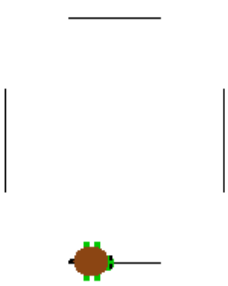
Informatik Abitur Bayern 2017 / IV - Lösung

Autoren:
Marek (1)/
Schulz (2)
Janus (3)

1a 3



In Worten (zur Lösung nicht erforderlich): Es werde 4 mal folgende Anweisungen wiederholt: Absenken des Stiftes, Vorwärtslaufen um 30 Einheiten (hier Millimeter), Drehen um 45 Grad gegen den Uhrzeigesinn, Anheben des Stiftes, Vorwärtslaufen um 30 Einheiten und Drehen um 45 Grad gegen den Uhrzeigersinn. Nach dem viermaligen Ausführung der Anweisungen steht die Turtle wieder auf ihrem Startpunkt (Schwanzspitze der Turtle).

1b Folgendes Alphabet wird für die Turtle-Sprache benötigt: 7

∑ = { 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; V; Z; R; L; O;U; W; ,; (;); !}

Die Produktionsregeln in der erweiterten Backus-Naur-Form (EBNF):

```
Programm = Befehl {"," Befehl} "!";
Befehl = "W" Zahl "(" Befehl {"," Befehl} ")" | "V" Zahl | "Z" Zahl |
         "R" Zahl | "L" Zahl | "O" | "U";
Zahl = Z1 {Z0};
Z1 = "1" | "2" | "3" | ... | "9";
Z0 = Z1 | "0";
```

Zur Erklärung: Anweisungen in geschweiften Klammern können weggelassen werden.
Z_0 beschreibt die Menge aller Ziffern zwischen 0 und 9, Z_1 die Ziffern ohne 0.

2a 3

Anweisung	k	a	t	erg
	5			
a=0		0		
t=1			1	
t=t+1			2	
t=t+1			3	
t=t+1			4	
t=t+1			5	
a=1		1		
erg=1				1

Anweisung	k	a	t	erg
	15			
a=0		0		
t=1			1	
t=t+1			2	
t=t+1			3	
a=1		1		
erg=0				0

2b Das angegebene Programmfragment setzt die bedingte Anweisung im Wiederholungsblock um. 2

2c
```
01:   lda 101
02:   dec
03:   jaz   27    //wenn k=1
04:   ldai  0
05:   sta   102   //a=0
06:   ldai  1
07:   sta   103   //t=1
08:   lda   103
09:   inc
10:   sta   103   //t=t+1
11:   lda   101
12:   mod   103   //k%t
13:   jap   16    //t kein Teiler von k
14:   ldai  1     //t Teiler von k
15:   sta   102   //a=1
16:   lda   102
17:   jaz   08    //solange a=0
18:   lda   103
19:   sub   101
20:   jaz   24    //wenn k=t
21:   ldai  0
22:   sta   104   //erg=0
23:   jmp   26
24:   ldai  1
25:   sta   104   //erg=1
26:   jmp   29
27:   ldai  0
28:   sta   104
29:   end
```

10

3a > mult(4,10) ⇒Rekursionsaufruf Zeile 6
> mult(8,5) ⇒Rekursionsaufruf Zeile 8
> 8 + mult(16,2) ⇒Rekursionsaufruf Zeile 6
> 8 + mult(32,1) ⇒Abbruch (Zeile 3)
> 8 + 32
> 40

3

3b Die Abbruchbedingung der Methode lautet wenn b gleich 1; in jedem anderen Fall wird in einem rekursiven Aufruf der Wert von b halbiert. Ist also der Wert von b bereits beim initialen Aufruf kleiner als 1, so wird die Abbruchbedingung niemals wahr und die Methode terminiert nie.

2

3c i) mult(2,3) = 2 + mult(4, 1) = 2 + 4 ⇒ein rekursiver Aufruf
mult(2,1024) = mult(4,512) = mult(8,256) = mult(16,128) = mult(32,64) = mult(64,32) = mult(128,16) = mult(256,8) = mult(512,4) = mult(1024,2) = mult(2048,1) = 2048
⇒10 rekursive Aufrufe

7

ii) Da im Algorithmus nur der Wert von b für die Terminierung entscheidend ist und b bei jedem Rekursionsaufruf halbiert wird, hat der Algorithmus logarithmische Laufzeitkomplexität. Da a, wie bereits gesagt, nicht für die Terminierung des Algorithmus relevant ist, gilt die gleiche Laufzeitkomplexität auch für mult(a,b) mit a > 2.

3d Wie bereits in Aufgabenteil cii erwähnt, hängt die Rekursionstiefe, d.h. die Anzahl der notwendigen Rekursionsaufrufe, nur von b und nicht von a ab. Weiterhin gilt für die Multiplikation das Kommutativgesetz, sodass 2×2017 = 2017×2. Wird also b mit dem größeren der beiden Faktoren belegt, wie hier 2017, kommt es zu unnötig vielen Rekursionsaufrufen. Die Funktion könnte optimiert werden, indem zu Beginn sichergestellt wird, dass b den kleineren Faktor enthält und die Faktoren falls nötig getauscht werden.

3

40