FernUniversität in Hagen

FAKULTÄT FÜR MATHEMATIK UND INFORMATIK

Matrikelnr.							

Name _____

Vorname _____

PRÜFUNGSKLAUSUR: **63811** Einführung in die imperative Programmierung
TERMIN: 19.09.2020
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:_____

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 1-9) 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 ist das Programm `WasPassiert`.

```
program WasPassiert(input,output);
  var
  e:real;
  a:real;
begin
  readln(e);
  a := 0;
  while (e >= 1) do
  begin
    e := e - 1;
    a := a + 1;
  end;
  writeln(a);
end.
```

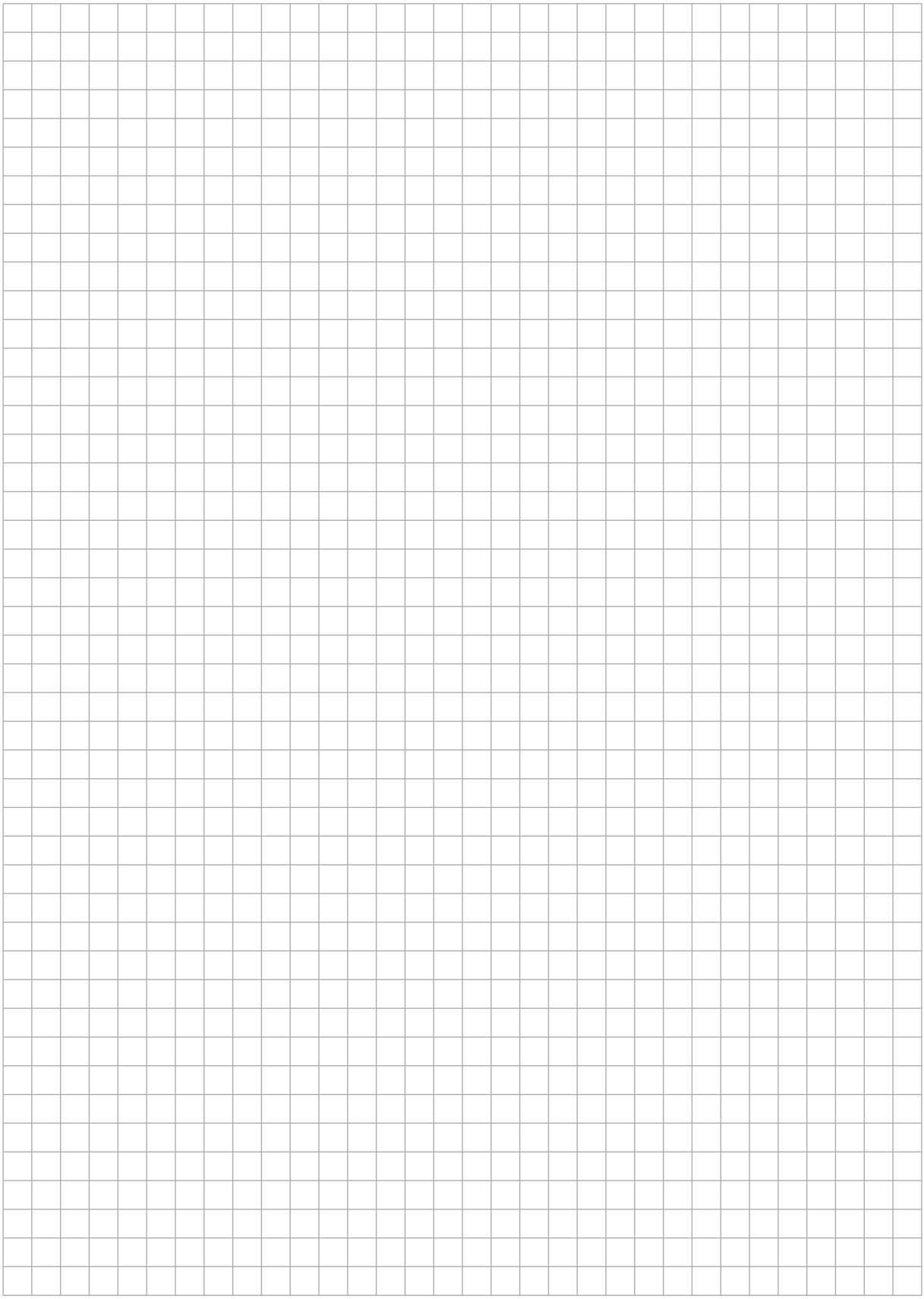
Geben Sie eine Problemspezifikation an, die durch das Programm `WasPassiert` gelöst wird. Wählen Sie dabei den Wertebereich der Eingabe so allgemein wie möglich. Seinen Sie beim Wertebereich der Ausgabe so restriktiv wie möglich.

