Universität Mannheim
Fakultät für Mathematik und Informatik
Lehrstuhl für Praktische Informatik IV
Professor Dr. W. Effelsberg

Scheinklausur Informatik
WS 99/00: Praktische Informatik I

Name: ..... Vorname: .....

Matrikel-Nr.: ..... Semester: ..... Fach: .....

Hinweise:

1. Bitte füllen Sie sofort den Kopf des Deckblattes aus.
2. Überprüfen Sie bitte Ihr Klausurexemplar auf Vollständigkeit (**12** Seiten).
3. Tragen Sie die Lösungen – soweit möglich – direkt in die Klausur ein.
4. Zugelassene Hilfsmittel: nicht programmierbarer Taschenrechner
5. Bearbeitungszeit: 90 Minuten.

Aufgabe	max. Punktzahl	Punkte
1	15	
2	15	
3	15	
4	15	
5	15	
6	15	
Summe	90	

Aufgabe 1: Logik *[4+5+6=15 Punkte]*

(a) [4 Punkte] Vereinfachen Sie die folgenden booleschen Ausdrücke (1 steht für „Wahr“, 0 für „Falsch“):

1. $a \wedge \neg a$
2. $a \wedge (a \vee b)$
3. $0 \leftrightarrow 0$
4. $(a \wedge b) \vee (a \wedge c)$

(b) [5 Punkte] Zeigen Sie mit Hilfe der Wertetabelle folgende Äquivalenz:

$$(a \rightarrow b) \vee (a \wedge \neg b) \Leftrightarrow 1$$

$a\ b$	$a \rightarrow b$	$a \wedge \neg b$	$(a \rightarrow b) \vee (a \wedge \neg b)$
0 0			
0 1			
1 0			
1 1			

(c) [6 Punkte] Zeigen Sie durch algebraische Umformungen folgende Äquivalenz. Geben Sie zu jeder Umformung die entsprechende Regel an.

$$(a \wedge b) \vee \neg(\neg a \vee b) \Leftrightarrow a$$

Aufgabe 2: Warteschlangen *[15 Punkte]*

Eine Warteschlange ist definiert als eine verkettete Liste, in die Elemente nur am Ende eingefügt und aus der sie nur am Anfang entnommen werden können. Effizienterweise implementiert man solche Warteschlangen als Arrays, wenn die Anzahl der Elemente fest ist. Das nachfolgende (unvollständige) Listing zeigt eine Java-Klassendeklaration für eine solche Warteschlange zur Verwaltung von positiven Integer-Werten. Die Warteschlange kann maximal 1000 Werte aufnehmen:

```
public class Warteschlange
{
  private static int[] daten= new int[1000]; // Datenfeld, das die
                                             // einzelnen Elemente der
                                             // Warteschlange enthaelt
  private static int anfang= 0; // Index auf den Anfang der Warteschlange
  private static int ende= 0;   // Index auf das Ende der Warteschlange

  /** ueberprueft, ob die Warteschlange leer ist
   *  Rueckgabe: true, wenn die Warteschlange leer ist
   *             false, sonst
   */
  public static boolean istLeer() {
  . . .
  }

  /** ueberprueft, ob die Warteschlange voll ist
   *  Rueckgabe: true, wenn die Warteschlange voll ist
   *             false, sonst
   */
  public static boolean istVoll() {
  . . .
  }


  /** fuegt ein Datenelement am Ende der Warteschlange ein
   *  Rueckgabe: true, wenn das Einfuegen erfolgreich ist
   *             false, sonst
   */
  public static boolean einfuegen(int element) {
  . . .
  }


  /** loescht ein Datenelement am Anfang der Warteschlange
   *  Rueckgabe: das geloeschte Element,
   *             -1 wenn die Warteschlange leer ist
   */
  public static int loeschen() {
  . . .
  }
}
```

Geben Sie eine Implementierung der Methoden `istLeer`, `istVoll`, `einfuegen` und `loeschen` an!

Aufgabe 3: Binärbäume *[5+10=15 Punkte]*

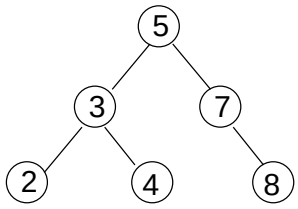
Gegeben sei folgende Deklaration zur Verwaltung positiver ganzer Zahlen innerhalb eines Binärbaums:

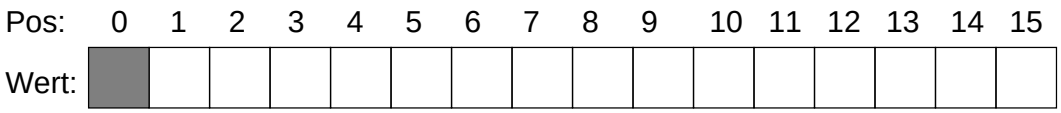
```
public class BinaerBaum {
  final static int MAXGROESSE = 1000;
  int[] daten = new int [MAXGROESSE+1];

  public int berechneHoehe (int position) {
    ...
  }
}
```

