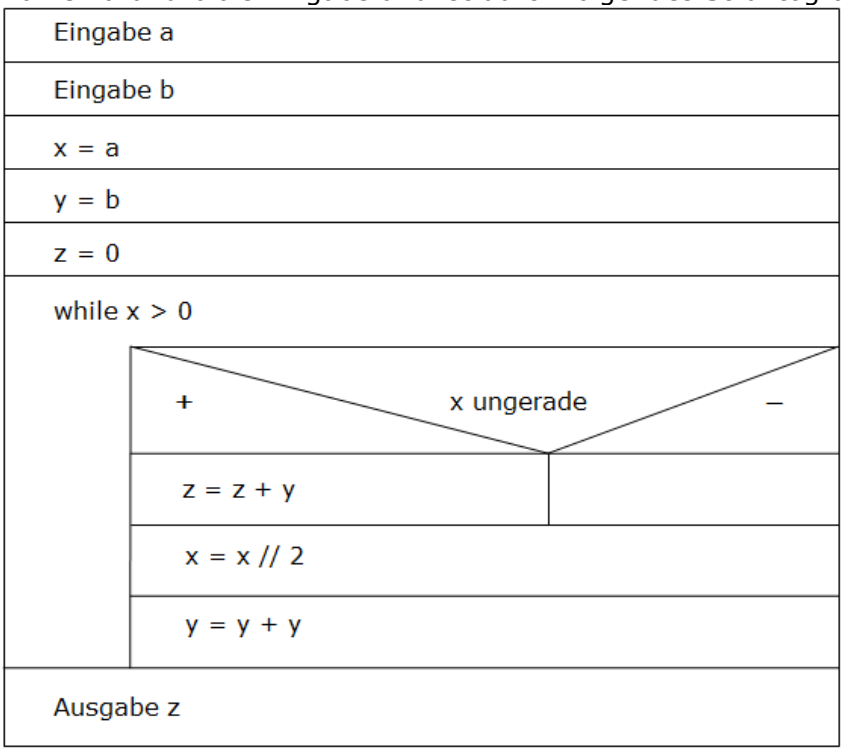
Aufgabe 18

Der Algorithmus **ÄGYPTISCHE MULTIPLIKATION** verlangt die nicht-negativen ganzen Zahlen **a** und **b** als Eingabe und ist durch folgendes Struktogramm gegeben:



Hinweis: *Unter* **`x // y`** *verstehen wir den ganzzahligen Quotient bei der Division von x durch y; der Rest wird in Python mit* **`x % y`** *berechnet.*

a) Bestätige anhand der Trace-Tabelle aus der Lösung von Aufgabe 4 der Kursarbeit: Die Beziehung

$$a \cdot b = x \cdot y + z$$

ist offensichtlich **Schleifeninvariante**, d. h. diese Beziehung ist vor und nach jedem Schleifendurchlauf erfüllt und damit invariant gegenüber Schleifendurchläufen (Auf den strengen Beweis verzichten wir hier; hierzu bedarf es des Beweisverfahrens „Vollständige Induktion".).

b) Begründe, daß der Algorithmus für jede zulässige Eingabe terminiert; zeige, daß bei Terminierung mit **z** das Produkt der eingegebenen Zahlen **a** und **b** ausgegeben wird.

c) Codiere und teste den Algorithmus in Python.

Aufgabe 19 Fibonacci-Folge

Für n ∈ {0, 1, 2, 3, ... } läßt sich die Fibonacci-Folge rekursiv definieren:

Rekursionsanfang: **fibo(0) = 0**
fibo(1) = 1

Rekursionsvorschrift: **fibo(n) = fibo(n–1) + fibo(n–2)** falls n > 1

(In Worten: für n > 1 erhält man das n-te Folgenglied als Summe der beiden vorangehenden Folgenglieder.)

a) Schreibe und teste ein Python-Programm mit rekursivem Funktionsaufruf, welches nach Eingabe von **n** den Wert **fibo(n)** ausgibt (oder: alle Werte fibo(0), . . . , fibo(n)); implementiere auch eine Variable **z**, welche die Anzahl der Funktionsaufrufe ermittelt und ausgibt.

Bestimme auch den Zeitbedarf, den die Berechnung von **fibo(n)** erfordert.

Bemerkung: Hier handelt es sich um einen Algorithmus mit exponentieller Komplexität, denn die Anzahl z der Funktionsaufrufe wächst exponentiell mit n; bei n = 38, 39, 40, . . nimmt die Berechnung bereits sehr viel Zeit in Anspruch.

b) Wenn man **`lru_cache`** des Python-Moduls **`functools`** nutzt, läßt sich die Laufzeit erheblich verbessern (hier werden bereits berechnete Werte in einem cache zwischengespeichert); allerdings kommt man mit **`lru_cache`** bei der Berechnung der Ackermann-Funktion (Aufgabe 20) wegen derer ungeheuren Rekursionstiefe kaum weiter: acker(3,9) läßt sich noch berechnen, bei acker(3,10) oder acker(4,n), n>0, ist Schluß.

```
from functools import lru_cache

n = int(input('n = '))
z = 0

@lru_cache(maxsize=64)
def fibo(n):
      . . . . . .
      . . . . . .
```

c) Schreibe und teste ein imperativ formuliertes Python-Programm, z. B. indem die Werte der Fibonacci-Folge in einem array (also einer Liste a) mit den Komponenten a[0], a[1], ... , a[n] abgelegt werden (setze a[0] = 0 und a[1] = 1). Vergleiche die Laufzeit mit dem funktional formulierten Algorithmus aus a).

Aufgabe 20 Ackermann-Funktion

Für m, n ∈ {0, 1, 2, 3, ... } ist die Ackermann-Funktion **f** wie folgt definiert:

Rekursionsanfang:	(1)	**f(0,n) = n+1**
Rekursionsvorschrift:	(2)	**f(m,0) = f(m-1, 1)**
	(3)	**f(m,n) = f(m-1, f(m,n-1))**

a) Man erhält: f(0,0) = 1, f(0,1) = 2, f(0,2) = 3, f(1,0) = f(0,1) = 2
Berechne f(2,0); f(1,1); f(1,2); f(3,0).

b) Schreibe den Algorithmus zur Berechnung der Ackermann-Funktion als Python-Programm mit rekursivem Funktionsaufruf.
Implementiere eine Zählvariable z, um die Anzahl der Funktionsaufrufe bestimmen; ermittle auch den Zeitbedarf zur Laufzeit.

Berechne f(3,7); f(3,8); f(4,0); f(3,8); f(3,9); f(4,1); f(4,2)

Bemerkung:
Die Ackermann-Funktion ist eine berechenbare Funktion, allerdings übersteigt deren ungeheure Rekursionstiefe sehr schnell die Möglichkeiten eines jeden auch noch so leistungsfähigen Computers!

Anhang: Implementierung einer Zählvariablen **z** am Beispiel Fakultät

```
z = 0
n = int(input('n = '))
def factorial(x):
  global z
  z += 1
  if x == 0: return 1
  else: return x * factorial(x - 1)
print (n,'! = ',factorial(n))
print ('# Aufrufe:',z)
```

```
z = 0
n = int(input('n = '))
if n == 0 or n == 1: fact = 1
else:
   fact = 1
   i = 2
   while i <= n:
        z = z + 1
        fact = fact * i
        i = i + 1
print(n,'! = ',fact)
print('# Schleifendurchläufe:',z)
```