Prof. Christian Bischof, Ph.D.
Jonathan Heinen, Michael Lülfesmann, Carsten Otto

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte ankreuzen):

- ○ Informatik Bachelor
- ○ Mathematik Bachelor
- ○ Informatik Lehramt
- ○ Sonstige: _____

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**11**	
Aufgabe 2	**14**	
Aufgabe 3	**15**	
Aufgabe 4	**26**	
Aufgabe 5	**16**	
Aufgabe 6	**18**	
Summe	**100**	

Hinweise:

- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit dokumentenechten Stiften, nicht mit roten Stiften oder mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern (benutzen Sie auch die Rückseiten).
- Zusätzlich bereitgestellte Blätter werden **sofort** an die Aufgabenblätter **geheftet**.
- **Auf alle Blätter** (inklusive zusätzliche Blätter) müssen Sie **Ihren Namen und Ihre Matrikelnummer** schreiben.
- Was nicht bewertet werden soll, streichen Sie bitte durch.
- Werden **Täuschungsversuche** beobachtet, so wird die Klausur mit **0 Punkten** bewertet.
- Geben Sie am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.

Matrikelnummer: **Name:**

Aufgabe 1 (Programmanalyse): (11 Punkte)

Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Schreiben Sie hierzu jeweils die ausgegebenen Zeichen in die Kästchen hinter den Kommentar „`OUT:`“.

```
class A {
  public static int x = 0;
  public int y = 3;

  public A() {
    this(5);
  }

  public A(int x) {
    this.x += x;
    this.y = x;
  }

  public void f(int x) {
    this.x = x + y;
  }
}

class B extends A {
  public int z;
  public A a;

  public B(int x) {
    a = new A(x);
    z = a.x;
  }

  public void f(double x) {
    z = (int) (10+x);
    a.x = z + 2;
    a.y += 5;
  }
}

public class M {

  public static void main(String[] args) {
    B b = new B(4);
    System.out.println(b.x + " " + b.y + " " + b.z); // OUT: [   ] [   ] [   ]
    b.f(5);
    System.out.println(b.y + " " + b.z);             // OUT: [   ] [   ]
    A a = b.a;
    System.out.println(a.y + " " + a.x);             // OUT: [   ] [   ]
    b.a = b;
    b.f(2.0);
    System.out.println(a.y + " " + a.x);             // OUT: [   ] [   ]
    System.out.println(b.y + " " + b.z);             // OUT: [   ] [   ]
  }


}
```

Matrikelnummer: **Name:**

Aufgabe 2 (Verifikation): **(12 + 2 = 14 Punkte)**

Gegeben sei folgender *Java*-Algorithmus P, der n^2 als Summe $\sum_{i=1}^{n}(2i-1)$ berechnet.

$\langle\varphi\rangle$ (Vorbedingung)

```
res = 0;
s = 1;
x = 0;
while (x < n) {
    res = res + s;
    s = s + 2;
    x = x + 1;
}
```

$\langle\psi\rangle$ (Nachbedingung)

a) Als Vorbedingung für den oben aufgeführten Algorithmus P gelte $n \geq 0$ und als Nachbedingung:

$$res = \sum_{i=1}^{n}(2i-1) \wedge s = 2 \cdot (n+1) - 1$$

Vervollständigen Sie die Verifikation des folgenden Algorithmus im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Hinweise:

- Eine leere Summe hat den Wert 0, beispielsweise gilt $\sum_{i=a}^{b} i := 0$ für $a > b$.
- Auf der nächsten Seite finden Sie eine Vorlage, die Sie direkt ausfüllen dürfen.

Matrikelnummer: **Name:**

$\langle n \geq 0 \rangle$

$\langle \_\_\_\_\_ \rangle$

```
res = 0;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
s = 1;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
x = 0;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
while (x < n) {
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    res = res + s;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    s = s + 2;
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    x = x + 1;
```

$\langle \_\_\_\_\_ \rangle$

```
}
```

$\langle \_\_\_\_\_ \rangle$

$\langle res = \sum_{i=1}^{n}(2i-1) \wedge s = 2 \cdot (n+1) - 1 \rangle$

b) Beweisen Sie die Terminierung des Algorithmus P. Geben Sie hierzu eine Variante für die `while`-Schleife an. Zeigen Sie, dass es sich tatsächlich um eine Variante handelt, und beweisen Sie damit unter Verwendung des Hoare-Kalküls mit der Voraussetzung $n \geq 0$ die Terminierung.

