Name: _____ Matrikelnr.: _____

Hinweise zur Bearbeitung der Klausur zum Kurs 01613 „Einführung in die imperative Programmierung"

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   - 2 Deckblätter
   - 1 Formblatt für eine Bescheinigung für das Finanzamt
   - diese Hinweise zur Bearbeitung
   - 4 Aufgaben (Seite 2-16)
   - 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.
   - 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, werden wir Ihre Klausur korrigieren!
3. Schreiben Sie Ihre Lösungen jeweils 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.
4. **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.
5. 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!
6. Neben unbeschriebenem Konzeptpapier und Schreibzeug (Füller oder Kugelschreiber, benutzen Sie **keinen Bleistift** und **keinen Rotstift**!) sind **keine weiteren Hilfsmittel** zugelassen.
7. Es sind maximal 24 Punkte erreichbar. Sie haben die Klausur bestanden, wenn Sie mindestens 12 Punkte erreicht haben.

Aufgabe 1 (6 Punkte)

Gegeben ist die folgende Funktion `WasPassiert`.

```
function WasPassiert(a:integer):boolean;

  var
  b:integer;
  c:integer;

begin
  b := 1;
  c := 1;
  while (a > b) do
  begin
    c := c + 2;
    b := b + c;
  end;
  WasPassiert := (a = b);
end;
```

Geben Sie eine Problemspezifikation an, die durch die Funktion `WasPassiert` gelöst wird. Wählen Sie dabei den Wertebereich der Eingabe so allgemein wie möglich.

Eingabe:

Ausgabe:

Nachbedingung:

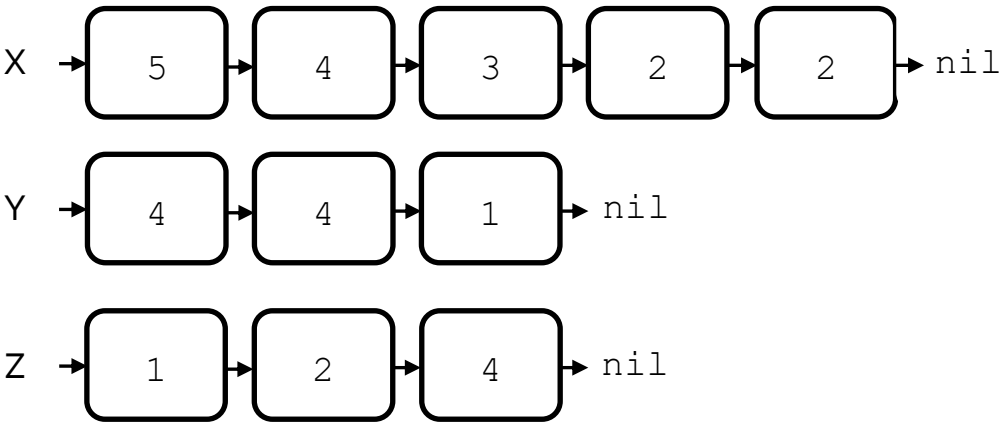
Name: _____ Matrikelnr.: _____

Aufgabe 2 (6 Punkte)

Gegeben sind die folgenden Typendefinitionen, die Funktionen `c,m,f` und die drei Listen `X,Y,Z`.

```
type
tRefListe = ^tListe;
tListe = record
           wert: integer;
           next: tRefListe
         end;

function c(inA:tRefListe;inB:tRefListe):boolean;
begin
  if ((inA = nil) or (inB = nil)) then
    c := ((inA = nil) and (inB = nil))
  else
    c := c(inA^.next, inB^.next)
end;

function m(inA:tRefListe;inB:tRefListe):integer;
begin
  if ((inA = nil)) then
    m := 0
  else
    m := inA^.wert * inB^.wert + m(inA^.next,inB^.next)
end;

function f(inA:tRefListe;inB:tRefListe):integer;
begin
  if c(inA,inB) then
    f := m(inA,inB)
  else
    f := 0
end;
```

