Mohamad Raafat Baki

Übung 1 – Algorithmen:

1.1. Schreiben Sie einen Algorithmus „absdiff“ in Pseudocode, der zu zwei gegebenen natürlichen Zahlen a und b den absoluten Betrag ihrer Differenz berechnet. Machen Sie dabei eine Fallunterscheidung.

func absdiff(a,b) var d = 0 if a >= b d = a - b else d = b - a return dprint absdiff(9,8)

1.2. Schreiben Sie einen Algorithmus „odd“ in Pseudocode, der zu einer gegebenen natürlichen Zahl a das Ergebnis 1 berechnet, wenn diese ungerade ist und 0 sonst. Ziehen Sie dazu von der Zahl wiederholt 2 ab, solange bis das Ergebnis feststeht.

func odd(a)

while a > 1

a = a - 2

return a

print odd(51)

1.3. Schreiben Sie einen Algorithmus "max3" in Pseudocode, der bei der Eingabe von drei natürlichen Zahlen a, b, und c die größte der drei Zahlen zurückgibt. Testen Sie den Algorithmus, indem Sie für alle möglichen Fälle von Eingaben jeweils ein Testbeispiel angeben und die Ausgabe ihres Algorithmus überprüfen. Verwenden Sie keine Arrays!

func max3(a,b,) if ( a >= b ) && ( a >= c ) return a if ( b >= a ) && ( b >= c ) return b else return cprint max3(1,2,3)

1.4. Wie würden Sie vorgehen, wenn Sie Algorithmen "max6" und "max9" schreiben sollten, die bei der Eingabe von sechs bzw. neun natürlichen Zahlen die größte der Zahlen zurückgibt? Wie können Sie überzeugend argumentieren, dass ihre Algorithmen für alle denkbaren Eingaben die richtigen Ergebnisse liefern? Verwenden Sie keine Arrays!

func max6(a,b,c,d,e,f)

var temp = a

if temp < b

temp = b

if temp < c

temp = c

if temp < d

temp = d

if temp < e

temp = e

if temp < f

temp = f

return temp

print max6(1,2,3,4,5,6)

1.5. Betrachten Sie den folgenden Pseudocode "quad". Was sind die Ergebnisse für die Eingaben 0, 1, 2, 3 und 4? Was vermuten Sie, berechnet dieser Pseudocode im Allgemeinen? Können Sie erklären, warum er dies korrekt tut? Finden Sie einen einfacheren Algorithmus, der dasselbe berechnet!

func quad(n)

var a = n\*n

return a

print quad(6)

1.6. Euklidischer Algorithmus (größter gemeinsamer Teiler)

nach Wikipedia: "Euklid berechnete den größten gemeinsamen Teiler, indem er nach einem gemeinsamen „Maß“ für die Längen zweier Linien suchte. Dazu zog er wiederholt die kleinere der beiden Längen von der größeren ab. Dabei nutzt er aus, dass sich der größte gemeinsame Teiler zweier Zahlen (oder Längen) nicht ändert, wenn man die kleinere von der größeren abzieht."

1.6.1. Probieren Sie den Euklidischen Algorithmus (in der obigen Textform) mit den zwei Zahlen (=Streckenlängen) 44 und 12 als Eingabe. Was ist das Ergebnis?

1.6.2. Überführen Sie den Euklidischen Algorithmus von der Textform nach Pseudocode.

func ggt(a,b)

while a != b

if a > b

a = a - b

else

b = b - a

return a

print ggt(44,12)

1.7. Beschreiben Sie einen Algorithmus "mod" (Modulo) in Pseudocode, der bei Eingabe zweier natürlicher Zahlen a und b den Rest ausgibt, der bei der ganzzahligen Division von a durch b bleibt. Benutzen Sie dabei wiederholte Subtraktionen und Vergleiche, aber keine Multiplikation oder Division. Zum Beispiel ist: mod(12,3) = 0, mod(23,3) = 2 und mod(17,4) = 1.

func Modulo(a,b)

while a >= b

a = a - b

return a

print Modulo(25,3)

