Arrays

Zie Scherp Scherper - Hoofdstuk 8

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- uitleggen waarom **arrays** veel data van hetzelfde type bundelen
- arrays **declareren**, **vullen** en met een loop **overlopen**
- begrijpen dat arrays **by reference** werken (en hoe je ze correct kopieert)
- de `System.Array`-methoden gebruiken (`Sort`, `Reverse`, `IndexOf`, …)
- arrays als **parameter** en **returntype** in methoden gebruiken
- **meerdimensionale** en **jagged** arrays herkennen

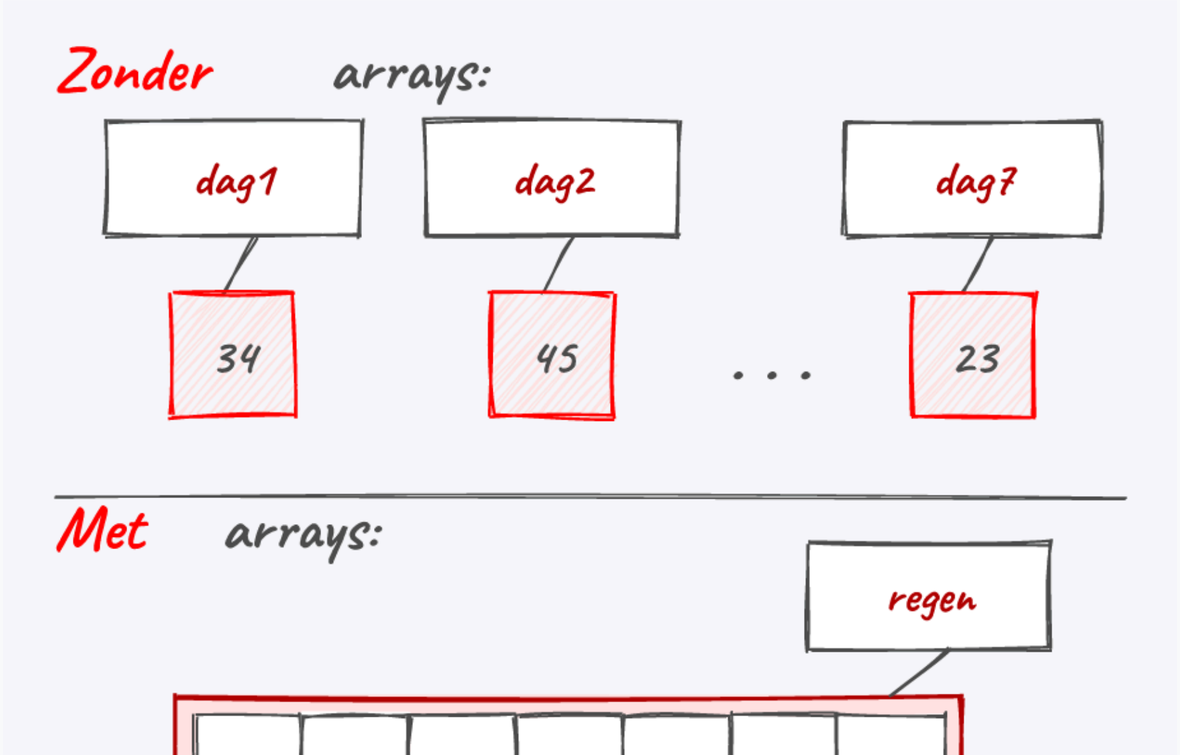
Het nut van arrays

Zeven losse variabelen voor zeven dagen regen?

```
1  int dag1 = 34; int dag2 = 45; int dag3 = 0; // ... dag7
```

Eén **array** bundelt dezelfde soort data in één variabele:

```
1  int[] regen = { 34, 45, 0, 34, 12, 0, 23 };
```



De kracht: een index in een loop

De index tussen `[ ]` mag een **variabele** zijn, dus genereer je ze in een loop:

```
int[] regen = { 34, 45, 0, 34, 12, 0, 23, 7, 20, 34, 7, 42 };
double som = 0;
for (int i = 0; i < regen.Length; i++)
{
    som += regen[i];
}
double gemiddelde = som / regen.Length;
```

Tip

Vanaf 3 of meer variabelen van dezelfde soort is een array bijna altijd het antwoord.

Even achteromkijken

Sorry dat we weer in het diepe duiken. Maar kijk eens achterom: schrik je hé? Je hebt al een aardige weg afgelegd sinds ik je voor het eerst in het zwembad gooide.

Alles wordt kinderspel als je er maar lang genoeg mee bezig bent.

