Modul
Datenstrukturen, Algorithmen und Programmierung 1
Sommersemester 2021

Klausur – 11.8.2021
17:00 - 19:30 Uhr

Dieses ist die Klausur zum Modul Datenstrukturen, Algorithmen und Programmierung 1 (DAP 1) im Sommersemester 2021. Für die Bearbeitung der Klausur haben Sie 120 Minuten Zeit. Zusätzlich stehen Ihnen weitere 30 Minuten zum Digitalisieren und Hochladen Ihrer Abgabe zur Verfügung. Insgesamt muss die Klausur bis *spätestens um 19:30 Uhr*, also 150 Minuten nach der Ausgabe, digital in Moodle abgegeben werden. Verspätete Abgaben werden nicht berücksichtigt.

Die Abgabe muss als *genau eine Datei* vom Typ PDF erfolgen. Ihre Abgaben können Sie sowohl handschriftlich als auch direkt digital anfertigen, wobei auch die Nutzung von Tools zur Programm- oder Diagrammerstellung oder zur Digitalisierung von Handschrift (z.B. Tablets) erlaubt sind. Das Hochladen von Archivformaten (wie .zip, .7z, usw.) ist explizit nicht gestattet und wird durch Moodle verhindert. Die Abgaben müssen lesbar sein und dürfen nur eine Bearbeitung je Aufgabe enthalten.

Am Anfang der ersten Seite Ihrer Abgabe müssen Ihr Vorname, Ihr Nachname und Ihre Matrikelnummer stehen. Der Name der Datei mit ihrer Abgabe muss dem folgenden Namensschema entsprechen:

DAP1_01_ <Matrikelnummer >_<Nachname >.pdf

Mit dem Beginn der Bearbeitung bestätigen Sie, dass Sie

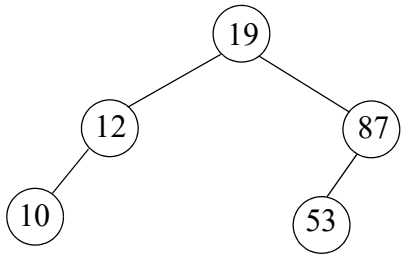
- durch ordnungsgemäße Anmeldung berechtigt sind, die Prüfung zu absolvieren,
- die Prüfung selbst ablegen werden,
- die Prüfung eigenständig durchführen werden,
- während der Klausur keine Kommunikation mit anderen Personen führen werden,
- während der Klausur die Aufgaben oder die zugehörigen Bearbeitungen nicht für andere Personen zugänglich machen werden,
- sich im Allgemeinen prüfungskonform verhalten werden und
- gesundheitlich prüfungsfähig sind.

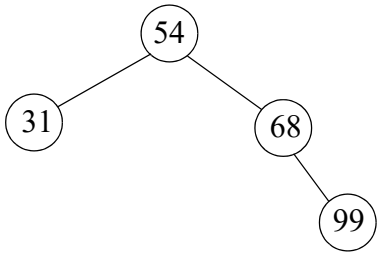
Täuschungsversuche werden nach der für Sie geltenden Prüfungsordnung geahndet.

Den Zugang zu dieser Klausur haben Sie nur nach Abgabe mehrerer eidesstattlicher Versicherungen erhalten. Eine Verletzung dieser eidesstattlichen Versicherungen kann strafrechtliche Konsequenzen haben.

