

Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____

Wintersemester 2010/2011
Hinweise zur Bearbeitung der Klausur
zum Kurs 1613 „Einführung in die imperative Programmierung“

Wir begrüßen Sie zur Klausur „Einführung in die imperative Programmierung“. Lesen Sie sich diese Hinweise vollständig und aufmerksam durch, bevor Sie mit der Bearbeitung der Aufgaben beginnen:

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
 - 2 Deckblätter
 - 1 Formblatt für eine Bescheinigung für das Finanzamt,
 - diese Hinweise zur Bearbeitung,
 - 6 Aufgaben (Seite 2-20)
 - die Muss-Regeln des Programmierstils.
2. Füllen Sie, **bevor** Sie mit der Bearbeitung der Aufgaben beginnen, folgende Seiten des Klausurexemplares aus:
 - Beide Deckblätter mit Namen, Anschrift sowie Matrikelnummer. **Markieren Sie vor der Abgabe auf beiden Deckblättern die von Ihnen bearbeiteten Aufgaben.**
 - Falls Sie eine Teilnahmebescheinigung für das Finanzamt wünschen, füllen Sie bitte das entsprechende Formblatt aus und belassen Sie es in der Klausur. Sie erhalten es dann zusammen mit der Korrektur abgestempelt zurück.

Nur wenn Sie die Deckblätter vollständig ausgefüllt haben, können wir Ihre Klausur korrigieren!
3. Schreiben Sie Ihre Lösungen auf den freien Teil der Seite unterhalb der Aufgabe bzw. auf die leeren Folgeseiten. Sollte dies nicht möglich sein, so vermerken Sie, auf welcher Seite die Lösung zu finden ist.

Streichen Sie ungültige Lösungen deutlich durch! (Sollten Sie mehr als eine Lösung zu einer Aufgabe abgeben, so wird nur eine davon korrigiert – und nicht notwendig die bessere.)
4. Schreiben Sie auf jedes von Ihnen beschriebene Blatt oben links Ihren Namen und oben rechts Ihre Matrikelnummer. Wenn Sie weitere eigene Blätter benutzt haben, heften Sie auch diese, mit Namen und Matrikelnummer versehen, an Ihr Klausurexemplar. Nur dann werden auch Lösungen außerhalb Ihres Klausurexemplares gewertet!
5. Neben unbeschriebenem Konzeptpapier und Schreibzeug (Füller oder Kugelschreiber, benutzen Sie **keinen Bleistift** und **keinen Rotstift!**) sind **keine weiteren Hilfsmittel** zugelassen. Die Muss-Regeln des Programmierstils finden Sie im Anschluss an die Aufgabenstellung.
6. Es sind maximal 36 Punkte erreichbar. Sie haben die Klausur sicher dann bestanden, wenn Sie mindestens 18 Punkte haben.

Wir wünschen Ihnen bei der Bearbeitung der Klausur viel Erfolg!

Aufgabe 1 (2+4 Punkte)

Gegeben sei das folgende Programm:

```
program WasPassiert(input,output);  
{  
  _____  
  _____ }  
  var  
    a:integer;  
    b:integer;  
begin  
  writeln('Geben Sie zwei natürliche Zahlen ein: ');  
  readln(a);  
  readln(b);  
  if a>b then  
    begin  
      b:=a+b;  
      a:=b-a;  
      b:=b-a  
    end;  
  while a<b do  
    begin  
      a:=a+1;  
      b:=b-1  
    end;  
  if a=b then  
    writeln('Ergebnis: ',b)  
  else  
    writeln('Ergebnis: ',b, '.5')  
end.
```

Überlegen Sie sich was das Programm leistet und wie es dabei vorgeht.

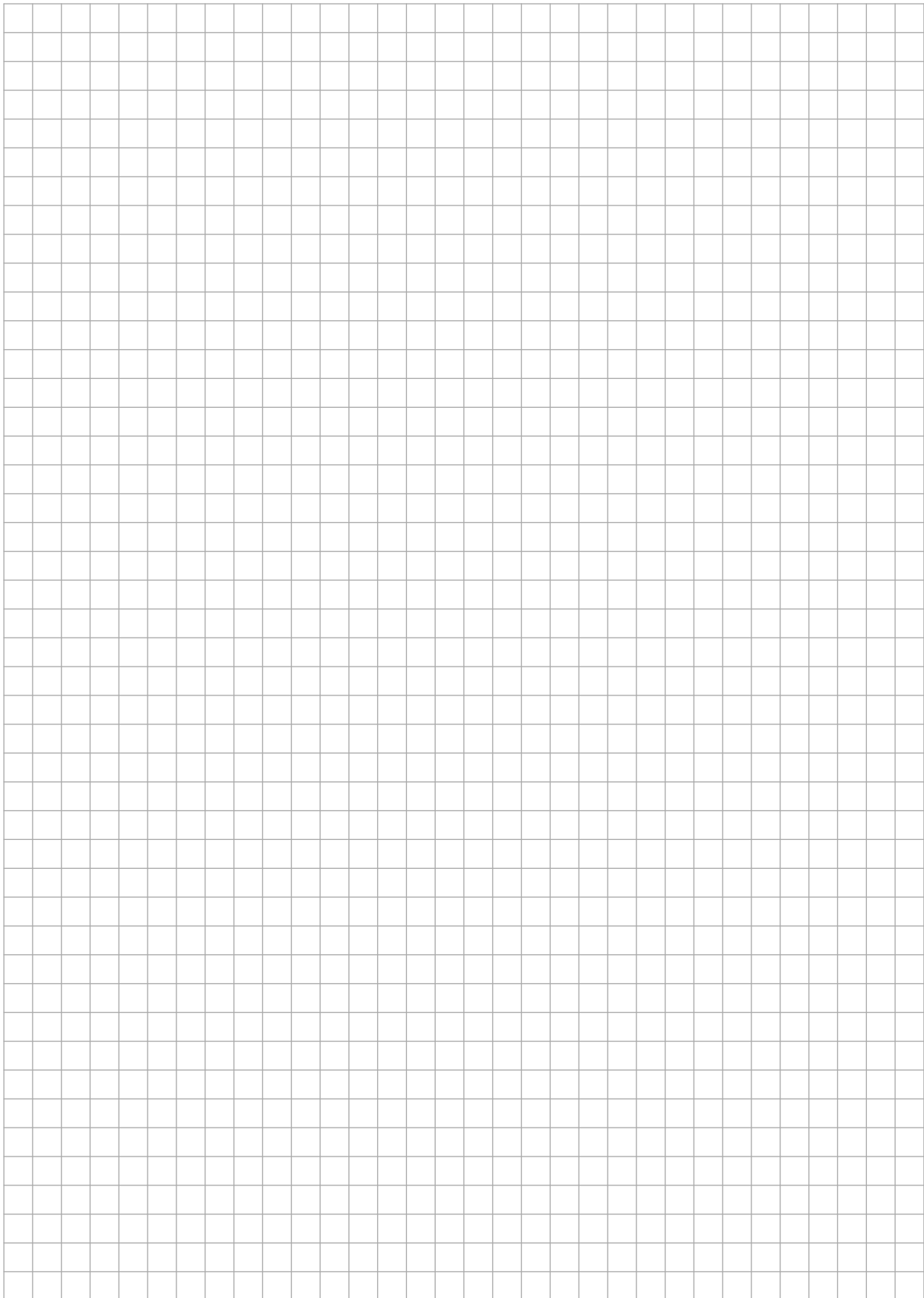
- a) Was gibt das Programm für die Eingaben a=7 und b=2 aus? 'Ergebnis: _____'
Was gibt das Programm für die Eingaben a=3 und b=5 aus? 'Ergebnis: _____'
- b) Ergänzen Sie im Programm einen erklärenden Kommentar an der grau eingefärbten Stelle und Schreiben Sie eine passende Problemspezifikation:

