Teil III

Grundlegende Datenstrukturen

Kapitel 9

Felder

Ziele:

- die Datenstruktur der Felder verstehen: mathematisch, als Objekte und im Speicher
- Grundlegende Algorithmen auf Feldern kennenlernen: Suche im ungeordneten und geordneten Feld
- Eindimensionale und Mehrdimensionale Felder verstehen

Ein Feld ist ein Tupel von Komponentengliedern, auf die über einen Index direkt zugegriffen werden kann. Mathematisch kann ein Feld mit n Komponenten vom Typ `type` als endliche Abblidung

$$I_n \longrightarrow \texttt{type}$$

mit Indexbereich $I_n = \{0, 1, \ldots, n-1\}$ beschreiben. n ist die Länge des Feldes.

Da `type` ein beliebiger Typ ist, kann man auch Felder als Komponenten haben. Dies führt zu mehrdimensionalen Feldern.

Ein Feld `a` der Länge 6 kann folgendermaßen dargestellt werden:

`a` :	`'V'`	`'E'`	`'R'`	`'L'`	`'A'`	`'G'`
Index :	0	1	2	3	4	5

`a` kann beschrieben werden als die Abbildung `a` : $\{0, \ldots, 5\} \longrightarrow$ **`char`**

$$\texttt{a}[i] = \begin{cases} \texttt{'V'} & \text{falls } i = 0 \\ \texttt{'E'} & \text{falls } i = 1 \\ \vdots & \\ \texttt{'G'} & \text{falls } i = 5 \end{cases}$$

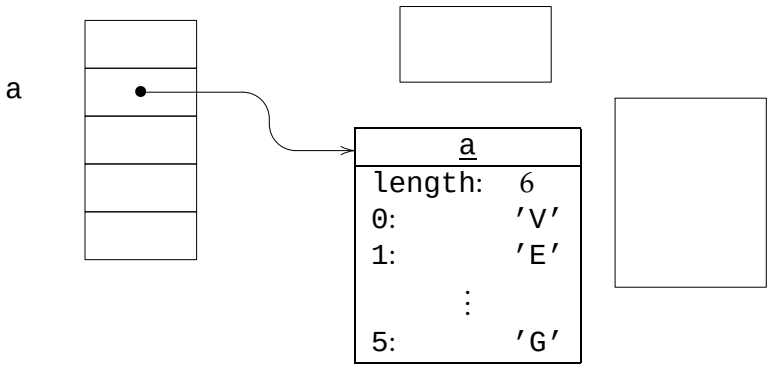
In Java wird ein Feld mit n Elementen aufgefaßt als ein Objekt mit den $n+1$ Attributen

`int`	`length`
`type`	0
⋮	
`type`	$(n-1)$

Das Feld `a` wird also folgendermaßen als Objekt aufgefaßt:

a	
`length:`	6
`0:`	`'V'`
`1:`	`'E'`
⋮	
`5:`	`'G'`

Die Speicherorganisation von `a` hat folgende Gestalt:



9.1 Deklaration von Feldtypen und -variablen

In Java haben Feldtypen die Form `TypeName[]...[]`. Beispiele für Feldtypen sind

```
int[], int[][], boolean[]
```

Variablendeklaration ist wie vorher, allerdings gibt es spezielle Ausdrücke zur Initialisierung von Feldern.

9.2 Eindimensionale Felder

Durch

```
type[] var = new type[n]
```

wird eine Variable `var` vom Typ `type[]` deklariert und Speicherplatz für ein einstufiges Feld der Länge n reserviert. Außerdem werden dadurch implizit n zusammengesetzte Variablen `var[0]`, ..., `var[n-1]` erzeugt, mit denen man auf die Werte der Komponenten von `var` zugreifen und diese Werte verändern kann. Man kann auch sofort Anfangswerte zuweisen („Initialisierung des Feldes“). Durch

`type[] var =` $\{v_0, \ldots, v_{n-1}\}$

wird eine neue Variable `var` der Länge n vom Typ `type[]` deklariert und es werden ihr die Werte $v_0, \ldots, v_{n-1}$ zugewiesen. Dabei müssen alle $v_0, \ldots, v_{n-1}$ vom Typ `type` sein.

Jede der obigen Deklarationen erzeugt ein (Objekt vom Typ) Feld, dessen Länge sich nicht mehr ändern kann.

Bemerkung: Da in Java die Länge des Feldes aber nicht Bestandteil des Typs ist, kann einer Feldvariablen ein Feld mit einer anderen als der initial angegebenen Länge zugewiesen werden.

Wir vereinbaren dieses Feld.

```
char[] a = new char[6];
char[] b = new char[6];
```

Typ von `a` ist **`char[]`**, d.h. der Typ eines einstufigen Feldes mit Elementen aus **`char`**. Mit `a[0]`, `a[1]`, ..., `a[5]` kann man auf die Komponenten von `a` zugreifen.

Man kann die Werte einzeln zuweisen:

```
a[0] = 'V';     a[3] = 'L';
a[1] = 'E';     a[4] = 'A';
a[2] = 'R';     a[5] = 'G';
```

Man kann beliebige einzelne Buchstaben ändern:

```
a[3] = 'R';
a[5] = 'T';
```

Das ergibt `'V' 'E' 'R' 'R' 'A' 'T'` als neuen Wert des Feldes. Außerdem hat `a[3]` nun den Wert `'R'`. Man kann das Feld auch durch direkte Angabe der Elemente erzeugen:

```
char[] c = {'V','E','R','L','A','G'};
```

Diese Art der Initialisierung ist aber nur in einer Deklaration zulässig.

