FernUniversität in Hagen

FAKULTÄT FÜR MATHEMATIK UND INFORMATIK

| Matrikelnr. | | | | | | | |
|---|---|---|---|---|---|---|---|

Name _____

Vorname _____

PRÜFUNGSKLAUSUR: **63811** Einführung in die imperative Programmierung
TERMIN: 22.02.2020
PRÜFENDE/R: Dr. Robin Bergenthum

## Aufbau und Bewertung der Prüfungsklausur

| Aufgabe | 1 | 2 | 3 | 4 | Gesamt/ Summe |
|---|---|---|---|---|---|
| maximal erreichbare Punktzahl | 6 | 6 | 6 | 6 | 24 |
| erreichte Punktzahl | | | | | |

Bewertung /Note:_____

Datum der Bewertung:_____

Unterschrift Prüfende/r:_____

**Kurs 01613 „Einführung in die imperative Programmierung“**

Matrikelnr.: _____

**Bearbeitungshinweise:**

- Tragen Sie Ihre Matrikelnummer sowie Ihren Vor- und Nachnamen auf dem **Deckblatt** ein. Versehen Sie bitte zusätzlich **jede Seite mit Ihrer Matrikelnummer** in dem oben rechts vorgesehenen Feld.
- **Prüfen Sie die Vollständigkeit Ihrer Unterlagen**. Die Klausur umfasst: Deckblatt, **4 Aufgaben** (Seite 1-13) und die Muss-Regeln des Programmierstils.
- Für die Bearbeitung der Prüfungsklausur haben Sie **120 Minuten** zur Verfügung.
- Verwenden Sie **keinen Bleistift oder Rotstift** für Ihre Lösungen.
- 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. Sollte der zur Verfügung stehende Platz nicht ausreichen, können Ihnen die Aufsichtspersonen weiteres durch das Prüfungsamt gestempeltes Papier aushändigen.
- **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.
- Die Prüfungsklausur ist komplett mit Deckblatt und allen Aufgaben- und Lösungsblättern abzugeben.
- Für die Prüfungsklausur sind neben Schreibutensilien **keine weiteren Hilfsmittel** zugelassen.
- Es sind maximal 24 Punkte erreichbar. Sie haben die Klausur bestanden, wenn Sie mindestens 12 Punkte erreicht haben.

## Aufgabe 1 (6 Punkte)

Gegeben sind die folgenden zwei Funktionen `p` und `WasPassiert`.

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

  var
  b:boolean;
  i:integer;

begin
  b := false;
  i := 2;
  while ((i < a) and (not b)) do
  begin
    b := (a mod i = 0);
    i := i + 1;
  end;
  p := (not b) and not (a < 2)
end;