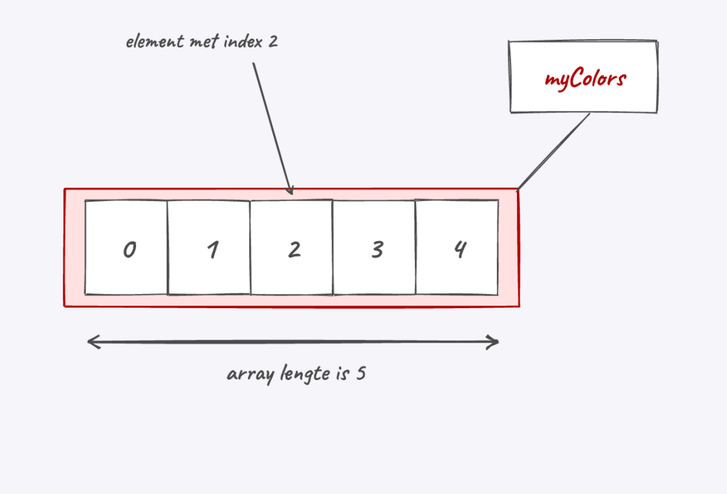
Een array declareren: 3 manieren

```
1  // 1) enkel declareren (nog geen array in het geheugen)
2  int[] verkoopCijfers;
3
4  // 2) meteen initialiseren met waarden
5  string[] kleuren = { "red", "green", "blue" };
6
7  // 3) een lege array van een vaste lengte
8  string[] namen = new string[5];
```

> **Belangrijk**
>
> Eens de lengte vastligt, kan een array **niet meer groeien of krimpen**. (Lists in hoofdstuk 12 lossen dat op.)

Index en accessor

- Benader een element met `array[index]`
- De index **start bij 0**
- Een array met **lengte 5** heeft indices **0 tot en met 4**



Lengte 5, indices 0 tot 4.

```
1  Console.WriteLine(myColors[1]); // tweede element
2  myColors[0] = "indigo";         // eerste element overschrijven
```

Overlopen met Length

`array.Length` geeft het aantal elementen. Ideaal als grens van een loop:

```
1  for (int i = 0; i < myColors.Length; i++)
2  {
3      Console.WriteLine(myColors[i]);
4  }
```

> ⚠ **Waarschuwing**
>
> `Console.WriteLine(myColors)` toont **niet** de inhoud. Overloop de array element per element.

Out of range

```
1  int[] getallen = { 1, 2, 3 };
2  Console.WriteLine(getallen[5]); // crash bij uitvoer
```

Een **`IndexOutOfRangeException`**. We bouwen aan een gebouw met 3 verdiepingen; vraag ik mijn metser om op de zesde te metsen, dan valt hij eraf.

Belangrijk

De compiler vangt dit **niet**: je ontdekt het pas bij de uitvoer.

Geheugen: value vs reference

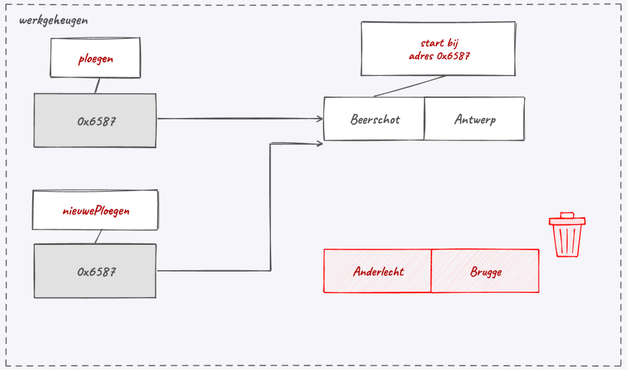
- **Value types** (`int`, `double`, …) bevatten de **waarde** zelf
- **Reference types** bewaren een **adres** naar de data elders in het geheugen

> **Belangrijk**
>
> **Arrays werken by reference.** Een array-variabele bevat dus niet de array zelf, maar een **verwijzing** ernaar.

De kopieer-val

```
1  string[] ploegen = { "Beerschot", "Antwerp" };
2  string[] nieuwe = { "Anderlecht", "Brugge" };
3  nieuwe = ploegen; // geen kopie!
```

Beide variabelen wijzen nu naar **dezelfde** array. `nieuwe[1] = "..."` past dus ook `ploegen` aan.



Een alias naar dezelfde array.

Een array echt kopiëren

`=` maakt een alias, geen kopie. Kopieer dus **element per element**:

```
1  for (int i = 0; i < ploegen.Length; i++)
2  {
3      nieuwe[i] = ploegen[i];
4  }
```

> **Tip**
>
> Of gebruik `Array.Copy(bron, doel, aantal)` uit de `System.Array`-bibliotheek.

