



 **Artemis** 6.4.0

 Course Overview




 ge64baw ▾

Courses > Praktikum: Grundlagen der Programmierung WS22/23 > Exercises > W04H01 - Rekursive Pingulogie


 **W04H01 - Rekursive Pingulogie**


**Hausaufgabe** **Easy**

Submission due: **9 months ago**

Points: 8 of 8    Assessment: manual 

Complaint due: **8 months ago**

 Resume practice in exercise

 100% (9 months ago) **GRADED**

Recent results:



Show all results ▾

Tasks:

## Rekursive Pingulogie

Die Forschuine der PUM (Pingu Universität München) haben vor kurzem große Fortschritte in ihrer Forschung gemacht. Einige ihrer Thesen müssen aber noch überprüft werden und dafür brauchen sie - natürlich - deine Hilfe! Da es sich bei der PUM aber um eine Exzellenzuniversität handelt, haben die Forschuine einige Anforderungen an dich, die du unter den einzelnen Aufgaben findest.

**Update 10.11.** Kritische Fehler in der Aufgabenstellung zu Aufgabe 4 wurden behoben (Sorry an alle Early-Solver und vielen Dank für die Hinweise). Außerdem wurden Hinweise zu Public Tests und Exception-Feedback hinzugefügt.


**Update 11.11.** Fehlerhafte Tests für Aufgabe 1 wurden durch korrekte Versionen ersetzt.

**Update 14.11.** Keine negativen Eingaben für PinguDNS

### Allgemeine Anforderungen

- Jede Teilaufgabe **muss** rekursiv gelöst werden. Insbesondere sind Schleifen in der gesamten Aufgabe verboten und eine Verwendung dieser führt zu einer Bewertung mit 0 Punkten bei den entsprechenden Teilaufgaben. Das Gleiche gilt für Streams (wer diese schon kennt).
- Private Hilfsmethoden, sowie (statische) Hilfsvariablen sind erlaubt. Die Methoden müssen aber ohne zusätzliche Aufrufe funktionieren (also so, wie sie in der **main**-Methode verwendet werden). Außerdem gilt zu beachten, dass die Tests mehrere Methodenaufrufe hintereinander ausführen und statische Hilfsvariablen dazwischen **nicht** zurückgesetzt werden.
- Die Signatur der vorgegebenen Methoden (Name, Rückgabetyt, Parametertypen) darf **nicht** verändert werden.

### Aufgaben

 **Aufgabe 1 - Rekursive Pinguin Reihe** No results

Als Erstes musst du die neuste Pinguin Reihe implementieren. Diese ist abhängig von einem Start-Vektor  $p = (p_0, p_1, p_2)$  und folgt dieser Berechnungsvorschrift:

$$\text{pinguSequenceRec}(0) = p_0$$
$$\text{pinguSequenceRec}(1) = p_1$$
$$\text{pinguSequenceRec}(2) = p_2$$
$$\text{pinguSequenceRec}(n) = 2 * \text{pinguSequenceRec}(-n), \text{ falls } n < 0$$
$$\text{pinguSequenceRec}(n) = \text{pinguSequenceRec}(n - 1) - \text{pinguSequenceRec}(n - 2) + 2 * \text{pinguSequenceRec}(n - 3), \text{ sonst}$$

Für deine Implementierung geben dir die Forschuine den Vektor komponentenweise mit, sodass deine Methode insgesamt vier Parameter übernimmt. Außerdem gelten folgende Einschränkungen:

- $-122 < n < 145$
- Die Berechnung muss auf einem durchschnittlichen PC für jede Eingabe in unter einer Sekunde terminieren. Für größere  $n$  ( $\sim > 30$ ) musst du dir also etwas einfallen lassen.
- Bei getesteten Kombinationen aus  $n$  und  $p$  wird es beim Ergebnis nicht zu long-Overflows kommen.

▼ Beispiel für `pinguSequenceRec` mit  $p = (1, 1, 2)$ . (**main**-Methode mit `testTask = 1` ausführen)

