Name: | Vorname: | Matr.–Nr.:

Technische Universität München
Fakultät für Informatik
Prof. Dr. Seidl

WS 2004/2005
11. Dezember 2004

Zwischenklausur zu Einführung in die Informatik I

Hinweis: In dieser Zwischenklausur können Sie insgesamt 30 Punkte erreichen. Zum Bestehen der gesamten Prüfung (Zwischen- und Endklausur) benötigen Sie mindestens 40 von insgesamt 100 Punkten.

Aufgabe 1 Syntax-Baum (5 Punkte)

Zeichnen Sie den Syntax-Baum für das folgende MiniJava-Programm. Benutzen Sie dazu als Blätter die Tokens am unteren Rand dieses Blattes.

(**Hinweis:** Die Grammatik aus der Vorlesung befindet sich im Anhang.)

```
int x;
x = read();
if (x % 2 == 0) {
    x = (-2) - x;
}
```

`int` `x` `;` `x` `=` `read()` `;` `if` `(` `x` `%` `2` `==` `0` `)` `{` `x` `=` `(` `-` `2` `)` `-` `x` `;` `}`

Aufgabe 2 Kontrollfluss-Diagramm (5 Punkte)

Konstruieren Sie ein Kontrollfluss-Diagramm für folgendes MiniJava-Programm:

```
int x, y, r;
x = read();
y = read();
r = 1;
if (x > 0) {
    while (y > 0) {
        if (x % 3 == 0) {
            r = x * x;
        } else {
            r = x + x;
        }
        y = y - 1;
    }
}
write(r);
```

Aufgabe 3 Übersetzung von MiniJava nach MiniJVM (4 + 1 = 5 Punkte)

a) Übersetzen Sie das folgende MiniJava-Programm in ein MiniJVM-Programm.
(**Hinweis:** Die Übersetzungsregeln aus der Vorlesung befinden sich im Anhang.)

```
int n,i,d;

n = read();
i = 2;
d = 1;
while (i<=n){
   if (n%i == 0) d = d+1;
   i = i+1;
}
write(d);
```

b) Was berechnet das Programm?

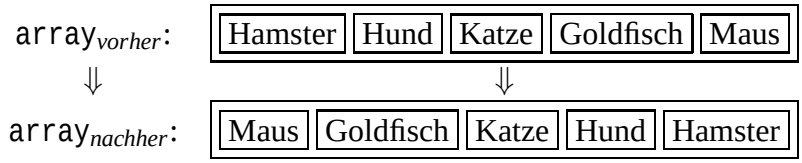
Aufgabe 4 Felder in Java **(3 + 3 = 6 Punkte)**

Implementieren Sie Methoden für die folgenden Aufgaben:

a) **Bestimmung von Minimum und Maximum**
Entwickeln Sie die Funktion `static void printMaxAndMin(int [] array)`, die im Feld `array` sowohl nach der kleinsten, als auch der größten Zahl sucht. Diese beiden Zahlen sollen dann ausgegeben werden.

b) **Umkehrung von Feldern**
Entwickeln Sie die Funktion `static void reverse(String [] array)`, die die Reihenfolge der Wörter im Feld `array` umdreht.

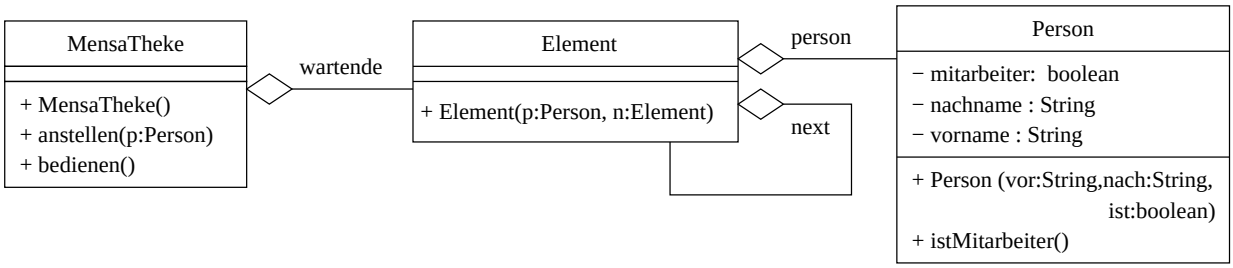
Beispiel:



(**Hinweis:**) Sie können hier einen Extrapunkt erhalten, indem Sie die Umkehrung ohne Zuhilfenahme eines zweiten Feldes realisieren.

Aufgabe 5 Mensa-Theke **(2 + 7 = 9 Punkte)**

Die Personen an einer Mensa-Theke sollen in der Reihenfolge bedient werden, in der sie sich angestellt haben – mit der Ausnahme, dass Mitarbeiter Vorrang vor allen Studenten haben und diese deshalb in der Schlange überholen. Diese Mensa-Theke sei durch das folgende UML-Modell beschrieben:



Die Wartenden werden in einer Liste von Elementen verwaltet. Die Klasse `Element` ist dabei wie folgt gegeben:

```
public class Element {
      public Person person;
      public Element next;

      public Element(Person p, Element n){
            data = p;
            next = n;
      }
}
```

a) Implementieren Sie die Klasse `Person` mit einem geeigneten Konstruktor und der Methode `istMitarbeiter`, mit der festgestellt werden kann, ob die Person ein Mitarbeiter ist.

b) Implementieren Sie die Klasse `MensaTheke` mit den Methoden `bedienen` und `anstellen`. Die Methode `bedienen` soll die nächste wartende Person bedienen und damit aus der Liste der Wartenden entfernen. Die Methode `anstellen` soll eine weitere Person in die Liste der Wartenden einreihen. Insbesondere sollen dabei Mitarbeiter vor allen Studenten platziert werden.

Anhang

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>	::=	`&&` | `||`

<*name*> und <*number*> für Bezeichner und Zahlen werden nicht weiter verfeinert.

Regeln aus der Vorlesung zur Übersetzung von MiniJava nach MiniJVM:

prog	=	ALLOC n Übersetzung von ss HALT — sofern `prog` aus einer Deklaration von n Variablen, gefolgt von der Statement-Folge ss besteht.
x	=	LOAD i — sofern x die Variable mit Adresse i ist.
c	=	CONST c — sofern c eine Konstante ist.
$expr_1$ + $expr_2$	=	Übersetzung von $expr_1$ Übersetzung von $expr_2$ ADD — Analog für die anderen Operatoren (SUB, MUL, DIV, MOD, AND, OR, LESS, LEQ, EQ, NEQ)
-expr	=	Übersetzung von expr NEG — Analog für NOT
x = expr;	=	Übersetzung von expr STORE i — sofern x die Variable mit Adresse i ist.
x = read();	=	READ STORE i — sofern x die Variable mit Adresse i ist.
write(expr);	=	Übersetzung von expr WRITE
if (cond) stmt	=	Übersetzung von cond FJUMP A Übersetzung von stmt A: ...
if (cond) $stmt_1$ else $stmt_2$	=	Übersetzung von cond FJUMP A Übersetzung von $stmt_1$ JUMP B A: Übersetzung von $stmt_2$ B: ...
while (cond) stmt	=	A: Übersetzung von cond FJUMP B Übersetzung von stmt JUMP A B: ...
$stmt_1$ ... $stmt_k$	=	Übersetzung von $stmt_1$ ... Übersetzung von $stmt_k$