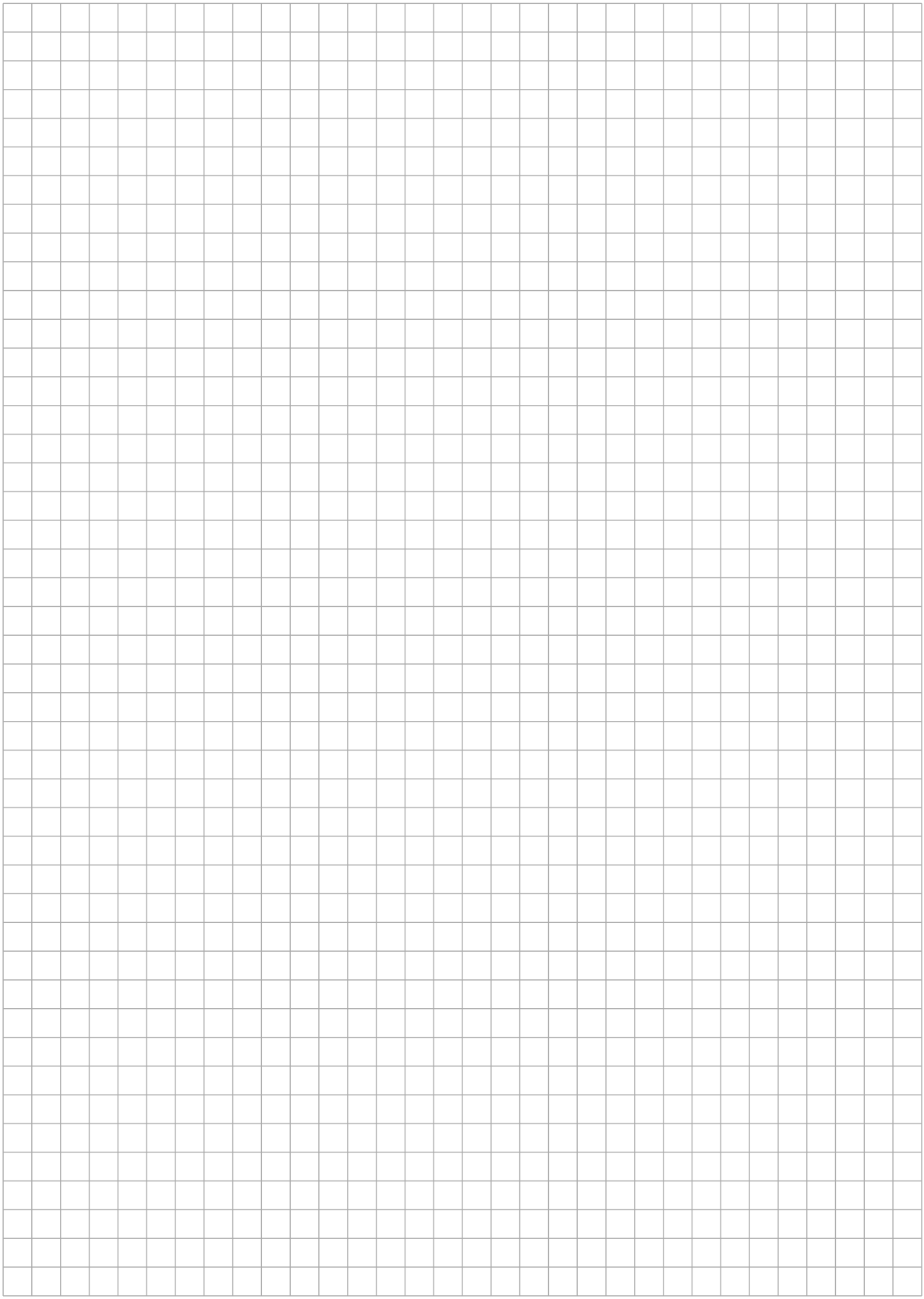
Name: _____ Matrikelnr.: _____



