ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. F. Mattern
Informatik II (D-ITET)

Disclaimer zur Veröffentlichung von Klausuren im Fach Informatik II (D-ITET)

Wir weisen Sie hiermit ausdrücklich auf Folgendes hin:

1. Die Aufgaben in den veröffentlichten Klausuren stellen nur Stichproben dar. Prüfungen aus vergangenen Jahren sind nicht mustergültig für zukünftige Prüfungen.
2. Die wöchentlichen Übungsaufgaben sind eine gute Vorbereitung. Oft erscheinen einige (leicht verändert) in der Prüfungsklausur.
3. Herauszufinden, ob man gut im Lösen von Prüfungen ist, ist nicht der Prüfungszweck.

Prüfung Informatik II (D-ITET)

23. August 2017, Räume HIL F41/F61, 10:00 – 11:00 Uhr

Name, Vorname: .....

ETH-Nummer: .....

Hinweise:

- Diese Prüfung umfasst **6 Aufgaben** auf **22 Seiten (inkl. Leerseiten)**. Prüfen Sie Ihr Exemplar auf Vollständigkeit!
- Schreiben Sie **deutlich lesbar** und ausschliesslich mit Tinte oder Kugelschreiber, **nicht** mit Bleistift. Wenn Sie Farben verwenden, benutzen Sie **auf keinen Fall rot**!
- Pro Aufgabe wird **höchstens eine Lösung** bewertet. Streichen Sie nicht zu wertende Lösungsversuche!
- Wo Java-Code verlangt wird, können Sie auch Pseudo-Code verwenden. Lösungen mit Pseudo-Code erhalten allerdings **nicht** die volle Punktzahl.
- Für die volle Punktzahl muss Ihr Code ausreichend mit **Kommentaren** versehen sein!
- Neben Papier und Schreibmaterial sind **KEINE Hilfsmittel** erlaubt.
- Die Blätter nicht voneinander trennen! Zusätzliche Leerblätter sind von der Aufsicht erhältlich.
- Tragen Sie Ihren Namen und Ihre ETH-Studierendennummer ein, und unterschreiben Sie dieses Blatt an der dafür vorgesehenen Stelle. Mit Ihrer Unterschrift bestätigen Sie, dass Sie die Aufgaben selbstständig gelöst, Ihre eigene Lösung abgegeben und störende äussere Einflüsse der Aufsicht gemeldet haben.
- Unehrliches Handeln führt zum sofortigem Ausschluss und kann rechtliche Folgen gemäss der Disziplinarordnung der ETH haben.
- **Bleiben Sie am Ende der Prüfung an Ihrem Platz, bis alle Unterlagen eingesammelt wurden!**

Unterschrift: .....

Viel Erfolg!

Aufgabe	Maximum	Erreicht
1	13	
2	14	
3	11	
4	10	
5	5	
6	7	
Gesamt	60	

Diese Seite wurde absichtlich leer gelassen!

Aufgabe 1: Datenstrukturen *(13 Punkte)*

1 a) (9 Punkte) Beschreiben Sie den Aufbau und die wichtigsten Eigenschaften der folgenden Datenstrukturen, und füllen Sie anschliessend in der folgenden Tabelle die Average-Case-Laufzeitkomplexitäten für die Operationen *insert*, *pop* und *search* in *O*-Notation aus.

Queue / Warteschlange (implementiert als ein Array).....
.....
.....
.....
.....
Stack (implementiert mit einer einfach-verketteten Liste) .....
.....
.....
.....
.....
Binärer Suchbaum (implementiert mit Knoten-Objekten und Referenzen auf solche Objekte) .....
.....
.....
.....
.....

Average-Case-Laufzeitkomplexitäten

Datenstruktur	***insert***	***pop****	***search***
Queue / Warteschlange	.....	.....	.....
Stack	.....	.....	.....
Binärer Suchbaum	.....	.....	.....

