**Hinweise zur Bearbeitung der Klausur zum Kurs 1661 Datenstrukturen I**

Bitte lesen Sie sich diese Hinweise vollständig und aufmerksam durch, bevor Sie mit der Bearbeitung der Klausur beginnen. Beachten Sie insbesondere die Punkte 7 und 8!

1. Die Klausurdauer beträgt 2 Stunden.
2. Prüfen Sie bitte die Vollständigkeit Ihrer Unterlagen. Die Klausur umfaßt:
   - 2 Deckblätter
   - diese Hinweise
   - 1 Formblatt für eine Teilnahmebescheinigung zur Vorlage beim Finanzamt
   - 3 Aufgaben auf den Seiten 2 – 4
3. Bevor Sie mit der Bearbeitung der Klausuraufgaben beginnen, füllen Sie bitte die folgenden Teile der Klausur aus:
   (a) *sämtliche* Deckblätter mit Name, Anschrift sowie Matrikelnummer. Markieren Sie vor der Abgabe auf allen Deckblättern die von Ihnen bearbeiteten Aufgaben.
   (b) die Teilnahmebescheinigung, falls Sie diese wünschen.
4. Schreiben Sie Ihre Lösungen auf ihr eigenes Papier (DIN A4) und nicht auf die Seiten mit den Aufgabenstellungen. Heften Sie vor Abgabe der Klausur die Deckblätter (und evtl. die Teilnahmebescheinigung) an Ihre Bearbeitung.
5. Schreiben Sie bitte auf jedes Blatt oben links Ihren Namen und oben rechts Ihre Matrikelnummer. Numerieren Sie Ihre Seiten bitte durch.
6. Wenden Sie bei der Lösung der Aufgaben, soweit dies möglich ist, die Algorithmen und Notationen aus den Kurseinheiten an.
7. Schreiben Sie, wenn ***Algorithmen*** gefordert sind, keine kompletten PASCAL- oder JAVA-Programme, sondern beschränken Sie sich auf die wesentlichen Teile des Algorithmus, die z.T. auch in Prosa formuliert werden können. Formulieren Sie Algorithmen aber so, daß die elementaren Einzelschritte erkennbar werden.

   Sparen Sie aber bei solchen Aufgaben nicht mit Kommentaren. Wenn Ihre Lösung aufgrund fehlender Kommentare nicht verständlich ist, führt das zu Punktabzug.
8. Vermeiden Sie in jedem Fall bei der Definition von Funktionen in ***Algebren*** PASCAL- oder JAVA-Programme sowie Algorithmen. Geben Sie lediglich mathematische Definitionen an wie sie im Kurstext verwandt worden sind!

   Ebenso sind Mengendefinitionen in mathematischer Mengennotation durchzuführen. Geben Sie auf keinen Fall Datentypdefinitionen an, und verwenden Sie keine konkreten Datenstrukturen!
9. Als Hilfsmittel sind nur unbeschriebenes Konzeptpapier und Schreibzeug (kein Bleistift!) zugelassen.
10. Es sind maximal 66 Punkte erreichbar. Zur Erlangung eines Übungsscheins bzw. Zertifikats benötigen Sie etwa 33 Punkte.

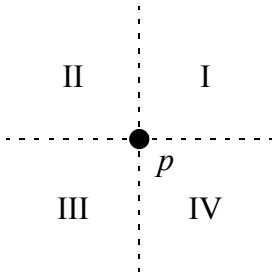
*19 Punkte*

## Aufgabe 1 Algebra für Punktmengen

Eine Algebra für eine Menge von Punkten im zweidimensionalen Raum ist das Thema dieser Aufgabe. Neben den üblichen Operationen auf solchen Mengen soll es nun unter anderem Operationen geben, die Teilmengen dieser Punktmengen berechnen und zurückgeben. Für die Spezifikation der Algebra können Sie auf eine Funktion *dist*: *point* × *point* → *real* zurückgreifen, die den Abstand zwischen zwei Punkten berechnet.