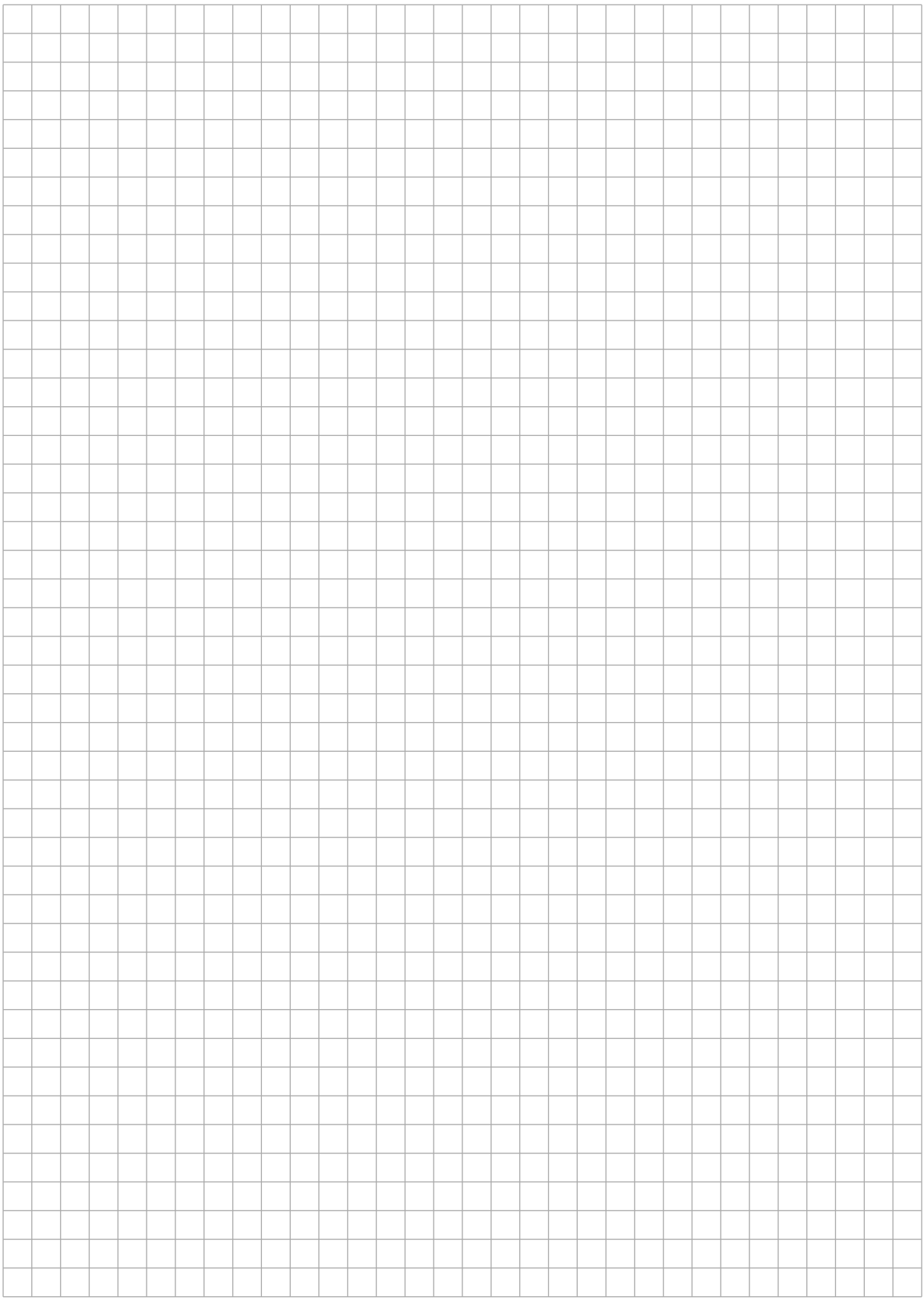
Welche Ergebnisse liefern die folgenden zwei Aufrufe?