Zur Erinnerung (Kopie von Seite 3):

$\langle \varphi \rangle$ (Vorbedingung)

```
res = 0;
s = 1;
x = 0;
while (x < n) {
    res = res + s;
    s = s + 2;
    x = x + 1;
}
```

$\langle \psi \rangle$ (Nachbedingung)

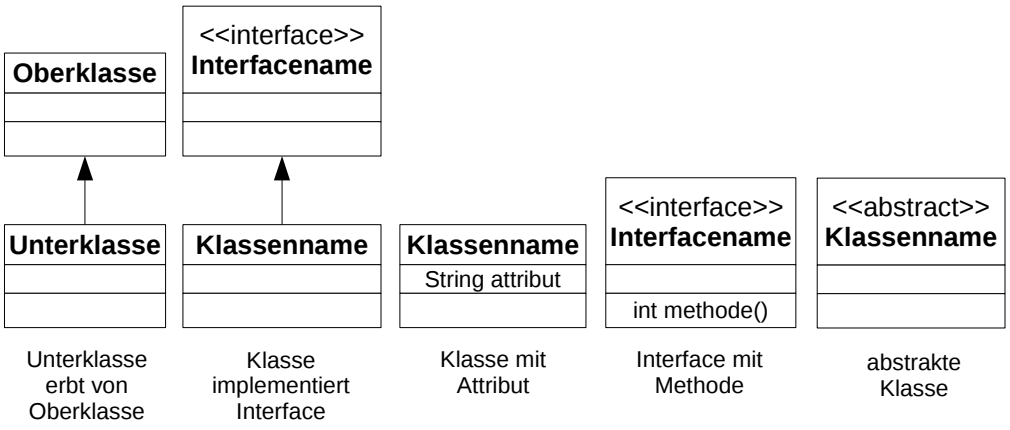
Matrikelnummer: **Name:**

Aufgabe 3 (Datenstrukturen in Java): **(6 + 9 = 15 Punkte)**

a) Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Nachrichten zu entwerfen. Bei der vorhergehenden Analyse wurden folgende Eigenschaften der verschiedenen Nachrichten ermittelt.

- Für jede Nachricht ist bekannt, ob sie wichtig ist.
- Eine Faxnachricht ist eine elektronische Nachricht, zu der die Seitenanzahl bekannt ist. Zusätzlich ist die Verfügbarkeit des Übertragungsdienstes bekannt, welche zwischen 0 und 1 liegt.
- Eine E-Mail ist eine elektronische Nachricht, zu der bekannt ist, ob es einen Anhang gibt. Auch bei E-Mails ist die Verfügbarkeit bekannt, welche zwischen 0 und 1 liegt.
- Jede Postnachricht hat eine Empfängeradresse.
- Ein Brief ist eine Postnachricht, zu der das Gewicht in Gramm bekannt ist.
- Ein Einschreiben ist ein spezieller Brief. Zu jedem Einschreiben ist bekannt, ob dieser nur eigenhändig an den Empfänger ausgeliefert werden darf.
- Eine weitere Postnachricht ist die Ansichtskarte. Hier ist bekannt, wie viele Sehenswürdigkeiten auf der Karte dargestellt sind.
- Die Nachrichtenübermittlung bei Fax- und allen Postnachrichten ist kostenpflichtig. Es gibt daher eine Methode zur Berechnung der Kosten einer Nachrichtenübermittlung.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Nachrichten. Achten Sie darauf, dass gemeinsame Merkmale in (eventuell abstrakten) Oberklassen zusammengefasst werden. Notieren Sie Ihren Entwurf graphisch und verwenden Sie dazu die folgende Notation:



Geben Sie für jede Klasse ausschließlich den jeweiligen Namen und die Namen und Datentypen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen und Ein- und Ausgabetypen seiner Methoden an.

