PGdP Woche #5 – Rekursion

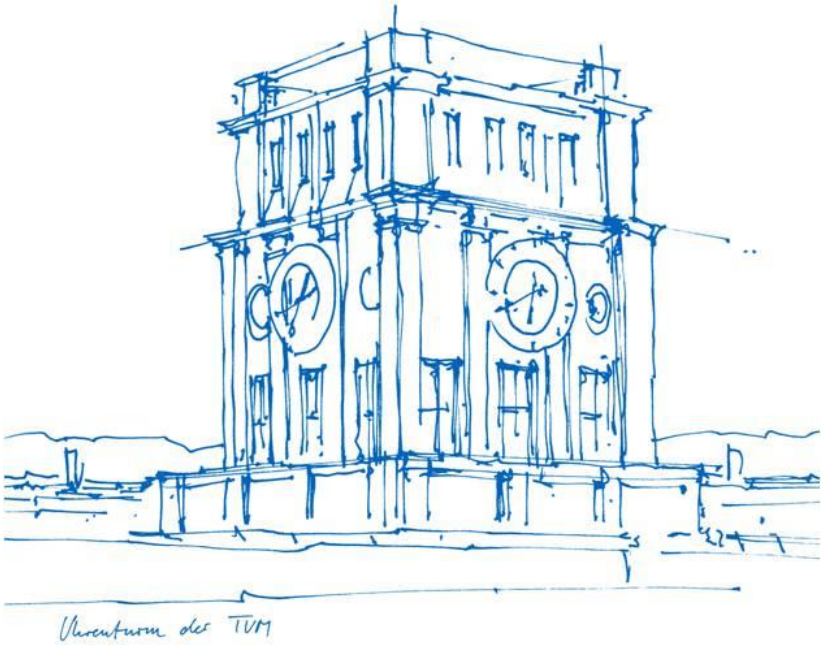
Sebastian Oßner

Technische Universität München

Garching, 18.November 2019

Ablauf:

1. P01 – StyleGuide
2. P02 – BinomialRekursion
3. P03 – Pinguinlabyrinth
4. P04 – MiniJava – Primzahlen



Lehrstuhl für Musterverfahren
Fakultät für Mustertechnik
Technische Universität München

Style Guide I

- Variablen und Methoden: lower CamelCase z.B.:

```
int anzahlEinwohner;
public void calculateResult() {…}
```

- Klassen: upper CamelCase z.B.:

```
public class Receptionist {…}
public class DefaultTextPrinter {…}
```

- Whitespace (Beispiele):

```
int calculateResult(int a, int b) {
        return a+b;
}
for (int i = 0; i < 10; i++) {
        System.out.println(i);
}
```

1 tab =
2/4 spaces

Lehrstuhl für Musterverfahren
Fakultät für Mustertechnik
Technische Universität München

Style Guide II - Kommentare

Do:

```
// Calculates Binomial coefficient n choose k
static int bino(int n, int k) {…}
```

Don‘t:

```
/* Calculates sum of two integers a and b and writes result
into third varialbe c, after creating said variable c*/
int c = a+b;
```

Do:

```
// If the Node has no Parent either, (Tree consists of a
single node), we'll set its value to -1
if (node.parent == null) {
        node.key = -1;
        return true;
}
```

Don’t:

```
// If a is greater than b, we return c
if(a > b) {return c;}
```

Rekursion

Eine rekursive Funktion ist eine Funktion, die sich direkt oder indirekt selbst aufruft.

Pros:
- Graph traversal
- Weniger Zeitkomplexität (bei richtiger Implementierung)
- Oft sind mathematische Funktionen rekursiv definiert => leicht zu implementieren
- Eleganter

Cons:
- Mehr Speicher benötigt
- Stack overflows
- Dauert länger (wenn man es schlecht implementiert)

Häufige Fehler:

Base Case vergessen/falsch platziert: (führt zu Stack Overflow)

```
static int factorial(int n) {
        return n*factorial(n-1);
}

static int factorial(int n) {
        return n*factorial(n-1);
        if (n == 1) {
                return 1;
        }
}
```

Programm terminiert nicht:

Problemgröße muss mit jedem Schritt verkleinert werden

Lehrstuhl für Musterverfahren
Fakultät für Mustertechnik
Technische Universität München

P02 - Binomialkoeffizient

```
static int bino(int n, int k) // Binomialkoeffizient k aus n

Vorraussetzung:
        n >= k >= 0 // Einlesen der Zahlen in main-Methode

Base case:
        bino(n,n) = bino(n,0) = 1

Rekursive Definition:
        bino(n,k) = bino(n-1,k-1) + bino(n-1,k)
```

Lehrstuhl für Musterverfahren
Fakultät für Mustertechnik
Technische Universität München

P03 - Pinguinlabyrinth

- BaseCases:
  - maxDistance < 0
  - Wir sind außerhalb vom Labyrinth
  - Wir befinden uns in einer Wand oder in einem alten aktiven Pfad

- Falls wir einen Pinguin finden, erhöhen wir die Anzahl der gefundenen Pinguine und MaxDistance

- Für alle Koordinaten im Umkreis von 1:
  - Dann updaten wir die Position unseres Players und des aktiven Pfads
  - Rekursiver Aufruf mit neuer Position und verringerter Distanz

- Position ist komplett durchgesucht => alten aktiven Pfad updaten und Pinguine zurückgeben

Lehrstuhl für Musterverfahren
Fakultät für Mustertechnik
Technische Universität München

P04 – Primzahlen MiniJava

- Virtual Machine herunterladen:
  JAR auf Artemis -> VAM (oben links) -> Select Machine -> vam.machines.minijvm

- Schreibt eine Textdatei, die dann in der JAR ausgeführt werden kann

Lehrstuhl für Musterverfahren
Fakultät für Mustertechnik
Technische Universität München

Verfügbare Operationen

Binäre Operationen: Verbrauchen oberste 2 Variablen und schreiben Ergebnis auf Stack

`Arith.:` NEG, ADD, SUB, MUL, DIV, MOD

`Logik:` NOT, AND, OR

`Comparison:` LESS, LEQ, EQ, NEQ

Schreiben von Konstante oben auf den Stack:

```
CONST i, TRUE, FALSE
```

Speicher:

```
LOAD i   //Schreibt i-te Konstante auf Stack
STORE i //Schreibt oberste Stackzelle in i-te Speicherzelle
ALLOC i //Reserviert i Zellen im Speicher
```

Sprünge:

```
JUMP i, FJUMP //Unbedingter, bzw. bedingter Sprung (false)
```

I/0:

```
READ, WRITE
```

Ende des Programms:

```
HALT
```