Aufgabe 1

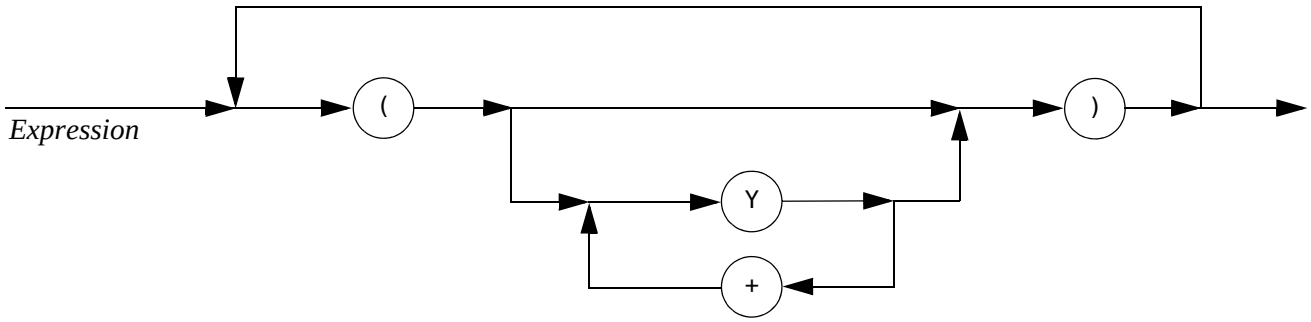
[4 Punkte] Fügen Sie die Werte *14*, *35*, *22*, *41* und *84* in *genau* dieser Reihenfolge in jeden der unten angegebenen binären Suchbäume ein.





Aufgabe 2

[5 Punkte] Das Syntaxdiagramm beschreibt, wie eine gültige *Expression* aussehen darf.



Geben Sie vier Zeichenfolgen an, die eine gültige *Expression* bilden und zusätzlich jeweils eine der folgenden Bedingungen erfüllen:

a) Die *Expression* enthält mindestens zweimal das Zeichen `Y`.

b) Die *Expression* enthält mindestens zweimal das Zeichen `+`.

c) Die *Expression* enthält mindestens zweimal das Zeichen `(` und mindestens zweimal das Zeichen `Y`.

d) Die *Expression* besteht aus genau zehn Zeichen und enthält genau vier Mal das Zeichen `)`.

e) Die *Expression* besteht aus genau acht Zeichen.

Aufgabe 3

[12 Punkte] Erstellen Sie eine Methode `public static int biggerThan( int[] allInts, int p )`. Die Methode `biggerThan` gibt den kleinsten Wert aus `allInts` zurück, der größer als `p` ist. Gibt es keinen solchen Wert, wird eine Ausnahme der Klasse `IllegalStateException` geworfen.

Aufgabe 4

[8 Punkte] Erstellen Sie eine Methode
`public static <T> int size( java.util.List<Iterable<T>> structures )`.
Die Methode `size` gibt die Anzahl der in `structures` enthaltenen Inhalte zurück.

Hinweis: Beachten Sie Folie 941 der Vorlesungsfolien für das Sommersemester 2021.

Aufgabe 5

[8 Punkte] Erstellen Sie eine Methode
`public static <T extends Comparable<T>> T max( Iterable<T> allObjects )`.
Die Methode `max` gibt den größten Inhalt aus `allObjects` zurück. Falls `allObjects` keinen Inhalt hat, soll `null` zurückgegeben werden.

Hinweis: Beachten Sie Folie 941 der Vorlesungsfolien für das Sommersemester 2021.

Aufgabe 6

[16 Punkte] Die bekannte Klasse `DoublyLinkedList<T>` soll um genau eine Methode ergänzt werden. Bei der Implementierung der geforderten Methode soll die Klasse `DoublyLinkedList<T>` benutzt werden, die in *Programmbeispiele aus Kapitel 12* bereitgestellt wird.

Erstellen Sie die Methode `public DoublyLinkedList<T> cutInFrontOf( T obj )`.

Die Methode `cutInFrontOf` gibt eine neu erzeugte Liste zurück. Die neue Liste enthält in unveränderter Reihenfolge alle Elemente der ausführenden Liste, die vor dem ersten Vorkommen des Inhalts `obj` liegen. Diese Elemente sollen zugleich aus der ausführenden Liste entfernt werden. Der Vergleich mit den Inhalten erfolgt mit der Methode `equals`. Ist der Inhalt `obj` nicht in der ausführenden Liste enthalten, wird eine leere Liste zurückgegeben.

