Aufgabe 2

33.Bundeswettbewerb Informatik 2014/'15

Der Script-Tim

Inhaltsverzeichnis

1 Entwicklung einer textuellen Darstellungsweise

Ich habe keine neue textuelle Darstellung erfunden; ich habe auf bereits vorhandene Datendarstellungen zurückgegriffen und sie in diesem Kontext angewendet.

1.1 JSON

Wie geschaffen für diese einfache textuelle Datendarstellung ist natürlich JSON. Hierbei wird ein Balken oder das Mobile allgemein durch ein Paar geschweifter Klammern repräsentiert.

$$\{\ldots\}$$

Eine Struktur befindet sich innerhalb geschweifter Klammern und besteht aus einem Wertepaar. Die Werte einer Struktur werden durch einen Doppelpunkt getrennt. Die einzelnen Strukturen werden durch Kommata voneinander getrennt;

$$\{\text{Abstand:Gewicht, Abstand:}\{\text{Abstand:Gewicht,Abstand:Gewicht}\}\}$$

Das Mobile aus der Angabe würde also folgendermaßen aussehen:

$$\{4:\{2:1,-1:2\},1:2,-2:3,-4:2\}$$

Abbildung 1 zeigt die Kontrolle der Balanciertheit des Mobiles in der JSON-Darstellung.

1.2 XML

Und hier noch - natürlich - die Mutter aller Datendarstellungen: XML. Von mir im Programm nicht verwendet, aber durchaus eine Möglichkeit, auch wenn sie wahrscheinlich nicht „einfach“ im Sinne der Aufgabenstellung ist.

```
<Mobile>
    <Balken abstand="4">
        <Figur abstand="2" gewicht="1" />
        <Figur abstand="-1" gewicht="2" />
    </Balken>
    <Figur abstand="1" gewicht="2" />
    <Figur abstand="-2" gewicht="3" />
    <Figur abstand="-4" gewicht="2" />
</Mobile>
```

Nicht, dass ich als Webprogrammierer nicht selbst auf XML gekommen wäre, aber in den öffentlich zugänglichen Lehrertipps steht explizit ein Hinweis auf XML (!).

2 Entwurf eines Programmes

2.1 Lösungsidee

Wie schon aus der textuellen Darstellungsweise ersichtlich, besteht für mich das Mobile bzw. die Strukturen aus zwei „Bausteinen“:

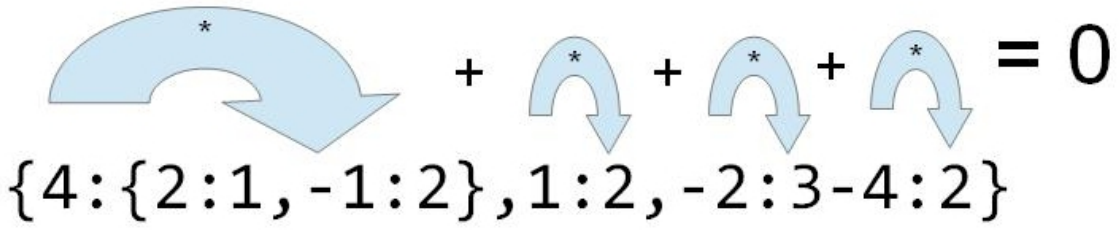

Abbildung 1: Kontrolle der Balanciertheit bei der JSON-Darstellung

1. **Figuren**: Sind immer das letzte Glied einer Struktur und besitzen eine Masse
2. **Balken**: besitzen minimal 2, maximal 4 weitere Balken oder Figuren.

