Die Programmiersprache repsub (Repeated Substitution)

Inhaltsverzeichnis

Inhaltsverzeichnis

Grundidee

Auf einem Eingabestring wird die allbekannte Funktion „Suchen und Ersetzen“ durchgeführt. Das wird mit einer ganzen Liste von Such und Ersetz-Befehlen so lange gemacht, bis sich nichts mehr ändert.

Entstehungsgeschichte

Mir war in einem Hotellzimmer mit Laptop aber ohne Internetanschluss langweilig, und da kam ich auf die Idee, eine demokratische Programmiersprache zu schreiben. Demokratisch im Sinne, dass sie nicht hierarchisch aufgebaut ist, sondern dass alle Funktionen und Programmzeilen und alle Speicherpositionen gleichberechtigt sind.

Befehlsumfang

Zusätzlich zum „Suchen und Ersetzen“ gibt's noch Wildcards:

```
find:
*? : Beliebiges Zeichen (außer char(0) )
** : Sternchen
*xyz* : x,y oder zett

subs:
*n : Pointer auf Buchstabe nummer n. m.a.w.
es wird der 'n'-te Buchstabe der gefundenen Textstelle eingefuegt.
** : Sternchen.
*0 : Sonderbefehle
*0?nXWort1:Wort2; if n-terBuchstabe == X then Wort1 else Wort2
*0-n n-ter Buchstabe -1
*0+n n-ter Buchstabe +1
```

Besondere Features von repsub

1. Die Programmiersprache ist vom Grundprinzip her echt parallel aufgebaut. Alle Speicherpositionen, Programmzeilen und Funktionen sind gleichberechtigt. Es können im Prinzip so viele Prozesse wie der gerade aktuelle Text Zeichen hat, parallel laufen. Diese Zeichenzahl kann bei jeder Iteration ver-n-facht werden, d.h. exponentiell mit der Zeit so ziemlich jede Computerresource ausnutzen.
2. Durch die Wildcards im find-Text kann man nicht nur nach Suchworten suchen, sondern nach Mustern. Repsub ist also eigentlich ein auf Mustererkennung basierendes System. Und Mustererkennung liegt grad so richtig im Trend.
3. Der sicherlich wichtigste Vorteil von repsub ist die Verbindung der beiden Umstände, dass hier erstens das Sternchen (*) das zentrale Sonderzeichen ist, und zweitens, dass gilt „*“=char(42).
4. Die Programmiersprache hat weniger Befehle als Brainfuck(*?+-), ist potentiell kryptischer als C, einfacher als FORTH und eleganter als Intercal.

Programmstruktur

Ein Repsubprogramm besteht aus fortlaufendem Text. Besteht eine Zeile aus zwei durch Space getrennten Worten, so werden diese als „Suchen und Ersetzen“ interpretiert, alle Zeilen die mehr oder weniger als zwei Worte enthalten, werden als Kommentarzeilen interpretiert.

Um nun auch Space und Sonderzeichen suchen und ersetzen zu können, werden alle zusammengeschriebenen Worte vorinterpretiert und zwar folgendermaßen: „_-“--> „ “ , „__“->“_“ und „_c042“ –> char(42)= „*“

Bsp.: „a_c042c_-d__e“-> „a*b c_d“

Ist der Eingabestring leer, so wird er zu Beginn auf „s“ gesetzt.

Beispielprogramme

Hello World:

```
s hello_-World!
```

Einfache Schleife:

Zunächst ohne Kommentarzeilen:

```
sterne(1) **
sterne(*?) **sterne(*0-8)
```

```
Dieses Programm gibt bei sterne(n) *** das ist eine Kommentarzeile
genau n sterne zurück: *** das auch
sterne(1) **
Ende der Schleife, sterne(1) ist ein stern
sterne(*?) **sterne(*0-8)
Schleifenkörper: (sterne(n) –> *sterne(n-1) )
```

```
sterne(4)
(1): repsub('sterne(*?) **sterne(*0-8)')
*sterne(3)
(2): repsub('sterne(*?) **sterne(*0-8)')
**sterne(2)
(3): repsub('sterne(*?) **sterne(*0-8)')
***sterne(1)
(4): repsub('sterne(1) **')
****
Append(repsub) repeated substitution
sterne(1) **
sterne(*?) **sterne(*0-8)
```

Addition zweier einstelliger Zahlen:

```
*?+0 *1
*?+*? *0+1+*0-3
```

```
3+4
(1): repsub('*?+*? *0+1+*0-3')
4+3
(2): repsub('*?+*? *0+1+*0-3')
5+2
(3): repsub('*?+*? *0+1+*0-3')
6+1
(4): repsub('*?+*? *0+1+*0-3')
7+0
(5): repsub('*?+0 *1')
7
Append(repsub) repeated substitution
*?+0 *1
*?+*? *0+1+*0-3

Kürzer wäre:
*?+*? *0?30*1:*0+1+*0-3;
```