`f(X,Y)`

`f(Y,Z)`



Name: _____ Matrikelnr.: _____



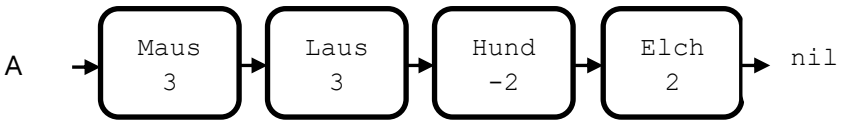
Aufgabe 3 (6 Punkte)

Gegeben sind die folgenden Typendefinitionen für eine Liste aus Zeichenketten mit einem zusätzlichen Wert `anzahl` und die Prozedur `add`.

```
tRefListe = ^tListe;
tListe = record
           text:String;
           anzahl:integer;
           next:tRefListe
         end;
```

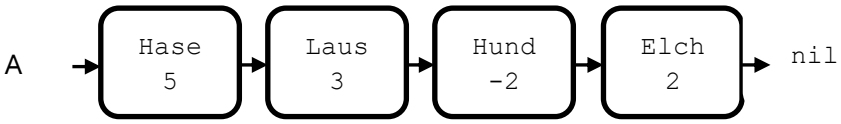
Die Prozedur `add` erfüllt die folgende Aufgabe: Für eine Liste, eine Zeichenkette und eine Anzahl, durchsucht die Prozedur die Liste. Existiert bereits ein Listenelement mit der gleichen Zeichenkette, wird die Anzahl dieses Elements um den eingegebenen Wert erhöht. Wird die Anzahl dadurch `0`, wird das Element aus der Liste ausgekettet. Existiert kein Listenelement mit der gleichen Zeichenkette, wird ein neues Element an den Anfang der Liste eingefügt. Gehen Sie davon aus, dass jede Liste jede Zeichenkette nur einmal enthält.

Hier sehen Sie Beispiele, wie die Prozedur `add` eine Liste `A` verändert:

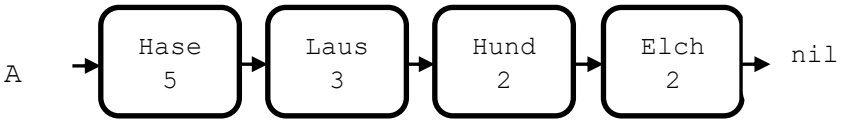



```
add(A,'Hase',5)
```


```
add(A,'Maus',-3)
```



