Kanaeva / Schweppe, FUB Informatik

Probeklausur zu Algorithmen und Programmieren III, WS 2001/2

- Jokerpunkte -

Bearbeitungszeit 3 Stunden.

Ausgabe: 6.02.02
Abgabe: 14.02.02 bis 14.00 Uhr

Spezifikationen & Hoare- Kalkül:

Aufgabe (3 P) (10 min.)
Gegeben sei das folgende Programmstück:

```
int i,j,x;
read(x);
i = 1;
j = 1;
while (i < 10) {
  j = j *i;
  i = i + 1;
  if (i == x) break;
}
```

Welche der folgenden Ausdrücken können als Effekte für das gegebene Programmfragment gelten:

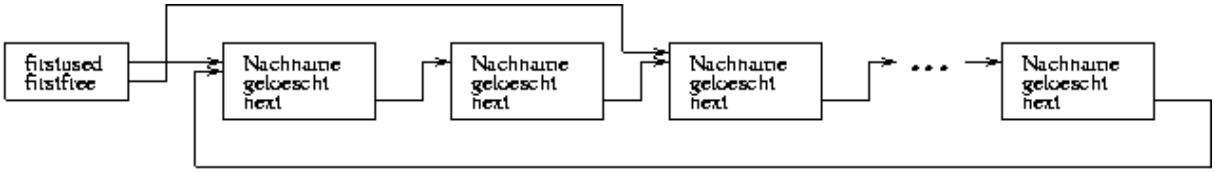
(A) $(j == (x-1)!) \wedge (i \geq x)$
(B) $(j == 9!) \wedge (i == 10)$
(C) $(j == 10\ !) \wedge (i == 10)$
(D) $((j == 10\ !) \wedge (i == 10)) \vee ((j == (x-1)!) \wedge (i == x))$
(E) $((j == 9!) \wedge (i \geq 10)) \vee ((j == (x-1)!) \wedge (i == x))$

Aufgabe (6 P) (25 min.)
Für ein Feld `b[0..n-1]` bestehend aus n Zahlen soll das Produkt der Zahlen berechnet werden. Formulieren Sie die Voraussetzung und den Effekt für dieses Problem. Beweisen Sie dann die Korrektheit des folgenden Programmstücks mit Hilfe des Hoare-Kalküls.

```
p = 1;
i = 1;
while (i < n) {
  p = p * b[i];
  i = i + 1;
}
```

ADTs & Repräsentationen:

Aufgabe (3 + 6 + 2 + 1 P) (30 min.)
Wir wollen eine (unendliche) Menge von Personen verwalten. Nötig ist dafür eine Datenstruktur, die es erlaubt, neue Personen einzufügen. Diese Datenstruktur soll ein Ringpuffer sein, der wie in der Abbildung gezeigt, aufgebaut ist.



`RingPuffer` ist eine Verwaltungsstruktur, über die ein Zugriff auf das erste leere Element und auf das zuerst eingefügte belegte Element möglich ist. Die Elemente selbst sind vom Typ `RingElem`. In ihnen werden die Namen der Personen und die Information, ob die Person gelöscht wurde oder nicht, eingetragen. Weiterhin erlauben sie den Zugriff auf das jeweils nächste Element. Dabei ist das letzte Element mit dem ersten verbunden, um einen Ring zu bilden. Wenn eine neue Person gespeichert werden soll, wird sie in dem ersten leeren Element eingetragen und der Verweis auf das erste leere Element wird ein Element weiter gesetzt. Die folgenden Methoden der Personenmenge seien wie folgt umgangsprachlich spezifiziert:

- `ringInsert(anzahl):` Im Ring wird für `anzahl` neuen Personen Platz gemacht. Dafür werden `anzahl` (leere) Elemente in den Ring eingefügt, und zwar in dem bisher nicht schon von Personen belegten Bereich.
- `ringNewPerson(nachname):` Eine neue Person wird in dem Ring gespeichert. Falls keine belegbaren Elemente mehr existieren, wird `ringInsert` ausgeführt, um Platz zu schaffen.
- `ringNextPerson(nachname):` Der Name der nächsten Person wird zurückgegeben. Das von ihm im Ringpuffer belegte Element wird durch Weitersetzen von `firstused` gelöscht. Bei den unbelegten oder gelöschten (gelöscht = = true) Elementen werden leere Strings anstatt Nachnamen zurückgegeben.