Zählen aller Worte „Karl“ (bis zu neun) in einem Text:

```
Karl*? *5Karl
*123456789*KarlKarl *0+1Karl
KarlKarl 2Karl
```

Idee: Alle Karls werden ans Ende geschoben (erste Zeile)

Beweis, dass repsub so mächtig wie eine Turing-Maschine ist

Man betrachte folgendes Programm:

Starttext: 1011010111000001010start10001001000000010101

Programm:

```
start z001
*?z001*? *0?61*11z002:z003*10;
*?z002*? *0?61*11z003:z003*10;
*?z003*? *0?61*11z003:z001*10;
.
.
.
```

Gehe in Zustand z001
Wenn Zahl hinter z001==1 dann schreibe 1, gehe nach rechts und in Zustand 002 andernfalls schreibe null gehe nach links und in Zustand 003. ...

Auf diese Weise lässt sich jede Turingmaschine in eine Folge von „Suchen und Ersetzen“-Befehlen umwandeln. q.e.d. (Das ist lateinisch und heißt w.z.b.w. Oder auf englisch w.w.w.w.w. :-)

Wir haben hier also immer ein aktives Zentrum, das bei jedem Durchlauf einmal gefunden und ersetzt wird. Die Idee des aktiven Zentrums lässt sich verallgemeinern zu mehreren aktiven Zentren. Diese können über „Boten“ miteinander kommunizieren:

Boten

Zwei aktive Zentren sind vorhanden, jetzt wird ein Bote Richtung Textende losgeschickt und vom aktiven Zentrum az137 empfangen, das beim Empfang in az138 verwandelt wird. Der Bote nimmt zwei Zeichen mit, die nach Erzeugung des az138 direkt hinter diesem stehen. (verfolge die Zeichen „xy“)

```
sodele, (xy)senden und zwar an az137.
(1): repsub('(*?*?)senden bote(*2*3)')
sodele, bote(xy) und zwar an az137.
(2): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, bote(xy)und zwar an az137.
(3): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, ubote(xy)nd zwar an az137.
(4): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, unbote(xy)d zwar an az137.
(5): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, undbote(xy) zwar an az137.
(6): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und bote(xy)zwar an az137.
(7): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zbote(xy)war an az137.
(8): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwbote(xy)ar an az137.
(9): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwabote(xy)r an az137.
(10): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwarbote(xy) an az137.
(11): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwar bote(xy)an az137.
(12): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwar abote(xy)n az137.
(13): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwar anbote(xy) az137.
(14): repsub('bote(*?*?)*? *9bote(*6*7)')
sodele, und zwar an bote(xy)az137.
(15): repsub('bote(*?*?)az137 az138(*6*7)')
sodele, und zwar an az138(xy).
Append(repsub) repeated substitution
(*?*?)senden bote(*2*3)
bote(*?*?)*? *9bote(*6*7)
bote(*?*?)az137 az138(*6*7)
```