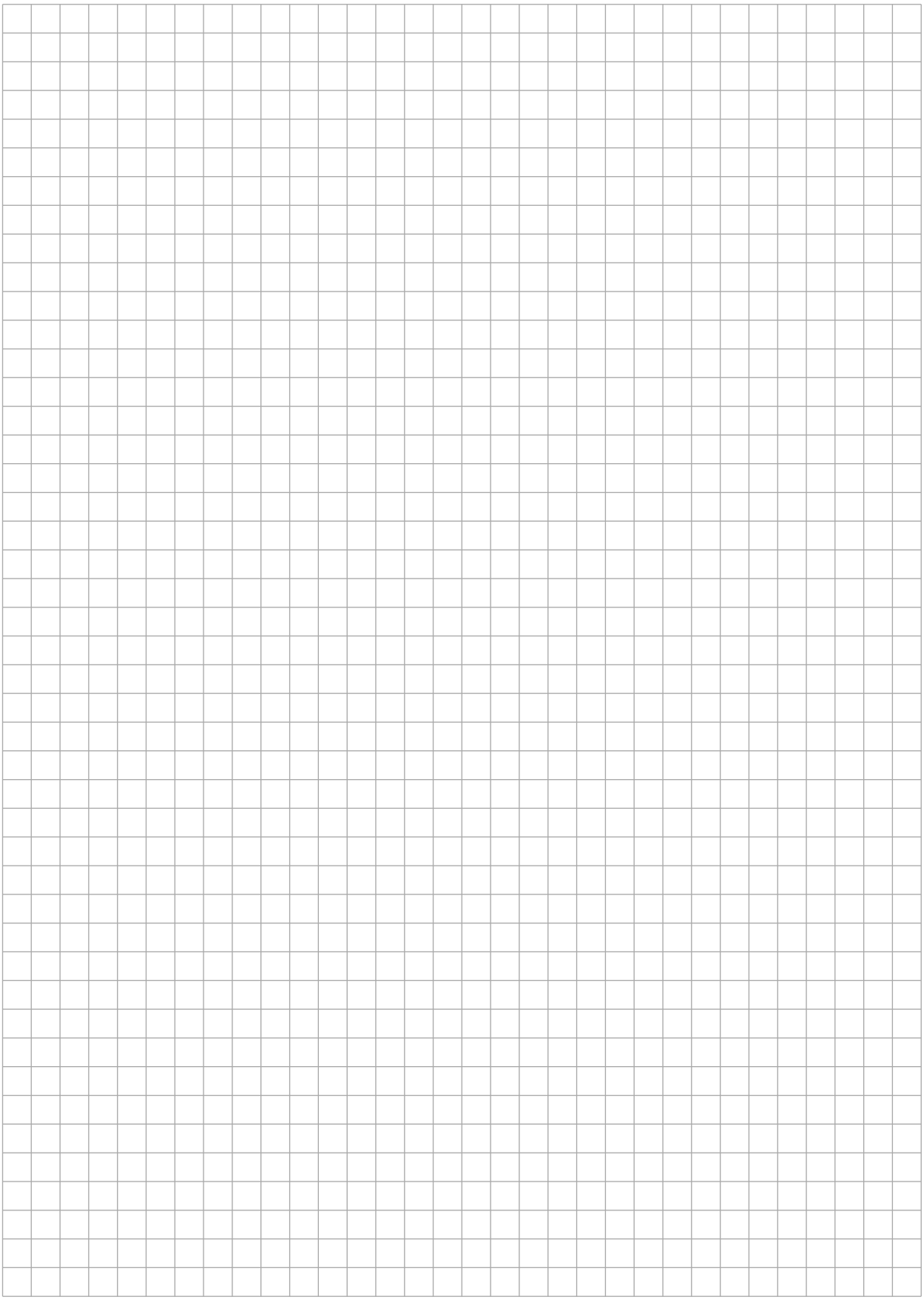

```
add(A,'Hund',4)
```



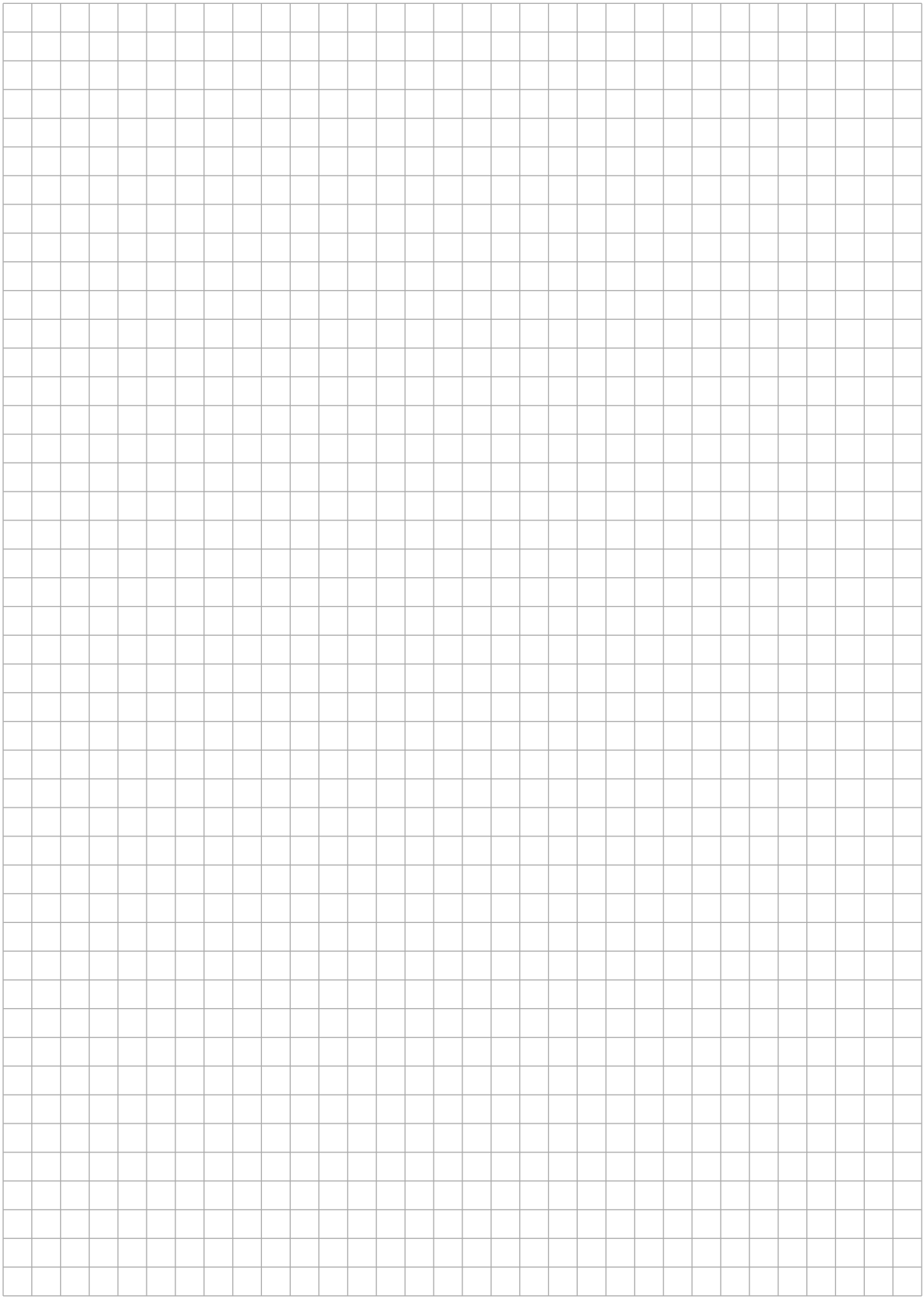
Name: _____ Matrikelnr.: _____

Hier finden Sie die Prozedur `add`. Die Prozedur ist jedoch noch unvollständig. Ergänzen Sie diese passend, an den grau eingefärbten Stellen, jeweils um eine Zeile.

```
procedure add(var ioSet:tRefListe; inText:String; inZahl:integer);

  var
  neu: tRefListe;
  lauf: tRefListe;
  lauf2: tRefListe;
  found: boolean;

  begin

    lauf2 := lauf;
    found := false;
    while (lauf <> nil) do
    begin
      if (lauf^.text = inText) then
      begin

        found := true;
        if (lauf^.anzahl = 0) then
        begin
          if (lauf2 = lauf) then
            ioSet := ioSet^.next
          else
          begin

          end
        end
      end;
      lauf2 := lauf;
      lauf := lauf^.next;
    end;

    if (not found) then
    begin
      new(neu);
      neu ^.text := inText;
      neu ^.anzahl := inZahl;
      neu ^.next := ioSet;
      ioSet := neu;
     end
  end;
```