Die Drähte spiele beim Programm keine Rolle und können vernachlässigt werden. Sowohl Figuren und Balken haben eine gemeinsame Eigenschaft: ihren Aufhängepunkt. Da ein Balken maximal vier gewichte tragen/aufnehmen kann (bzw. darf) müssen ggf. Unterbalken erstellt werden. Diese Balken müssen aber ihrerseits auch ausbalanciert werden. Meine Lösungsidee besteht darin, die Erstellung des Balkens an seine Ausbalancierung zu knüpfen. Die Ausbalancierung fußt darauf, dass es immer mindestens zwei Paare von Gewichten auf den unterschiedlichen Seiten des Aufhängepunktes geben muss, die sich gegenseitig ausgleichen (vgl.Abb.2). Bei vier Strukturen an einem Balken sind es dann zwei Paare; hier muss lediglich aufgepasst werden, dass es nicht zwei gleiche Werte auf einer Seite der Aufhängung geben darf (ggf. wird ein 'Differenzierungsfaktor' x erhöht, der an ein Paar angesetzt wird) (vgl. Abb.3). Der 'Nachteil' dieses 'jeweils-zwei-gleichen-sich-aus-Prinzips' ist, dass es keinen Balken geben kann, an dem eine ungerade Anzahl an Strukturen hängt (sprich: 3).

2.2 Wahl der Programmiersprache

Da ich die grundlegenden Bestandteile des Mobiles durch Klassen darstellen und das Mobile als dynamisches Array/Container speichern wollte, kam C++ dafür nicht in Frage (speichern von Instanzen verschiedener Klassen im selben Container meines Wissens nicht möglich). Ich habe deshalb auf die weniger typisierte (C++-ähnliche) Scriptsprache PHP zurückgegriffen.

Information
Für die Ausführung des PHP-Scriptes wird ein Webserver benötigt. Genau wie in Junioraufgabe 1 habe ich das Script wieder auf meinen Webserver geladen:
`http://www.tim-hollmann.de/BwInf/Aufgabe2/index.php`. Der Hash lautet
945651736746a670d897901a50c4e29e.

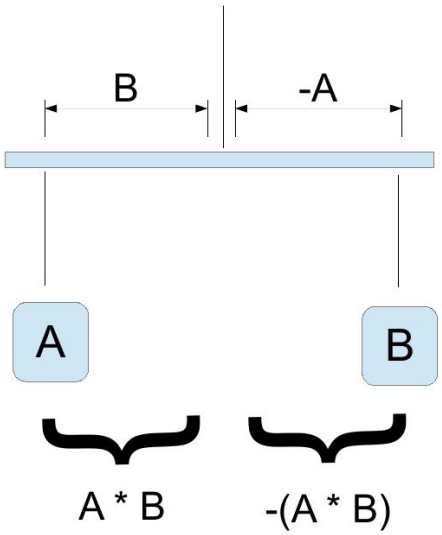

Abbildung 2: Einfache Ausbalancierung (ein Paar)

2.3 Klassen und Funktionen

Allgemein ist das Mobile in zwei Bestandteile aufgeteilt: Balken und Figuren. Folglich definiere ich zwei Klassen:

```
39 class Figur{
40     public $abstand = 0;
41     public $gewicht = 0;
42 
43     public function __construct($gewicht){
44         $this->gewicht = $gewicht;
45     }
46 
47     public function gewicht(){ return $this->gewicht; }
48 
49 }
50 
51 class Balken{
52     public $abstand = 0;
53 
54     public $strukturen = array();
55 
56     public function gewicht(){
57         $gewicht_insg = 0;
58         foreach($this->strukturen as $struktur){
59             $gewicht_insg += $struktur->gewicht();
60         }
61         return $gewicht_insg;
62     }
63 
64     public function __construct($gewichte = array()){
```

