

Ludwig-Maximilians-Universität München

Institut für Informatik

Priv.-Doz. Dr. Thom Frühwirth, Dr. Slim Abdennadher, Alexander Knapp

**Einführung in die Informatik:
Programmierung und Softwareentwicklung, WS 99/00
Probeklausur**

Aufgabe P-1

(6 Punkte)

BNF-Grammatik

Es soll die Syntax einer einfachen Form arithmetischer Ausdrücke beschrieben werden. Diese Ausdrücke verwenden nur positive ganze Zahlen ohne führende Nullen sowie die zweistelligen Operatoren „+“, „-“, „*“ und „/“. Es kommen keine Klammern vor, die Teilausdrücke zusammenfassen. Zusätzlich können in diesen Ausdrücken die einstelligen Funktionen „sin“ (für Sinus) und „cos“ (für Cosinus) verwendet werden. Diese Funktionen haben ein einziges Argument, das in Klammern gesetzt direkt an den Funktionsnamen angeschlossen wird. Dies ist die einzige Art, wie Klammern in den Ausdrücken verwendet werden. Funktionen können an jeder Stelle im Ausdruck verwendet werden, an der auch eine Zahl stehen könnte. Das Argument einer solchen Funktion kann wiederum ein beliebiger Ausdruck sein.

1. Geben Sie eine passende BNF Grammatik an.

Lösungsvorschlag:

Mit dem Startsymbol *Term* beschreibt

$$NonNull = "1" | "2" | "3" | \dots | "9"$$
$$Null = "0"$$
$$Ziffer = Null | NonNull$$
$$Zahl = Null | NNZahl$$
$$NNZahl = NonNull \{ Ziffer \}$$
$$Term = Zahl | "sin" "(" Term ")" | "cos" "(" Term ")" |$$
$$Term "+" Term | Term "-" Term | Term "*" Term | Term "/" Term$$

das Gewünschte

2. Geben Sie eine Ableitung des folgenden Ausdrucks anhand Ihrer Grammatik an:
 $11 * \sin(3+4) / 2.$

Lösungsvorschlag:

Eine mögliche Ableitung ist

```
Term → Term "*" Term
      → Term "*" Term "/" Term
      → Term "*" "sin" "(" Term ")" "/" Term
      → Term "*" "sin" "(" Term "+" Term ")" "/" Term
      → Zahl "*" "sin" "(" Term "+" Term ")" "/" Term
      → Zahl "*" "sin" "(" Zahl "+" Term ")" "/" Term
      → Zahl "*" "sin" "(" Zahl "+" Zahl ")" "/" Term
      → NNZahl "*" "sin" "(" Zahl "+" Zahl ")" "/" Zahl
      → NonNull Ziffer "*" "sin" "(" Zahl "+" Zahl ")" "/" Zahl
      → "1" Ziffer "*" "sin" "(" Zahl "+" Zahl ")" "/" Zahl
      → "1" NonNull "*" "sin" "(" Zahl "+" Zahl ")" "/" Zahl
      → "1""1" "*" "sin" "(" Zahl "+" Zahl ")" "/" Zahl
      → "1""1" "*" "sin" "(" NNZahl "+" Zahl ")" "/" Zahl
      → "1""1" "*" "sin" "(" NonNull "+" Zahl ")" "/" Zahl
      → "1""1" "*" "sin" "(" "3" "+" NNZahl ")" "/" Zahl
      → "1""1" "*" "sin" "(" "3" "+" NonNull ")" "/" Zahl
      → "1""1" "*" "sin" "(" "3" "+" "4" ")" "/" Zahl
      → "1""1" "*" "sin" "(" "3" "+" "4" ")" "/" NNZahl
      → "1""1" "*" "sin" "(" "3" "+" "4" ")" "/" NonNull
      → "1""1" "*" "sin" "(" "3" "+" "4" ")" "/" "2"
```

Aufgabe P-2

(8 Punkte)

Komplexität

In der Telefonzentrale einer Firma muß erfaßt werden, welche Nebenstelle mit welcher anderen Nebenstelle telephoniert. Da es nur eine festgelegte Anzahl von Nebenstellen gibt (wir nehmen an, 100), kann jede Nebenstelle durch eine Zahl zwischen 0 und 99 identifiziert werden. Die Verbindungen können dann in einem zweidimensionalen Array `verbindungen` von `boolean`-Werten erfaßt werden. In dieser Matrix wird jeweils an der Stelle `verbindungen[i][j]` ein `true` eingetragen, wenn Nebenstelle Nr. *i* mit Nebenstelle Nr. *j* verbunden ist, und ein `false` sonst. Jede Nebenstelle kann zu einer Zeit immer nur mit maximal einer anderen Nebenstelle verbunden sein.

1. Es wird eine Klasse `Telefonzentrale` vorgegeben, die die vermittelten Verbindungen verwaltet. Vervollständigen Sie die Methoden `verbinde`, `frei` und `anzahlVerbindungen`.

```
public class Telefonzentrale {
    public boolean[][] verbindungen = new boolean[100][100];
    public void verbinde(int nebenstelle1,
```

```

        int nebenstelle2) { ... }
// trägt eine neue Verbindung zwischen den gegebenen
// Nebenstellen ein ohne (!) zuvor zu überprüfen, ob
// beide Nebenstellen frei sind.

public boolean frei(int nebenstelle) { ... }
// überprüft, ob die angegebene Nebenstelle augenblicklich
// frei ist und gibt in diesem Fall true zurück.

public int anzahlVerbindungen() { ... }
// gibt die Gesamtanzahl der augenblicklich in der
// Telefonzentrale vermittelten Verbindungen zurück.

