252-0027-00: Einführung in die Programmierung

Theorie-Probeprüfung (Teil I)

Nachname: _____ Vorname: _____

Aufgabe	Wert	Punkte	Aufgabe	Wert	Punkte
1	3		5	8	
2	5		6	4	
3	8		7	8	
4	4				
Σ			Total	40	

Dies ist *kein* offizielles Eprog-Dokument und es wurde *nicht* vom EProg-Team auf Korrektheit oder Sinnhaftigkeit überprüft. Es dient dem Zweck, in der Übungsstunde eine 40-minütige Theorieprüfung zu simulieren.

Aufgabe 1 (3)

Gegeben sei eine Methode main in einer Java Klasse

```
public static void main(String[] args) { /* body */ }
```

Die folgenden Anweisungen sollen als "Body" (Rumpf) anstelle des Kommentars /* body */ eingefügt werden. Geben Sie für jede Anweisung an, was für eine Ausgabe erzeugt wird – entweder was gedruckt wird, oder ob ein Laufzeitfehler auftritt (schreiben Sie "Exception"), oder ob der Compiler einen Fehler feststellt (schreiben Sie "Compile-Fehler"). Achten Sie auf die korrekte Formatierung der verschiedenen Typen, also z.B. 7.0 statt 7 für eine reelle Zahl (double).

1) `System.out.println(7 + 5 / 4 + (2 + "==") + 5 * 2);`

   _____

2) `System.out.println(6 / 3 % 6 + (double) 5 / 4 + 3 / 4);`

   _____

3) `System.out.println(false != !false && ( 12/4 ) > 2 && (12 / (2 / 4)) == -1);`

   _____

Aufgabe 2 (5)

Gegeben sei eine Methode main in einer Java Klasse

```
public static void main(String[] args) { /* body */ }
```

Die folgenden Anweisungen sollen als "Body" (Rumpf) anstelle des Kommentars /* body */ eingefügt werden und den angegebenen Output produzieren. Bestimmen Sie für jede Anweisung die fehlenden Operatoren so, das die Anweisung die gezeigte Ausgabe erzeugt. Mögliche Operatoren sind +, -, *, / und %.

Wenn es für eine Anweisung mehrere mögliche Lösungen gibt, so genügt eine Lösung. Sollte es keine Lösung geben, so schreiben Sie bitte "nicht möglich".

Bereits existierende Operatoren (oder sonstige Teile der Anweisung) dürfen Sie nicht verändern. Auch dürfen Sie Klammern weder hinzufügen noch entfernen.

1) `System.out.println(24 / (2 * 12  4));`

   // Output: 4

2) `System.out.println(72 % (5 * 9  1) / 4);`

   // Output: 6

3) `System.out.println(36  (4  10));`

   // Output: -6

Aufgabe 3 (8)

Gegeben sei in Abbildung 1 die EBNF-Beschreibung von expr.

digit ⇐ `0` | `1` | ... | `9`
char ⇐ `A` | `B` | ... | `Z` | `a` | `b` | ... | `z`
num ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]
var ⇐ *char* { *char* }
func ⇐ *char* { *char* } `(`
op ⇐ `+` | `-` | `*` | `/` | `^`
open ⇐ `(`
close ⇐ `)`

atom ⇐ *num* | *var*
term ⇐ *open expr close* | *func expr close* | *atom*
expr ⇐ *term* [*op term*]

Abbildung 1: **EBNF-Beschreibung** von *expr*

Geben Sie für jeden folgenden Ausdruck an, ob er nach der EBNF-Beschreibung von expr in Abbildung 1 gültig ist.

Ausdruck	**Gültig**	**Ungültig**	**Ausdruck**	**Gültig**	**Ungültig**
1337			(o)^(o)		
x = 5			10.5 / 0		
y + z - 7			x + x(1 - x)		
sin(cos(sin(x)))			(x0 + x1)		
sqrt(-4 + 20)			fun1(3.0 * q)		

Aufgabe 4 (4)

Gegeben sind die Precondition und Postcondition für das folgende Programm.

```
public int compute(int n, int m) {
     // Precondition: m >= n && n >= 0
     int x = n;
     int y = m;

     while (x<y){
       x = x + 1;
       y = y - 1;
     }

     // Postcondition: y == (n+m)/2
     return y;
}
```

Bitte geben Sie die Loop Invariante als Java Expression an.