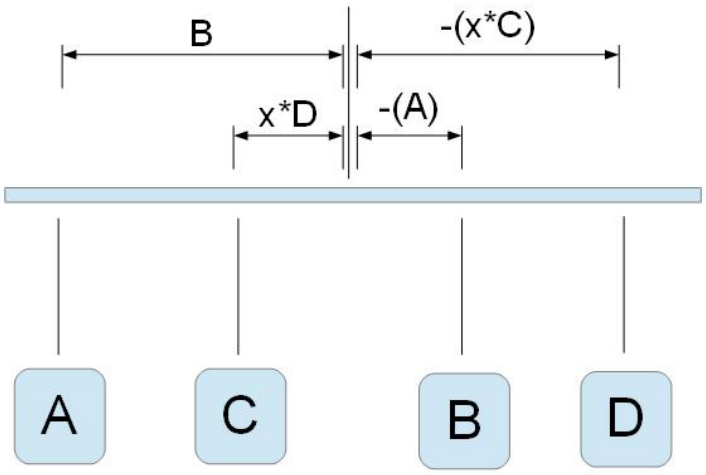

Abbildung 3: Ausbalancierung von zwei Paaren an einem Balken

Zwar könnte man auf die Membervariable *$gewicht* von *Figur* auch direkt zugreifen, da sie als *public* definiert ist, jedoch ist es während des Programmablaufes nötig, das Gewicht einer gegebenen Instanz zu ermitteln, ohne berücksichtigen zu können, von welcher Klasse sie abgeleitet wurde. Deshalb definiere ich einfach genau wie in *Balken* eine Memberfunktion *Figur::gewicht()*, die ich dann einfach für beide Klassen verwende.

```
133 function Struktur($gewichte = array()){
134     if (sizeof($gewichte) == 1){
135         return new Figur($gewichte[0]);
136     }else{
137         return new Balken($gewichte);
138     }
139 }
```

Die Funktion *Struktur* ist dafür da, eine Instanz der Klassen Figur oder Balken je nach Anzahl der in *$gewichte* übergebenen Gewichte zu erzeugen.(1 Gewicht → Figur, alles andere → Balken). So muss während des Programmablaufes bei der Erstellung von Strukturen nicht zwischen *Figur* und *Balken* unterschieden werden.

Zusätzlich gibt es noch eine Funktion *ausgeben*, die aber erst bei der Ausgabe der textuellen Datenrepräsentation eine Rolle spielt.

2.4 Programmablauf

2.4.1 Laden der Gewichtsdaten

Die Gewichtsdaten werden vom Benutzer über das HTML-Formular übergeben; sie werden in der Variable *$_REQUEST[gewichte]* an das PHP-Script übergeben.

```
161     $gewichte = $_REQUEST["gewichte"];
```

Vor Verarbeitung der Daten werden noch einige Korrekturen vorgenommen

```
162     $gewichte = preg_replace("| |", "", $gewichte); //evtl. Leerzeichen entfernen
163     $gewichte = preg_replace("|,|", ";", $gewichte); //evtl. Kommata in Semikolon
164     $gewichte = explode(";", $gewichte);//am Semikolon trennen -> Array
```

Schlussendlich sind die Gewichte als Array in *$gewichte* gespeichert.