* Beim binären Suchbaum entspricht *pop* dem Löschen von Wurzel, wobei anschliessend die Suchbaumeigenschaft wieder hergestellt sein soll.

1 b) (4 Punkte) Geben Sie an, ob die folgenden Aussagen entweder wahr oder falsch sind. Für richtige Antworten erhalten Sie einen Punkt, pro falsche Antwort gibt es allerdings einen halben Punkt Abzug. Die Teilaufgabe gibt mindestens 0 Punkte.

	wahr	*falsch*
"Der Zeitaufwand zur Suche nach einem Element in einer doppelt verketteten Liste (d.h. jeder Knoten hat zusätzlich noch eine Referenz auf den Vorgängerknoten) beträgt im Worst-Case$O(\log(n))$."	☐	☐
"Ein binärer Baum mit der Tiefe h hat $O(2^h)$ viele Knoten."	☐	☐
"Verwendet man zum Sortieren einen binären Suchbaum, dann ist die Best-Case- und Worst-Case-Laufzeitkomplexität in $O(n \log(n))$."	☐	☐
"Bei der Remove-Operation in einem Max-Heap wird die Wurzel immer mit einem Wert aus dem rechten Teilbaum ersetzt."	☐	☐

Aufgabe 2: Objektorientierung *(14 Punkte)*

Sie sind damit beauftragt, eine Datenbank für eine Bank mitsamt ihren Angestellten zu implementieren. Unter den Angestellten (`Employee`) gibt es eine Gruppe (`Clerk`), die Transaktionen ausführen können. Der maximale Betrag, den ein `Clerk` mit einer Transaktion überweisen kann, ist durch ein individuelles Freigabeniveau (`clearanceLevel`) begrenzt. Aus Datenschutzgründen kann nicht direkt über eine `public`-Methode auf das Gehalt der Angestellten zugegriffen werden, aber die Klasse `Employee` implementiert das Interface `Comparable`, wodurch Angestellte anhand ihres Gehalts verglichen werden können.

2 a) (2 Punkte) Implementieren Sie die Methode `compareTo` in der Klasse `Employee`. Falls das Gehalt der `this`-Referenz kleiner ist als das von `obj`, soll -1 zurückgegeben werden, bei Gleichheit 0 und ansonsten 1. Falls `obj` nicht vom Typ `Employee` ist, soll 0 zurückgegeben werden.

```
1
2  public class Employee implements Comparable {
3      private int salary;
4
5      public Employee(int salary) {
6          this.salary = salary;
7      }
8
9      @Override
10     public int compareTo(Object obj) {
11
12         ..................................
13
14         ..................................
15
16         ..................................
17
18         ..................................
19
20         ..................................
21
22         ..................................
23
24         ..................................
25
26     }
27 }
```

2 b) (3 Punkte) Implementieren Sie die Klasse `Clerk`, die die Klasse `Employee` erweitert. Schreiben Sie eine getter-Methode für `clearanceLevel`.
Hinweis: Mit `super` ist es möglich, auf Konstruktoren oder Methoden der Basisklasse zuzugreifen.

```
1
2  public class Clerk ..................................... {
3
4      private int clearanceLevel;
5
6      public Clerk(int salary, int clearanceLevel) {
7
8          ....................................
9
10         ....................................
11
12     }
13
14     public int getClearanceLevel() {
15
16         ....................................
17
18     }
19 }
```

2 c) (2 Punkte) Implementieren Sie den Konstruktor der Klasse `Bank`, der das Attribut `employees` als leere `ArrayList` initialisiert, und die Methode `addEmployee`, die der `ArrayList` einen `Employee` hinzufügt.

2 d) (6 Punkte) Implementieren Sie die Methode `getClerkForTransaction`. Die Methode soll denjenigen `Clerk` zurückgeben, der unter den `Clerk`s mit der notwendigen Freigabe (d.h. `clearanceLevel` muss mindestens so hoch sein wie `amount`) das niedrigste Gehalt hat.