Was passiert, wenn sich zwei Boten treffen?

```
toend(ab) Lueckenfueller tostart(xy).
(1): repsub('toend(*?*?)*? *:toend(*7*8)')
toend(ab)Lueckenfueller tostart(xy).
(2): repsub('*?tostart(*?*?) tostart(*:*;)*1')
toend(ab)Lueckenfuellertostart(xy) .
(3): repsub('toend(*?*?)*? *:toend(*7*8)')
Ltoend(ab)ueckenfuellertostart(xy) .
(4): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Ltoend(ab)ueckenfuelletostart(xy)r .
(5): repsub('toend(*?*?)*? *:toend(*7*8)')
Lutoend(ab)eckenfuelletostart(xy)r .
(6): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Lutoend(ab)eckenfuelltostart(xy)er .
(7): repsub('toend(*?*?)*? *:toend(*7*8)')
Luetoend(ab)ckenfuelltostart(xy)er .
(8): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luetoend(ab)ckenfueltostart(xy)ler .
(9): repsub('toend(*?*?)*? *:toend(*7*8)')
Luectoend(ab)kenfueltostart(xy)ler .
(10): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luectoend(ab)kenfuetostart(xy)ller .
(11): repsub('toend(*?*?)*? *:toend(*7*8)')
Luecktoend(ab)enfuetostart(xy)ller .
(12): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luecktoend(ab)enfutostart(xy)eller .
(13): repsub('toend(*?*?)*? *:toend(*7*8)')
Luecketoend(ab)nfutostart(xy)eller .
(14): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luecketoend(ab)nftostart(xy)ueller .
(15): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentoend(ab)ftostart(xy)ueller .
(16): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Lueckentoend(ab)tostart(xy)fueller .
(17): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckenttoend(ab)ostart(xy)fueller .
(18): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentotoend(ab)start(xy)fueller .
(19): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostoend(ab)tart(xy)fueller .
(20): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentosttoend(ab)art(xy)fueller .
(21): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostatoend(ab)rt(xy)fueller .
(22): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostartoend(ab)t(xy)fueller .
(23): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostarttoend(ab)(xy)fueller .
(24): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostart(toend(ab)xy)fueller .
(25): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostart(xtoend(ab)y)fueller .
(26): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostart(xytoend(ab))fueller .
(27): repsub('toend(*?*?)*? *:toend(*7*8)')
Lueckentostart(xy)toend(ab)fueller .
(28): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luecketostart(xy)ntoend(ab)fueller .
(29): repsub('toend(*?*?)*? *:toend(*7*8)')
Luecketostart(xy)nftoend(ab)ueller .
(30): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luecktostart(xy)enftoend(ab)ueller .
(31): repsub('toend(*?*?)*? *:toend(*7*8)')
Luecktostart(xy)enfutoend(ab)eller .
(32): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luectostart(xy)kenfutoend(ab)eller .
(33): repsub('toend(*?*?)*? *:toend(*7*8)')
Luectostart(xy)kenfuetoend(ab)ller .
(34): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Luetostart(xy)ckenfuetoend(ab)ller .
(35): repsub('toend(*?*?)*? *:toend(*7*8)')
Luetostart(xy)ckenfueltoend(ab)ler .
(36): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Lutostart(xy)eckenfueltoend(ab)ler .
(37): repsub('toend(*?*?)*? *:toend(*7*8)')
Lutostart(xy)eckenfuelltoend(ab)er .
(38): repsub('*?tostart(*?*?) tostart(*:*;)*1')
Ltostart(xy)ueckenfuelltoend(ab)er .
(39): repsub('toend(*?*?)*? *:toend(*7*8)')
Ltostart(xy)ueckenfuelletoend(ab)r .
(40): repsub('*?tostart(*?*?) tostart(*:*;)*1')
tostart(xy)Lueckenfuelletoend(ab)r .
(41): repsub('toend(*?*?)*? *:toend(*7*8)')
tostart(xy)Lueckenfuellertoend(ab) .
(42): repsub('*?tostart(*?*?) tostart(*:*;)*1')
tostart(xy) Lueckenfuellertoend(ab) .
(43): repsub('toend(*?*?)*? *:toend(*7*8)')
tostart(xy) Lueckenfueller toend(ab).
(44): repsub('toend(*?*?)*? *:toend(*7*8)')
tostart(xy) Lueckenfueller .toend(ab)
Append(repsub) repeated substitution
Botenkollisionsprogramm:
toend(*?*?)*? *:toend(*7*8)
*?tostart(*?*?) tostart(*:*;)*1
```

Antwort: sie durchdringen sich mehr oder weniger ungehindert.

Ach ja, die Addressierung *1,*2,... läuft über die Neun hinaus: *8, *9, *:, *;, *<, *=,...

So kann man also bis zu 256 Byte relativ zum gefundenen Wort addressieren. Wieso denke ich gerade an Kartoffeln und die gute alte PDP7 :-)

Strukturiertes programmieren :-)

Das ist selbstverständlich auch möglich. Muss aber nicht sein.
Funktionales Programmieren hat aber echte Vorteile:

```
Teil für die einstellige Addition
(*?,0)pluseinstellig *2
(*?,*?)pluseinstellig (*0+2,*0-4)pluseinstellig

Teil für die einstellige Subtraktion
(*?,0)minuseinstellig *2
(*?,*?)minuseinstellig (*0-2,*0-3)minuseinstellig

Teil für die zweistellige Addition
(*?*?,00)pluszweistellig (*2*3)übertragtest
(*?*?,*?*?)pluszweistellig ((*2,*5)pluseinstellig(*3,*6)pluseinstellig,00)pluszweistellig
(*?*0123456789*)übertragstest *2*3
(*?*?)übertragstest ((*2,1)pluseinstellig(*3,:)minuseinstellig)übertragstest

Teil für die zwei- und einstellige Multiplikation
(*?*?,0)multzweistellig 00
(*?*?,1)multzweistellig (*2*3)übertragstest
(*?*?,*?)multzweistellig ((*2*3,0*5)pluszweistellig,*0-5)multzweistellig
```