function WasPassiert(a:integer):boolean;
begin
  WasPassiert := p(a) and p(a+2)
end;
```

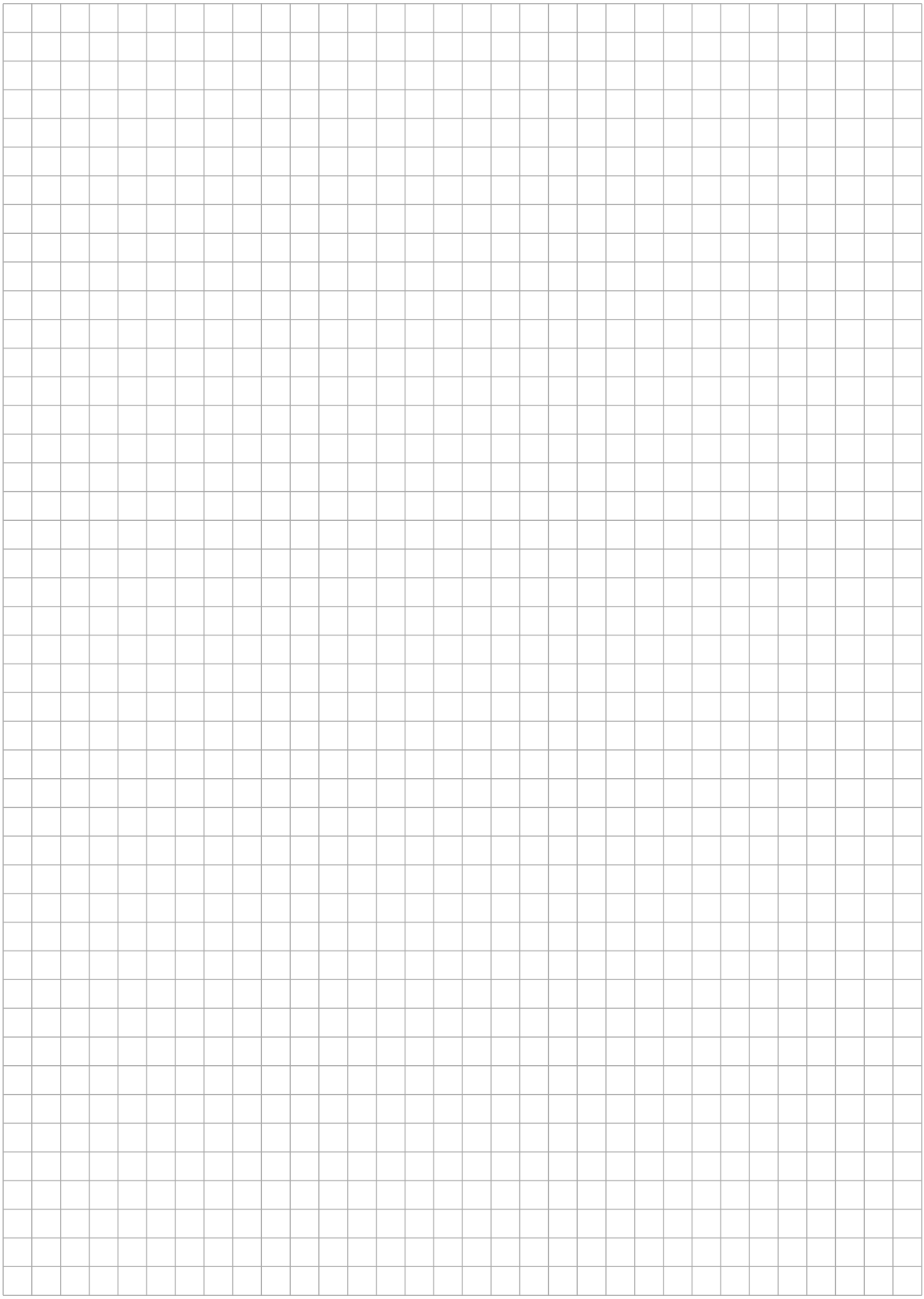
Geben Sie eine Problemspezifikation an, die durch die Funktion `WasPassiert` gelöst wird. Wählen Sie dabei den Wertebereich der Eingabe so allgemein wie möglich.

Eingabe:

Ausgabe:

Nachbedingung:

Matrikelnr.: _____



## Aufgabe 2 (6 Punkte)

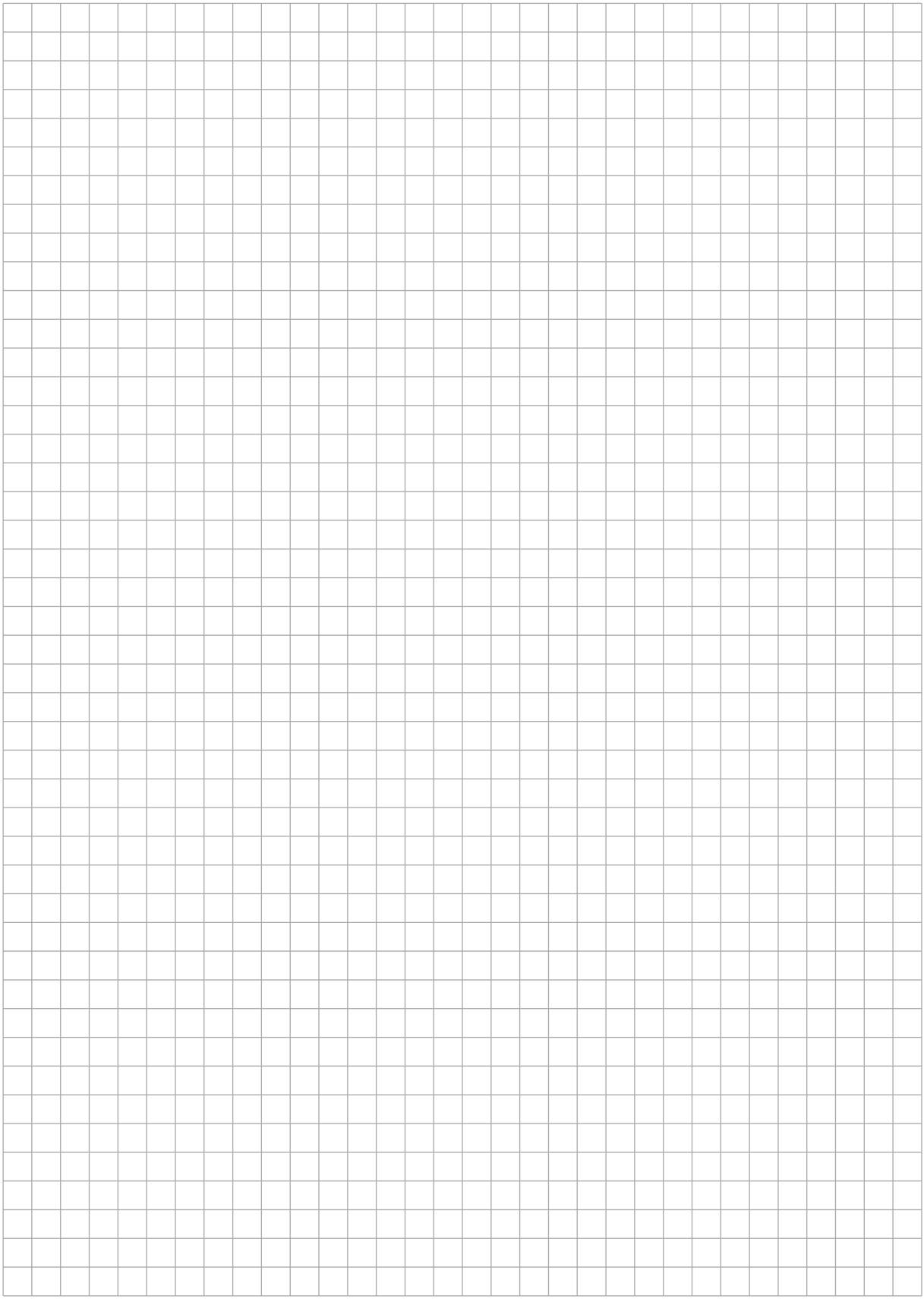
Gegeben sind die folgenden Typendefinitionen, die Funktion `x` und die Liste `A`.

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

function x(A:tRefListe):tRefRing;

  var
  B:tRefRing;
  S:tRefRing;
  L:tRefListe;

begin
  L := A;
  new(B);
  S := B;
  B^.wert := L^.wert;
  while L^.next <> nil do
  begin
    new(B^.next);
    B^.next^.wert := L^.next^.wert;
    B^.next^.prev := B;
    B := B^.next;
    L := L^.next
  end;
  B^.next := S;
  S^.prev := B;
  x := S
end;
```