Die Konstruktion des Baums unterliegt dabei folgenden Regeln:

- Knoten 1 ist die Wurzel. Das dazugehörige Datenelement befindet sich im Feld `daten` an Position 1. (Position 0 des Felds bleibt unbenutzt.)
- Der Knoten $2i$ ist das linke Kind des Knotens i. Das dazugehörige Datenelement befindet sich im Feld `daten` an der Position $2i$. Existiert kein linkes Kind, so wird an der Position $2i$ der Wert (-1) eingetragen.
- Der Knoten $2i+1$ ist das rechte Kind des Knotens i. Das dazugehörige Datenelement befindet sich im Feld `daten` an der Position $2i+1$. Existiert kein rechtes Kind, so wird an der Position $2i+1$ der Wert (-1) eingetragen.

(a) [5 Punkte] Bilden Sie untenstehenden Binärbaum mit Hilfe der Konstruktionsregeln in das darunterliegende Feld ab!





(b) [10 Punkte] Entwickeln Sie eine rekursive Methode in Java, welche die Höhe des Binärbaums berechnet. Verwenden Sie dazu die oben angegebenen Deklarationen. Gehen Sie davon aus, daß ein leerer Baum die Höhe 0, ein Baum, der nur aus der Wurzel besteht die Höhe 1 hat usw.

Aufgabe 4: Endliche Automaten *[12+3=15 Punkte]*

Ein Geheimdienst lauscht an einer Datenleitung, über die beliebige Folgen von 0 und 1 übertragen werden. Die Agenten haben herausgefunden, daß die für sie interessanten Informationen mit der Bitfolge 101 oder mit der Bitfolge 110 beginnen. Unterstützen Sie die Agenten, indem Sie einen deterministischen endlichen Automaten entwerfen, der seinen Endzustand erreicht, sobald eine der beiden oben genannten Bitfolgen im Datenstrom auftritt.

(a) [12 Punkte] Geben Sie den Automaten grafisch an.

(b) [3 Punkte] Geben Sie für die Bitfolge 01001110 die Zustandsübergänge Ihres Automaten an.

Aufgabe 5: Komplexität *[15 Punkte]*

Die Methode `bsuche` ermittelt, ob sich ein ganzzahliger Suchwert in einem aufsteigend sortierten Feld befindet:

```
public boolean bsuche (int[] feld, int suchwert,
                       int linkeGrenze, int rechteGrenze) {
  if (linkeGrenze > rechteGrenze)
     return false;

  int mitte = (linkeGrenze + rechteGrenze) / 2;

  if (suchwert == feld[mitte]) {
     return true;
  } else if (suchwert < feld[mitte]) {
     return bsuche (feld, suchwert, linkeGrenze, mitte - 1);
  } else {
     return bsuche (feld, suchwert, mitte + 1, rechteGrenze);
  }
}
```

Beispiel für einen Aufruf:

```
...
int[] feld = { 1, 2, 3, 4 };
int suchwert = 10;
boolean resultat = bsuche(feld, suchwert, 0, feld.length-1);
...
```

Berechnen Sie die Zeitkomplexität der Methode `bsuche`. Stellen Sie dazu die entsprechende Rekurrenzrelation auf, beweisen Sie diese für beliebige Rekursionstiefen, ermitteln Sie die geschlossene Darstellung und geben Sie das O-Kalkül an.

Gehen Sie von folgenden Einschränkungen aus:

- Der Suchwert ist größer als alle Werte, die sich im zu durchsuchenden Feld befinden.
- Die Größe des Felds ist eine Potenz von 2.

Aufgabe 6: C-Programmierung *[5+10=15 Punkte]*

Nachfolgend ist ein Ausschnitt aus einer C-Implementierung einer einfach verketteten Liste abgedruckt.

```
struct ListenElement {
  char* name;
  struct ListenElement* nachfolger;
};

typedef struct ListenElement ListenElement;


void wasmacheich (ListenElement* pos)
{
  ListenElement* min;
  char* tmp;
  ListenElement* i;

  if (pos != NULL) {
      min = pos;
      for (i = pos; i != NULL; i = i->nachfolger) {
        /* strcmp (s1, s2) vergleicht die Strings s1 und s2
           Rueckgabewert: -1, s1 < s2
                          0, s1 == s2
                          1, s1 > s2
           Hinweis: Die entsprechende Java-Methode
                    heisst s1.compareTo(s2) */
        if (strcmp (i->name, min->name) < 0) {
          min = i;
        }
      }

      tmp = min->name;
      min->name = pos->name;
      pos->name = tmp;

      wasmacheich (pos->nachfolger);
  } /* if (pos != NULL) */
}
```

(a) [5 Punkte] Was leistet die Funktion `wasmacheich` in obigem Quelltext? Begründen Sie Ihre Antwort, indem Sie die Funktionsweise des Algorithmus kurz beschreiben.

(b) [10 Punkte] Geben Sie eine Java-Implementierung dieser Funktion sowie der zugehörigen Datenstruktur an.