

Funktionale und objektorientierte Programmierkonzepte



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Präsenz-Sprechstunde B

Simon Hock, Nhan Huynh, Daniel Mangold

12.01.2022 Präsenzsprechstunde B im Raum in Raum S103/**123!**



Rekursion [25min]

Vorgehensweise bei rekursiven Aufgaben

Rekursive Auswertung von Arithmetischen Ausdrücken in Präfix-Notation

Arbeitsphase [75min]



„Eine Rekursion ist eine Funktion, die sich selbst aufruft. Dabei wird versucht das Problem auf eine einfachere Variante des **gleichen** Problems rekursiv zu reduzieren, welches dann gelöst wird.“



1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann
 - ▣ Teillösungen der Gesamtlösung
2. Das kleinste Problem ist ein Rekursionsanker
3. Überlege, wie die kleinen Problemen kombiniert werden können, um die ursprüngliche Problemstellung zu lösen
4. Kombiniere die Teillösungen zu einer Gesamtlösung

Rekursive Auswertung von Arithmetischen Ausdrücken in Präfix-Notation



TECHNISCHE
UNIVERSITÄT
DARMSTADT

■ Blatt 06 H4

■ Präfix-Notation Beispiele:

- $+ 2\ 3 = 5$
- $* - 12\ 3 = *(-12)3 = -3$
- $+ 4 - 5\ 1 = +4(-5)1 = 8$

Notation	Addition zweier Zahlen
Präfix	$+ x\ y$
Infix	$x + y$
Postfix	$x\ y +$

Tabelle: Überblick von Notationen

H4.1: Vorbereitung: Klasse ReturnData



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Schreiben Sie zunächst eine `public`-Klasse `ReturnData`, welche zwei `public`-Objektattribute vom Typ `int` hat: `result` und `nextIndex`. Diese Klasse muss einen parameterlosen `public`-Konstruktor besitzen. Sie dürfen aber gerne auch andere Konstruktoren implementieren, solange weiterhin ein parameterloser `public`-Konstruktor zur Verfügung steht.

```
1 public class ReturnData {  
2  
3     public int result;  
4     public int nextIndex;  
5  
6     public ReturnData() {}  
7  
8     // Optional  
9     public ReturnData(int result, int nextIndex) {  
10         this.result = result;  
11         this.nextIndex = nextIndex;  
12     }  
13 }
```


H4.2: Methoden `evaluate` und `evaluateRecursively`



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erweitern Sie Klasse `StrangeThings` aus 2 um eine `public`-Klassenmethode `evaluate`, die den Wert eines arithmetischen Ausdrucks in Präfix-Notation (das heißt: zuerst Operator, dann Operanden) berechnet. Es geht also um eine Notation wie in Racket – aber im Gegensatz zu Racket haben wir keine Klammern. Außerdem beschränken wir uns bei den Operanden auf einstellige Zahlen (also 0 ... 9) und bei dem Operator um die Subtraktion.

Information: „Ein arithmetischer Ausdruck besteht aus einer Folge von arithmetischen Operanden, die durch (arithmetische) Operatoren und Klammern voneinander getrennt (oder miteinander verknüpft) sind.“

Beispiel für arithmetische Ausdrücke



TECHNISCHE
UNIVERSITÄT
DARMSTADT

$$\begin{array}{rcl} & & 1 & (1) \\ & & - 12 & (2) \\ & - & - 123 & (3) \\ - & - & 12 - 34 & (4) \end{array}$$

1

(1)

- 1 2

(2)

$1-2=-1$

- - 1 2 3

(3)

$1-2=-1$

-1-3=-4

- - 1 2 - 3 4

(4)

$1-2=-1$ $3-4=-1$

-1-(-1)=0

Konkret hat Methode `evaluate` einen Parameter vom formalen Typ „`Array`“ vom primitiven Typ `char`“ und Rückgabotyp `int`. Die Methode `evaluate` darf ohne Überprüfung davon ausgehen, dass das Array mindestens die Länge 1 hat und jede Komponente des Arrays entweder eine Ziffer `'0'`, ..., `'9'` oder das Zeichen `'-'` enthält und der mit der `char`-Kette repräsentierte Präfix-Ausdruck ein korrekt gebildeter Präfix-Ausdruck ist, in dem jede Subtraktion genau zwei Operanden hat.