Eingabe: _____
Ausgabe: _____
Nachbedingung: _____

Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____

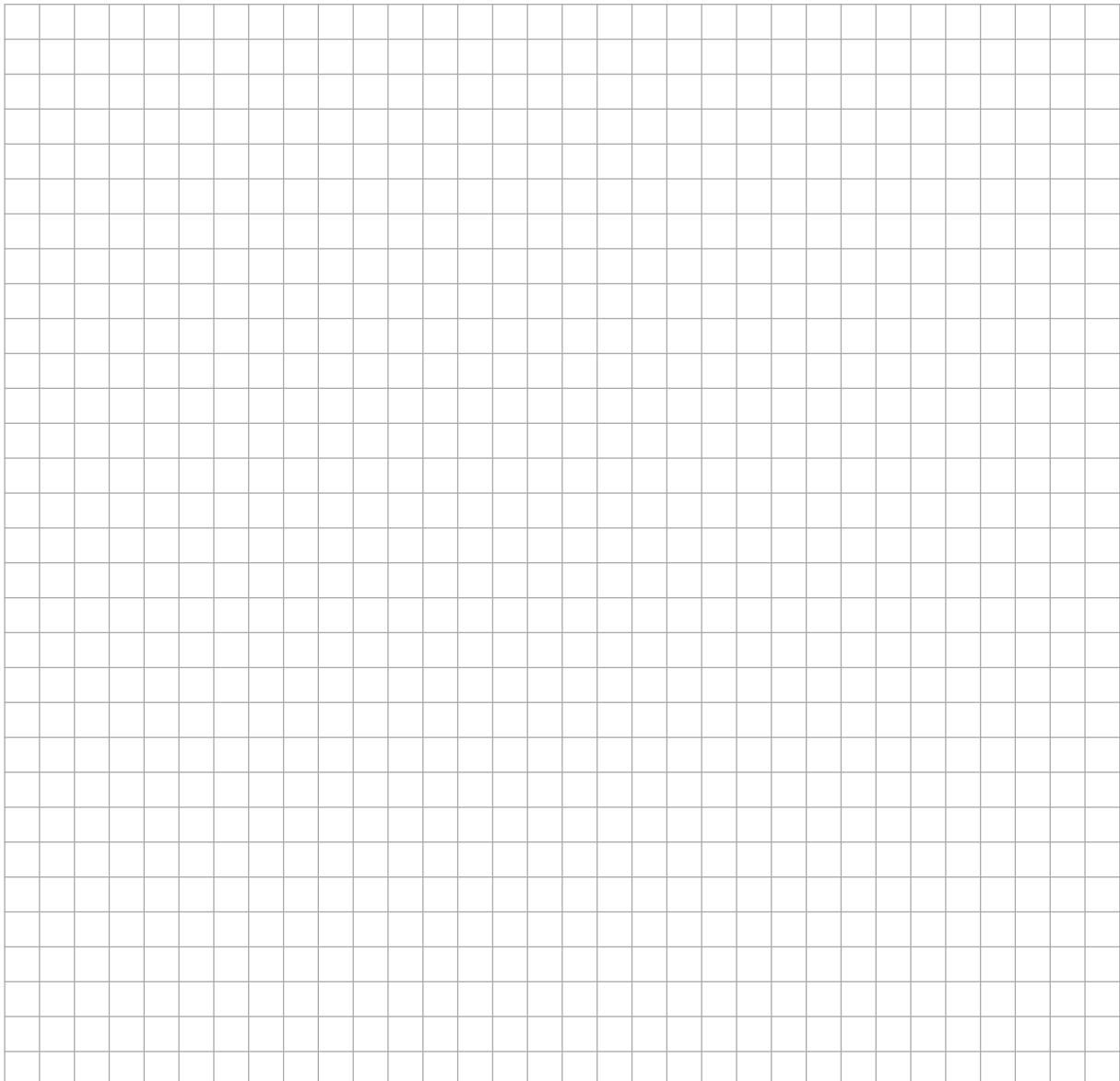


Aufgabe 2 (5 Punkte)

Schreiben Sie ein Programm, das eine natürliche Zahl n einliest und ein Dreieck aus X-Symbolen auf der Standardausgabe erzeugt. Die erste Zeile des Dreiecks soll aus einem X-Symbol, die zweite aus zwei X-Symbolen, usw. bestehen. Insgesamt soll sich das Dreieck aus n solchen Zeilen zusammensetzen.

Für die Zahl $n=4$ erscheint zum Beispiel folgendes Dreieck:

```
X  
XX  
XXX  
XXXX
```



Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____



Aufgabe 3 (6 Punkte)

Für eine endliche Folge von Werten ist die Standardabweichung die Wurzel der durchschnittlichen quadratischen Abweichungen vom Mittelwert der Werte der Folge.

