text{ hoch } expo) \text{ mod } m$ („Powermod“):

Es gilt folgende Formel:

$$pMod = basis^{expo} \text{ mod } m = \begin{cases} (basis^{expo/2} \text{ mod } m)^2 \text{ mod } m ; \text{ expo gerade} \\ (basis^{expo-1} \text{ mod } m) \cdot basis \text{ mod } m ; \text{ expo ungerade} \end{cases}$$

Algorithmus(optimiert):

```
pMod = 1;
solange expo > 0
  falls expo mod 2 = 1 ; expo ungerade
    pMod = (pMod·basis) mod m;
  expo = expo >> 1; // Shift right !
  basis = (basis·basis) mod m;
```

Beispiel: $basis = 3$ $expo = 9$ $m = 17$

$pMod = 1$

$pMod = 1*3 \text{ mod } 17 = 3$

$expo = 9 \text{ div } 2 = 4$

$basis = 3^2 \text{ mod } 17 = 9$

$expo = 4 \text{ div } 2 = 2$

$basis = 9^2 \text{ mod } 17 = 13$

$expo = 2 \text{ div } 2 = 1$

$basis = 13^2 \text{ mod } 17 = 16$

$pMod = 3*16 \text{ mod } 17 = \mathbf{14}$

$expo = 1 \text{ div } 2 = 0$

$basis = 16^2 \text{ mod } 17 = 1$

5) $ggT = ggT(a, b)$:

Es wird der Euklidische Algorithmus verwendet ($a, b \geq 0$, $ggT > 0$):

```
solange b > 0
  tmp = a mod b
  a = b
  b = tmp
ggT = a
```

Anschließend kann noch das kgV berechnet werden mit: $kgV = a*b / ggT$.

6) zufZahl = random(x0,a,c,m) :

Zufallszahlen berechnet man am besten mit der LEHMER-Methode, wobei eine Folge von "zufälligen" Zahlen generiert wird, die in eine Periode läuft:

Linearer Kongruenzgenerator nach D.H.Lehmer (1948)

Formel: $x_{n+1} = (a \cdot x_n + c) \bmod m$

Soll die Periode = m sein, so beachte man 3 Bedingungen:

- (1) c und m teilerfremd
- (2) a-1 ist ein Vielfaches jedes Primteilers von m
- (3) falls m ein Vielfaches von 4 ist, so ist es auch a-1

Algorithmus:

```
zufZahl = x0
wiederhole
    zufZahl = (a·zufZahl + c) mod m
bis Abbruch
```

7) Primzahltest (probabilistisch) :

Der Primzahltest nach Miller-Rabin wird verwendet:

Hier eine Java-Methode mit dem Datentyp long.

Für BigInteger's müssen die entsprechenden Anweisungen umgeschrieben werden.

```
public static boolean istPrimMillerRabin(long n) {
    // probabilistische Methode für ungerade n>2
    long a, x, k = 20; // Test wird max. 20-mal durchlaufen

    // n-1 = 2^s*d ; d ungerade erzeugen
    long d = n - 1;
    int s = 0;
    while (d % 2 == 0) {
        d = d / 2;
        s = s + 1;
    }
    System.out.println("d= " + d + " s=" + s); // Test

    for (long i = 1; i <= k; i++) {
        a = 2 + (long) (Math.random() * (n - 2)); // a = random(2,n-1);
        System.out.println("a= " + a); // Test

        // Berechnung von x = a^d mod n
        x = powerMod(a,d,n);
        System.out.println("x= " + x); // Test
        if (x == 1 || x == n - 1)
            continue;
        for (int r = 1; r < s; r++) {
            x = x * x % n;
            if (x == 1)
                return false;
            if (x == n - 1)
                break;
        }
        if (x != n - 1)
            return false;
    }
    return true; // wahrscheinlich ist n prim
}
```


8) atan = arctan(x) :

Auch reelle Ergebnisse kann man ermitteln, z.B. den Arcustangens:

$$\arctan(x) \approx \sum_{k=0}^n (-1)^k \cdot \frac{x^{2k+1}}{2k+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \dots + (-1)^n \cdot \frac{x^{2n+1}}{2n+1} ; |x| \leq 1$$

Hier muss man zusätzlich zu der vorgegebenen (gewünschten) Stellenzahl noch einige Zusatzstellen (Schutzstellen) verwenden, damit die Ergebnisse nicht ungenau werden.