Aufgabe 7

[16 Punkte] Die bekannte Klasse `DoublyLinkedList<T>` soll um genau eine Methode ergänzt werden. Bei der Implementierung der geforderten Methode soll die Klasse `DoublyLinkedList<T>` benutzt werden, die in *Programmbeispiele aus Kapitel 12* bereitgestellt wird.

Erstellen Sie die Methode `public int countDoubles()`.

Die Methode `countDoubles` gibt die Anzahl der Inhalte zurück, die in der ausführenden Liste *genau zweimal* vorkommen. Der Vergleich der Inhalte erfolgt mit der Methode `equals`.

Aufgabe 8

[16 Punkte] Die bekannte Klasse `BinarySearchTree<T extends Comparable<T>>` soll um genau eine Methode ergänzt werden. Bei der Implementierung der geforderten Methode soll die Klasse `BinarySearchTree<T extends Comparable<T>>` benutzt werden, die in *Programmbeispiele aus Kapitel 12* bereitgestellt wird.

Erstellen Sie die Methode `public int deleteLeavesAbove( int level )`.

Die Methode `deleteLeavesAbove` löscht alle Blätter aus dem Baum, die oberhalb der Ebene `level` liegen – also auf einer Ebene mit einer kleineren Nummer. Die Anzahl der gelöschten Blätter wird zurückgegeben. Die Wurzel liegt auf der Ebene `0`. Wird ein negatives Argument übergeben, wird eine Ausnahme der Klasse `IllegalStateException` geworfen.

Aufgabe 9

[15 Punkte] Gegeben sind die Interfaces `BoolFunction`, `IntFunction` und die Klasse `Data`.

```
public interface BoolFunction {
    boolean apply( int x, boolean b);
}
```

```
public interface IntFunction {
     int apply( int x, int y );
}
```

```
public class Data
{
    private int[] intValues;

    public Data( int[] iV ) {
        intValues = iV;
    }

    public boolean test( BoolFunction f, boolean init ) {
        boolean check = init;
        for ( int v : intValues ) {
            check = f.apply( v, check );
        }
        return check;
    }

    public int make( IntFunction f, int init ) {
        int result = init;
        for ( int v : intValues ) {
            result = f.apply( v, result );
        }
        return result;
    }
}
```

Hinweis: In den folgenden Aufgabenbeschreibung wird als Kurzschreibweise der Name `intValues` immer dann verwendet, wenn das Attribut `intValues` des als Argument übergebenen `Data`-Objekts gemeint ist.

Erstellen Sie die folgenden drei Methoden außerhalb der Klasse `Data`. Sofern Sie aus der Klasse `Data` die Methoden `test` und `make` verwenden, dürfen Sie als Argumente für den Parameter `f` *nur Lambda-Ausdrücke* verwenden.

a) Erstellen Sie eine Methode `static int sizeIf( Data d, int p )`.

Die Methode `sizeIf` gibt die Länge von `intValues` zurück, falls `intValues` mindestens einmal den Wert `p` enthält. Enthält `intValues` nicht den Wert `p`, wird `-1` zurückgegeben.

b) Erstellen Sie eine Methode `static boolean containsIf( Data d, int p )`.

Die Methode `containsIf` gibt `true` zurück, falls alle in `intValues` enthaltenen Werte größer als `p` sind und mindestens einer dieser Werte größer als `2*p` ist . Sonst wird `false` zurückgegeben.

c) Erstellen Sie eine Methode `static int multiplyIf( Data d, int limit )`.

Die Methode `multiplyIf` gibt das Produkt aller Werte aus `intValues` zurück, falls die Länge von `intValues` kleiner als `limit` ist. Sonst wird `0` zurückgegeben.