Matrikelnummer: **Name:**

b) Implementieren Sie in Java eine Methode **berechneKosten**. Die Methode bekommt als Parameter ein Array von Nachrichten übergeben und gibt die aufsummierten Kosten für die Übermittlung dieser Nachrichten zurück.

Gehen Sie dabei davon aus, dass das übergebene Array nicht **null** ist und dass es keinen **null**-Eintrag enthält. Kennzeichnen Sie die Methode mit dem Schlüsselwort **static**, falls angebracht.

Matrikelnummer: **Name:**

Aufgabe 4 (Programmierung in Java): **(6 + 4 + 12 + 4 = 26 Punkte)**

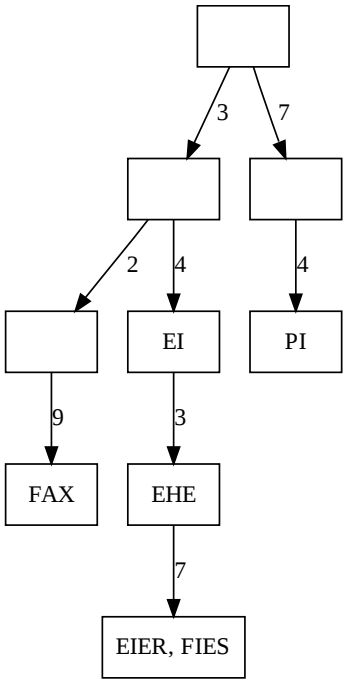
Zum einfacheren Schreiben von SMS (*Short Message Service*) mit dem Handy gibt es das T9-System (*Text on 9 keys*). Hierbei sind den Zifferntasten des Handys mehrere Buchstaben zugeordnet. So sind zum Beispiel die Buchstaben **j**, **k** und **l** der Ziffer **5** zugeordnet. Die weiteren Kombinationen entnehmen Sie bitte der Abbildung.

Für jedes Wort gibt es exakt eine Folge von entsprechenden Ziffern. Beispielsweise ergibt sich für das Wort **EIER** die Ziffernfolge **3,4,3,7**. Eine Ziffernfolge beschreibt nicht zwingend nur ein Wort, so dass beispielsweise das Wort **FIES** ebenfalls durch die Ziffernfolge **3,4,3,7** dargestellt wird.

Gegeben sei die folgende baumartige Datenstruktur **TNeun**, um ein Wörterbuch für das T9-System zu verwalten.

```
public class TNeun  {
  private Liste<String> woerter;
  private TNeun[] nachfolger;

  public TNeun() {
    this.woerter = new Liste<String>();
    this.nachfolger = new TNeun[10];
  }
}
```



In der Grafik sehen Sie, wie ein Beispiel-Wörterbuch mit dieser Struktur realisiert ist. Die Klasse **Liste** ist hierbei eine generische Liste. Für die durch leere Kästchen dargestellten Knoten enthält die jeweilige **woerter**-Liste kein Wort. In den Listen der Knoten auf Stufe *i*, wobei die Wurzel auf Stufe 0 ist, sind die Wörter der Länge *i* gespeichert. Die Ziffern, die an den Kanten auf dem Pfad zu einem Knoten stehen, geben die Ziffernfolge der in diesem Knoten gespeicherten Wörter an. Der Zugriff auf einen Knoten wird durch das **nachfolger**-Array des Vorgängerknotens ermöglicht, wobei der entsprechende Array-Index an der Kante notiert ist. Der Knoten mit dem Wort **EHE** ist also über **nachfolger[3]** des Knotens mit dem Wort **EI** erreichbar. Fehlende Nachfolger sind in diesem Array **null**-Einträge.

Verwenden Sie in den Implementierungen dieser Aufgabe keine Schleifen, sondern ausschließlich Rekursion. Versehen Sie Methoden- und Klassendeklarationen mit **static**, falls angebracht. Verwenden Sie die Lösungen vorheriger Teilaufgaben, falls angebracht.

a) Die Ziffernfolgen für Wörter dürfen nur die Ziffern **2** bis **9** enthalten. Implementieren Sie für eine entsprechende Fehlerbehandlung die Exception **UngueltigeZifferException**. Diese Klasse erweitert direkt die Klasse **Exception**. Jede Instanz dieser Exception soll bei der Erzeugung eine ungültige Zahl übergeben bekommen und diese speichern. Weiterhin soll die gespeicherte Zahl mit einer geeigneten Methode zurückgegeben werden können.

