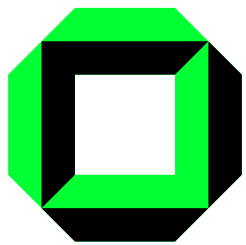
Info I – Übungsblatt 10

Joachim Breitner

mit Aufgaben von Martin Kiefel und Felix Brandt

http://www.joachim-breitner.de/wiki/Infotut

23. Januar 2006



Unser Programm heute

Übungsblatt-Rückblick

Statistik

- Schnitt: 20 von 31 Punkten
- Blätter liegen zu hause, ihr bekommt sie in der Rechnerübung oder nächsten Montag.

Häufige Fehler

- Anfangszustand bei Automaten in Graphendarstellung nicht markiert
- Beim SchniPoMat die Ein- bzw. Ausgabe nicht erklärt
- Reguläre Ausdrücke bitte am Stück: „$0+(1+2+3+4+5+6+7+8+9)(\varepsilon+0+1+2+3+4+5+6+7+8+9)$“

Übungsblatt-Rückblick

Statistik

- Schnitt: 20 von 31 Punkten
- Blätter liegen zu hause, ihr bekommt sie in der Rechnerübung oder nächsten Montag.

Häufige Fehler

- Anfangszustand bei Automaten in Graphendarstellung nicht markiert
- Beim SchniPoMat die Ein- bzw. Ausgabe nicht erklärt
- Reguläre Ausdrücke bitte am Stück:
  „0+(1+2+3+4+5+6+7+8+9)(ε+0+1+2+3+4+5+6+7+8+9)“

Vorlesung nachher

Die Vorlesung direkt im Anschluss
findet im Gerthsen-Hörsaal statt!

Anmeldung für die Klausur

Für die Informatik-I-Klausur müsst ihr euch anmelden:
Dazu braucht ihr (Informatiker) den blauen Schein vom Studienbüro, und müsst damit zum Sekretariat.
Zusätzlich solltet ihr euch online eintragen, damit es keine Probleme wegen falschem Abtippen gibt:

`https://i71ipo.cm-tm.uni-karlsruhe.de:444/`

(Den Link findet ihr auch auf der IPO-Startsete)

Probeklausur

Die Tutoren veranstalten eine **inoffizielle** Probeklausur. Nehmt die Gelgenheit wahr, ohne Erstickungsgefahr Klausurluft zu schnuppern!
Die Klausur findet statt am:

Samstag, 28.1.2006 um 9:45
im Audimax

Die notwendige Anmeldung ist möglich auf:

http://info1probeklausur.dl.am/

Konstruktoraufgabe

Aufgabe

Erstellt Konstruktoren für die Crayon-Klasse vom letzen Tutorium:

- Einen, ohne Parameter
- Einen, mit einem Parameter für die Länge

Erstellt außerdem eine Auswahl von `set-` und `get`-Methoden.

Malstiftfabriken

```
class Crayon {
  // Same attributes as last time

  public Crayon() { // default constructor with fixed values
    length = 10.0;
    red = green = blue = 0.5;
  }

  // constructor that sets the length
  public Crayon(double newLength) {
    red = green = blue = 0.5;
    crayon.length = newLength;
  }

}
```

Malstiftfabrikbenutzer

```
...
public static void main(String[] argh) {
  // using the default constructor:
  Crayon crayon1 = new Crayon();

  // using another constructor:
  Crayon longCrayon = new Crayon(20.0);
}
```

Malstiftverfärber

```
class Crayon {
  ...
  public void setGreen(double green) {
    this.green = green;
  }

  public void setRed(double newRed) {
    red = newRed;
  }

  public double getBlue() {
    return blue;
  }
}
```

Damit es was zu erben gab, musste einer ...

- Häufig benötigt man Klassen, die sich in vielen Eigenschaften gleichen aber die man dennoch nicht zusammenfassen kann, da bestimmte Teile der einen Klasse für die andere uninteressant sind und umgekehrt.
- die Gemeinsamkeiten können in eine Oberklasse „ausgelagert“ werden
- danach werden die beiden Klassen nur noch spezialisiert, also die Elemente welche noch nicht in der Oberklasse auftauchen hinzugefügt.
- So kann die Coderedundanz verringert und Wartbarkeit erleichtert werden.