Lösen Sie bitte folgende Aufgaben:

1. Definieren Sie Java-Klassen, die nötig sind, um die in den Abbildungen gezeigte Datenstruktur repräsentieren zu können. Dazu gehören `RingPuffer` und `RingElem` und evtl. benötigte Hilfstypen.
2. Implementieren Sie die Methode `ringInsert.`
3. Geben Sie die Repräsentationsinvariante an.
4. Geben Sie die Abstraktionsfunktion an.

Objektorientierung & Polymorphie:

Aufgabe (3 + 2 P) (20 min.)
(a) Analysieren Sie das folgende java-Programm:

```
class A {
  int i,j;

  public A(int ii) {
    i = ii;
    j = ii + 1;
  }
  public boolean lessThanEq(Object a1) {
    return i <= ((A) a1).i;
  }
  public void foo() {
    System.out.println("A: "+i+"  "+j);
  }
}
```

```
class C extends A {
  int i,m;

  public C(int ii) {
    super(ii*2);
    i = ii;
    m = ii + 1;
  }
  public void foo() {
    System.out.println(“C: “+i+”  ”+j+”  “+m);
  }
}

class TestA {
  public static void main(String[] args) {
    A a1,a2;
    a1 = new A(5);
    System.out.println(a1.i);
    System.out.println(a1.j);
    System.out.println(a1.m);
    a2 = a1;
    a1.foo();
    a1.i = 2;
    System.out.println(a2.i);
    a2.foo();
    a2 = null;
    C c = new C(333);
    c = a1;
    if (((A) a1).lessThanEq(A) c))
       System.out.println(“a1 ist kleiner als c”);
    System.out.println(a2.i);
  }
}
```

Welche Anweisungen führen zu Übersetzungsfehlern, welche zu Laufzeitfehlern? Geben Sie bitte für Ihre Antworten kurze Begründungen an. Welche Ausgabe liefert das Programm, wenn man die fehlerhaften Anweisungen auskommentiert?

(b) Wozu braucht man in Java die Typumwandlung (mehrere Antworten sind möglich):

(A) Um „information hiding“-Prinzip zu ermöglichen
(B) Um das dynamische Binden auszuschließen
(C) Für die Umwandlung einer Superklassen-Referenz in eine Subklassen-Referenz
(D) Um die fehlende Generizität zu ermöglichen
(E) Für die Umwandlung einer Subklassen-Referenz in eine Superklassen-Referenz

Bäume:

Aufgabe (6 P) (20 min.)
Für die Klasse `BinSTree`:

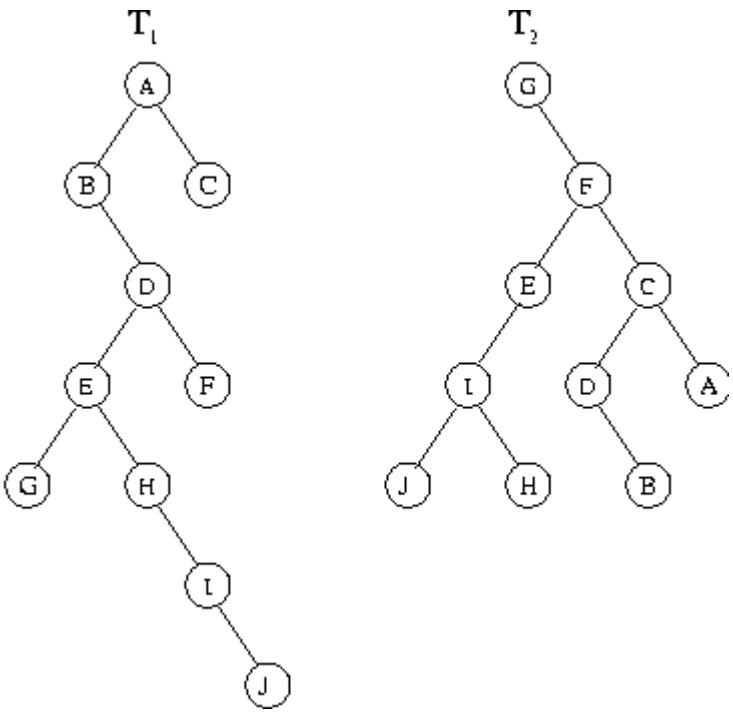
```
class BinSTree {
    class TreeNode {
      int elem;
      TreeNode left = null, right = null;
      TreeNode (int e) { elem = e; }
      TreeNode getLeft () { return left; }
      TreeNode getRight () { return right; }
      int getElement () { return elem; }
      void setLeft (TreeNode n) { left = n; }
      void setRight (TreeNode n) { right = n; }
      boolean isLeaf () { return left == null && right == null;  }
```

```
  }
  public int[] intervall(int low, int high) { ... }

  ...
}
```

soll eine Methode `intervall(int low, int high)` geschrieben werden, die alle Werte x zwischen ihren Argumenten low und high liefert. Das Programm soll möglichst wenig Knoten des Baums traversieren. Verwenden Sie keinen inorder-Iterator. Geben Sie die mittlere Laufzeit der Methode und die Laufzeit im schlechtesten Fall an.