Loop Invariante: _____

Aufgabe 5 (8)

Gegeben sei die Methode maxEven(int[]) einer Klasse. In dieser Aufgabe stehen { und } – die Klammern für einen Block – auf einer separaten Zeile. Welche dieser Statements können entfernt werden ohne dass das Ergebnis der Methode verändert wird – egal, was für Parameter der Methodeübergeben werden?

Zum Beispiel sind im Codesegment

```
if (true) {     // Redundant
     x++;
} else {        // Redundant
     x--;        // Redundant
}
```

die markierten Statements redundant und können entfernt werden. Damit würden auch die Klammern überflüssig aber Sie müssen nicht angeben, ob Klammern redundant sind.

Markieren Sie bitte im folgenden Programm welche der Statements der Methode maxEven redundant sind. Der Parameter arr ist niemals null. Schreiben Sie “R” (oder “Redundant”) rechts in der entsprechenden Zeile. Sie brauchen nur angeben, welche Statements entfernt werden können, Sie brauchen nicht das Programm anderweitig zu verbessern. Sollte es mehrere Optionen geben, so wählen Sie die Option, bei welcher mehr Zeilen als redundant markiert werden.

```
   public static int maxEven(int[] arr){      // Platz fuer Antwort

     int max = -1;                          //

     for (int i=0; i<arr.length; i++) {     //

          if(arr[i] % 2 == 0) {             //
               if ( max == -1 ) {           //
                    max = arr[i];           //
               }
               if ( arr[i] > max ) {        //
                    max = arr[i];           //
               }
               if (arr[i] == max) {         //
                    max = arr[i];           //
               }
          } else {                          //
               if ( max % 2 != 0) {         //
                    max = -1;               //
               }
          }
     }

     if (max >= 0) {                        //
          return max;                       //

     } else {                               //
          return max++;                     //
     }
}
```

Aufgabe 6 (4)

Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest pre-condition) an. Bitte verwenden Sie Java Syntax (die Klammern { und } können Sie weglassen).

1)
```
P: ??
y = 2 * x;
z = z * y + 1;
Q: { z > 0 && y < 5 }
```

2)
```
P: ??
if (a > b) {
  c = (a-b)*(b-a);
} else {
  c = b*b;
}
Q: { c > 0 }
```

Aufgabe 7 (8)

1) Welche dieser Hoare-Tripel sind (un)gültig? Bitte geben Sie für ungültige Tripel ein Gegenbeispiel an. Das Gegenbeispiel muss Werte für Variable angeben, so dass nach Ausführung der Anweisung(en) die Postcondition nicht gilt. Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ int und es gibt keinen Overflow.

   a) `{ x > 0 } y = x / 2; { y > 0 }`

   b) `{ true } a = b / 2; a--; { a < b }`

   c)
```
{ x > 0 || y > 0 }
if (x > y) {
  z = x-y;
} else {
  z = y-x;
}
{ z > 0 }
```

2) Gegeben sind jeweils ein Programm zusammen mit einer Precondition und einer Postcondition. Auf der rechten Seite finden Sie jeweils drei potentielle Invarianten für die Schleife im Programm. Höchstens eine Option davon ist eine gültige Invariante. Sofern eine davon gültig ist, kreuzen Sie das dazugehörige Feld an. Sollte keine davon gültig sein, kreuzen Sie die vierte Option an.

a)
```
P: { x >= 0 }
int i = 0;
int s = 0;
while(i <= x) {
  s = s + i;
  i = i + 1;
}
Q: { s == x*(x+1)/2 }
```

Wählen Sie die passende Option:

☐ `{i <= x && s == (i-1)*i/2}`
☐ `{i <= x && s == i*(i+1)/2}`
☐ `{i <= x + 1 && s == (i-1)*i/2}`
☐ Keine der drei Optionen ist eine gültige Invariante

b)
```
P: { n >= 0 }
int y = 0;
int z = 0;
while(y*y < n) {
  y = y + 1;
  if (y*y <= n) {
    z = z + 1;
  }
}
Q: {z == Math.floor(Math.sqrt(n))}
```

Wählen Sie die passende Option:

☐ `{y*y <= n && z*z <= n}`
☐ `{y*y <= n && (y == z || y == z+1)}`
☐ `{z*z <= n && (y == z || y == z+1)}`
☐ Keine der drei Optionen ist eine gültige Invariante