Erben in Java

Mittels `extends` wird bei der Klassendefinition die Oberklasse angegeben. Die neue Unterklasse „besitzt“ dann auch alle Eigenschaften und Methoden der Oberklasse.

Beispiel

```
class Ship{
        int weight, length, height;
        String owner, country;

        public void makeSound(){ dingding(); }
}

class Supertanker extends Ship{
        int volume, crew, power;

        public void makeSound(){ huuuuuuuup(); }
}

class Nussschale extends Ship{
        boolean omaAnBoard;

        public void makeSound(){ scream(); }
}
```

Sichtbarkeit

In `Java` kann die Sichtbarkeit von Klassen, Methoden und Variablen durch folgende Schlüsselwörter geregelt werden:

`public`:	von überall Sichtbar
`protected`:	sichtbar in der aktuellen Klasse, im gleichen Verzeichnis (bzw. Paket) befindenden Klassen und in abgeleiteten Klassen
`private`:	sichtbar nur in der aktuellen Klasse

Ohne Schlüsselwort sind alle Element nur in der aktuellen Klasse und in den sich im gleichen Verzeichnis (bzw. Paket) befindenden Klassen sichtbar.

FIFOs

Was ist ein FIFO.

Ein FIFO („First In, First Out“) ist eine Warteschlange, in die man Werte (Objekte,...) einfügen und auslesen kann. Dabei werden die Objekte in der Reihenfolge ausgegeben, in der sie eingegeben werden.
Vergleiche das mit den LIFO („Last In, First Out“), auch Stacks genannt, die die Eingaben in der anderen Reihenfolge ausgeben.

FIFOs

Was ist ein FIFO.

Ein FIFO („First In, First Out“) ist eine Warteschlange, in die man Werte (Objekte,...) einfügen und auslesen kann. Dabei werden die Objekte in der Reihenfolge ausgegeben, in der sie eingegeben werden.
Vergleiche das mit den LIFO („Last In, First Out“), auch Stacks genannt, die die Eingaben in der anderen Reihenfolge ausgeben.

Python

Auf der nächsten Folie sehen wir ein FIFO, implementiert in (schlechten, Java-ähnlichem) Python. Versucht, zu verstehen, wie der Code funktioniert!
(Eigentlich kann Python das selbst: `l.append(x)` und `l.pop[0]`)

Eine Schlange in Python

```
#!/usr/bin/python
class Queue:
  def __init__(self, len=5):
    self.array = [None]*len
    self.pos   = 0
  def enqueue(self, x):
    self.array[self.pos] = x
    self.pos += 1
  def dequeue(self):
    self.pos -= 1
    value = self.array[0]
    for i in range(0,self.pos):
      self.array[i] = self.array[i+1]
    return value

q = Queue(10)
q.enqueue(1); q.enqueue(2); print q.dequeue();
q.enqueue(3); print q.dequeue(); print q.dequeue();
```

Einen Baum pflanzen

Aufgabe

- Baue die folgenden Werte der Reihe nach in einen binären Baum ein:
  4,6,6,8,5,1,2,7
- Wieviele Vergleiche brauchst du, bis du weisst, ob die 8 bzw. die 3 gespeichert ist.
- Wieviele Vergleiche bräuchtest du bei der linearen Suche.

Warum Hashen?

Motivation

Oft muss man Daten so abspeichern, dass man sie schnell wiederfinden kann. Daher wollen wir aus den Daten selber ihre Speicher-Adresse (z.B. Array-Index) berechnen.

Was brauchen wir?

Dazu brauchen wir eine Funktion, die Hash-Funktion, die aus unseren Daten eine Adresse, den Hash-Wert, berechnet. Diese sollte daher möglichst kollisionsfrei sein. Eine Kollision tritt auf, wenn zwei verschiedene Daten den gleichen Hashwert haben.

Probleme mit Kollisionen

Kollisionen lassen sich nicht vermeiden. Wenn man die Daten nur naiv im Array speichet (wie wir jetzt), können Daten überschrieben werden. Dagegen gibt es Strategien (z.B. lineares Sondieren).

Warum Hashen?