Falls Sie die erste Teilaufgabe nicht erledigt haben, dürfen Sie `compareTo` hier trotzdem verwenden. Falls es keinen `Clerk` mit der passenden Freigabe gibt, geben Sie `null` zurück.
Hinweis: In der Klasse `Bank` ist es nicht möglich, auf das Gehalt eines Angestellten zuzugreifen.

```
1  import java.util.ArrayList;
2
3  public class Bank {
4
5      private ArrayList<Employee> employees;
6
7      public Bank() {
8
9          .....................................
10
11
12     }
13
14     public void addEmployee(Employee emp) {
15
16         .....................................
17
18     }
19
20     public Clerk getClerkForTransaction(int amount) {
21
22         .....................................
23
24         .....................................
25
26         .....................................
27
28         .....................................
29
30         .....................................
31
32         .....................................
33
34         .....................................
35
36         .....................................
37
38         .....................................
39
40     }
41 }
```

```
1
2  public class Main {
3
4      public static void main(String[] args) {
5          Bank bank = new Bank();
6
7          bank.addEmployee(new Employee(55000));
8          bank.addEmployee(new Clerk(90000, 15000));
9          bank.addEmployee(new Employee(190000));
10         bank.addEmployee(new Clerk(50000, 1000));
11         bank.addEmployee(new Clerk(75000, 13500));
12         bank.addEmployee(new Clerk(95000, 20000));
13         bank.addEmployee(new Clerk(150000, 100000));
14
15         Clerk c = bank.getClerkForTransaction(14000);
16
17         if (c == null) {
18             System.out.println("No clerk available.");
19         } else{
20             System.out.println("The clearance level of the clerk is "
21                     + c.getClearanceLevel());
22         }
23     }
24
25 }
```

2 e) (1 Punkt) Was ist die Ausgabe des Programmstücks der obigen *main*-Methode.

.....

.....

Aufgabe 3: Heaps *(11 Punkte)*

3 a) (1 Punkt) Stellt ein absteigend sortiertes Array einen Max-Heap dar? Begründen Sie Ihre Antwort.

.....
.....
.....
.....

3 b) (2 Punkte) Ein Min-Heap habe $2^k - 1$ Elemente (mit $k > 1$), d.h. der Baum ist voll besetzt. Unter den Elementen gebe es keine gleich grossen Elemente. Wie wahrscheinlich ist es, dass sich das zweitgrösste Element auf dem untersten Niveau befindet? Begründen Sie Ihre Antwort.

☐ 1 ☐ $\frac{1}{2}$ ☐ $\frac{1}{k}$ ☐ $\frac{1}{k-1}$ ☐ $\frac{1}{\log_2 k}$ ☐ $\frac{1}{2k}$ ☐ $\frac{1}{2k-1}$ ☐ $\frac{1}{2^k}$ ☐ $\frac{1}{2^k-1}$ ☐ 0

.....
.....
.....
.....

3 c) (1 Punkt) Folgende Aussage ist falsch: "Traversiert man einen Min-Heap in postorder, dann werden die Knotenwerte immer absteigend sortiert ausgegeben." Geben Sie ein Gegenbeispiel an.

.....
.....
.....
.....

3 d) (1 Punkt) Folgende Aussage ist falsch: "Traversiert man einen Max-Heap in postorder, dann werden die Knotenwerte nie aufsteigend sortiert ausgegeben." Geben Sie ein Gegenbeispiel an.

.....
.....
.....
.....

3 e) (2 Punkte) Heapsort besteht aus zwei Phasen. Welche Laufzeitkomplexität (in O-Notation) hat die erste Phase für einen Min-Heap, wenn die Elemente nacheinander mit "insert" in einen (anfangs leeren) Heap eingefügt werden und die zu sortierende Eingangsfolge bereits aufsteigend sortiert ist?
Begründen Sie Ihre Antwort.

.....
.....
.....
.....