Name: _____ Matrikelnr.: _____



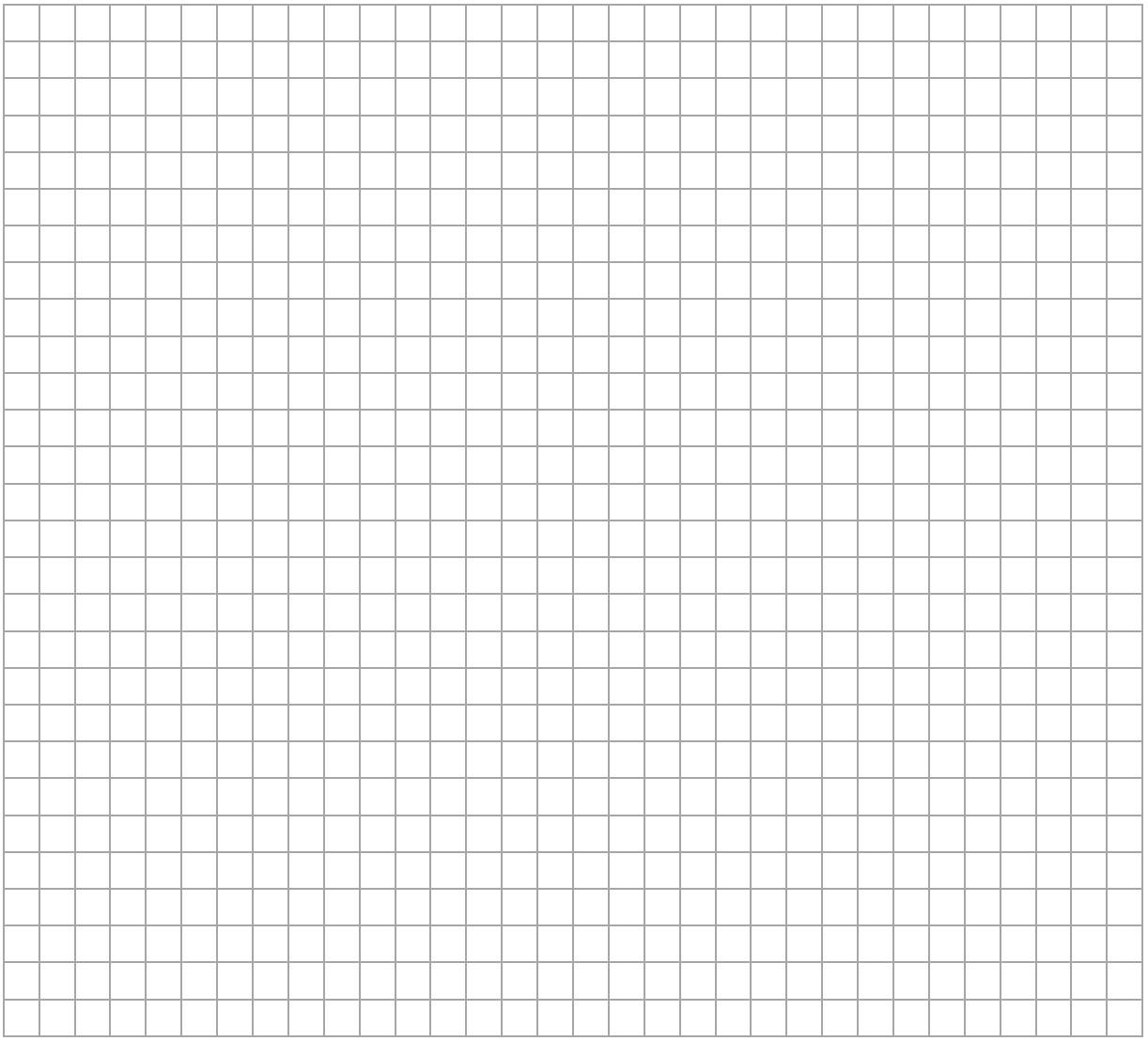
Aufgabe 4 (6 Punkte)

Eine Funktion gibt für eine nicht-negative ganze Zahl aus, welcher Rest verbleibt, wenn die Zahl (ganzzahlig) durch 3 geteilt wird. Sie soll einem funktionsorientierten Test unterzogen werden.