Motivation

Oft muss man Daten so abspeichern, dass man sie schnell wiederfinden kann. Daher wollen wir aus den Daten selber ihre Speicher-Adresse (z.B. Array-Index) berechnen.

Was brauchen wir?

Dazu brauchen wir eine Funktion, die Hash-Funktion, die aus unseren Daten eine Adresse, den Hash-Wert, berechnet. Diese sollte daher möglichst kollisionsfrei sein. Eine Kollision tritt auf, wenn zwei verschiedene Daten den gleichen Hashwert haben.

Probleme mit Kollisionen

Kollisionen lassen sich nicht vermeiden. Wenn man die Daten nur naiv im Array speichet (wie wir jetzt), können Daten überschrieben werden. Dagegen gibt es Strategien (z.B. lineares Sondieren).

Warum Hashen?

Motivation

Oft muss man Daten so abspeichern, dass man sie schnell wiederfinden kann. Daher wollen wir aus den Daten selber ihre Speicher-Adresse (z.B. Array-Index) berechnen.

Was brauchen wir?

Dazu brauchen wir eine Funktion, die Hash-Funktion, die aus unseren Daten eine Adresse, den Hash-Wert, berechnet. Diese sollte daher möglichst kollisionsfrei sein. Eine Kollision tritt auf, wenn zwei verschiedene Daten den gleichen Hashwert haben.

Probleme mit Kollisionen

Kollisionen lassen sich nicht vermeiden. Wenn man die Daten nur naiv im Array speichet (wie wir jetzt), können Daten überschrieben werden. Dagegen gibt es Strategien (z.B. lineares Sondieren).

Ein einfacher Hash-Algorithmus

Aufgabe

Verwendet die unten angegebene Hash-Funktion um:

- die Werte 0,5,23,28,42,1001,123456789 zu speichern.
- überprüfe, ob die 28, die 5 bzw. die 2006 in der Liste enthalten waren.

Hash-Funktion: Die „wiederholte Quersumme“

Berechne wiederholt die Quersumme, bis sich das Ergebnis nicht mehr ändert, d.h., bis das Ergebnis einstellig ist:

$$q^0(n) := q(n) = \text{Quersumme von } n$$
$$q^i(n) = q(q^{i-1}(n)) \quad \forall i \in \mathbb{N}$$
$$f(n) = q_i(n) \text{ mit } i = \min\{i \in \mathbb{N}_0 | q_i(n) = q_{i+1}(n)\}$$

Aufgabenstellung

Ziel dieser Aufgabe ist es, den Java Taschenrechner aus früheren Übungsaufgaben so zu erweitern, dass er auch Terme mit mehr als einem Operator verarbeiten kann.

Randbedingungen

- Ausdrücke in Postfixnotation
- Verwendung eines Kellers als Datenstruktur
- Zur Vereinfachung nur ganze Zahlen

Postfixnotation

Bei Ausdrücken in Postfixnotation (auch umgekehrte polnische Notation genannt) befindet sich im Gegensatz zur gewohnten Infixnotation der Operator jeweils hinter seinen Operanden. So wird der Term “2 + 3“(Infixnotation) in Postfixschreibweise als “2 3 +“ ausgedrückt. Bei dieser Darstellung kann auf Klammern verzichtet werden.

Beispiele

- $1 + 2 + 3 + 4$
- $2 * (3 + 4)$
- $(3 * 2) + (4 * 3)$

Postfixnotation

Bei Ausdrücken in Postfixnotation (auch umgekehrte polnische Notation genannt) befindet sich im Gegensatz zur gewohnten Infixnotation der Operator jeweils hinter seinen Operanden. So wird der Term “2 + 3“(Infixnotation) in Postfixschreibweise als “2 3 +“ ausgedrückt. Bei dieser Darstellung kann auf Klammern verzichtet werden.