3 f) (2 Punkte) Angenommen, eine Anwendung nutzt eine Priority Queue in einer Weise, dass sehr viele *insert*-Operationen, aber nur wenige *get_min*-Operationen auftreten. Welche Implementierung der Priority Queue ist hierfür am besten geeignet: ein Heap, eine aufsteigend sortierte verkettete Liste oder eine unsortierte verkettete Liste?
Begründen Sie Ihre Antwort.

.....
.....
.....
.....

3 g) (2 Punkte) Geben Sie den Ansatz für einen Algorithmus an, der in linearer Zeit prüft, ob ein Array einen Min-Heap repräsentiert.

.....
.....
.....
.....

Aufgabe 4: Laufzeitkomplexität von Mergesort *(10 Punkte)*

In der Vorlesung wurde mit einem Induktionsbeweis gezeigt, dass Mergesort die Zeitkomplexität $O(n \log n)$ hat, wobei n die Problemgrösse repräsentiert ($n \geq 1$). Dazu wurde folgende Gleichungsfolge für die Zahl der Schritte $t(n)$ verwendet:

$$\begin{aligned} t(n) &= 2 * t(\frac{n}{2}) + n \\ &= 2 * (\frac{n}{2} * \log_2(\frac{n}{2}) + \frac{n}{2}) + n \\ &= n + n + n * \log_2(\frac{n}{2}) \\ &= n + n - n + n * \log_2(n) \\ &= n + n * \log_2(n) \end{aligned}$$

Der Einfachheit halber gehen wir hier davon aus, dass n eine Zweierpotenz ist.

4 a) (2 Punkte) Wieso gilt die erste Zeile, also $t(n) = 2 * t(\frac{n}{2}) + n$?

.....
.....
.....
.....

4 b) (2 Punkte) Wie lautet die im Beweis benutzte Induktionsannahme (manchmal auch Induktionsvoraussetzung genannt), und wo wird diese im Beweis genau verwendet?

.....
.....
.....
.....

4 c) (2 Punkte) Typischerweise gibt es in einem Induktionsbeweis einen Induktionsschritt (manchmal auch Induktionsschluss genannt) von n auf $n + 1$. Dies ist hier aber nicht der Fall. Was tritt hier statt dessen an diese Stelle?

.....
.....
.....
.....

4 d) (2 Punkte) Der Induktionsanfang (manchmal auch Induktionsbasis oder Induktionsverankerung genannt) für $n = 1$ ist hier nicht ausgeführt. Streng genommen muss aber die Korrektheit von $t(1)$ gezeigt werden. Damit die Gleichungsformel auch noch für $t(1)$ gilt, darf der Wert von $t(1)$ nicht 0 sein. Begründen Sie, warum dies so ist und geben Sie an, welchen Wert $t(1)$ stattdessen haben muss.

.....
.....
.....
.....

4 e) (1 Punkt) Wieso folgt aus obiger Gleichungsfolge die Behauptung, dass die Zeitkomplexität von Mergesort $O(n * \log(n))$ beträgt?

.....
.....
.....
.....

4 f) (1 Punkt) Begründen Sie kurz: Wurde damit die Worst-Case- oder die Average-Case-Komplexität gezeigt?

.....
.....
.....
.....