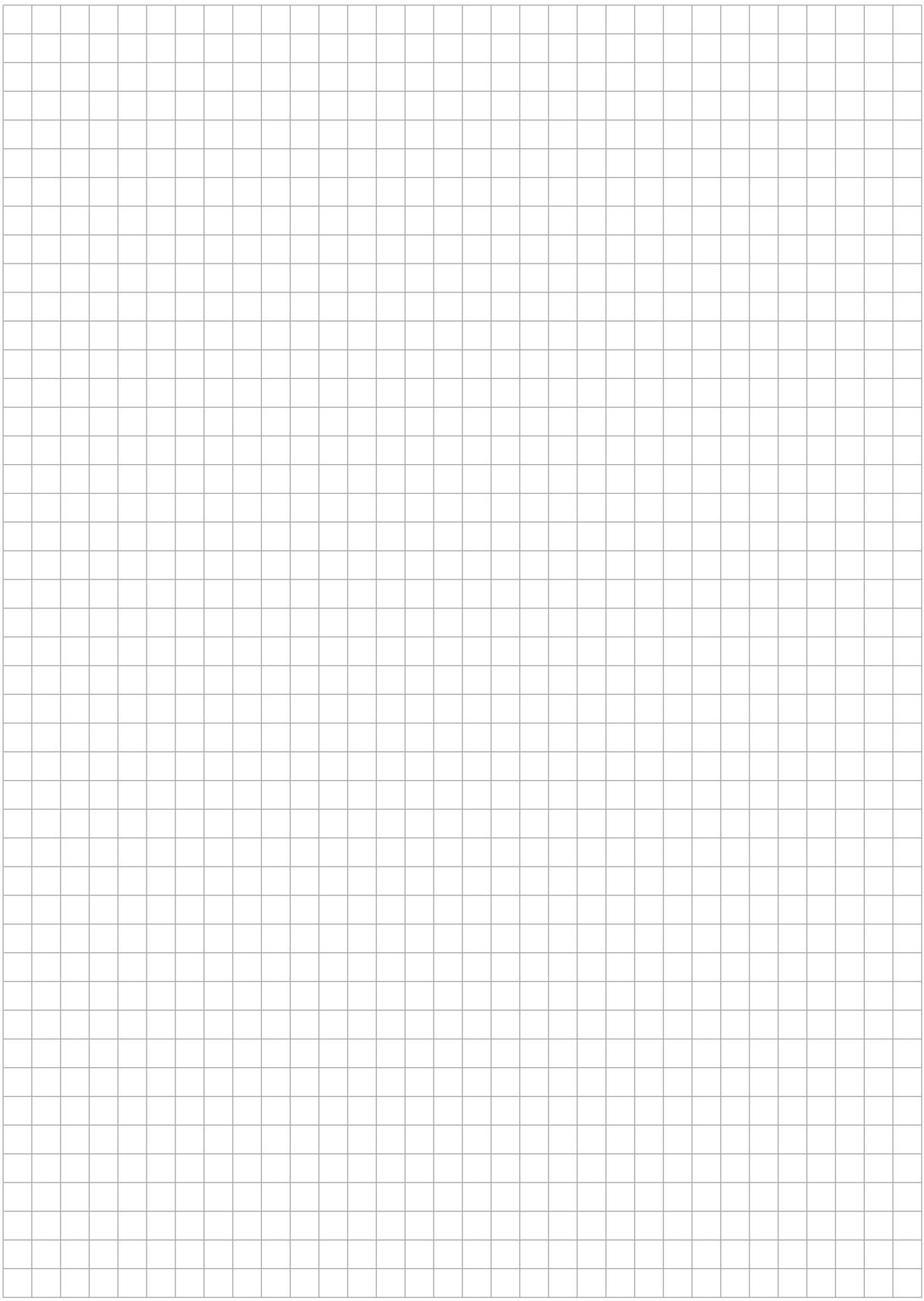
Bekannt sind die Typdefinitionen sowie der Funktionskopf:

```
type
tZahl = 0..maxint;
tRest = 0..2;

function divDrei(inZahl:tZahl):tRest;
{berechnet den Wert inZahl % 3}
```

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



Name: _____ Matrikelnr.: _____



Name: _____ Matrikelnr.: _____

Kurs 01613 „Einführung in die imperative Programmierung"

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.