Für Zahlen x_1, \dots, x_n berechnet sich der Mittelwert m als:

$$m = \frac{(x_1 + \dots + x_n)}{n}$$

Damit ergibt sich die Standardabweichung s als:

$$s = \sqrt{\frac{(x_1 - m)^2 + \dots + (x_n - m)^2}{n}}$$

Für die Zahlen 1,2,3,4,5 ist beispielsweise:

$$m = \frac{(1 + 2 + 3 + 4 + 5)}{5} = 3$$

$$s = \sqrt{\frac{(1 - 3)^2 + (2 - 3)^2 + (3 - 3)^2 + (4 - 3)^2 + (5 - 3)^2}{5}} = \sqrt{2} \approx 1.41$$

Gegeben seien folgende Typdefinitionen für ein Feld mit Real-Zahlen:

const

FELDGROESSE=10;

type

tIndex=1..FELDGROESSE;

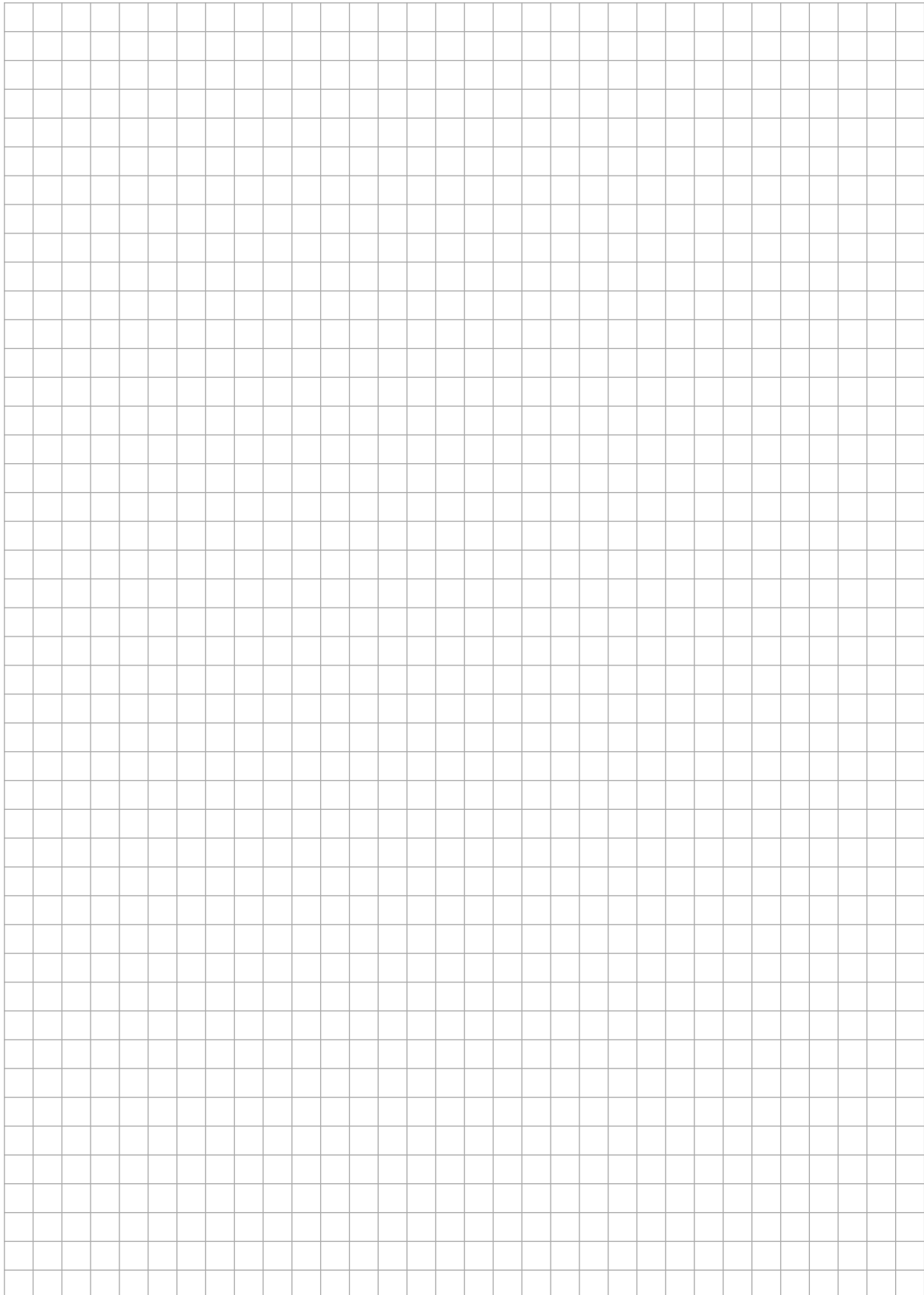
tFeld=array[tIndex] of real;