Konkret hat Methode `evaluate` einen Parameter vom formalen Typ „`Array`“ vom primitiven Typ `char`“ und Rückgabotyp `int`. Die Methode `evaluate` darf ohne Überprüfung davon ausgehen, dass das Array mindestens die Länge 1 hat und jede Komponente des Arrays entweder eine Ziffer `'0'`, ..., `'9'` oder das Zeichen `'-'` enthält und der mit der `char`-Kette repräsentierte Präfix-Ausdruck ein korrekt gebildeter Präfix-Ausdruck ist, in dem jede Subtraktion genau zwei Operanden hat.

Information: Beim Parameterwert wird zwischen dem aktuellen Wert (der Parameterwert, der eingesetzt wird) und dem formalen Wert (der in der Definition des Parameters in der Methode steht) unterschieden.

Neben der Methode `evaluate` soll in Klasse `StrangeThings` eine `private`-Klassenmethode `evaluateRecursively` mit einem ersten Parameter namens `array` vom Typ „`Array`“ vom primitiven Typ `char`, einem zweiten Parameter namens `startIndex` vom Typ `int` und Rückgabotyp `ReturnData` existierten.

Die Methode `evaluateRecursively` soll aus dem `Array array` den numerischen (Teil-)Ausdruck ab Index `startIndex` auswerten und ein `ReturnData`-Objekt zurückliefern, dessen Objektattribut `result` den Zahlenwert des ausgewerteten numerischen (Teil-) Ausdrucks ab Index `startIndex` und Objektattribut `nextIndex` den ersten Index des nächsten numerischen Teilausdrucks enthält.

Die Methode `evaluate` ruft die Methode `evaluateRecursively` auf, wobei `evaluateRecursively` als ersten aktuellen Parameter den ersten aktuellen Parameter von `evaluate` erhält (beides jeweils „Array von `char`“). Als zweiten aktuellen Parameter erhält `evaluateRecursively` den Wert `0`. Nach dem Aufruf von `evaluateRecursively` liefert die Methode `evaluate` den Wert des Objektattributs `result` des von `evaluateRecursively` gelieferten Objekts vom Typ `ReturnData`.

Das Verhalten der Methode `evaluateRecursively` ist vom Zeichen am Index `startIndex` in Array `array` abhängig:

Im Fall, dass der `char`-Wert `array[startIndex]` eine Ziffer darstellt, wurde ein atomarer arithmetischer Ausdruck gefunden und die Methode liefert ein `ReturnData`-Objekt zurück, dessen Objektattribut `result` den durch das Zeichen dargestellten Wert (für `'4'` zum Beispiel 4) und `nextIndex` den Wert `startIndex+1` enthält.

Im anderen Fall, also wenn der `char`-Wert gleich `'-'` ist, müssen die nächsten beiden arithmetischen Teil-Ausdrücke `e1` und `e2` ausgewertet und ein geeignetes `ReturnData`-Objekt zurückgegeben werden.

Implementieren Sie die Methoden `evaluate` und `evaluateRecursively` wie beschrieben.

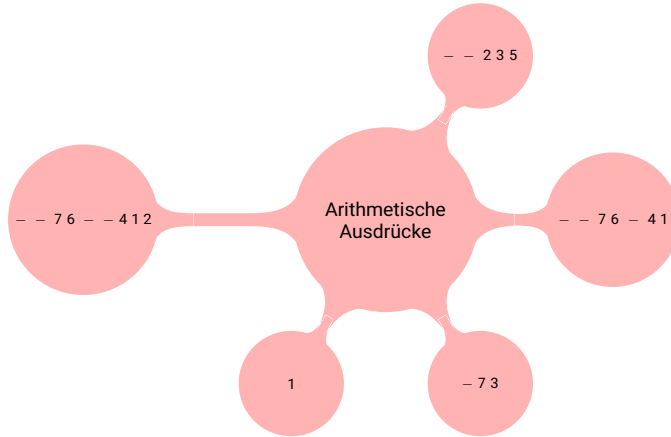
- array: Enthält den arithmetischen Ausdruck, bspw. ['-', '1', '2']
- startIndex: Aktueller Index beim Durchlauf durch das Array array
- ReturnData
 - ▣ result: Enthält das Ergebnis des bisher berechneten Teilausdrucks
 - ▣ nextIndex: Gibt den ersten Index des nächsten Teilausdrucks an

```
1 private static ReturnData evaluateRecursively(char[] array, int startIndex) {  
2     // TODO  
3 }
```