```

Lösungsvorschlag:

```

public void verbinde(int nebenstelle1,
                    int nebenstelle2) {
    verbindungen[nebenstelle1][nebenstelle2] = true;
    verbindungen[nebenstelle2][nebenstelle1] = true;
}

public boolean frei(int nebenstelle) {
    boolean verbunden = false;
    for (int i = 0; i < 100; i++)
        if (nebenstelle != i)
            verbunden |= verbindungen[nebenstelle][i];
    return !verbunden;
}

public int anzahlVerbindungen() {
    int sum = 0;
    for (int i = 0; i < 100; i++)
        for (int j = i+1; j < 100; j++)
            if (verbindungen[i][j])
                sum++;
    return sum;
}

```

2. Geben Sie die worst-case Zeitkomplexitäten der von Ihnen in (1) implementierten Methoden in O-Notation unter der Annahme an, daß die maximale Anzahl von Nebenstellen als Parameter für die Komplexität aufgefaßt wird. Begründen Sie Ihre Antwort.

Lösungsvorschlag:

Die Suche nach dem Worst-Case erübrigt sich, da alle Methoden unabhängig vom Inhalt des Arrays immer die gleiche Komplexität haben. Die Methode `verbinde` hat offensichtlich die Komplexität $O(1)$, da sie parameterunabhängig nur zwei Indexzuweisungen vornimmt. Die Methode `frei` führt eine einfache Schleife über eine Array-Zeile mit $O(n)$ Schleifendurchläufen aus, in denen eine Operation mit konstanter Zeit vorgenommen wird. Ihre Komplexität ist also $O(n)$. `anzahlVerbindungen` entspricht einer Doppelschleife über die Hälfte

aller Arrayelemente. Für jedes Element wird konstante Zeit verbraucht. Sie hat also die Komplexität $O(n^2/2) = O(n^2)$.

Aufgabe P-3

(6 Punkte)

Klassenstrukturen

Sie sollen die Objektstruktur eines kleinen Systems beschreiben, das Himmelskörper bearbeitet. Es gibt hier zwei Formen von Himmelskörpern: Sterne und Planeten. Jeder Himmelskörper wird durch einen Namen (als Zeichenkette) identifiziert. Für Sterne wird zusätzlich ihre Helligkeits-Stufe (durch eine ganze Zahl gegeben) gespeichert, für Planeten wird zusätzlich die Eigenschaft gespeichert, ob Sie bewohnt sind. Jeder Planet kreist um genau einen Stern oder um genau einen anderen Planeten.

1. Implementieren Sie eine geeignete Java-Klassenstruktur für dieses Problem.
2. Formulieren Sie die folgende Methode in der Klasse der Planeten:

```
public String starName() { ... }  
// gibt für den Planeten, auf dem die Methode aufgerufen wird,  
// einen String mit dem Namen des Sterns zurück, um den die  
// Planeten in der Galaxie kreisen, zu der auch dieser Planet  
// gehört.
```

Als Galaxie betrachten wir in (2) einen Stern mit der Gesamtheit aller Planeten, die um diesen Stern kreisen. Damit gehören natürlich auch alle Planeten zu einer Galaxie, die um einen Planeten kreisen, der bereits zur Galaxie gehört. Galaxien sollen nicht explizit gespeichert werden.

Lösungsvorschlag:

```
public class Himmelskoerper {  
    String name;  
  
    public String starName() {  
        return name;  
    }  
}  
  
class Stern extends Himmelskoerper {  
    int helligkeit;  
}  
  
class Planet extends Himmelskoerper {  
    boolean bewohnt;  
    Himmelskoerper orbit;  
  
    public String starName() {  
        return orbit.starName();  
    }  
}
```

```
}  
}
```

Aufgabe P-4

(4 Punkte)

Rekursive Datenstrukturen

Listen aus ganzen Zahlen können in Java als Objektstrukturen aus Instanzen der folgenden Klasse repräsentiert werden:

```
class List {  
    boolean isempty;    // true genau dann, wenn leere Liste  
    int contents;       // Listenelement  
    List next;         // Rest der Liste  
  
    public List() {  
        isempty = true;  
    }  
  
    public List(int c) {  
        contents = c;  
        next = new List();  
        isempty = false;  
    }  
  
    public List(int c, List n) {  
        contents = c;  
        next = n;  
        isempty = false;  
    }  
}
```

1. Implementieren Sie eine Methode `int average()` für die Klasse `List`, die den ganzzahligen Durchschnittswert aller Zahlen in der aktuellen Liste berechnet.

Lösungsvorschlag:

```
public int sum() {  
    if (isempty)  
        return 0;  
    return contents + next.sum();  
}  
  
public int length() {  
    if (isempty)  
        return 0;  
    return 1 + next.length();  
}
```

```
public int average() {
    if (isempty)
        return 0;
    return sum()/length();
}
```

2. Implementieren Sie eine Methode `List below(int x)` für die Klasse `List`, um die Teil-
liste aller ganzen Zahlen der aktuellen Liste zu berechnen, die kleiner oder gleich x sind.

Lösungsvorschlag:

```
public List below(int x) {
    if (isempty)
        return new List();
    else
        if (contents <= x)
            return new List(contents, next.below(x));
        else
            return next.below(x);
}
```