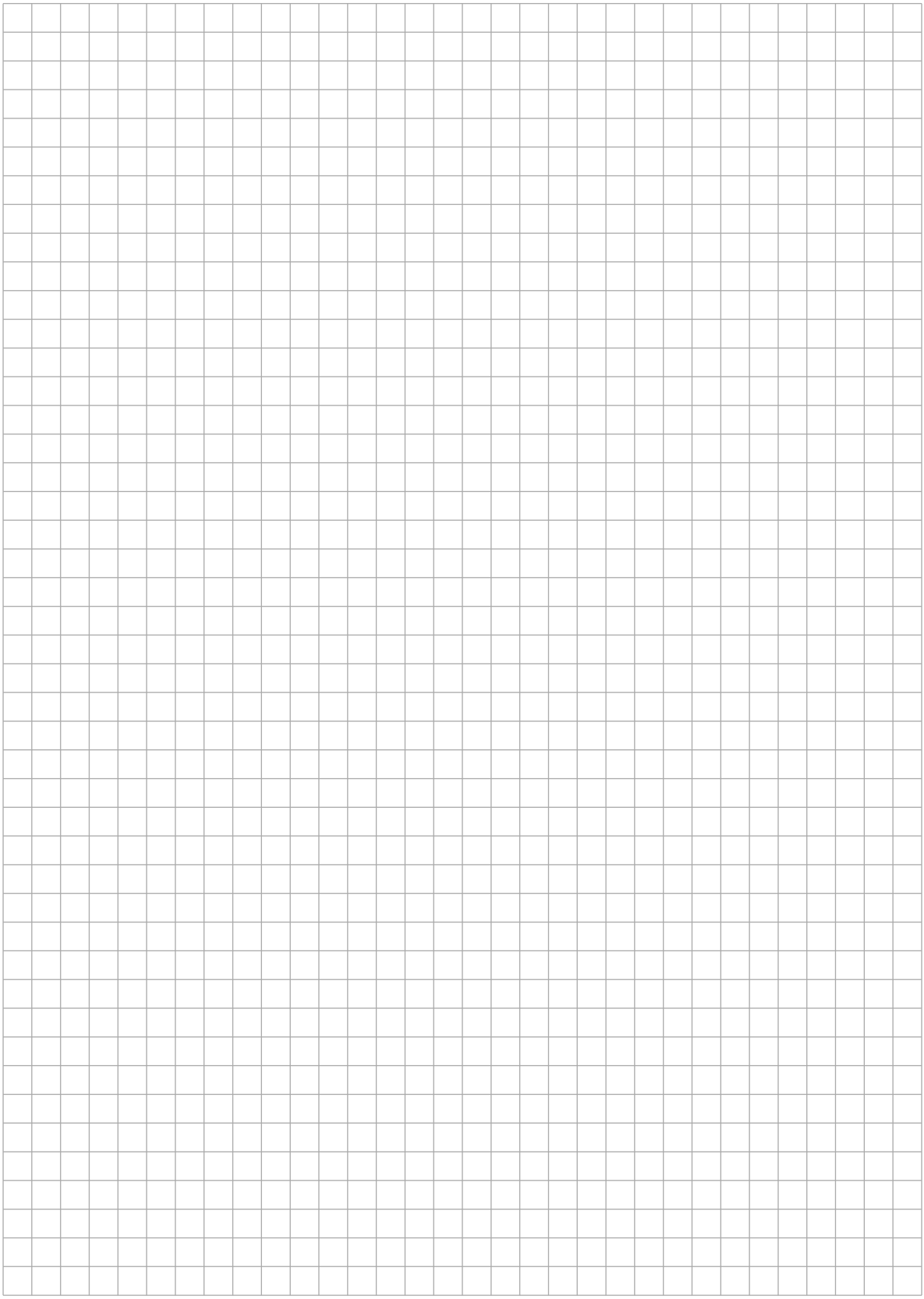
Matrikelnr.: _____


**Aufgabe**: Welches Ergebnis liefert der Aufruf `x(A)`?

`x(A)` →



Matrikelnr.: _____



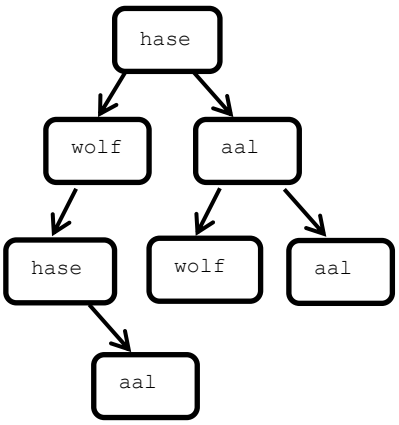
## Aufgabe 3 (6 Punkte)

Gegeben sind die folgenden Typendefinitionen für einen Baum aus Zeichenketten und die Funktionen `x` und `c`.

```
tRefBinBaum = ^tBinBaum;
tBinBaum = record
             text:string;
             li:tRefBinBaum;
             re:tRefBinBaum
           end;
```

Mit Hilfe der Funktion `x` erfüllt die Funktion `c` die folgende Aufgabe: Genau dann, wenn die Anzahl der Knoten im Baum, die die erste Zeichenkette enthalten, genau so groß ist, wie die Anzahl der Knoten im Baum, die die zweite Zeichenkette enthalten, gibt die Funktion `true` zurück.

**Beispiel**: Wird der folgende Baum und die zwei Zeichenketten `hase` und `wolf` übergeben, so liefert `c` den Wert `true`. Wird der folgende Baum und die zwei Zeichenketten `hase` und `aal` übergeben, so liefert `c` den Wert `false`.