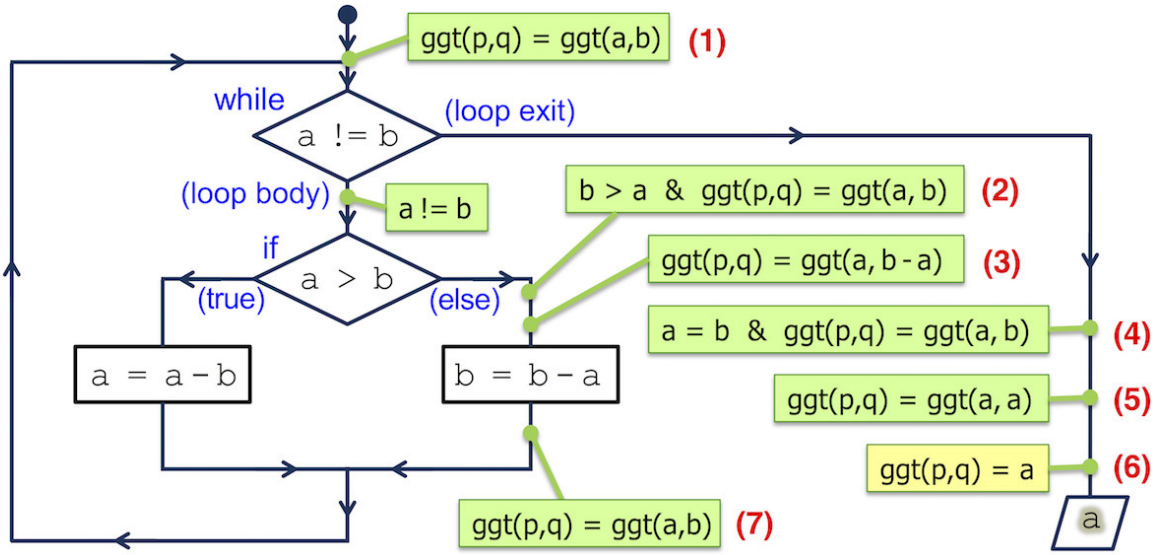
Diese Seite wurde absichtlich leer gelassen!

Aufgabe 5: Programmverifikation *(5 Punkte)*

Das folgende Programm berechnet den grössten gemeinsamen Teiler (ggT) zweier positiver ganzer Zahlen p,q:

```
1  public int ggT(int p, int q) {
2    int a = p;
3    int b = q;
4    while (a != b) {
5      if (a > b)
6        a = a - b;
7      else
8        b = b - a;
9    }
10   return a;
11 }
```

Die nachfolgende Skizze enthält wesentliche Elemente des Korrektheisbeweises in Form von Annotationen (Zusicherungen) am Kontrollfluss-Diagramm:



5 a) (1 Punkt) Wie lautet die Schleifeninvariante, mit der sich die Korrektheit beweisen lässt?

.....

.....

5 b) (2 Punkte) Innerhalb der Schleife wird jedes Mal entweder nur a oder nur b verändert. Die Wertänderung einer Variablen wird also nicht durch die Wertänderung einer anderen Variablen kompensiert. Wie ist es möglich, dass trotz der Änderung des Wertes von genau einer Variablen die Schleifeninvariante dennoch invariant bleibt?

.....
.....
.....
.....

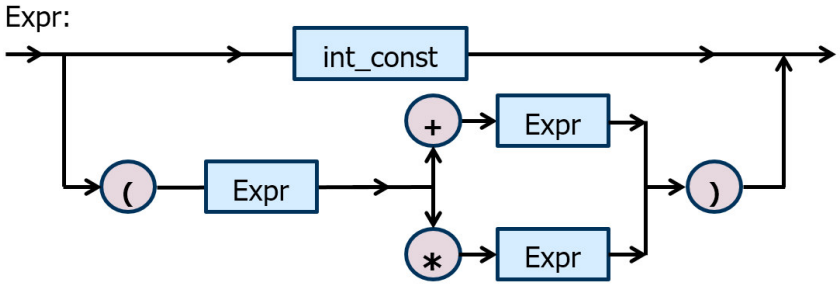
5 c) (2 Punkte) Wieso gilt Zusicherung (4)? Das heisst, aufgrund welcher Beweisregeln und / oder anderen Zusicherungen lässt sich die Gültigkeit von (4) folgern?

.....
.....
.....
.....

Diese Seite wurde absichtlich leer gelassen!

Aufgabe 6: Syntaxdiagramme *(7 Punkte)*

Folgendes Syntaxdiagramm dient der Erzeugung vollständig geklammerter Infix-Ausdrücke. *int_const* repräsentiert hierbei eine Zahl von 0 bis 9.