Aufgabe (3 P) (15 min.)
Gegeben seien zwei binäre Bäume T_1 und T_2:



Welche Traversierungen dieser zwei Bäume werden das gleiche Ergebnis liefern?

	T_1	T_2
(A)	Preorder	Postorder
(B)	Postorder	Inorder
(C)	Postorder	Postorder
(D)	Inorder	Inorder
(E)	Postorder	Preorder

Aufgabe (6 P) (20 min.)
Sei T ein nicht leerer binärer Baum, in dem jeder Knoten entweder ein Blatt ist oder zwei Kinder hat (also innerer Knoten). Es seien weiter

- n(T) die Anzahl der inneren Knoten des Baums T (wo n(T) = 0, falls T ein Blatt ist),
- h(T) die Höhe des Baums T (wo h(T) = 0, falls T ein Blatt ist
- T_L der linke Teilbaum von T und T_R der rechte Teilbaum von T.

Wenn F eine Funktion mit der folgenden Vorschrift ist:

F(T) = 0, falls T ein Blatt ist
F(T) = $F(T_L) + F(T_R) + \min(h(T_L), h(T_R))$, sonst,

dann gilt F(T) =

(A) $n(T) + h(T) - 1$
(B) $n(T) + h(T)$
(C) $n(T) + h(T) + 1$
(D) $n(T) - h(T) - 1$
(E) $n(T) - h(T)$

Begründen Sie bitte Ihre Entscheidung.

Hashing:

Aufgabe (4 P) (15 min.)
Gegeben sei eine Hash-Tabelle D mit der offenen Adressierung und mit vier Buckets (verkettete Listen), die mit 0,1,2,3 durchnumeriert sind. Die Integer-Zahlen werden in diese Tabelle mittels Hash-Funktion h(x) = x mod 4 eingefügt. Wenn man die Folge der perfekten Quadrate 1,4,9, ..., i^2, ... in die anfangs leere Tabelle D einfügt, dann gilt folgendes:

(A) Zwei der Buckets bleiben leer, die anderen zwei bekommen die näherungsweise gleiche Anzahl von Einträgen.
(B) Alle Buckets werden (näherungsweise) gleich voll sein.
(C) Alle Einträge werden in einem bestimmten Bucket gespeichert.
(D) Alle Buckets werden gefüllt, aber die Differenz zwischen der minimalen Bucketgröße und der maximalen wird wachsen.
(E) Drei der Buckets werden (näherungsweise) gleich groß sein, der vierte wird leer bleiben.

Begründen Sie bitte Ihre Wahl.

Datenstrukturen & Laufzeit:

Aufgabe (3 P) (10-15 min.)
Angenommen, es gibt N Datensätze, die gespeichert werden müssen. Jeder Datensatz wird eindeutig mit seiner Identifikationsnummer versehen. Betrachten Sie folgende Möglichkeiten um die N Datensätze zu speichern:

(I) Ein nach den Identifikationsnummern sortierter Array (mit binärer Suche).
(II) Eine einfach verkettete Liste, die nach den Identifikationsnummern sortiert ist.
(III) Eine nicht sortierte einfach verkettete Liste.
(IV) Ein AVL-Baum mit den Identifikationsnummern als Schlüssel.

Wählen Sie für die Datenstrukturen (I-IV) die richtige mittlere Laufzeit, die man braucht, wenn man für eine gegebene Identifikationsnummer nach dem entsprechenden Datensatz sucht.

	I	II	III	IV
(A)	O(log N)	O(N)	O(N)	O(log N)
(B)	O(N)	O(log N)	O(N)	O(N)
(C)	O(log N)	O(log N)	O(N)	O(1)
(D)	O(N)	O(N)	O(N)	O(log N)
(E)	O(N)	O(log N)	O(log N)	O(1)