b) Erweitern Sie die Methode **pruefe** um die Benutzung der **UngueltigeZifferException** aus der vorherigen Teilaufgabe. Die Methode überprüft, ob eine bestimmte Zahl der Ziffernfolge einen gültigen Wert (2 bis 9) hat. Vervollständigen Sie den Code so, dass für eine ungültige Zahl eine entsprechende neue Instanz der Exception geworfen wird.

```
public static void pruefe(int[] folge, int index)


{
  int zahl = folge[index];
  if (zahl < 2 || zahl> 9) {


  }


}
```

c) Implementieren Sie die Methode **getWoerterListe** in der Klasse **TNeun**. Diese Methode soll für eine Ziffernfolge die entsprechende Wörterliste zurückgeben. Hierbei wird die Ziffernfolge als **int**-Array übergeben.

Nicht existierende Knoten auf dem durch die Ziffernfolge definierten Pfad werden hierbei mit Hilfe des **TNeun**-Konstruktors erzeugt. Diese neuen Knoten werden entsprechend in das **nachfolger**-Array eingetragen.

Die Methode gibt die im der Ziffernfolge entsprechenden **woerter**-Attribut gespeicherte Wörterliste zurück.

Beispielsweise liefert der Aufruf von **getWoerterListe** auf dem Wurzelknoten der Grafik mit der Ziffernfolge **3,4,3,7** das Array mit den Wörtern **EIER, FIES**. Für die Ziffernfolge **3,4,5** wird ein neuer Knoten als Nachfolger von dem Knoten mit dem Wort **EI** angelegt und die (leere) Wortliste dieses neuen Knotens zurückgegeben.

Gehen Sie hierbei davon aus, dass die übergebene Ziffernfolge nicht **null** ist und nur gültige Werte (also **2** bis **9**) enthält. Gehen Sie weiterhin davon aus, dass die **nachfolger**-Arrays groß genug für die Indizes bis **9** sind und nicht **null** sind, allerdings **null**-Werte enthalten können!

Hinweis: Zur Lösung kann es helfen, eine Hilfsmethode zu definieren und zu benutzen.

d) Implementieren Sie die Methode **addWort** in der Klasse **TNeun**, die ein übergebenes Wort an der Stelle hinzufügt, die durch die übergebene Ziffernfolge definiert ist. Diese Methode bekommt also sowohl ein Wort als auch eine Ziffernfolge (als Array von **int**-Werten) übergeben und hat keine Rückgabe. Auch hier sollen fehlende **TNeun**-Knoten entlang des Pfades hinzugefügt werden.

Gehen Sie hierbei davon aus, dass die übergebene Ziffernfolge nicht **null** ist und nur gültige Werte (also **2** bis **9**) enthält. Gehen Sie weiterhin davon aus, dass die **nachfolger**-Arrays groß genug für die Indizes bis **9** sind und nicht **null** sind, allerdings **null**-Werte enthalten können!

Um ein Wort in eine Wortliste hinzuzufügen, verwenden Sie die in der Klasse **Liste** vordefinierte Funktion **add** mit der folgenden Signatur (implementieren Sie diese Methode also **nicht**!):

```
public void add(E element)
```

Hierbei ist **E** der generische Typ der Elemente in der Liste. Gehen Sie davon aus, dass die **woerter**-Liste eines jeden Knotens nicht **null** ist.

Matrikelnummer: **Name:**

Aufgabe 5 (Funktionale Programmierung in Haskell): (3+2+(2+1+4+4) = 16 Punkte)

a) Geben Sie den allgemeinsten Typ der Funktionen f und g an, die wie folgt definiert sind. Gehen Sie davon aus, dass **1** den Typ **Int** hat.

```
f y x = 1
f [x:xs] y = x + (f [] (x:y))
```

```
g x y = (\y -> [y] ++ [1]) x
```

b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

- `foldr (\x y -> y) 0 "Viel Erfolg!"`

- `map (\x -> 1) [map (\x -> 1), foldr (\[x] y -> [2]) [3]]`

c) Wir betrachten eine Datenstruktur *Aufgabenliste*, in der zu erledigende und bereits erledigte *Aufgaben* gespeichert werden können. Zu jeder solchen Aufgabe ist ein Bezeichner (eine Zeichenfolge) bekannt. Zusätzlich ist jeder Aufgabe eine Priorität (als nicht-negative Zahl) zugeordnet. In der Datenstruktur wird für jede Aufgabe unterschieden, ob sie zu erledigen ist oder bereits erledigt wurde.

