

Re: Seeking efficient sequential-Carmichael-number generator

Usage:

=====

Step 1: cd into the directory where the file Carmichael.SCM is located.

Step 2: Call Guile with

```
guile -l Carmichael.SCM
```

Step 3: Enter

```
(carmichael-bis <integer-number>)
```

Place a integer number for <integer-number>.
The program will compute all Carmichael Numbers up to this Integer.

Example:

```
(carmichael-bis 1800)
```

This will produce the output:

```
561  
1105  
1729
```

These are all Carmichael Numbers up to 1800.

A German manual is also included.

World Wide Web:

<http://www.norman-interactive.com> (my Homepage)

<http://www.informatik.uni-stuttgart.de> (University Stuttgart, Dept. of Computer Science)

=====

```
-----Carmichael.SCM-----  
; Carmichael.SCM  
; Programm zum Auffinden von Carmichael-Zahlen  
; Carmichael-Zahlen sind Zahlen, die den Fermat-Test überlisten.  
; 2. überarbeitete Version.  
; Autor: Norman Walter  
; Version 2.0
```

Re: Seeking efficient sequential–Carmichael–number generator

; Datum : 8.11.2002

```
(define (gerade? n)
  (= (remainder n 2) 0)
)
```

```
(define (quadrat x)
  (* x x)
)
```

; Euklidischer Algorithmus zur Ermittlung des ggT

```
(define (ggT a b)
  (if (= b 0)
      a
      (ggT b (remainder a b)))
)
```

; Prozedur zur Berechnung der Potenz einer Zahl modulo m.
; Sie verwendet sukzessive Quadratbildung, so daß die Anzahl der
; Schritte logarithmisch mit dem Exponenten wächst.

```
(define (potmod basis exp m)
  (cond ((= exp 0) 1)
        ((gerade? exp)
         (remainder (quadrat (potmod basis (/ exp 2) m))
                    m))
        (else
         (remainder (* basis (potmod basis (- exp 1) m))
                    m)))
)
```

; Fermat–Test
; Gibt der Fermat–Test true zurück, so ist die Zahl n mit Sicherheit
; zusammengesetzt. Ergibt er false, so ist nicht ganz sicher, ob
; n wirklich eine Primzahl ist. Sie könnte z.B. eine
; "Basis–2–Pseudoprimzahl" sein.
; In dieser Version wird nicht nur auf $2^{(n-1)} \bmod n = 1$
; getestet, sondern auf $b^{(n-1)} \bmod n = 1$.

```
(define (fermat b n)
  (not (= (potmod b (- n 1) n) 1))
)
```

```
(define (kleinster–teiler n)
  (finde–teiler n 2)
)
```

; n hat einen Teiler kleiner oder gleich Wurzel n, wenn es keine

Re: Seeking efficient sequential-Carmichael-number generator

; Primzahl ist. Der Algorithmus muß also nur Teiler zwischen 1
; und Wurzel n überprüfen.

```
(define (finde-teiler n pruef-teiler)
  (cond ( (> (quadrat pruef-teiler) n) n)
        ( (teilt? pruef-teiler n) pruef-teiler)
        (else (finde-teiler n (+ pruef-teiler 1))
          )
        )
  )
)
```

```
(define (teilt? a b)
  (= (remainder b a) 0)
)
```

; Gibt #t zurück, falls n DEFINITIV eine Primzahl ist, sonst #f

```
(define (primzahl? n)
  (= n (kleinster-teiler n))
)
```

; Liefert #t, falls n Carmichael-Zahl ist, sonst #f

```
(define (carmichael? n)
  (if (eq? (primzahl? n) #f)
      (carmichael-iter 1 n)
      #f)
)
```

; Liefert alle Carmichaelzahlen bis zu einer Obergrenze

```
(define (carmichael-bis obergrenze)
  (carmichael-bis-iter 1 (+ obergrenze 1))
)
```

```
(define (carmichael-bis-iter zaehler obergrenze)
  (cond ( (< zaehler obergrenze)
        (cond ( (carmichael? zaehler)
              (display zaehler)
              (newline)
              )
          )
        )
        (carmichael-bis-iter (+ zaehler 1) obergrenze)
        )
)
```

; Iteration über a:
; Falls für alle $0 < a < n$ gilt:
; n ist keine Primzahl und

Re: Seeking efficient sequential-Carmichael-number generator

; ggT(a,n) impliziert a hoch (n-1) kongruent 1 modulo n,
; dann ist n eine Carmichael-Zahl.

```
(define (carmichael-iter a n)
```

```
; Implikation:
```

```
(if (or (not (= (ggT a n) 1))  
(eq? (fermat a n) #f))  
)  
(if (< a (- n 1))  
(carmichael-iter (+ a 1) n)  
#t  
)  
#f  
)  
)
```

----- History txt

history.txt

31.10.2002 – Erste Version 1.0

8.11.2002 – Version 2.0:

Änderungen im Vergleich zu Version 1.0:

Aus

```
(define (fermat n)  
(not (= (potmod 2 (- n 1) n) 1))  
)
```

wurde

```
(define (fermat b n)  
(not (= (potmod b (- n 1) n) 1))  
)
```

Der zusätzliche Parameter b ermöglicht es nun, den Fermat-Test nicht nur zur Basis 2, sondern zur Basis b durchzuführen.

Die Prozedur

```
(define (kongruent? a n)  
(= (potmod a (- n 1) n)  
(modulo 1 n))  
)
```

Re: Seeking efficient sequential-Carmichael-number generator

flog komplet heraus.

Zum einen ist $1 \bmod n$ immer 1, zum anderen können wir den gleichen Test jetzt auch mit der erweiterten Fermat-Prozedur durchführen:

```
(define (carmichael-iter a n)
```

```
  ; Implikation:
```

```
  (if (or (not (= (ggT a n) 1))
```

```
        (eq? (fermat a n) #f)
```

```
      )
```

```
      (if (< a (- n 1))
```

```
          (carmichael-iter (+ a 1) n)
```

```
          #t
```

```
      )
```

```
      #f
```

```
    )
```

```
  )
```

.