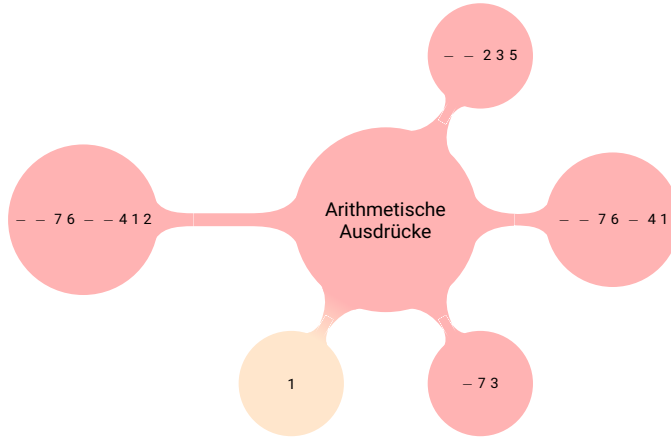
- Hauptrekursion wird in der Hilfsmethode `evaluateRecursively` durchgeführt
- `ReturnData` enthält das Ergebnis der Auswertung

```
1 public static int evaluate(char[] array) {  
2     ReturnData data = evaluateRecursively(array, 0);  
3     // Alternatively: return evaluateRecursively(array, 0).result;  
4     return data.result;  
5 }
```

1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann



1. Identifiziere, wie das Problem auf eine einfachere Variante des Problems zerlegt werden kann und direkt gelöst werden kann



2. Das kleinste Problem ist ein Rekursionsanker

Algorithm evaluateRecursively(array, startIndex)

- 1: **if** array[startIndex] is a digit **then**
 - 2: **return** array[startIndex], startIndex + 1
 - 3: **end if**
-

```
1 private static ReturnData evaluateRecursively(char[] array, int startIndex) {  
2     char token = array[i];  
3     if (Character.isDigit(token)) {  
4         return new ReturnData(  
5             Character.getNumericValue(token),  
6             startIndex + 1  
7         );  
8     }  
9     // TODO  
10 }
```

-
3. Überlege, wie die kleinen Problemen kombiniert werden können, um die ursprüngliche Problemstellung zu lösen

1. kleines Problem und 2. kleines Problem
Kombinieren

- Wann werden die beiden Probleme zusammengefügt?

-
3. Überlege, wie die kleinen Problemen kombiniert werden können, um die ursprüngliche Problemstellung zu lösen

1. kleines Problem und 2. kleines Problem

Kombinieren

- Wann werden die beiden Probleme zusammengefügt? \Rightarrow Wenn ein Operator gelesen wird

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then
2:   return array[startIndex], startIndex + 1
3: else
4:   leftOperand  $\leftarrow$  evaluateRecursively(array, ...)
5:   rightOperand  $\leftarrow$  evaluateRecursively(array, ...)
6:   return leftOperand.result + rightOperand.result, ...
7: end if
```

- Von welchen Indizes starten die Operanden bzw. was ist der nächste Index nach der Kombination?

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then  
2:   return array[startIndex], startIndex + 1  
3: else  
4:   leftOperand  $\leftarrow$  evaluateRecursively(array, ...)  
5:   rightOperand  $\leftarrow$  evaluateRecursively(array, ...)  
6:   return leftOperand.result + rightOperand.result, ...  
7: end if
```

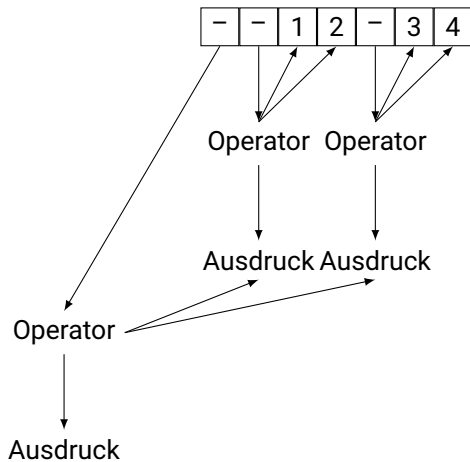


Abbildung: Visualisierung der Auswertung

- Wir müssen rekursiv den nächsten Index abprüfen

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then  
2:   return array[startIndex], startIndex + 1  
3: else  
4:   leftOperand  $\leftarrow$  evaluateRecursively(array, startIndex + 1)  
5:   rightOperand  $\leftarrow$  evaluateRecursively(array, ...)  
6:   return leftOperand.result + rightOperand.result, ...  
7: end if
```

- Wir müssen rekursiv nach dem letzten besuchten Index von dem linken Operand abprüfen

Algorithm evaluateRecursively(array, startIndex)

