TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen WS 12/13
Einführung in die Informatik I **Wiederholungsklausur**
Prof. Dr. Helmut Seidl, A. Lehmann, A. Herz, A. Reuß 12.4.2013

Name	Vorname	Studiengang	Matrikelnummer
Hörsaal	Reihe	Sitzplatz	Unterschrift

Allgemeine Hinweise:

- **Bitte füllen Sie die oben angegebenen Felder vollständig aus und unterschreiben Sie!**
- **Schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!**
- **Es sind keine Hilfsmittel zugelassen. Verwenden Sie nur die bereitgestellten Klausurbögen und einen dokumentenechten blauen oder schwarzen Stift!**
- **Was nicht bewertet werden soll, kennzeichnen Sie bitte eindeutig durch Durchstreichen.**
- **Die Arbeitszeit beträgt 120 Minuten.**
- **Prüfen Sie, ob Sie alle 12 Seiten erhalten haben.**
- **In dieser Klausur können Sie insgesamt 43 Punkte erreichen. Zum Bestehen werden 21 Punkte benötigt.**
- **Falls Ihnen der Platz bei einer Aufgabe nicht ausreicht, verwenden Sie die leere Rückseite der Aufgabe.**

1	2	3	4	5	6	7	8	$\sum$

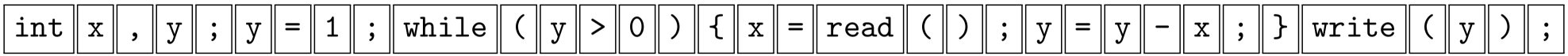
.....
Erstkorrektur

.....
Zweitkorrektur

Aufgabe 1 [10 Punkte] **MiniJava - Syntaxbaum**

Zeichnen Sie den Syntaxbaum zum folgenden MiniJava-Programm (Grammatik siehe Anhang am Ende der Klausur). Platz für den Syntaxbaum finden Sie auf der nächsten Seite!

```
int x, y;
y = 1;
while (y > 0) {
  x = read();
  y = y - x;
}
write(y);
```


int x , y ; y = 1 ; while (y > 0) { x = read ( ) ; y = y - x ; } write (y) ;

Aufgabe 2 [6 Punkte] **RLE**

Das *Run Length Encoding* (RLE) ist eine einfache Möglichkeit zur Datenkompression, bei der die Elemente einer Sequenz jeweils durch das Zeichen selbst sowie die Anzahl, wie oft es nacheinander vorkommt, kodiert werden. Beispielsweise lautet der String `aaabbcaa` in kodierter Form `a3b2c1a2`.

Implementieren Sie die Methode `static String rlEncode(String txt)`. Die Methode soll den String `txt` in RLE-kodierter Form zurückgeben. Gehen Sie davon aus, dass `txt` mindestens ein Zeichen und keine Ziffern enthält. *Hinweise*:

- `s.charAt(i)` liefert das Zeichen am Index `i` des Strings `s`.
- `s.length()` liefert die Länge des Strings `s`.

```
static String rlEncode(String txt) {









}
```

Aufgabe 3 [4 Punkte] **Nadel im sortierten Heuhaufen**

Implementieren Sie *rekursiv* die *Binäre Suche* in der Methode

```
static boolean findRec(int[] values, int elt, int left, int right)
```

Die Methode soll `true` zurückgeben, falls der Wert `elt` in `values` an einem Index zwischen (jeweils einschließlich) `left` und `right` enthalten ist, sonst `false`. Die Indizes `left` und `right` repräsentieren also den zu durchsuchenden Bereich. Gehen Sie davon aus, dass `values` mindestens ein Element enthält und die Elemente in aufsteigender Reihenfolge sortiert sind.

```
static boolean findRec(int[] values, int elt, int left, int right) {
    if (left > right) {
        return [                              ];
    }

    int pivotIndex = [                              ];


    if ( [                              ] ) {
        return true;
    }
    if (values[pivotIndex] > elt) {
        return findRec(values, elt, [              ] , [              ] );

    if (values[pivotIndex] < elt) {
        return findRec(values, elt, [              ] , [              ] );
    }
    return [                    ] ;
}
```

Aufgabe 4 [4 Punkte] **Erbsenzählerei**

Implementieren Sie die Methode

```
static int count(int[] unsorted)
```

Die Methode soll zählen, wie oft im Array `unsorted` eine größere Zahl links von einer kleineren Zahl steht. *Links* bedeutet hierbei, dass der zugehörige Index kleiner ist. Die Anzahl entsprechender Index-Paare soll zurückgegeben werden. Gehen Sie davon aus, dass gilt: `unsorted != null`. Beispiel: Das Ergebnis für das Array `{2,3,0,1}` ist 4.

```
static int count(int[] unsorted) {









}
```

Aufgabe 5 [5 Punkte] **Java verstehen**

Betrachten Sie die folgende Funktion `rec`.

```
static int rec (int pull) {
    System.out.print (pull + ",");
    for (int next = pull - 1; next > 1; next = next - 1) {
        if (next % 2 == 0) {
            return rec (next);
        }
    }
    return pull;
}
```