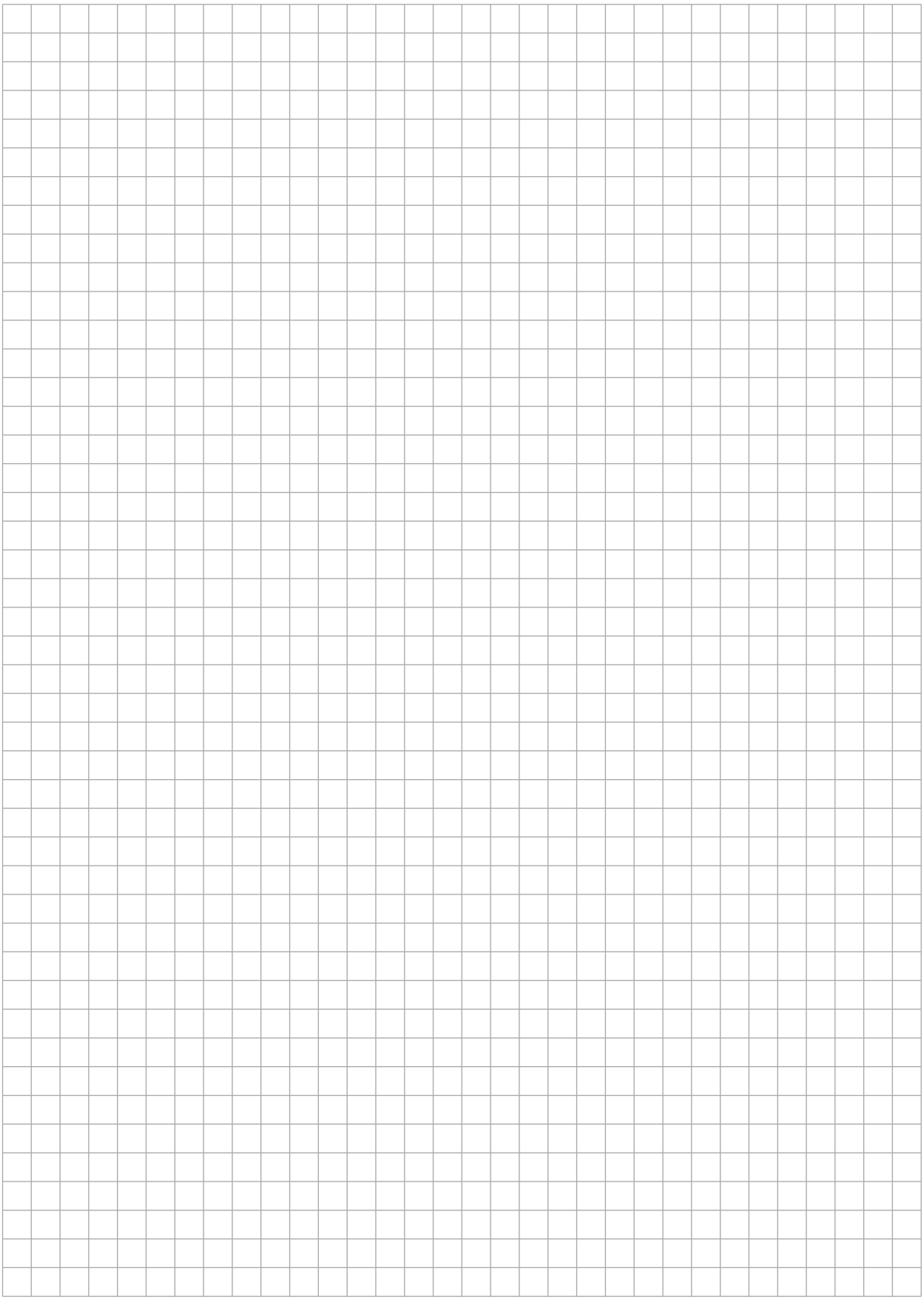


Matrikelnr.: _____

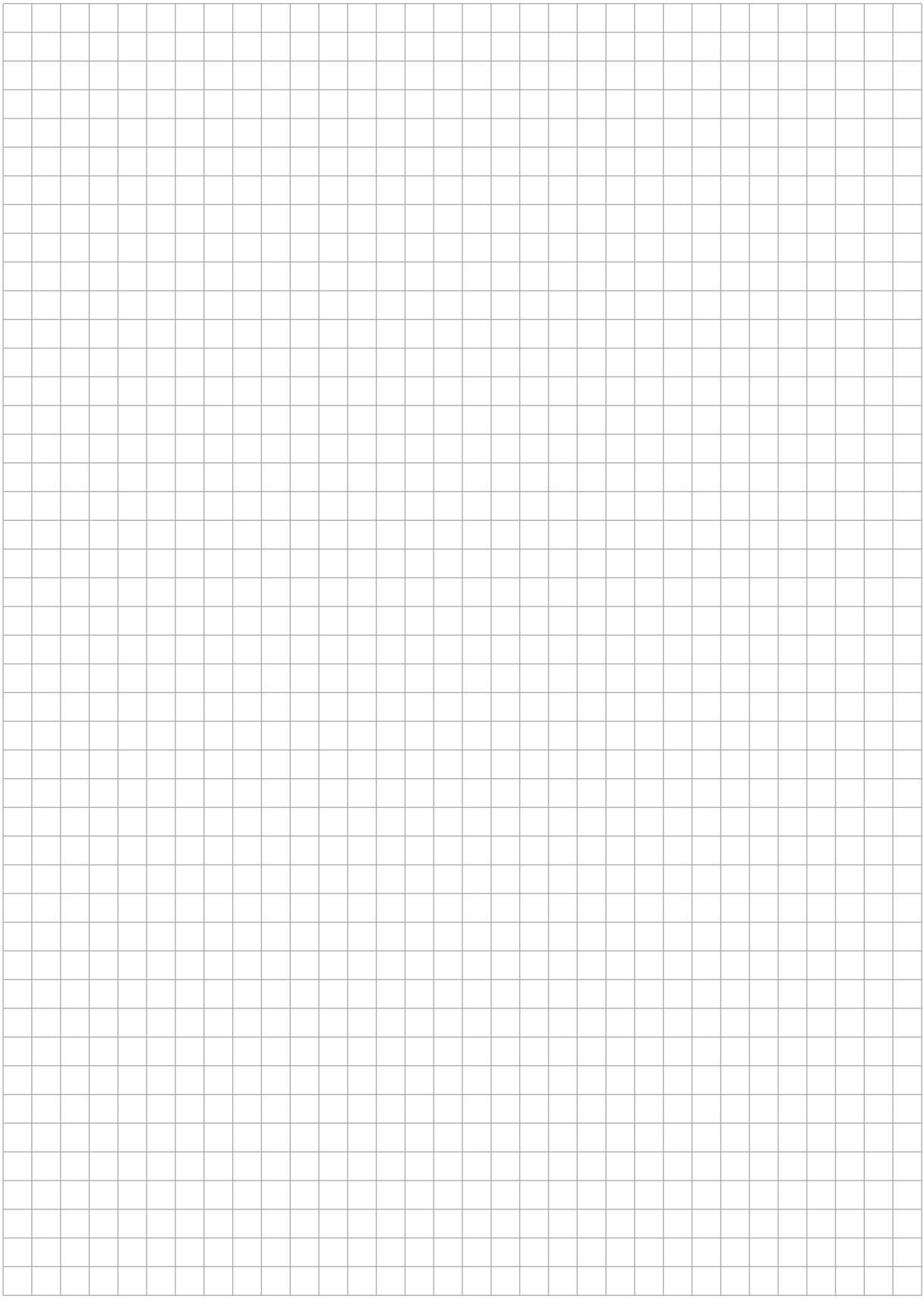
**Aufgabe a**: Hier finden Sie die Funktionen `c` und `x`. Die Funktion x ist jedoch noch unvollständig. Ergänzen Sie sie passend, an den grau eingefärbten Stellen, jeweils um eine Zeile.

```
function x(inText1:string; inText2:string; A:tRefBinBaum):integer;

  var
  i: integer;

begin

  if (A=nil) then
    x := i
  else
  begin
    if (A^.text = inText1) then
    begin
      i := 1
    end;
    if (A^.text = inText2) then
    begin
      i := i - 1
    end;

  end
end;