Beispiele

- $1 + 2 + 3 + 4 \Longleftrightarrow$ 1 2 + 3 + 4 + oder 1 2 3 4 + + +
- $2 * (3 + 4) \Longleftrightarrow$ 2 3 4 + *
- $(3 * 2) + (4 * 3) \Longleftrightarrow$ 3 2 * 4 3 * +

Das Kellerprinzip

- Zugriffsprinzip: Das zuletzt auf den Stapel gelegte Element wird als erstes wieder entfernt (Last In First Out)
- Stapeloperationen sind
  - `push(x)`
  - `x = pop()`

Das Kellerprinzip

- Zugriffsprinzip: Das zuletzt auf den Stapel gelegte Element wird als erstes wieder entfernt (Last In First Out)
- Stapeloperationen sind
  - `push(x)`
  - `x = pop()`

Das Ganze in Java

- Interne Darstellung als verkettete Liste (Klasse `Node`)
- Die Klasse `Stack` bietet folgende Funktionen:
  - `void Stack.push(int)`
  - `int Stack.pop()`
  - `boolean Stack.isEmpty()`
- Man darf alle Klassen in eine Datei, benannt nach der Klasse mit der `main`-Methode, schreiben.

Node.java

```
class Node{
  int value;      // value of this node
  Node next;      // reference to following node

  Node(int x, Node n){
    this.value = x;
    this.next  = n;
  }

    // return following node (deleting this one)
  public Node pop(){ return this.next; }
    // send own value, like getValue()
  public int top(){ return value; }
}
```

Stack.java

```
class Stack{
  private Node start;     // reference to the top element

    // standard constructor
  public Stack(){ this.start = null; }
    // delete node on top of the stack
  public int pop(){
    int x = start.top();
    start = start.pop();
    return x;
  }
    // create new node on top of the stack
  public void push(int x){
    start = new Node(x, start);
  }
    // true if no elements on stack
  public boolean isEmpty(){ return start == null; }
}
```

Vorgehensweise

- Wie verarbeiten wir die Eingabe (`String`?)
- Unterscheidung zwischen Operanden und Operatoren?
- Speichern der Operanden?
- Erkennung der Operation?
- Implementierung der Operationen?
- Ausgabe

Vorgehensweise

- Wie verarbeiten wir die Eingabe (`String`?)
- Unterscheidung zwischen Operanden und Operatoren?
- Speichern der Operanden?
- Erkennung der Operation?
- Implementierung der Operationen?
- Ausgabe

Vorgehensweise

- Wie verarbeiten wir die Eingabe (`String`?)
- Unterscheidung zwischen Operanden und Operatoren?
- Speichern der Operanden?
- Erkennung der Operation?
- Implementierung der Operationen?
- Ausgabe

Vorgehensweise

- Wie verarbeiten wir die Eingabe (`String`?)
- Unterscheidung zwischen Operanden und Operatoren?
- Speichern der Operanden?
- Erkennung der Operation?
- Implementierung der Operationen?
- Ausgabe

Vorgehensweise

- Wie verarbeiten wir die Eingabe (`String`?)
- Unterscheidung zwischen Operanden und Operatoren?
- Speichern der Operanden?
- Erkennung der Operation?
- Implementierung der Operationen?
- Ausgabe

Vorgehensweise

- Wie verarbeiten wir die Eingabe (`String`?)
- Unterscheidung zwischen Operanden und Operatoren?
- Speichern der Operanden?
- Erkennung der Operation?
- Implementierung der Operationen?
- Ausgabe

CalcRPN.java

```
class CalcRPN{
  private static Stack input;

  public static void main(String[] argv){
    String[] args; input = new Stack(); // initialize stack
    String tmp = In.readLine();   // read term to compute
    str = tmp.split(" ");
      // loop for each operand and operator
    for(int x = 0; x<str.length; x++){
      if(isInteger(str[x])){        // operand to push on stack
        input.push(Integer.parseInt(str[x]));
      } else {                      // apply operation to stack
        switch(str[x].charAt(0)){
          case '+': add(); break;
          ...
        }
      }
    }
    Out.println("Result is: " + input.pop());
  }
```

CalcRPN.java (Fortsetzung)

```
...
    public static void add(){
      int y = input.pop();   // get second operand from stack
      int x = input.pop();   // get first operand from stack
      input.push(x+y);       // put result of operation on stack
    }

    ...

        // true if s only constists of digits (0-9)
    static boolean isInteger(String s){
      for(int x=0; x<s.length(); x++)
        if(s.charAt(x)<'0' || s.charAt(x)>'9') return ↵
            false;
      return true;
    }
}
```