Welche *Ausgaben* erzeugen die Aufrufe der Funktion mit den Werten 6 bzw. 11?

a) Ausgabe für `rec(6)`:

b) Ausgabe für `rec(11)`:

Aufgabe 6 [5 Punkte] **Schmeckt fast so gut wie Filter-Kaffee**

Das generische Interface `Predicate` deklariert eine Methode `applies`, die dazu dient, ein gegebenes Objekt `toWhat` auf eine bestimmte Eigenschaft hin zu überprüfen. Sie gibt `true` zurück, wenn die Eigenschaft zutrifft, sonst `false`:

```
public interface Predicate<T> {
    boolean applies(T toWhat);
}
```

Gegeben sei zudem die generische Klasse `List`:

```
public class List<T> {
    private final T    info;
    private final List next;

    public List(T info, List next) {
        this.info = info;
        this.next = next;
    }
}
```

Implementieren Sie die Methode

```
static <S> List<S> filter(List<S> node, Predicate<S> pred)
```

Die Methode soll eine neue Liste zurückgeben, die genau diejenigen Elemente der Liste `node` enthält, für die das Prädikat `pred` zutrifft. Die leere Liste sei durch `null` repräsentiert.

```
static <S> List<S> filter(List<S> node, Predicate<S> pred) {









}
```

Aufgabe 7 [9 Punkte] **Threads**

In dieser Aufgabe soll in einem Array die größte Zahl gesucht werden. Dazu soll zunächst das Array in Bereiche der Größe `size` (und ggf. einen kleineren Restbereich) aufgeteilt und dann parallel nach den größten Elementen der Bereiche gesucht werden. Anschließend soll dann das größte Element der ermittelten größten Elemente der Bereiche bestimmt werden.

Ein Master-Thread soll dafür die benötigte Anzahl an Slave-Threads erzeugen. Er teilt das Array auf und startet für jeden Bereich der Größe `size` einen Slave. Die Slaves ermitteln jeweils das größte Element ihres Bereichs und speichern es in dem Attribut `max`. Danach ermittelt der Master das größte Element aus den Ergebnissen der Slaves und des evtl. vorhandenen Restbereichs.

Master und Slave werden beide in der Klasse `FindMax` realisiert. Zur Unterscheidung von Master- und Slave-Threads wird im Konstruktor das Attribut `isMaster` gesetzt.

```
public class FindMax extends Thread {

    private final boolean isMaster;
    private final int start;    // Bereichsanfang
    private final int size;     // Bereichsgroesse
    private final int[] data;

    // Stores the result of this thread:
    public int max = Integer.MIN_VALUE;

    // Constructor for master.
    public FindMax (int[] data, int size) {
        this.data = data;
        this.start = 0;
        this.size = size;
        this.isMaster = true;
    }

    // Constructor for slave.
    public FindMax (int[] data, int start, int size) {
        this.data = data;
        this.start = start;
        this.size = size;
        this.isMaster = false;
    }

    // Rest of class FindMax...

}
```

Implementieren Sie im vorgegebenen Template auf der nächsten Seite die `run`-Methode. Gehen Sie beim Aufruf des Konstruktors für den Master-Thread davon aus, dass `size` größer als 0 ist und `data` mindestens ein Element enthält.

Beispielaufruf:

```
public static void main (String[] args) throws InterruptedException {
    int[] testData = { 477, 9, 1, 177, 23 };

    FindMax master = new FindMax(testData, 2);
    master.start();
    master.join();
    System.out.println(master.max);
}
```

```
public void run() {
  try { // Handles all possible InterruptedExceptions
    if (isMaster) { // Code for master
        System.out.println("master");
```

```
    } else { // Code for slaves
       System.out.println("slave, "+ start +" to "+ (start+size-1));
```

```
  } catch (InterruptedException ie) {
     System.err.println("Oh No!"); System.exit(-1);
  }
```

Hinweis: Diese und die folgenden Seiten dürfen Sie von der Klausur abtrennen.

Die Grammatik von MiniJava (aus der Vorlesung)

<program> ::= <decl>* <stmt>*

<decl> ::= <type> <name> (`,` <name>)* `;`

<type> ::= `int`

<stmt> ::= `;`
| `{` <stmt>* `}`
| <name> `=` <expr>`;`
| <name> `=` `read();`
| `write(`<expr>`);`
| `if (`<cond>`)` <stmt>
| `if (`<cond>`)` <stmt> `else` <stmt>
| `while (`<cond>`)` <stmt>

<expr> ::= <number>
| <name>
| `(`<expr>`)`
| <unop> <expr>
| <expr> <binop> <expr>

<unop> ::= `-`

<binop> ::= `-` | `+` | `*` | `/` | `%`

<cond> ::= `true`
| `false`
| `(` <cond> `)`
| <expr> <comp> <expr>
| <bunop> <cond>
| <cond> <bbinop> <cond>

<comp> ::= `==` | `!=` | `<=` | `<` | `>=` | `>`

<bunop> ::= `!`

<bbinop> ::= `&&` | `||`