System.Array

Handige ingebouwde methoden:

```
1  Array.Sort(myColors);     // alfabetisch / van klein naar groot
2  Array.Reverse(myColors);  // volgorde omkeren
3  Array.Clear(myColors, 0, myColors.Length); // alles op default
```

Zoeken in een array

```
1  int i = Array.IndexOf(myColors, "green"); // index, of -1
```

- **`IndexOf`** werkt op een **ongesorteerde** array: voor beginners bijna altijd de juiste keuze
- **`BinarySearch`** is sneller, maar **enkel op een gesorteerde** array

> **Belangrijk**
>
> `BinarySearch` op een ongesorteerde array geeft onzin. Sorteer eerst, of gebruik `IndexOf`.

Zelf zoeken met een loop

Staat rugnummer 12 in de top 5?

```
int teZoeken = 12;
int[] top5 = { 5, 10, 12, 25, 16 };
bool gevonden = false;
int index = 0;
do
{
    if (top5[index] == teZoeken) gevonden = true;
    index++;
} while (!gevonden && index < top5.Length);
```

Tip

Twee **synchrone** arrays (producten + prijzen op dezelfde index) doorzoek je op dezelfde manier.

Een string is een char-array

```
1  string zin = "Ik ben Tom";
2  char[] karakters = zin.ToCharArray();
3  karakters[8] = 'i';                  // "Ik ben Tim"
4  string terug = new string(karakters);
```

Een `string` is in feite een `char[]`, vandaar deze omzettingen.

Nuttige string-methoden

```
"Tim".Length;                 // 3
"Ik ben Reinhardt".IndexOf("ben"); // 3
"  tekst  ".Trim();           // "tekst"
"Tim".ToUpper();              // "TIM"
"Ik ben Mercy".Replace("Mercy", "Reinhardt");
"Ik ben Mercy".Remove(3, 4);  // "Ik Mercy"
```

Opmerking

Deze methoden geven een **nieuwe** string terug; de originele blijft ongewijzigd. Bewaar het resultaat.

Split en Join

```
1  string data = "12,13,20";
2  string[] delen = data.Split(',');        // {"12","13","20"}
3
4  string samen = String.Join(";", delen);  // "12;13;20"
```

`Split` levert een **array** op, `Join` voegt een array terug samen tot één string.

Arrays en methoden

Arrays gaan **by reference** naar een methode. Pas je de array aan in de methode, dan verandert de **originele** array mee:

```
1  static void PasAan(int[] arr) { arr[0] = 0; }
2
3  int[] nrs = { 1, 2, 3 };
4  PasAan(nrs);
5  Console.WriteLine(nrs[0]); // 0, niet 1
```

Array-grootte en returntype

In de signatuur geef je **geen** grootte mee; gebruik `.Length` in de methode:

```
static int[] MaakArray(int lengte, int beginwaarde)
{
    int[] result = new int[lengte];
    for (int i = 0; i < lengte; i++)
        result[i] = beginwaarde;
    return result;
}

int[] nieuw = MaakArray(4, 666);
```

> ⚠ **Waarschuwing**
>
> `static void LeesData(int[6] arr)` is **fout**: een grootte in de parameterlijst mag niet.

Opstartparameters: args

Nu snap je `string[] args` in `Main`: het zijn de **opstartparameters**.

```
1  for (int i = 0; i < args.Length; i++)
2      Console.WriteLine(args[i]);
3
4  if (args.Length >= 3 && args[2] == "cool")
5      Console.WriteLine("Ik ga akkoord!");
```

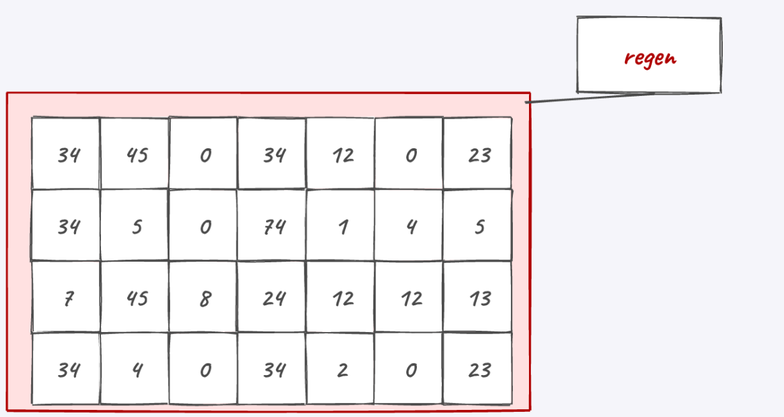
Belangrijk

Test **eerst** `args.Length >= 3`, dan pas `args[2]`. De `&&` stopt links bij `false` en zo vermijd je een crash.