Eingabe:

Ausgabe:

Nachbedingung:

Matrikelnr.: _____



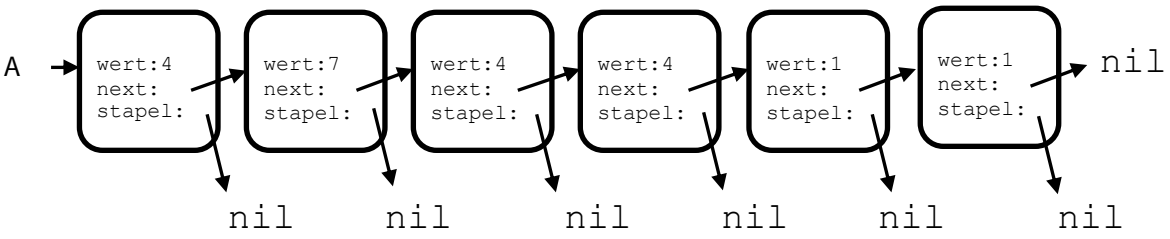
Aufgabe 2 (6 Punkte)

Gegeben sind die folgenden Typendefinitionen, die Prozedur `x` und die StapelListe `A`.

```
type
tRefStapelListe = ^tStapelListe;
tStapelListe = record
                 wert:integer;
                 next:tRefStapelListe;
                 stapel:tRefStapelListe
               end;

procedure x(A: tRefStapelListe);
  var
  S:tRefStapelListe;
  T:tRefStapelListe;
  T2:tRefStapelListe;
begin
  S := A;
  T := A^.next;
  T2 := A;
  while T <> nil do
  begin
    while (S <> T) AND (S^.wert <> T^.wert) do
    begin
      S := S^.next
    end;
    if S <> T then
    begin
      T2^.next := T^.next;
      while S^.stapel <> nil do
      begin
        S := S^.stapel
      end;
      S^.stapel := T;
      T^.next := nil;
      T := T2^.next
    end
    else
    begin
      T2 := T;
      T := T^.next
    end;
    S := A
  end
end;
```

Matrikelnr.: _____



Aufgabe: Wie sieht die Stapelliste A nach dem Aufruf `x(A)aus`?

A →

Aufgabe 3 (6 Punkte)

Gegeben sind die folgenden Typendefinitionen für einen Baum aus Zahlen und die Funktion `x`.

```
tRefBinBaum = ^tBinBaum;
tBinBaum = record
             wert:integer;
             li:tRefBinBaum;
             re:tRefBinBaum
           end;
```

Die Funktion `x` erfüllt die folgende Aufgabe: Zu jedem Blatt des Baumes gibt es einen Weg von der Wurzel zu diesem Blatt. Für jeden Weg kann man die Summe der Werte der Knoten auf dem Weg berechnen und wir nennen diese Summe den Wert des Weges. Die Funktion gibt den maximalen Wert aller Wert der Wege zurück.

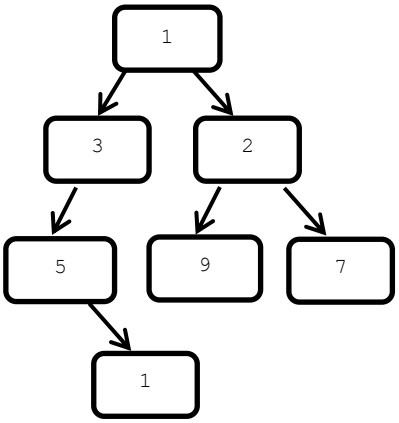
Beispiel: Wird der folgende Baum übergeben, so existieren drei Wege deren Werte sich wie folgt berechnen:

1 + 3 + 5 + 1 = 10

1 + 2 + 9 = 12

1 + 2 + 7 = 10

Die Funktion gibt in diesem Beispiel 12 zurück.