9.3 Algorithmen auf Feldern

1. **Suche nach dem Index eines minimalen Elements eines Feldes.** Gegeben sei folgendes Feld:

3	-1	15	1	-1

   Man sucht ein minimales Element mit folgendem Algorithmus:

a) Bezeichne $minIndex$ den Index des kleinsten Elements

b) Initialisierung $minIndex = 0$

c) Durchlaufe das ganze Feld. In jedem Schritt i vergleiche den Wert von $minIndex$ (d.h. `a[minIndex]`) mit dem Wert des aktuellen Elements (d.h. `a[i]`). Falls `a[i] < a[minIndex]` setze $minIndex = i$

Quelltext:

```
class C {
    int[] a;

    int minSuch()
    {
        int minIndex = 0;
        for (int i = 1; i < a.length; i++)  // Optimierung, da
        {                                   // a[0] < a[0] falsch ist
            if (a[i] < a[minIndex])
                minIndex = i;
        }
        return a[minIndex];
    }
}
```

2. **Binäre Suche eines Elements e im geordneten Feld.** Sei a ein geordnetes Feld mit den Grenzen j und k, d.h. $a[i] \leq a[i+1]$ für $i = j, \ldots, k$; also z.B.:

a :	3	7	13	15	20	25	28	29
	j	$j+1$	$\cdots$					k

Um den Wert e in a zu suchen, teilt man das Feld in der Mitte und vergleicht e mit dem Element in der Mitte:

- Ist $e < a[mid]$, so sucht man weiter im linken Teil $a[j], \ldots, a[mid-1]$.
- Ist $e = a[mid]$, hat man das Element gefunden.
- Ist $e > a[mid]$, so sucht man weiter im rechten Teil $a[mid+1], \ldots, a[k]$.

Quelltext:

```
class C
{
    int[] a;

    boolean binSuch(int e)
    {
        int j = 0,
```

```
            k = a.length - 1,
            mid;
        boolean found = false;

        while (j <= k & !found) {
            mid = (j + k) / 2;

            if (e < a[mid])
                k = mid - 1;
            else
            {
                if (e = a[mid])
                    found = true;
                else
                    j = mid + 1;
            }
        }
        return found;
    }
}
```

9.4 Mehrdimensionale Felder

Allgemeiner wird durch

`type[]...[] var = new type[`n_1`]...[`n_i`][]...[]` $(i > 0)$

eine Variable `var` vom Typ `type[]...[]` deklariert und Speicherplatz für ein mehrstufiges Feld reserviert. Es muß mindestens die Länge n_1 des ersten Indexbereiches angegeben werden.

Zweidimensionale Felder können initialisiert werden mit

`type[][] var =` $f_0, \ldots, f_{n_1-1}$

wobei $f_0, \ldots, f_{n_1-1}$ jeweils Felder von Werten vom Typ `type` sind. Dabei können die Längen von $f_0, \ldots, f_{n_1-1}$ unterschiedlich sein. Analoges gilt für höherdimensionale Felder.

Ausdrücke

Die Syntax der Ausdrücke wird um zusammengesetzte Variablen

`var[`i_1`]...[`i_n`]`

und Initialisierungsausdrücke der Form

`new type[`n_1`]...[`n_i`][]...[] bzw. {`$v_0, \ldots, v_{n_1-1}$`}`

erweitert.

Expression ::=
*Var { "[" Expression "]" }**
| "new" TypeName "[" Expression "]" { "[" Expression "]" } { "[" "]" }**
| "{" Expression { "," Expression } "}"*

Beispiel: Tastentelefon
Ein Tastentelefon besteht aus 4 Zeilen und 3 Spalten:

	0	1	2
3	1	2	3
2	4	5	6
1	7	8	9
0	*	0	#

```
char[][] tastenwert = { { '*', '0', '#' },
                        { '7', '8', '9' },
                        { '4', '5', '6' },
                        { '1', '2', '3' } };
```

Dann gilt z.B.:

```
tastenwert[3][0] = '1';
tastenwert[0][2] = '#';
```

Bei der Initialisierung eines mehrstufigen Feldes durch **new** muß nur die Länge des am weitesten links stehenden Index festgelegt werden. Dadurch können die Längen der Elemente des Feldes verschieden sein. Ähnlich ist das mit der direkten Initialisierung möglich:

```
int[][] = { {1}, {1,2}, {1,2,3} };
System.out.println(a[0][0] + "-" + a[1][1] + "-" + a[2][2]);
```

ergibt die Ausgabe

```
1-2-3
```

9.5 Zusammenfassung

1. Felder sind mathematisch endliche Abbildungen von einem Indexbereich auf einen Elementbereich.
2. In Java sind Felder Objekte mit einer speziellen Syntax für den Zugriff auf die Attribute.
3. Klassische Suchalgorithmen sind die binäre Suche im geordneten Feld und die Suche nach dem Index mit dem kleinsten Element im ungeordneten Feld.