6 a) (1 Punkt) Ist entsprechend diesem Diagramm $((2)+(4*3))$ ein korrekter vollständig geklammerter Infix-Ausdruck? Begründen Sie Ihre Antwort.

.....
.....
.....
.....

6 b) (2 Punkte) Geben Sie zum Ausdruck $2 + 3 * 4 + 5$ einen äquivalenten vollständig geklammerten Infix-Ausdruck nach obigem Syntaxdiagramm an und skizzieren Sie den zugehörigen Operatorbaum.

.....
.....
.....
.....

6 c) (2 Punkte) Geben Sie zum Syntaxdiagramm einen Parser als Java-Programm an. (Sie dürfen davon ausgehen, dass ein zu analysierender Ausdruck keine Leerzeichen oder falsche Zeichen enthält.) Zur Erinnerung ist nachfolgend der Parser aus der Vorlesung für die dort behandelten (nicht vollständig geklammerten) Infix-Ausdrücke aufgeführt. `KbdInput` puffert die Zeichenfolge und gibt mittels der statischen Methode `getc()` den nächsten `char` zurück.

```
1  public static void main(String args[]) throws Exception {
2      Parser p = new Parser();
3      p.c = KbdInput.getc();
4      try {
5          p.Expression();
6      } catch (StringIndexOutOfBoundsException s){
7          System.out.println("Correct");
8      }
9  }
10
11 public static class Parser {
12
13     private char c;
14
15     public void Expression() throws Exception {
16         Term();
17         while (c == '+') {
18             c = KbdInput.getc();
19             Term();
20         }
21     }
22
23     public void Term() throws Exception {
24         Faktor();
25         while (c == '*') {
26             c = KbdInput.getc();
27             Faktor();
28         }
29     }
30
31     public void Faktor() throws Exception {
32         if ((c >='0') && (c <='9')) {
33             int_const();
34         } else {
35             if (c == '(') {
36                 c = KbdInput.getc();
37                 Expression();
38                 if (c == ')') {
39                     c = KbdInput.getc();
40                 } else {
41                     throw new Exception("Falsche Klammerung Schluss.");
42                 }
43             } else {
44                 throw new Exception("Falsche Klammerung Beginn.");
45             }
46         }
47
48     }
49
50     public void int_const() {
51         c = KbdInput.getc();
52     }
53 }
```

Füllen Sie das folgende Codegerüst mit den Anweisungen für Ihren Parser aus. `KbdInput` darf weiterhin verwendet werden.

```
1  public static void main(String args[]) throws Exception {
2      Parser p = new Parser();
3      p.c = KbdInput.getc();
4      try {
5          p.Expression();
6      } catch (StringIndexOutOfBoundsException s){
7          System.out.println("Correct");
8      }
9  }
10
11 public class MyParser {
12     private char c;
13     public void Expression() throws Exception {
14         ..............................
15         ..............................
16         ..............................
17         ..............................
18         ..............................
19         ..............................
20         ..............................
21         ..............................
22         ..............................
23         ..............................
24         ..............................
25         ..............................
26         ..............................
27         ..............................
28         ..............................
29         ..............................
30         ..............................
31         ..............................
32         ..............................
33         ..............................
34         ..............................
35         ..............................
36         ..............................
37         ..............................
38         ..............................
39         ..............................
40         ..............................
41         ..............................
42         ..............................
43     }
44
45     public void int_const() {
46         c = KbdInput.getc();
47     }
48 }
```

6 d) (2 Punkte) Ergänzen Sie Ihren Parser aus der vorherigen Teilaufgabe mit Anweisungen `System.out.print(...)` so, dass ein äquivalenter Ausdruck in Postfix-Notation ausgegeben wird. Erläutern Sie kurz Ihre Konstruktion.

Diese Seite wurde absichtlich leer gelassen!

Diese Seite wurde absichtlich leer gelassen!