Die folgende Beispiel-Aufgabenliste enthält drei Aufgaben:

1. noch nicht erledigt: „Party“, Priorität 5
2. bereits erledigt: „Urlaub“, Priorität 25
3. noch nicht erledigt: „Klausur“, Priorität 20

1. Geben Sie die Definition einer Datenstruktur **Aufgabenliste** entsprechend obiger Beschreibung an. Verwenden Sie hierfür jeweils einen eigenen Konstruktor für zu erledigende und bereits erledigte Aufgaben. Verwenden Sie *nicht* die in Haskell vordefinierten Listen.

2. Geben Sie die im Beispiel dargestellte Aufgabenliste als einen **Aufgabenliste**-Ausdruck an. Übernehmen Sie hierbei die Aufgaben in der Reihenfolge des Beispiels.

3. Schreiben Sie eine Funktion **markiereErledigt**, mit der Sie einen nicht erledigten Eintrag als erledigt markieren können. Diese Eintrag wird durch seinen Bezeichner identifiziert. Diese Funktion erhält als Eingabe eine Aufgabenliste und einen Aufgaben-Bezeichner. Die Rückgabe ist eine Aufgabenliste, in der die erste unerledigte Aufgabe mit dem gesuchten Bezeichner als erledigt markiert ist. Wenn eine entsprechende unerledigte Aufgabe nicht vorhanden ist, gibt **markiereErledigt** also die unveränderte Aufgabenliste zurück.

   Geben Sie auch den Typ der Funktion an! Sie dürfen außer der für Zeichenfolgen vordefinierten Funktion == *keine* Hilfsfunktion definieren und benutzen.

4. Schreiben Sie eine Funktion **hoechstePrioritaet**, die in der Eingabeliste die *unerledigte* Aufgabe mit der höchsten Priorität findet und diese Priorität als Ergebnis zurückgibt. Für die Beispiel-Aufgabenliste liefert der Aufruf von **hoechstePrioritaet** also „20“. Falls die Aufgabenliste keine unerledigte Aufgabe enthält, soll „-1“ zurückgegeben werden.

   Geben Sie auch den Typ der Funktion an! Sie dürfen außer den für Zahlen vordefinierten Funktionen >, <, <=, >= *keine* Hilfsfunktion definieren und benutzen.

Matrikelnummer: **Name:**

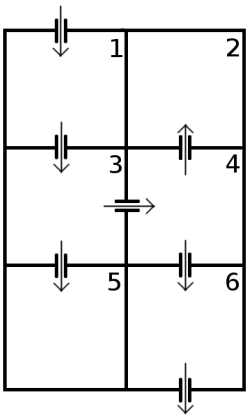
Aufgabe 6 (Logische Programmierung in Prolog): (3 + (3 + 4 + 4) + 4 = 18 Punkte)

a) Geben Sie für die folgenden Paare von Termen den allgemeinsten Unifikator an oder begründen Sie kurz, warum dieser nicht existiert.

f(b,g(h(Z)),Y,X) und **f(Z,g(X),Z,h(Y))**

g(h(g(Z)), X, g(Z)) und **g(X, h(Y), Y)**

b)



In dieser Teilaufgabe betrachten wir einfache *Labyrinthe*, die aus einzelnen *Räumen* bestehen. Jeder Raum hat exakt einen Eingang und an jeder der drei anderen Seiten entweder eine Wand, einen Durchgang zu einem anderen Raum oder einen Ausgang des gesamten Labyrinths. Für jeden Raum betrachten wir hierbei, was relativ zum Raumeingang sichtbar ist – also ob links / vorne / rechts eine Wand, ein Durchgang oder ein Ausgang ist. In der Grafik ist ein solches Labyrinth mit sechs Räumen dargestellt. Beispielsweise betrachten wir für den ersten Raum links eine Wand, vorne einen Durchgang (zu Raum 3) und rechts ebenfalls eine Wand.