2.4.2 Ausbalancierung und Erstellen von Unterbalken

Bei jeder neuen Erzeugung einer Instanz der Klasse *Balken* wird dem Konstruktor ein Array der Gewichte übergeben, die am Balken „untergebracht“ werden sollen. Wenn es mehr als vier sind, kann der Konstruktor natürlich auch weitere Unterbalken erstellen. Bei der Zuteilung der Gewichte auf die Aufhängungen nimmt der Konstruktor gleich die Ausbalancierung vor:

```
64     public function __construct($gewichte = array()){
65
66         if (sizeof($gewichte) == 2){
67             $aufhaengungLinks = new Figur($gewichte[0]);
68             $aufhaengungRechts = new Figur($gewichte[1]);
69
70
71             $aufhaengungLinks->abstand = $aufhaengungRechts->gewicht();
72             $aufhaengungRechts->abstand = -($aufhaengungLinks->gewicht());
73
74             $this->strukturen[] = $aufhaengungLinks;
75             $this->strukturen[] = $aufhaengungRechts;
76         }
77
78         if (sizeof($gewichte) == 3){
79             $aufhaengungLinks = new Balken(array($gewichte[0],$gewichte[1]));
80             $aufhaengungRechts = new Figur($gewichte[2]);
81
82             $aufhaengungLinks->abstand = $aufhaengungRechts->gewicht();
83             $aufhaengungRechts->abstand = -($aufhaengungLinks->gewicht());
84
85             $this->strukturen[] = $aufhaengungLinks;
86             $this->strukturen[] = $aufhaengungRechts;
87         }
88
89         if (sizeof($gewichte) >= 4){
90             $aufhaengung = array();
91             $aufhaengung[0] = array();
92             $aufhaengung[1] = array();
93             $aufhaengung[2] = array();
94             $aufhaengung[3] = array();
95
96             for($x = 0; $x < sizeof($gewichte); $x++){
97                 $aufhaengung[$x % 4][] = $gewichte[$x];
98             }
99
100            $aufhaengung1 = Struktur($aufhaengung[0]);
101            $aufhaengung2 = Struktur($aufhaengung[1]);
```

```
102             $aufhaengung3 = Struktur($aufhaengung[1]);
103             $aufhaengung4 = Struktur($aufhaengung[1]);
104 
105             $OK = false;
106             for($faktor = 1; ($faktor <= 3 && $OK == false) ; $faktor++){
107                 //Paar 1: 1vs2
108                 $aufhaengung1->abstand = $aufhaengung2->gewicht();
109                 $aufhaengung2->abstand = -($aufhaengung1->gewicht());
110 
111                 //Paar 2: 3vs4
112                 $aufhaengung3->abstand = $faktor*$aufhaengung4->gewicht();
113                 $aufhaengung4->abstand = -($faktor*$aufhaengung3->gewicht());
114 
115                 if ($aufhaengung1->abstand != $aufhaengung3->abstand && $aufhaengung2
                        ->abstand != $aufhaengung4->abstand){
116                     $OK = true;
117                 }else{
118                     $OK = false;
119                 }
120             }
121 
122             if ($OK == false){ Die("FEHLER - STRATEGIE FEHLGESCHLAGEN."); }
123 
124             $this->strukturen[] = $aufhaengung1;
125             $this->strukturen[] = $aufhaengung2;
126             $this->strukturen[] = $aufhaengung3;
127             $this->strukturen[] = $aufhaengung4;
128         }
129 
130     }
```

Die Art der Ausbalancierung ist abhängig von der Anzahl der Gewichte:

- **2**: *[Z.66-76]* Wenn es genau zwei Gewichte auszubalancieren gilt, gibt es logischer Weise nur ein Paar und der Differenzierungsfaktor x fällt weg. Zunächst werden die zwei Figuren erstellt (*[Z.67+68]*) und dann genau wie in Abbildung 2 „ausbalanciert“: das Gewicht des einen ist der Abstand des anderen; mit der Bedingung, dass die Aufhängung rechts einen negativen Abstand hat. Anschließend werden die Strukturen in der Membervariable *Balken::strukturen* als Array gespeichert.

- **3**: *[Z.78-87]* Die Anzahl von drei Gewichten ist ungünstig; es können direkt keine Pärchen gebildet werden. Deshalb werden die ersten beiden zu einem Unterbalken zusammengeschlossen (*[Z.79]*) und dieser bildet dann mit dem dritten Gewicht ein Paar. Der restliche Ablauf deckt sich mit der Ausbalancierung von zwei Gewichten.

- $\geq$ **4**: *[Z.89-130]* Da es mehr als vier Gewichte geben kann, werden die Gewichte zunächst mit dem *modulo*-Operator in eine von vier Aufhängungen zugeteilt (*[Z.90-98]*). Diese Aufhängungen werden dann zu einem Balken oder einer Figur zusammengefasst (hier kommt *Struktur* zum Einsatz, da hier die Anzahl der zugewiesenen Gewichte nicht jeweils erst noch überprüft wird)(*[Z.100-103]*). Es gibt so