## Task 1 example output

0: 1  
1: 1  
2: 2  
3: 3  
4: 3  
5: 4  
6: 7  
7: 9  
8: 10  
9: 15  
10: 23  
11: 28  
12: 35  
13: 53  
14: 74  
15: 91  
16: 123  
17: 180  
18: 239  
19: 305  
20: 426  
21: 599  
22: 783  
23: 1036  
24: 1451  
25: 1981  
26: 2602  
27: 3523  
28: 4883  
29: 6564  
30: 8727  
31: 11929  
32: 16330  
33: 21855  
34: 29383  
35: 40188  
36: 54515  
37: 73093  
38: 98954  
39: 134891  
40: 182123  
41: 245140  
42: 332799  
43: 451905  
44: 609386  
45: 823079  
46: 1117503  
47: 1513196  
48: 2041851  
49: 2763661  
50: 3748202  
51: 5068243  
52: 6847363  
53: 9275524  
54: 12564647  
55: 16983849  
56: 22970250  
57: 31115695  
58: 42113143  
59: 56937948  
60: 77056195  
61: 104344533  
62: 141164234  
63: 190932091  
64: 258456923  
65: 349853300  
66: 473260559  
67: 640321105  
68: 866767146  
69: 1172967159  
70: 1586842223  
71: 2147409356  
72: 2906501451  
73: 3932776541  
74: 5321093802  
75: 7201320163  
76: 9745779443  
77: 13186646884  
78: 17843507767

79: 24148419769  
80: 32678205770  
81: 44216801535  
82: 59835435303  
83: 80975045308  
84: 109573213075  
85: 148269038373  
86: 200645915914  
87: 271523303691  
88: 367415464523  
89: 497183992660  
90: 672815135519  
91: 910462071905  
92: 1232014921706  
93: 1667183120839  
94: 2256092342943  
95: 3052939065516  
96: 4131212964251  
97: 5590458584621  
98: 7565123751402  
99: 10237091095283  
100: 13852884513123  
101: 18746040920644  
102: 25367338598087  
103: 34327066703689  
104: 46451809946890  
105: 62859420439375  
106: 85061743899863  
107: 115105943354268  
108: 155763040333155  
109: 210780584778613  
110: 285229431153994  
111: 385974927041691  
112: 522306665444923  
113: 706790600711220  
114: 956433789349679  
115: 1294256519528305  
116: 1751403931601066  
117: 2370014990772119  
118: 3207124098227663  
119: 4339916970657676  
120: 5872822853974251  
121: 7947154079771901  
122: 10754165167113002  
123: 14552656795289603  
124: 19692799787720403  
125: 26648473326656804  
126: 36060987129515607  
127: 48798113378299609  
128: 66034072902097610  
129: 89357933782829215  
130: 120920087637330823  
131: 163630299658696828  
132: 221426079587024435  
133: 299635955202989253  
134: 405470474933358474  
135: 548686678904418091  
136: 742488114377038123  
137: 1004742385339336980  
138: 1359627628771135039  
139: 1839861472185874305  
140: 2489718614093413226  
141: 3369112399449808999  
142: 4559116729728144383  
143: 6169441558465161836  
144: 8348549627636635451

**Aufgabe 2 - Pingu Zwillinge** No results

Als Nächstes musst du für die Forschuine die Zwillingreihe `pinguF` und `pinguM` implementieren, die sich wie folgt berechnet:

$$\text{pinguF}(0) = 1$$
$$\text{pinguF}(n) = n - \text{pinguM}(\text{pinguF}(n - 1)), \text{ falls } n > 0$$
$$\text{pinguM}(0) = 0$$
$$\text{pinguM}(n) = n - \text{pinguF}(\text{pinguM}(n - 1)), \text{ falls } n > 0$$

Die Forschuine verichern dir außerdem, dass die Eingabe `n` niemals negativ sein wird und beim Ergebnis zu keinem `int`-Overflow führen wird.

▼ Beispiel für `pinguF` und `pinguM` mit  $n = 0 \dots 9$ . (`main`-Methode mit `testTask = 2` ausführen)

```
Task 2 example output
pinguF: 1, 1, 2, 2, 3, 3, 4, 5, 5, 6,
pingM: 0, 0, 1, 2, 2, 3, 4, 4, 5, 6,
```

**Aufgabe 3 - Der Pingu Code** No results

Im Land der Pinguine gilt der "Pingu Code" als eins der am schwersten zu verstehenden Geheimnisse. Die Forscher der PUM haben jüngst einen Ansatz gefunden, diesen Code zu berechnen.

Das Ergebnis ist von zwei Eingabezahlen  $n$  und  $m$  abhängig und lässt sich außerdem rekursiv unter Beachtung des Zwischenergebnisses berechnen. Anfangs ist das Zwischenergebnis natürlich 0. Falls  $n = 0$  gilt, so ist das Endergebnis die Summe aus  $m$  und dem Zwischenergebnis. Ansonsten muss unterschieden werden, ob die Summe aus  $n$  und dem Zwischenergebnis gerade ist oder nicht. Falls es gerade ist, wird rekursiv weitergerechnet, wobei das neue  $n$  den Wert von  $m$  erhält und das neue  $m$  die Hälfte des alten  $n$  (abgerundet). Außerdem werden  $n$ -halbe (altes  $n$ , abgerundet) zum Zwischenergebnis addiert. Falls obige Summe ungerade ist, wird auch rekursiv weitergerechnet. In diesem Fall wird jedoch  $m$  zum Zwischenergebnis addiert,  $n$  um eins reduziert und  $m$  halbiert (abgerundet).