Folgende Operationen sollen unterstützt werden:

- *empty*: erzeugt eine leere Punktmenge
- *newPoint*: erzeugt einen neuen Punkt
- *insert*: fügt einen Punkt in eine Punktmenge ein
- *delete*: löscht einen Punkt aus einer Punktmenge
- *member*: gibt *true* zurück, falls der gegebene Punkt in der Punktmenge enthalten ist, sonst *false*
- *isEmpty*: gibt *true* zurück, falls die Punktmenge leer ist, sonst *false*
- *horizontalSize*: gibt die horizontale Ausdehnung der Punktmenge zurück
- *verticalSize*: gibt die vertikale Ausdehnung der Punktmenge zurück
- *quadrant*: gibt die Teilmenge einer Punktmenge zurück, die in dem durch eine Zahl angegebenen Quadranten liegt (ohne Koordinatenachsen), wobei der Ursprung des Koordinatensystems durch einen übergebenen Punkt bestimmt wird (siehe Bild)




- *windowQuery*: gibt die Teilmenge einer Punktmenge zurück, die durch ein Fenster eingeschlossen ist (die Punkte liegen nicht auf der Grenze); das Fenster wird durch zwei Punkte, den oberen linken und den unteren rechten Punkt, bestimmt
- *circleQuery*: gibt die Teilmenge einer Punktmenge zurück, die durch einen Kreis eingeschlossen ist (die Punkte liegen nicht auf der Grenze); der Kreis wird durch seinen Mittelpunkt und seinen Radius gegeben

*5 Punkte* (a) Geben Sie die Sorten und Operationen für die Algebra an.

*7 Punkte* (b) Geben Sie Trägermengen und Funktionen für die Algebra an.

*7 Punkte* (c) Jeder Punkt soll nun einen eindeutigen Identifikator bekommen, z.B. einen Ortsnamen, und Punkte sollen nun wo möglich über ihre Identifikatoren angesprochen werden (z.B. bei *delete*). Der Identifikator eines Punktes wird bei der Konstruktion des Punktes vom Nutzer übergeben. Beachten Sie, daß über die Funktionen der Algebra geregelt werden muß, daß keine Duplikate in die Punktmenge aufgenommen werden. Für die letzten drei Operationen sollen

nun nicht länger Punkte übergeben werden, sondern Identifikatoren für Punkte aus der Punktmenge spezifiziert werden. Ändern Sie die Algebra entsprechend ab.

## Aufgabe 2 Sortierverfahren *24 Punkte*

(a) *4 Punkte*

Gegeben sei eine Listenimplementierung, welche ein ausgewiesenes Element (im folgenden als aktuelles Element bezeichnet) besitzt. Diese unterstützt die folgenden Operationen:

- *first* : setzt das aktuelle Element auf den Anfang der Liste
- *next* : setzt aktuelles Element auf das nächste Listenelement, falls vorhanden
- *prev*: setzt aktuelles Element auf seinen Vorgänger, falls vorhanden
- *get*: liefert den Inhalt des aktuellen Listenelements
- *set*: setzt den Inhalt des aktuellen Listenelements auf den Wert des Arguments
- *onEnd*: liefert true, wenn das aktuelle Element das letzte in der Liste ist
- *position*: gibt die Position des aktuellen Listenelements zurück
- *append*: hängt ein Element ans Ende der Liste an
- *emptyList*: erzeugt eine neue, leere Liste

Beschreiben Sie, wie Heapsort geändert werden muß, damit Mengen, die in solchen Listen gespeichert sind, direkt (also ohne Kopieren in eine andere Darstellung) sortiert werden können. Eine formale Angabe des Algorithmus ist nicht erforderlich.

Überlegen Sie, ob Heapsort ein geeignetes Verfahren ist, um solche Listen zu sortieren? Geben Sie dazu die Laufzeit an und begründen Sie diese.