immer zwei Paare von Aufhängungen: 1+2 und 3+4. Diese Paare werden jeweils unabhängig wie bei den einfachen Paaren ausbalanciert. Da aber nun auf der einen Seite des Balkens ein Abstand doppelt belegt worden sein könnte, kommt - falls wirklich doppelt belegt wurde - ein Differenzierungsfaktor x zum Einsatz (*[Z.106]*): er wird mit den Abständen eines Paares multipliziert. Da beide Figuren des Paares damit multipliziert werden, bleiben sie ausgeglichen, aber ihre Abstände verändern sich und sind somit nicht mehr mit denen des ersten Paares identisch. Sehr schön sieht man dies, wenn man sich die Ausgabe für die Gewichte *{2,2,2,2}* ansieht:

$$\{2:2,-2:2,4:2,-4:2\}$$

Hier wurde dem letzten Paar der Differenzierungsfaktor *2* zugewiesen, damit es nicht jeweils auch den Abstand *2* und *-2* hätte.

2.4.3 Ausbalancieren des Mobiles

Das Ausbalancieren des Mobiles geschieht dann durch den primitiven Befehl

```
170     //Ausbalancierung in Gang setzen
171     $Mobile = Struktur($gewichte);
```

2.4.4 Darstellen des Mobiles

Ich habe als textuelle Darstellung JSON gewählt. Jedoch ist es mir auch nach vielen Versuchen nicht gelungen, den C++-eigenen Parser die Darstellung für mich erstellen zu lassen (meine Objekte sind zu abstrakt). Manuelle Erzeugung der textuellen Ausgabe:

```
141 function ausgeben($strukturen){
142     $ausgabe = "{";
143     $didfirst = false;
144 
145     foreach($strukturen as $struktur){
146         if ($struktur instanceof Balken){
147             $ausgabe .= (($didfirst == true) ? "," : "") . $struktur->abstand . ":" .
                    ausgeben($struktur->strukturen);
148         }elseif($struktur instanceof Figur){
149             $ausgabe .= (($didfirst == true) ? "," : "") . $struktur->abstand .":" .
                    $struktur->gewicht();
150         }
151         $didfirst = true;
152     }
153     $ausgabe .= "}";
154     return $ausgabe;
155 }
```

```
173     //Mobile darstellen (Textform)
174     $ausgabe = "";
175 
176     if ($Mobile instanceof Balken){
177         $ausgabe = ausgeben($Mobile->strukturen);
```

```
178     }else{
179         //Eine Figur!
180         $ausgabe = "{0:".$Mobile->gewicht()."}";
181     }
182 
183     Echo "<a href=\"?gewichte\">Gewichte neu eingeben</a><h2>Mobile (ausbalanciert;
        textural)</h2><hr />".$ausgabe."<hr />";
```

Die Kontrolle der Balanciertheit wird in Abbildung 1 gezeigt.

2.5 Beispiele

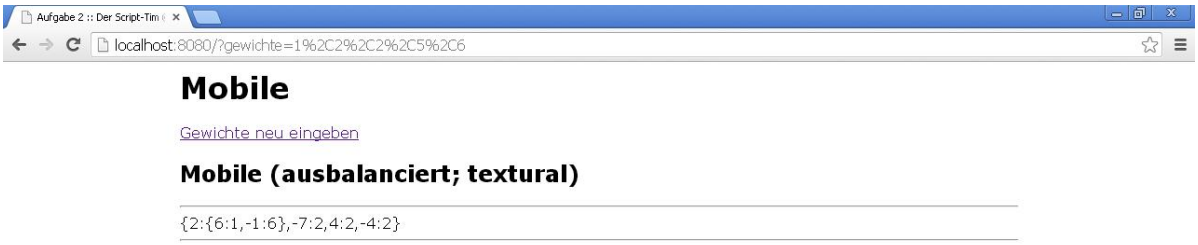

Abbildung 4: Beispielhafte textuelle Darstellung im Browser

2.5.1 Einige beliebige Beispiele

Gewichte $w_1 \dots w_n$	**Ausgabe**
1,2,2,5,6	{2:{6:1,-1:6},-7:2,4:2,-4:2}
3,5,9,1	{5:3,-3:5,10:5,-10:5}
1	{0:1}
3,3,3,3,3	{3:{3:3,-3:3},-6:3,9:3,-9:3}