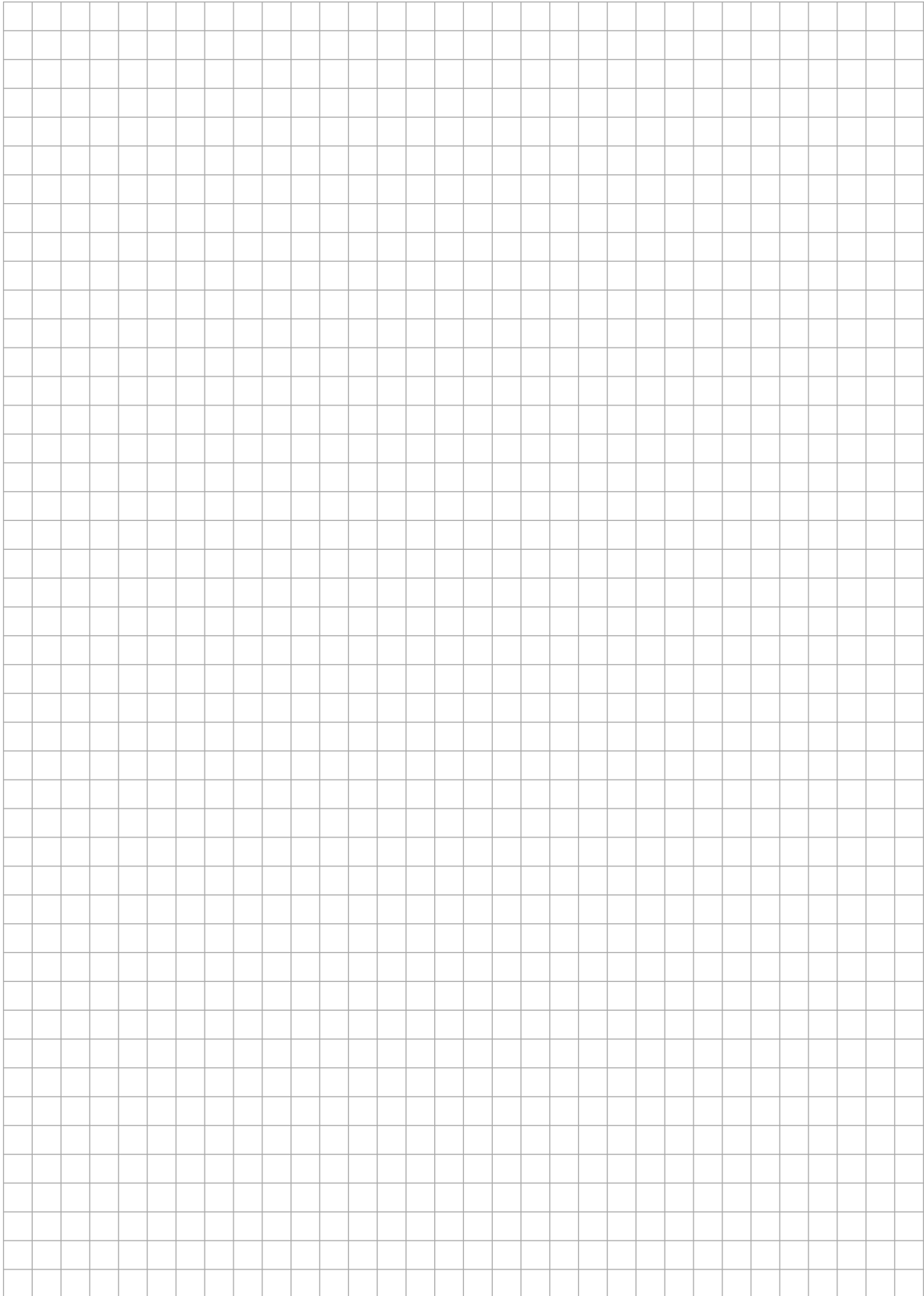
Schreiben Sie eine Funktion, die die Standardabweichung der Elemente eines Feldes vom Typ tFeld berechnet und als Real-Wert zurückgibt.

Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____





Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____



Aufgabe 4 (7 Punkte)

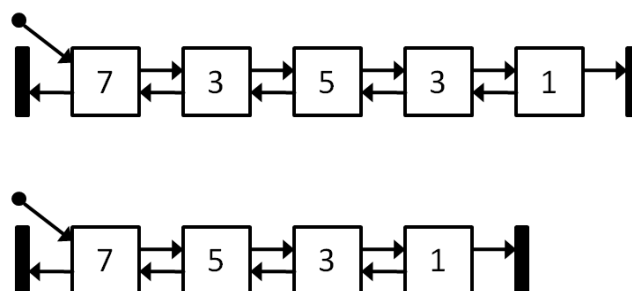
Gegeben sei die folgende Typvereinbarung:

```
type  
tRefDVElement = ^tDVElement;  
tDVElement = record  
    Wert: integer;  
    next, prev: tRefDVElement  
end;
```

Die Typen `tRefDVElement` und `tDVElement` dienen zur Bildung einer doppelt verketteten linearen Liste von Integer-Zahlen. Im Gegensatz zu einer einfach verketteten Liste zeichnet sich eine doppelt verkettete Liste dadurch aus, dass jedes ihrer Elemente nicht nur einen Zeiger `next` auf das nachfolgende, sondern zusätzlich einen Zeiger `prev` auf das vorhergehende Element besitzt. Analog zum Zeiger `next` des letzten Listenelementes zeigt auch der Zeiger `prev` des ersten Listenelementes auf `nil`.

Schreiben Sie eine iterative Prozedur `DVListenElementEntfernen`, die einen Zeiger auf das erste Element einer doppelt verketteten Liste und einen Integer-Wert übergeben bekommt. Das erste Element der Liste mit diesem Wert wird gesucht und aus der Liste gelöscht. Existiert kein Listenelement mit dem angegebenen Wert, wird die Liste nicht verändert.

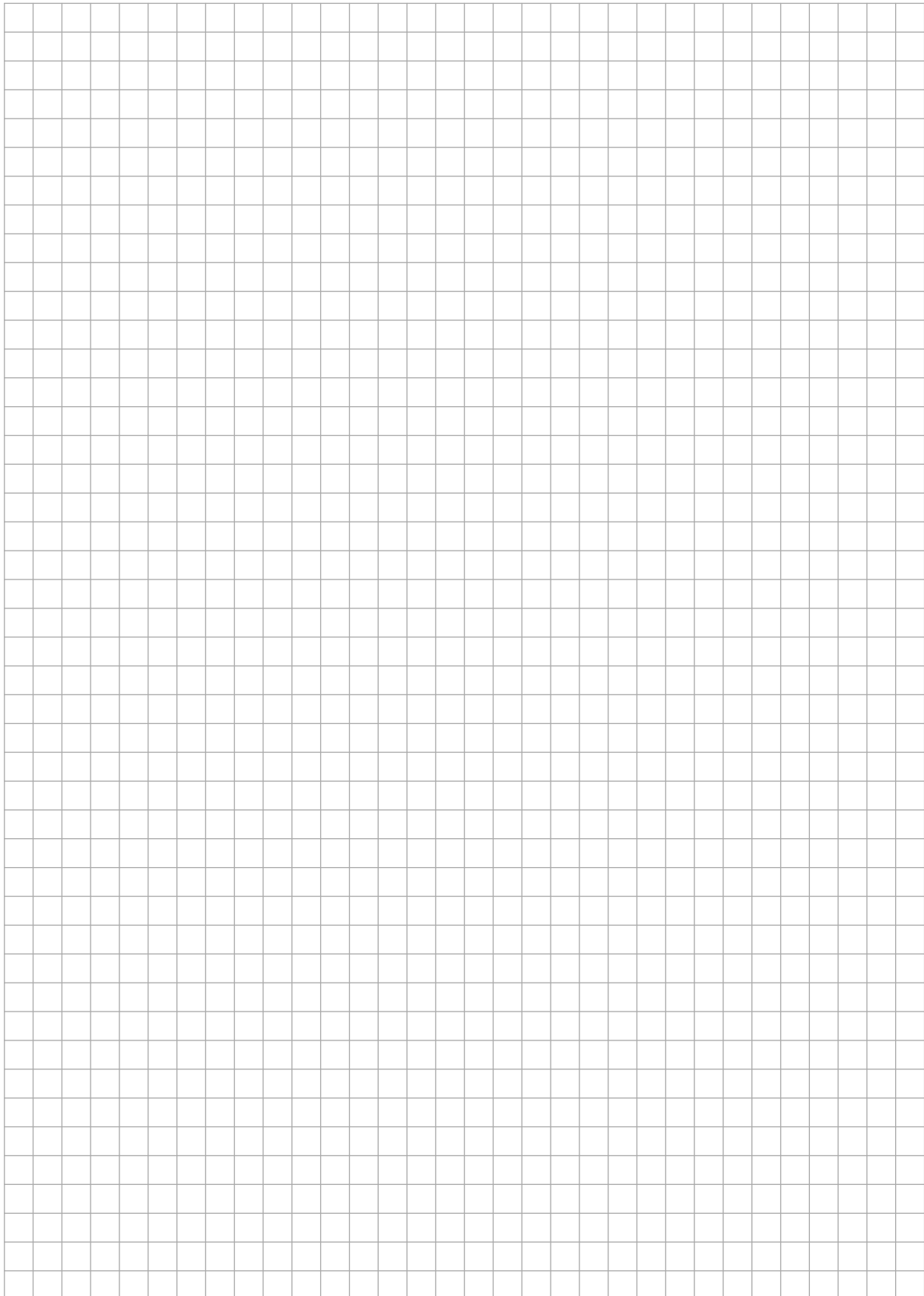
Die folgende Abbildung zeigt ein Beispiel einer doppelt verketteten linearen Liste oben vor und unten nach dem Aufruf der Prozedur `DVListenElementEntfernen` mit dem Integer Wert 3.