1.8. Ändern (Erweitern) Sie den Algorithmus "mod" so zu einem neuen Pseudocode Algorithmus "div", der bei Eingabe von a und b das Divisionsergebnis der ganzzahligen Division von a geteilt durch b berechnet. Benutzen Sie wiederum keine Multiplikation oder Division.

func div(a,b)

var q = 0

while a >= b

a = a - b

q = q + 1

return q

print div(24,3)

1.10. Implementieren Sie einen Algorithmus teilbar(x,y) in Pseudocode, der 1 zurückgibt, wenn die positive ganze Zahl x durch die positive ganze Zahl y teilbar ist und der 0 zurückgibt falls dies nicht der Fall ist. Z.B.: teilbar(10,5) = 1; teilbar(13,3) = 0. Verwenden Sie keine Funktionsaufrufe und bei arithmetischen Berechnungen nur + und –

func teilbar(a,b)

while a >= b

a = a - b

if a == 0

return 1

else

return 0

print teilbar(13,3)

1.11. Implementieren Sie einen Algorithmus teiler(n), der bei Eingabe einer positiven ganzen Zahl n den kleinsten positiven echten Teiler dieser Zahl ausgibt, also teiler(15)=3, teiler(13)=13, Sie können dazu den Algorithmus teilbar benutzen.

func teilbar(a,b)

while a >= b

a = a - b

if a == 0

return 1

else

return 0

func teiler(n)

if n == 1

return 1

for i = 2 to n

if teilbar(n,i) == 1

return i

print teiler(15)

1.12. Implementieren Sie einen Algorithmus is\_primz(n) in Pseudocode, der als Eingabe eine positive ganze Zahl n hat und 1 zurückgibt, falls n eine Primzahl ist und 0 falls n keine Primzahl ist. Sie können die Algorithmen von oben wiederverwenden.

func div(a,b)

while a >= b

a = a - b

return a

func is\_prime(n)

for i = 2 to n-1

if div(n,i) == 0

return 0

return 1

print is\_prime(11)

1.13. Falls Sie für den Algorithmus "teiler" eine for-Schleife benutzt haben: transformieren Sie diese in eine while-Schleife. Falls Sie für diesen Algorithmus eine while-Schleife benutzt haben: transformieren Sie diese in eine for-Schleife.

func div(a,b)

while a >= b

a = a - b

return a

func teilbar(a,b)

while a >= b

a = a - b

if a == 0

return 1

else

return 0

func teiler(n)

var i = 2

while i <= n

if teilbar(n,i) == 1

return i

i=i+1

print teiler(13)

1.14. Implementieren Sie einen Algorithmus "next\_prime", der als Eingabe eine natürliche Zahl n hat und als Ausgabe die erste natürliche Zahl k, die größer als n und eine Primzahl ist. Benutzen Sie dazu "is\_primz".

func div(a,b)

while a >= b

a = a - b

return a

func is\_prime(n)

for i = 2 to n-1

if div(n,i) == 0

return false

return true

func next\_prime(n)

n = n + 1

while ! is\_prime(n)

n = n + 1

return n

print next\_prime(911)

1.15. Welchen Schleifentyp haben Sie für den "next\_prime" Algorithmus verwendet? Könnten Sie den Algorithmus auch mit dem anderen Schleifentyp schreiben? Begründen Sie kurz Ihre Entscheidung!

while schleife wurde verwendet .

for-schleife könnte auch verwendet werden ,dazu braucht man aber auch if-schleife um next\_prime zu bistemmen.

1.16. a) Nennen Sie die vier Anforderungen an einen Algorithmus .

b) Überlegen Sie sich für jede der vier genannten Anforderungen ein konkretes Gegenbeispiel, das diese Anforderung verletzt.

• Allgemeinheit :

– löst potenziell unendlich viele Instanzen eines Problems.

• Ausführbarkeit :

– die auszuführenden Schritte müssen offensichtlich erledigt werden können (d.h. elementare Schritte).

• Endlichkeit :

– die Beschreibung besteht aus endlich vielen Schritten.

• Eindeutigkeit :

– an jeder Stelle ist eindeutig bestimmt (determiniert), was zu tun ist und mit welchem Schritt es weitergeht.