In Raum 4 gibt es links einen Durchgang (zu Raum 2), vorne eine Wand und rechts einen weiteren Durchgang (zu Raum 6). Der einzige Weg zu einem Ausgang führt über die Räume 1, 3, 4 und 6 (in dieser Reihenfolge). Räume, Wände und Ausgänge stellen wir in Prolog durch die Atome **raum**, **wand** und **ausgang** dar. Bei Räumen wird die Darstellung der angrenzenden Räume (bzw. Wände, Ausgänge) links / vorne / rechts entsprechend als erstes / zweites / drittes Argument kodiert. Die Darstellung des 6. Raums ist also beispielsweise **raum**(**wand**, **ausgang**, **wand**). Das gesamte Beispiellabyrinth wird also durch

$$
\begin{aligned}
&\mathbf{raum}_1(\mathbf{wand}, \mathbf{raum}_3(\\
&\quad \mathbf{raum}_4(\mathbf{raum}_2(\mathbf{wand}, \mathbf{wand}, \mathbf{wand}), \mathbf{wand}, \mathbf{raum}_6(\mathbf{wand}, \mathbf{ausgang}, \mathbf{wand})),\\
&\quad \mathbf{raum}_5(\mathbf{wand}, \mathbf{wand}, \mathbf{wand}),\\
&\quad \mathbf{wand}\\
&), \mathbf{wand})
\end{aligned}
$$

dargestellt. Die Zahlen dienen nur zur Orientierung und sind **nicht** Teil der Prolog-Darstellung!

Matrikelnummer: **Name:**

1. Programmieren Sie das zweistellige Prädikat **pfad**, was zu einem Labyrinth alle Pfade zu einem Ausgang berechnet. Ein *Pfad* ist hierbei eine in Prolog vordefinierte Liste mit den Elementen **links**, **vorne** und **rechts** (wobei diese Angabe relativ zum Raumeingang gilt). Beispielsweise liefert die Anfrage „?- pfad(L, X).“ die Antwortsubstitution **X = [vorne, links, rechts, vorne]**, wenn **L** durch obige Darstellung des Beispiel-Labyrinths belegt ist.

2. Programmieren Sie das vierstellige Prädikat **maxDrei**, das für drei Zahlen die größte dieser drei Zahlen bestimmt. Beispielsweise liefert die Anfrage „?- maxDrei(5,27,12,X).“ die Antwortsubstitution **X = 27** und die Anfrage „?- maxDrei(5,5,5,5).“ das Ergebnis **true**. Sie dürfen nur das vordefinierte Prädikat > verwenden.

   Hinweis: Schreiben Sie zusätzlich ein dreistelliges Prädikat **maxZwei** zur Bestimmung des Maximums zweier Zahlen. Verwenden Sie dieses Prädikat für die Implementierung von **maxDrei**.

3. Programmieren Sie das zweistellige Prädikat **laenge**, was zu einem Labyrinth die maximale Anzahl der besuchbaren Räume bis zu einem beliebigen Ausgang oder zu einer beliebigen Wand berechnet. Beispielsweise liefert die Anfrage „?- laenge(L, X).“ die Antwortsubstitution **X = 4**, wenn **L** durch obige Darstellung des Beispiel-Labyrinths belegt ist.

   Hinweis: Verwenden Sie **maxDrei**. Sie dürfen nur die vordefinierten Prädikate + und **is** verwenden.

c) Erstellen Sie für das folgende Logikprogramm zur Anfrage **?- h(A, B).** den Beweisbaum und geben Sie alle Antwortsubstitutionen an. Sie dürfen dabei abbrechen, sobald die Anfrage aus mindestens vier Teilen (Atomen) bestehen würde. Kennzeichnen Sie solche Knoten durch „. . . “. Kennzeichnen Sie abgebrochene Pfade durch „↯“.

```
f(b).
h(X,X).
h(X,b) :- f(Y), h(X,Y), f(b).
h(X,a) :- f(X), h(X,c).
```