function c(inText1:string; inText2:string; A:tRefBinBaum):boolean;
begin
  c := (x(inText1,inText2,A) = 0);
end;
```

**Aufgabe b**: Ersetzen Sie in der Funktion `x` die Zeile `i := i - 1` durch die Zeile `i := - 1`. Geben Sie ein Testdatum an, für den die Funktion `c` Ihre Aufgabe jetzt nicht mehr erfüllt.



Matrikelnr.: _____



## Aufgabe 4 (6 Punkte)

Eine Funktion, die für eine natürliche Zahl größer als Eins die Primfaktorzerlegung bestimmt, soll einem funktionsorientierten Test unterzogen werden. Die Primfaktorzerlegung stellt die Zahl als Produkt von Primzahlen dar. Jede natürliche Zahl hat eine eindeutige Primfaktorzerlegung. Sortieren wir die Primfaktoren aufsteigend, können wir die Zerlegung eindeutig als Zeichenkette darstellen.

Beispiele: Die Primfaktorzerlegung der Zahl `18` ist `2*3*3`. Die Primfaktorzerlegung der Zahl `10` ist `2*5`

Bekannt sind die folgende Typdefinition und der Funktionskopf:

```
type
tZahl = 2..maxint;

function faktorZerlegung(inZahl:tZahl):string;
{berechnet die Primfaktorzerlegung der Zahl inZahl und gibt diese
als Zeichenkette aus.}
```

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

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.