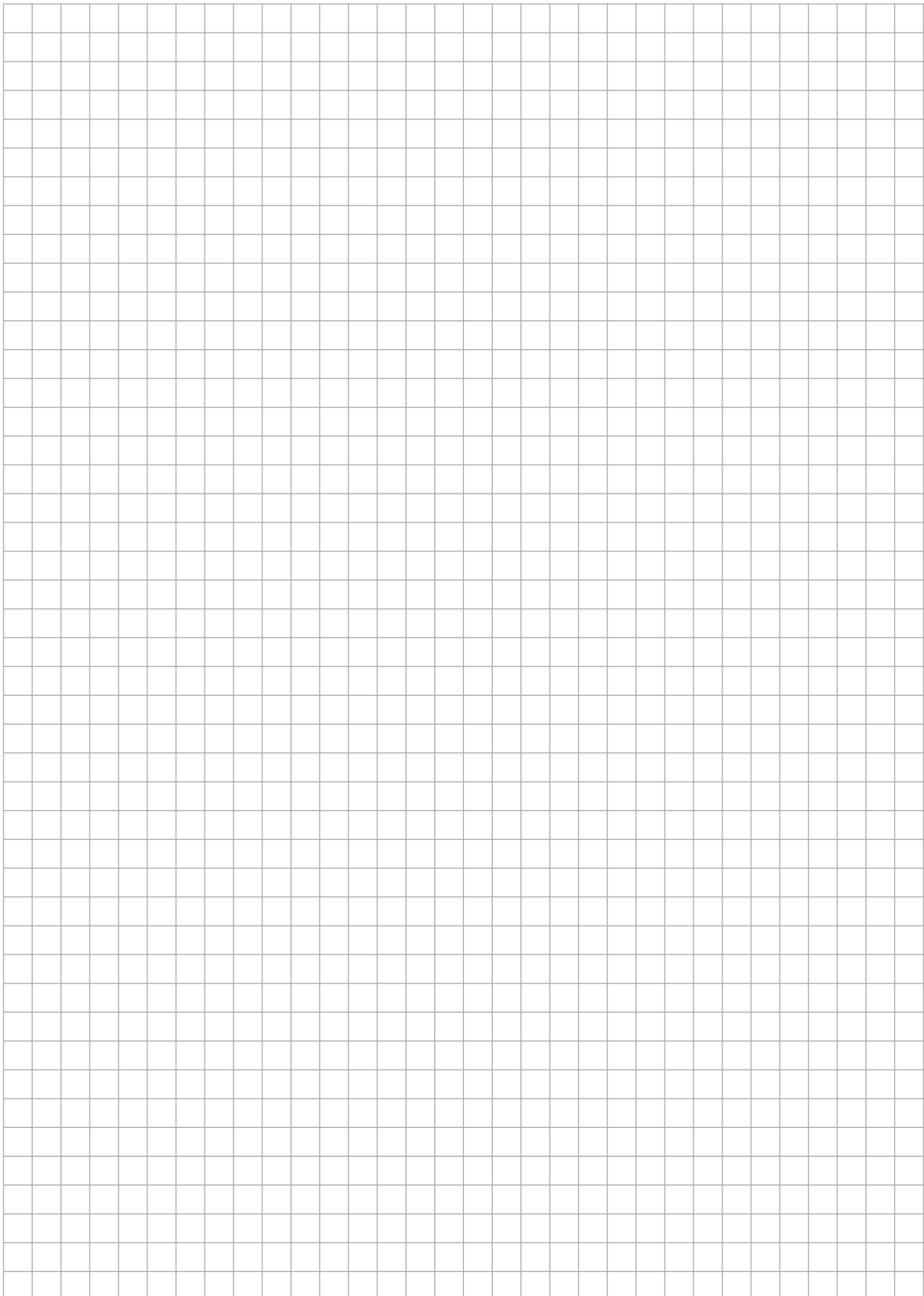


Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____





Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____

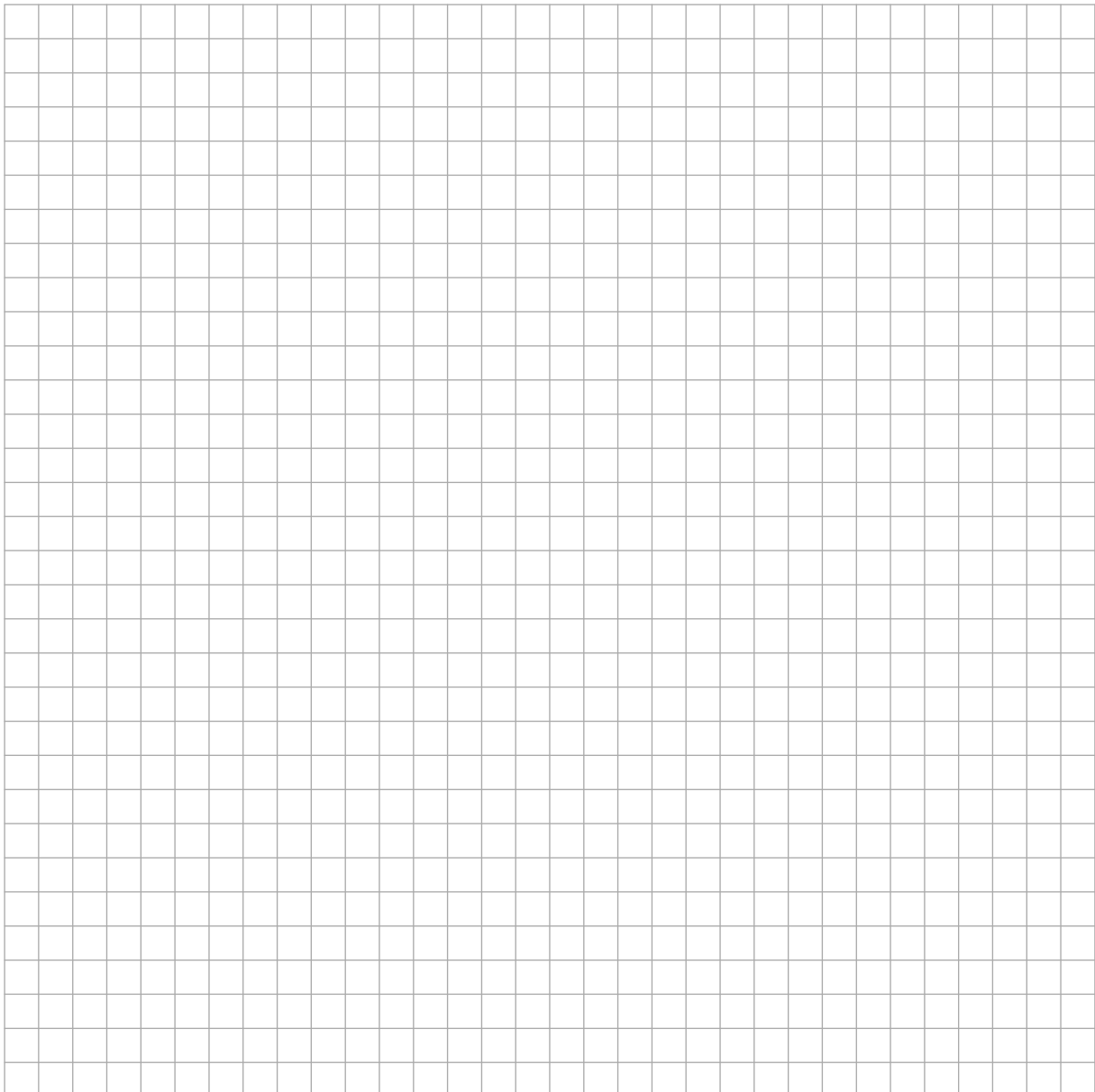


Aufgabe 5 (6 Punkte)

Gegeben seien folgende Typdefinitionen für einen Binärbaum von Integer-Zahlen:

```
type  
tRefBinBaum=^tBinBaum;  
tBinBaum=record  
    Wert:Integer;  
    links,rechts:tRefBinBaum  
end;
```

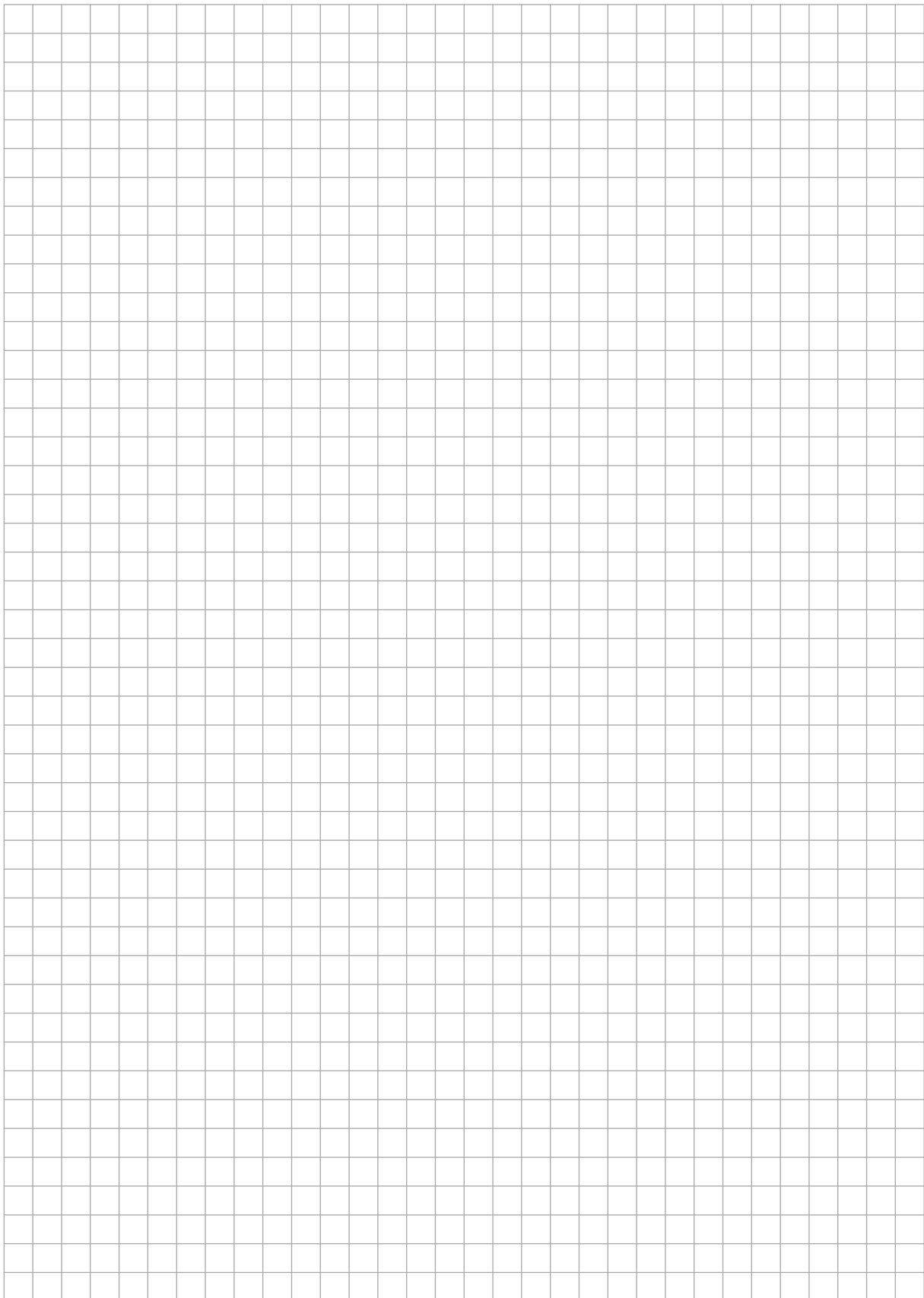
Schreiben Sie eine rekursive Funktion, die die Werte aller Knoten eines Baumes vom Typ tRefBinBaum addiert und das Ergebnis als Integer zurückgibt.



Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____



Aufgabe 6 (6 Punkte)

Zu zwei natürlichen Zahlen a und b existiert eindeutig eine nicht-negative ganze Zahl c und eine ganze Zahl r mit $0 \leq r < b$, so dass gilt:

$$a = c * b + r.$$

Die Zahl r heißt Rest der ganzzahligen Division von a und b . In Zeichen schreiben wir für diesen Rest $a \bmod b$. So ist zum Beispiel

$7 \bmod 3 = 1$ da $7 = 2 * 3 + 1$ gilt,
 $18 \bmod 5 = 3$ da $18 = 3 * 5 + 3$ gilt und
 $12 \bmod 4 = 0$ da $12 = 3 * 4 + 0$ gilt.

Eine Funktion, die für zwei natürliche Zahlen den Rest bei ganzzahliger Division bestimmt, soll einem funktionsorientierten Test unterzogen werden. Bekannt seien die Typdefinitionen sowie der Funktionskopf:

type

```
tNatZahl=1..maxint;
```

```
function mod(inA:tNatZahl,inB:tNatZahl):integer;
```

```
{Ermittelt den Rest bei ganzzahliger Division der zwei Zahlen inA  
und inB.}
```

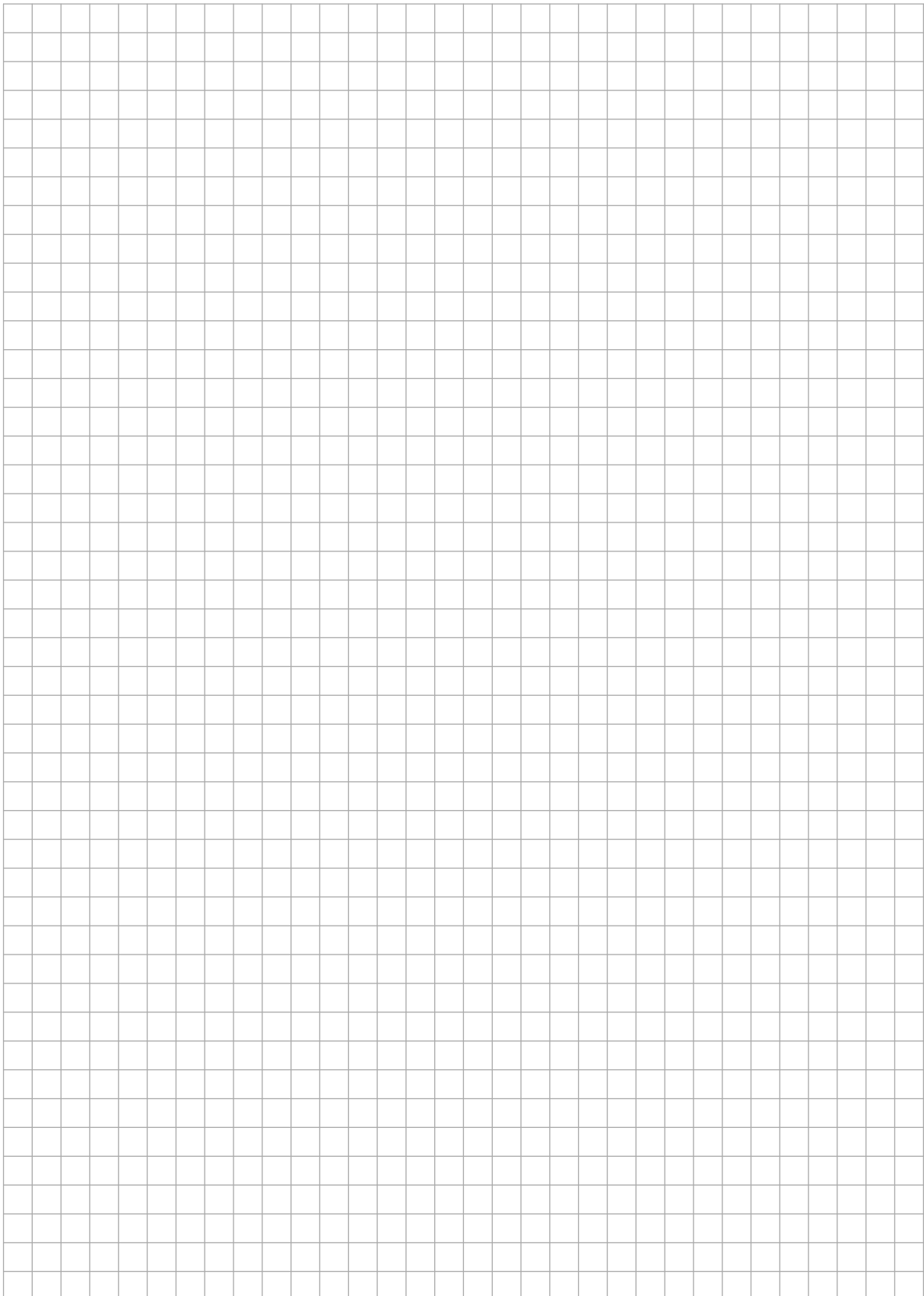
Geben Sie eine für einen Black-Box-Test sinnvolle Zerlegung der Menge der zulässigen Eingabedaten in Äquivalenzklassen an. Geben Sie hierbei zu jeder Äquivalenzklasse jeweils ein Testdatum an.

Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____

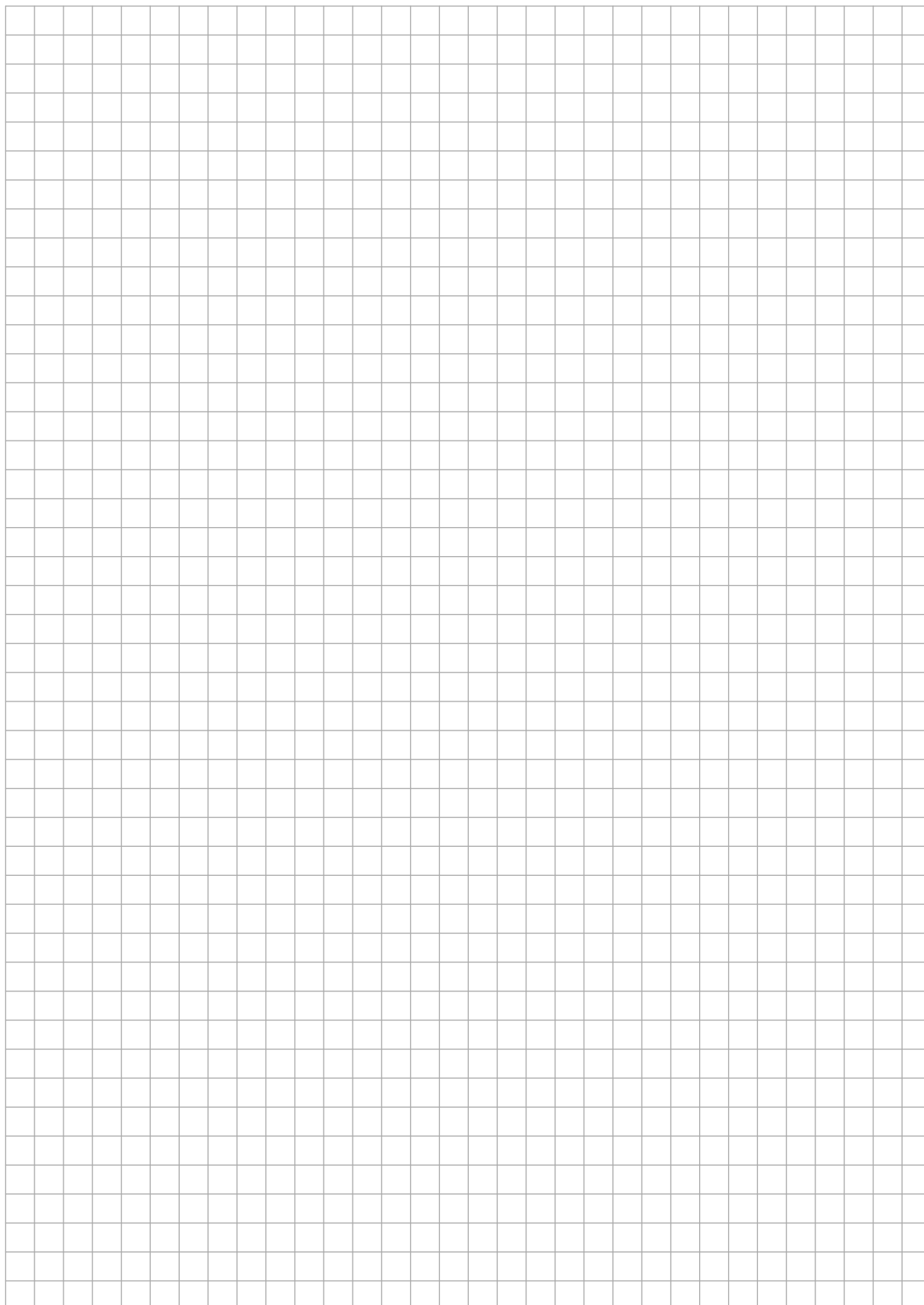




Kurs 1613 „Einführung in die imperative Programmierung“
Hauptklausur 05.02.2011

Name: _____

Matrikelnr.: _____





Zusammenfassung der Muss-Regeln

1. Selbstdefinierte Konstantenbezeichner bestehen nur aus Großbuchstaben. Bezeichner von Standardkonstanten wie z.B. `maxint` sind also ausgenommen.
2. Typenbezeichnern wird ein `t` vorangestellt. Bezeichnern von Zeigertypen wird ein `tRef` vorangestellt. Bezeichner formaler Parameter beginnen mit `in`, `io` oder `out`.
3. Jede Anweisung beginnt in einer neuen Zeile. `begin` und `end` stehen jeweils in einer eigenen Zeile.
4. Anweisungsfolgen werden zwischen `begin` und `end` um eine konstante Anzahl von 2-4 Stellen eingerückt. `begin` und `end` stehen linksbündig unter der zugehörigen Kontrollanweisung, sie werden nicht weiter eingerückt.
5. Anweisungsteile von Kontrollanweisungen werden genauso eingerückt.
6. Im Programmkopf wird die Aufgabe beschrieben, die das Programm löst.
7. Jeder Funktions- und Prozedurkopf enthält eine knappe Aufgabenbeschreibung als Kommentar. Ggf. werden zusätzlich die Parameter kommentiert.
8. Die Parameter werden sortiert nach der Übergabeart: Eingangs-, Änderungs- und Ausgabeparameter.
9. Die Übergabeart jedes Parameters wird durch Voranstellen von `in`, `io` oder `out` vor den Parameternamen gekennzeichnet.
10. Das Layout von Funktionen und Prozeduren entspricht dem von Programmen.
11. Jede von einer Funktion oder Prozedur benutzte bzw. manipulierte Variable wird als Parameter übergeben. Es werden keine globalen Variablen manipuliert.
12. Jeder nicht von der Prozedur veränderte Parameter wird als Wertparameter übergeben. Lediglich Felder können auch anstatt als Wertparameter als Referenzparameter übergeben werden, um den Speicherplatz für die Kopie und den Kopiervorgang zu sparen. Der Feldbezeichner beginnt aber stets mit dem Präfix `in`, wenn das Feld nicht verändert wird.
13. Pascal-Funktionen werden wie Funktionen im mathematischen Sinne benutzt, d.h. sie besitzen nur Wertparameter. Wie bei Prozeduren ist eine Ausnahme nur bei Feldern erlaubt, um zusätzlichen Speicherplatz und Kopieraufwand zu vermeiden.
14. Wertparameter werden nicht als lokale Variable missbraucht.
15. Die Laufvariable wird innerhalb einer `for`-Anweisung nicht manipuliert.
16. Die Grundsätze der strukturierten Programmierung sind strikt zu befolgen.