Matrikelnr.: _____

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

```
function x(A:tRefBinBaum):integer;

  var
  l: integer;
  r: integer;

begin
  if A=nil then
                                                    
  else
  begin
    l:=x(A^.li);
    r:=x(A^.re);
    if l > r then
    begin
                                                    
    end
    else
    begin
                                                    
    end
  end
end;
```

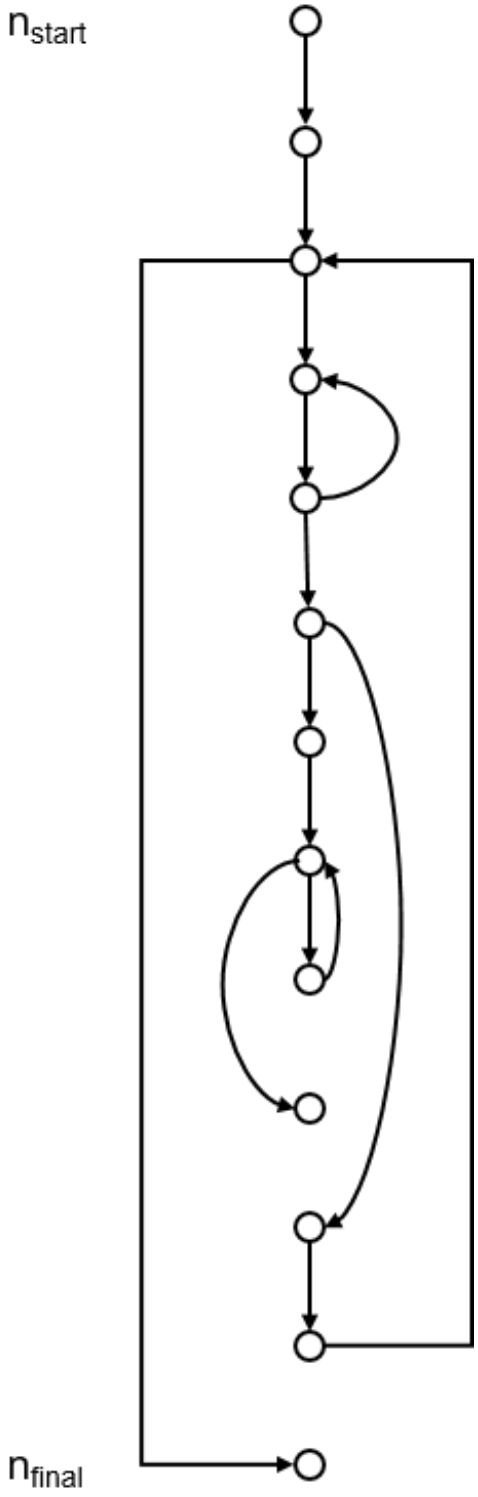
Aufgabe 4 (6 Punkte)

Hier sehen Sie nochmals die Prozedur x aus Aufgabe 2. Auf der nächsten Seite finden Sie den kompakten Kontrollflussgraphen der Prozedur x. Der Graph enthält jedoch drei Fehler.

Aufgabe: Markieren Sie den einen falschen Pfeil und ergänzen Sie die zwei fehlenden Pfeile!

```
procedure x(A: tRefStapelListe);
  var
  S:tRefStapelListe;
  T:tRefStapelListe;
  T2:tRefStapelListe;
begin
  S := A;
  T := A^.next;
  T2 := A;
  while T <> nil do
  begin
    while (S <> T) AND (S^.wert <> T^.wert) do
    begin
      S := S^.next
    end;
    if S <> T then
    begin
      T2^.next := T^.next;
      while S^.stapel <> nil do
      begin
        S := S^.stapel
      end;
      S^.stapel := T;
      T^.next := nil;
      T := T2^.next
    end
    else
    begin
      T2 := T;
      T := T^.next
    end;
    S := A
  end
end;
```

Matrikelnr.: _____



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.