Meerdimensionale arrays

Met een komma in de haken maak je een **rechthoekige** array:

```
1  int[,] matrix = new int[5, 10];
2
3  string[,] boeken =
4  {
5      { "Macbeth", "Shakespeare" },
6      { "Zie scherp", "Tim Dams" }
7  };
8  Console.WriteLine(boeken[1, 1]); // Tim Dams
```



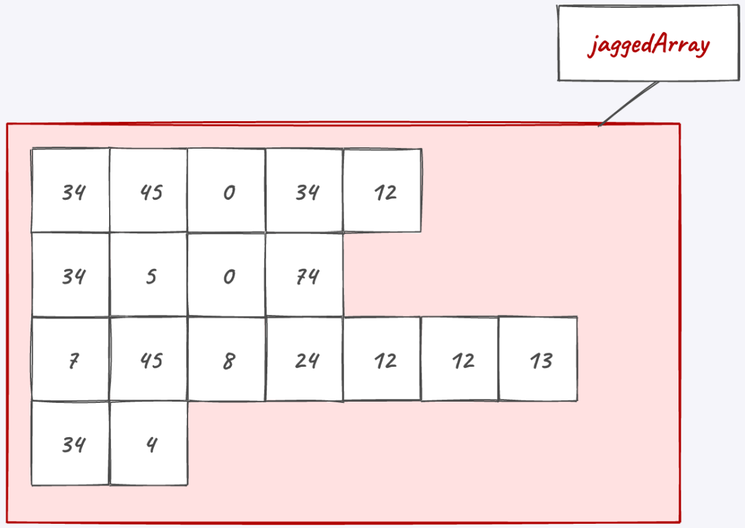
Een 2D-array.

> **Tip**
>
> `.Length` geeft het **totaal**; `GetLength(0)` en `GetLength(1)` de afzonderlijke dimensies, `.Rank` het aantal dimensies.

Jagged arrays

Een **array van arrays** met **verschillende** lengtes. Let op de dubbele `[][]`:

```
1  double[][] tickets =
2  {
3      new double[] { 3.0, 40, 24 },
4      new double[] { 123, 31.3 },
5      new double[] { 2.1 }
6  };
7  Console.WriteLine(tickets[0][1]); // 40
```



Een jagged array.

> **Opmerking**
>
> Vaak omslachtig en foutgevoelig. De collecties uit hoofdstuk 12 zijn meestal een gezonder alternatief.

Conclusie

- Een **array** bundelt veel waarden van hetzelfde type onder één naam; index start bij **0**
- Overloop arrays met een **loop** en `.Length`; let op de **`IndexOutOfRangeException`**
- Arrays werken **by reference**: `=` maakt een alias, kopieer dus element per element
- **`System.Array`** en string-methoden (`Split`, `Replace`, …) besparen veel werk
- Arrays gaan **by reference** naar methoden; **meerdimensionale** en **jagged** arrays voor speciale gevallen

> **Tip**
>
> Volgende halte: **object-georiënteerd programmeren**, met klassen en objecten.

Meer weten

Kennisclips

- Arrays
- Arrays en geheugen
- System.Array
- Algoritmes en arrays
- Strings en arrays
- Arrays en methoden
- Meer-dimensionale arrays

Oefeningen H8 · Quizlet flashcards

Zie verder: andere talen

Dezelfde concepten uit dit hoofdstuk, maar dan in andere talen. Handig om later code in Python of JavaScript te leren lezen.

Zie verder: vaste vs dynamische lengte

In **Python** bestaat geen array met vaste grootte, wel een dynamische `list`:

```
1 getallen = [1, 2, 3]
2 getallen.append(4)        # groeit gewoon mee
3 getallen.append("vier")  # mag zelfs een ander type zijn
```

Totaal anders dan een C#-array (vaste lengte, 1 type). Lijkt eerder op de `List` uit hoofdstuk 12.

Zie verder: geen bounds-checking in C

C# gooit netjes een `IndexOutOfRangeException`. In **C** is er geen enkele grenscontrole:

```c
int getallen[3] = {1, 2, 3};
printf("%d", getallen[5]); // geen foutmelding!
```

Je leest dan zomaar willekeurige geheugeninhoud of je programma crasht. Net dat maakt buffer overflow attacks mogelijk.

Zie verder: dezelfde val in JavaScript

Ook **JavaScript**-arrays werken by reference, dus `=` maakt een alias, geen kopie:

```
let ploegen = ["Beerschot", "Antwerp"];
let nieuwe = ploegen;     // zelfde array!
nieuwe[1] = "Brugge";
console.log(ploegen[1]);  // "Brugge"
```

Heel gelijkaardig aan C#. Een echte kopie: `let kopie = [...ploegen];`.

Samenvattende poster

Een handige samenvattende poster (Opgelet: gemaakt m.b.v. ChatGPT Image Gen 2).