

# Powermod: Programmierung von pmod(a,b,m)

Prof. Dr. Dörte Haftendorn, Mathematik mit MuPAD 4.02, (ex. in 3.11 Sept. 05) Feb.07

<http://haftendorn.uni-lueneburg.de> [www.mathematik-verstehen.de](http://www.mathematik-verstehen.de)

#####

$$a^b \text{ mod } m$$

Der wichtigste Term der Kryptographie ist

Dabei sind alle drei Variablen riesige Zahlen mit einer Dezimalen Stellenlänge weit über 100.

Aber auch schon im bescheidenen Rahmen einer Berechnung mit 3-stelligen Zahlen in der Lehre

kann man (ohne großes CAS) nicht einfach die Potenz ausrechnen und dann den modularen Rest bestimmen:

```

pot:=109^47
5741764862085664539988180789670220849290368606745530758352920831581818850615883676792554
pot mod 211
125

```

Alle großen CAS haben dafür den Befehl **powermod(a,b,m)** (in irgendeiner Syntax), bei dem

mit geschickten Zwischenrechnungen immer sofort modular "heruntergebrochen" wird.

Kleinen CAS, insbesondere CAS-Taschenrechnern wie dem TI-voyage, fehlt diese Fkt.

Die unten entwickelte Prozedur kann -unter Beachtung kleiner Sytaxanpassungen- sofort auf den TI-voyage (oder TI92 ) übertragen werden. download der TI-Datei ist möglich.

**Grundidee** Es werden nacheinander "Potenztürme" gebaut:

$$a, a^2, (a^2)^2, \left( (a^2)^2 \right)^2, \dots$$

Bei der Entscheidung, ob ein solcher Turm als Faktor

zum

Aufbau von  $a^b$  benötigt wird hilft die Dualdarstellung des Exponenten b.

Zum Beispiel b=11. Dann ist  $b = \text{IOII}$  und  $a^{11} = a^{\text{IOII}} = a^{2^3} a^2 a$ .

Nun braucht man aber diese Dualdarstellung nicht **explizit** zu erzeugen, sondern man nutzt die "Doublel-Daddel-Methode" statt zum Erzeugen der Dualzahl gleich zu Erzeugen der Potenz.

```

[

```

"Doublel-Daddel-Methode" zum Erzeugen der Dualzahl ---->Extraseite

## Programmierung von pmod(a,k,m) #####

```

pmod:=proc(a: Type::Rational,k: Type::Integer,m: Type::Integer)
  local x, kk, pot;
  begin
    kk:=k; x:=1: pot:=a:
    if k<0 then kk:=-k: pot:=1/a: end_if;
    //Abfangen negativer Exponenten
    if k=0 then
      if a<>0 then return(1);
      elif a=0 then return ("unbestimmt");
    
```

```

        elif a=0 then return ("unbestimmt");
        end_if;
    end_if;
    //Abfangen von a^0 und 0^0
repeat
    if kk mod 2 =1 then
        x:= x*pot mod m ;
        if kk=1 then return(x); end_if;
        kk:=kk-1; //hiernach ist k gerade
    end_if;
    kk:= kk/2; //immer möglich
    pot:= pot*pot mod m;
until 3=5 end_repeat;
end_proc:

```

```

pmod(27,5,31);
powermod(27,5,31); // MuPAD-Funktion

```

30

30

Die folgende Matrix zeigt, dass das eigene pmod und das eingebaute powermod für einige positive Eingaben identische Werte liefern.

```

matrix([[ [pmod(a,k,31), powermod(a,k,31)] $ k=25..31]
        $ a=27..30]);

```

$$\begin{pmatrix} [30, 30] & [4, 4] & [15, 15] & [2, 2] & [23, 23] & [1, 1] & [27, 27] \\ [25, 25] & [18, 18] & [8, 8] & [7, 7] & [10, 10] & [1, 1] & [28, 28] \\ [30, 30] & [2, 2] & [27, 27] & [8, 8] & [15, 15] & [1, 1] & [29, 29] \\ [30, 30] & [1, 1] & [30, 30] & [1, 1] & [30, 30] & [1, 1] & [30, 30] \end{pmatrix}$$

Übrigens kann man an dieser Liste schon sehen, dass bei Potenzen im Modul allerlei Besonderheiten auftauchen, die man erkunden kann. Siehe Extraseite zu Potenzen.

## Tests zu Sonderfällen #####

```

powermod(2,0,7);
pmod(2,0,7);

```

1

1

```

powermod(0,0,7);
pmod(0,0,7);

```

1

"unbestimmt"

Na, da ist das eigene pmod sogar besser!

Als Basis können rationale Zahlen genommen werden, denn der mod-Befehl von MuPAD kann diese verarbeiten.

denn der mod-Befehl von MuPAD kann diese verarbeiten.

```
[ pmod(15/13, 1, 17)
```

```
9
```

```
[ 15/13 mod 17
```

```
9
```

**Probe**

```
[ i13:=(1/13 mod 17);
```

```
i13*13 mod 17;
```

```
(15*i13)*13 mod 17;
```

```
4
```

```
1
```

```
15
```

**Auch negative Basen können genommen werden.**

```
[ -3 mod 11;
```

```
pmod(-3, 5, 11);
```

```
powermod(-3, 5, 11);
```

```
8
```

```
10
```

```
10
```

**Beide Funktionen verarbeiten auch negative Exponenten,**

```
[ powermod(2, -3, 11);
```

```
pmod(2, -3, 11);
```

```
7
```

```
7
```

```
[ 7*2^3 mod 11 //Probe
```

```
1
```