Statt x verwenden wir den Bruch p / q mit den Integers p und q .

```
public static void init(int gesStellen) {
    int z = gesStellen + cZusatzRegister*cRegStellen;
    kmax = (int) (1.0 + z / Math.log10(q/p));
    anzahlRegs = z / cRegStellen;
    regSumX = new long[anzahlRegs+1];    regSum5[0]=p;
    regXpot = new long[anzahlRegs+1];    regXpot[0]=p;
    regXdurchK = new long[anzahlRegs+1];
}

public static int[] atan(int p, int q) {
    if (p == 0) return {0};
    boolean negativ;
    int k;
    // berechne arctan (p / q)
    regXpot = RegQuotient(regXpot, q, anzahlRegs); // x = p/q
    regSumX = RegQuotient(regSumX, q, anzahlRegs); // summe = p/q
    negativ = true;
    k = 1;
    while (k < kmax) {
        k += 2;
        regXpot = RegQuotient(regXpot, q*q, anzahlRegs);
        regXdurchK = RegQuotient2(regXpot, k, anzahlRegs);
        if (negativ) {
            regSumX = RegDifferenz(regSumX, regXdurchK, anzahlRegs);
            negativ = false;
        } else { // Vorzeichen +
            regSumX = RegSumme(regSumX, regXdurchK, anzahlRegs);
            negativ = true;
        }
    }
    // while
    return regSumX;
}
```

Anhang: Alternative Basis verwenden:

Da man mit Zweierpotenzen vorteilhaft rechnen kann, eignet sich als Basis auch $b = 2^{31} = \underline{2147483648}$.

Ein Datentyp, der die Basis $b = 2^{31}$ verarbeiten kann, ist **long** (0 bis $2^{63} - 1 = 9223372036854775807$). Sowohl Addition als auch Multiplikation können innerhalb des Bereichs von long durchgeführt werden.

Verwendet wird also ein array vom Datentyp long mit long-Basis $b = 2147483648$.

Die **größte Zahl**, die ein array-Element aufnehmen kann, ist dann Integer.MAX_VALUE = **2147483647** .

Addiert man die beiden größten array-Werte, so erhält man 4294967294L (= $2^{32}-2$). $summe = a[i] + b[i]$.
Anschließend wird der Übertrag bzgl. der Basis folgendermaßen ermittelt :
Ist $summe < Basis$, dann ist der Übertrag = 0, andernfalls ist der Übertrag = 1 und es muss von der summe noch die Basis subtrahiert werden, damit das Ergebnis im Bereich $[0 ; Basis-1]$ liegt.

Multipliziert man die beiden größten Zahlen miteinander, so erhält man 4611686018427387904 , was kleiner ist als long.MAX_VALUE = 9223372036854775807.

Auch hier muss der Übertrag bestimmt werden.

Da dieser jedoch wesentlich größer als 1 werden kann, muss man folgendermaßen vorgehen:

$produkt = (long)(a[i]) * b[i]$. Es muss also zunächst die Zahl $a[i]$ in long umgewandelt werden, weil andernfalls bei der Multiplikation Integer „überlaufen“ könnte.

Ist $produkt < Basis$, dann ist der Übertrag 0 (genauso wie bei der Summe) .

Andernfalls muss der Übertrag durch Division bestimmt werden: $Übertrag = produkt \div Basis$.

Der Produktwert ergibt sich zu: $produkt = produkt \bmod Basis$.

Schnelle Operationen für Zweierpotenzen:

```
procedure setBit(var x:LongWord; bitNr:Byte);  
  begin x:=x or (1 shl bitNr); end;
```

```
procedure clearBit(var x:LongWord; bitNr:Byte);  
  begin x:=x and not (1 shl bitNr); end;
```

```
procedure mal2(var x,carry:LongWord);  
  begin carry:=x shr 15; x:=x shl 1; end;
```

```
procedure durch2(var x:LongWord);  
  begin x:=x shr 1; end;
```

```
procedure differenz(var diff:LongWord; a,b:LongWord);  
begin  
  b:=not b; // Zweierkomplement des Subtrahenden b  
  b:=b+1; // um 1 erhöhen  
  diff:=a+b; // und zu a addieren!  
end;
```

```
procedure produkt(var prod:LongWord; a,b:LongWord); // russische Bauernmultiplikation  
begin  
  prod = 0;  
  while b>0 do begin  
    if odd(b) then prod:=prod+a; // odd(b): prüfen, ob bit Nr. 0 von b gesetzt ist !  
    a:=a shl 1; //a:=a*2;          b: 111010010000101 ungerade !  
    b:=b shr 1; //b:=b/2;  
  end;  
end;
```

Umwandlung der Zahl (des Strings) x in „Integer-Pakete“ :

Umwandlung der „Integer-Pakete“ in die Zahl (den String) x :