(b) *8 Punkte*

Gegeben sei die Zahlenfolge:

123 - 15 - 61 - 500 - 16 - 32 - 94 - 68 - 541 - 983 - 673 - 981 - 78

Sortieren Sie diese Folge mittels Heapsort in aufsteigender Reihenfolge. Zeichnen Sie zuerst den Baum, der diese Folge beinhaltet. Bauen Sie dann den Heap auf. Geben Sie dabei die Pfade an, auf denen die Elemente einsinken. Geben Sie den Heap als Baum dargestellt an.
Beschreiben Sie für die zweite Phase, welche Elemente getauscht werden und geben Sie wieder die Pfade des Einsinkens der Elemente an.

(c) *6 Punkte*

Geben Sie einen auf Radixsort basierenden Algorithmus zum Sortieren von (fast) beliebig langen Strings an. Ein einzelnes Zeichen sei dabei durch seinen ASCII-Wert im Bereich von 1 bis 127 bestimmt. Das Zeichen an einer beliebigen Position eines String kann über die Funktion *get*: $string \times int \rightarrow int$ ermittelt werden. Ist der übergebene *int*-Wert größer als die Länge des Strings, die sich über *length*: $string \rightarrow int$ bestimmen läßt, wird 0 zurückgegeben.
Welche Laufzeit hat dieser Algorithmus in Abhängigkeit von der Größe der

zu sortierenden Menge und der Längen der in dieser Menge enthaltenen Strings? Die Strings seien dabei in einer Liste gegeben, die alle Operationen aus Aufgabenteil (a) beherrscht. Wann ist dieser Algorithmus anderen guten Sortierverfahren vorzuziehen?

*6 Punkte* (d) Sortieren Sie die in Aufgabenteil (b) gegebene Zahlenfolge mittels Radixsort.

## *23 Punkte* Aufgabe 3 Hashing

*15 Punkte* (a) Der Datentyp *dictionary* soll mittels offenem Hashing mit $m$ Behältern implementiert werden. Implementieren Sie Algorithmen für die Operationen *insert, delete* und *member*. Dabei soll eine Reorganisation der Hashtabelle stattfinden, sobald $n / m > c$ gilt ($n$: Anzahl der im Dictionary gespeicherten Elemente, $c$: eine beliebige Konstante). Geben Sie zusätzlich eine Funktion $t$ für die durchschnittliche Laufzeit der Operation *member* an, wenn sich Anfragen mit der Wahrscheinlichkeit $p$ auf einen in der Hashtabelle gespeicherten Wert beziehen.

Benutzen Sie in den Algorithmen die folgenden Typen, Variablen und Konstanten:

**type** *listelem* = **record** *value*: *elem*; *next*: ↑*listelem* **end**;
**type** *set* = ↑**array of** ↑*listelem*;
**const** *c*; **var** *n*, *m*;

*8 Punkte* (b) Beim geschlossenen Hashing mit $m$ Behältern, die genau ein Element aufnehmen können, ist die Kollisionswahrscheinlichkeit $P_k$ ein bedeutender Faktor für die durchschnittliche Laufzeit der *dictionary* Operationen. Leiten Sie, unter der Annahme, daß die Hashfunktion ideal ist, eine Formel für $P_k$ her.

Führen Sie eine Abschätzung durch, die es Ihnen ermöglicht, zu einer gegebenen Anzahl $n$ ein $m$ zu bestimmen, so daß $P_k < x$ gilt, für ein vorgegebenes $x \in ]0, 1]$. Gesucht ist also eine Beziehung zwischen $m$, $n$ und $x$.

*Hinweis*: Bei einer idealen Hashfunktion ist die Wahrscheinlichkeit $P(i)$ für das Ereignis, daß der $i$-te Schlüssel auf einen freien Behälter abgebildet wird gegeben durch $P(i) = (m - i + 1) / m$. Ersetzen Sie bei der Berechnung an geeigneter Stelle einen Term durch eine möglichst große untere Schranke, die es Ihnen erlaubt, die entstehende Ungleichung nach $m$ aufzulösen.