```
1: if array[startIndex] is a digit then  
2:   return array[startIndex], startIndex + 1  
3: else  
4:   leftOperand  $\leftarrow$  evaluateRecursively(array, startIndex + 1)  
5:   rightOperand  $\leftarrow$  evaluateRecursively(array, leftOperand.nextIndex)  
6:   return leftOperand.result + rightOperand.result, ...  
7: end if
```

- Der nächste Index nach der Kombination ist der Index nach der Auswertung von dem rechten Teilausdruck

Algorithm evaluateRecursively(array, startIndex)

- 1: **if** array[startIndex] is a digit **then**
 - 2: **return** array[startIndex], startIndex + 1
 - 3: **else**
 - 4: leftOperand \leftarrow evaluateRecursively(array, startIndex + 1)
 - 5: rightOperand \leftarrow evaluateRecursively(array, leftOperand.nextIndex)
 - 6: **return** leftOperand.result + rightOperand.result, rightOperand.nextIndex
 - 7: **end if**
-

```
1 private static ReturnData evaluateRecursively(char[] array, int startIndex) {
2     char token = array[i];
3     if (Character.isDigit(token)) {
4         return new ReturnData(
5             Character.getNumericValue(token), startIndex + 1);
6     }
7     ReturnData leftOperand = evaluateRecursively(array, startIndex + 1);
8     ReturnData rightOperand = evaluateRecursively(array,
9         leftOperand.nextIndex);
10    return new ReturnData(
11        leftOperand.result + rightOperand.result,
12        rightOperand.nextIndex);
13 }
```



- `array = '-', '-', '1', '2', '-', '3', '4'`
- Auswertung von `evaluateRecursively(array, 0)`

-	-	1	2	-	3	4
---	---	---	---	---	---	---

1. `startIndex = 0`
2. Ist das aktuelle Zeichen eine Zahl? Nein
3. Bestimme rekursiv zuerst den linken und dann den rechten Operanden

-	-	1	2	-	3	4
---	---	---	---	---	---	---

4. `startIndex = 1`
5. Ist das aktuelle Zeichen eine Zahl? Nein
6. Bestimme rekursiv zuerst den linken und dann den rechten Operanden

-	-	1	2	-	3	4
---	---	---	---	---	---	---

7. startIndex = 2
8. Ist das aktuelle Zeichen eine Zahl? Ja
9. Gebe die Zahl und den nächsten Index zurück
 - ▣ ReturnData(1, 3)
10. Rückkehr zu 6.
11. Bestimme den rechten Operanden

-	-	1	2	-	3	4
---	---	---	---	---	---	---

7. startIndex = 3
8. Ist das aktuelle Zeichen eine Zahl? Ja
9. Gebe die Zahl und den nächsten Index zurück
 - ▣ ReturnData(2, 4)
10. Rückkehr zu 6.
11. Kombiniere beide Operanden und gebe das Ergebnis der Kombination zurück
 - ▣ ReturnData(-1, 4)
12. Rückkehr zu 3.
13. Bestimme den rechten Operanden

-	-	1	2	-	3	4
---	---	---	---	---	---	---

- 12. startIndex = 4
- 13. Ist das aktuelle Zeichen eine Zahl? Nein
- 14. Bestimme rekursiv zuerst den linken und dann den rechten Operanden

-	-	1	2	-	3	4
---	---	---	---	---	---	---

- 15. startIndex = 5
- 16. Ist das aktuelle Zeichen eine Zahl? Ja
- 17. Gebe die Zahl und den nächsten Index zurück
 - ▣ ReturnData(3, 6)
- 18. Rückkehr zu 14.
- 19. Bestimme den rechten Operanden

-	-	1	2	-	3	4
---	---	---	---	---	---	---

- 20. startIndex = 6
- 21. Ist das aktuelle Zeichen eine Zahl? Ja
- 22. Gebe die Zahl und den nächsten Index zurück
 - ▣ ReturnData(4, 7)
- 23. Rückkehr zu 14.
- 24. Kombiniere beide Operanden und gebe das Ergebnis der Kombination zurück
 - ▣ ReturnData(-1, 7)
- 25. Rückkehr zu 3.
- 26. Kombiniere beide Operanden und gebe das Ergebnis der Kombination zurück
 - ▣ ReturnData(0, 7)
- 27. Abbruch: Ergebnis ist 0

Selbstständiges Arbeiten