Auch bei dieser Aufgabe werden nur nicht-negative Eingaben verwendet und es wird nicht zu `int`-Overflows kommen. "Abgerundet" meint in dieser Aufgabe das normal Java-Rundungsverhalten (Richtung 0).

► Beispiel für `pinguCode` mit  $n = 0 \dots 9, m = 0 \dots 9$ . (`main`-Methode mit `testTask = 3` ausführen)

**Aufgabe 4 - Pingu DNS** No results

Zu guter Letzt musst du den Forschuinen noch helfen, die DNS eines Pinguins zu bestimmen, bei dem nur die Pingu-Werte der Eltern ( $f$  und  $m$ ) bekannt sind. Für die rekursive Berechnung ist vor allem die binäre Darstellung dieser Werte relevant, die dir in Form eines (nicht negativen) `int` übergeben wird. Zur Bestimmung des DNA-Strings haben die Forscher folgende Prozedur gefunden:

Sollten die Pingu-Werte beider Eltern 0 sein, so ist die resultierende DNS der leere String "".

Falls nur  $f = 0$  ist, so wird für jedes Bit von  $m$  abzüglich führender 0er-Bits ein "A" angehängt, im umgekehrten Fall (nur  $m = 0$ ) ein "T". `pinguDNA(0, 2)` gibt also "AA" zurück, da  $2 = 10b$  entspricht. Für `pinguDNA(10, 0)` ist die Lösung "TTTT", da  $10 = 1010b$  ist.

Interessant wird es, falls beide Pingu-Werte nicht 0 sind. In diesem Fall muss zuerst unterschieden werden, ob  $f$  und  $m$  mit dem selben Bitwert enden (Tipp: gerade/ungerade).

Falls ja hängt das Basenpaar davon ab, ob  $f > m$  ("GT"),  $f < m$  ("GA") oder  $f = m$  ("GC") gilt.

Falls nein, ist es relevant, welcher Pingu-Wert mit einem 1-er Bit endet. Sollte  $f$  so enden, wird "TC" angehängt, sonst "AC".

Jedes Bit wird für genau eine Nukleinbase im Ausgabestring verwendet und kann daher für die Rekursion verworfen werden (Tipp: Division durch 2).

Ein Schritt-für-Schritt Beispiel findest du, wenn du das folgende Beispiel aufklappst.

▼ Beispiel für `pinguDNA` mit Erklärung. (`main`-Methode mit `testTask = 4` ausführen)

```
pinguDNA(21, 25) = GCACTCGAGA
```

21 in binär ist "10101b"  
25 in binär ist "11001b"

- 1.  $21 \neq 0 \neq 25$ , beide enden mit 1er Bit und  $21 < 25 \Rightarrow$  "GA"
- 2. Bits entfernen  $\Rightarrow f = "1010b" = 10, m = "1100b" = 12$
- 3.  $f, m \neq 0$ , enden mit 0er Bit,  $f$  "GA"  $\Rightarrow$  insgesamt "GAGA"
- 4. Bits entfernen  $\Rightarrow f = "101b" = 5, m = "110b" = 6$
- 5.  $f, m \neq 0$ , enden mit unterschiedlichem Bitwert,  $m$  ist durch 2 teilbar  $\Rightarrow$  "TC"  $\Rightarrow$  insgesamt "TCGAGA"
- 6. Bits entfernen  $\Rightarrow f = "10b" = 2, m = "11b" = 3$
- 7.  $f, m \neq 0$  enden mit unterschiedlichem Bitwert,  $f$  ist durch 2 teilbar  $\Rightarrow$  "AC"  $\Rightarrow$  insgesamt "ACTCGAGA"
- 8. Bits entfernen  $\Rightarrow f = "1b" = 1, m = "1b" = 1$
- 9.  $f, m \neq 0$ , enden mit 1er Bit und sind gleich  $\Rightarrow$  "GC"  $\Rightarrow$  insgesamt "GCACTCGAGA"
- 10. Bits Entfernen  $\Rightarrow f = 0, m = 0 \Rightarrow$  "", fertig  $\Rightarrow$  insgesamt "GCACTCGAGA"

Hinweise

Exercise details

Release date:	Nov 10, 2022 18:30
Submission due:	Nov 20, 2022 18:00
Assessment due:	Nov 27, 2022 18:00
Complaint due:	Dec 4, 2022 18:00

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have 998 complaints left. ⓘ

How useful is this feedback to you?



About

[Request change](#)

[Release notes](#)

[Privacy Statement](#)

[Imprint](#)