

1.0.0

## H6: Arrays

# Array principes

## Array principes



Kennisclip

### Arrays

Stel je voor dat je, met de kennis die je nu hebt, een boodschappenlijstje moet programmeren. Met andere woorden, een programma dat vraagt om een aantal regels tekst in te typen, waarbij je dan items kan aanduiden als gekocht.

Je zou ruimte kunnen voorzien voor 3 items, voorgesteld als `string`, als volgt:

```
string item1, item2, item3;
Console.WriteLine("Wat is het 1e item dat je nodig hebt?");
item1 = Console.ReadLine();
Console.WriteLine("Wat is het 2e item dat je nodig hebt?");
item2 = Console.ReadLine();
Console.WriteLine("Wat is het 3e item dat je nodig hebt?");
item3 = Console.ReadLine();
// hier nog wat code om deze items te tonen
// of om te markeren dat ze in de kar geplaatst zijn
```

Een groot nadeel van deze aanpak: als je vier items wil ondersteunen, heb je meer code nodig. Als je vijf items wil ondersteunen, heb je nog meer code nodig. Deze code kan geen basis zijn voor een degelijk boodschappenlijstje.

Een tweede mogelijkheid bestaat erin heel het lijstje voor te stellen als een string, als volgt:

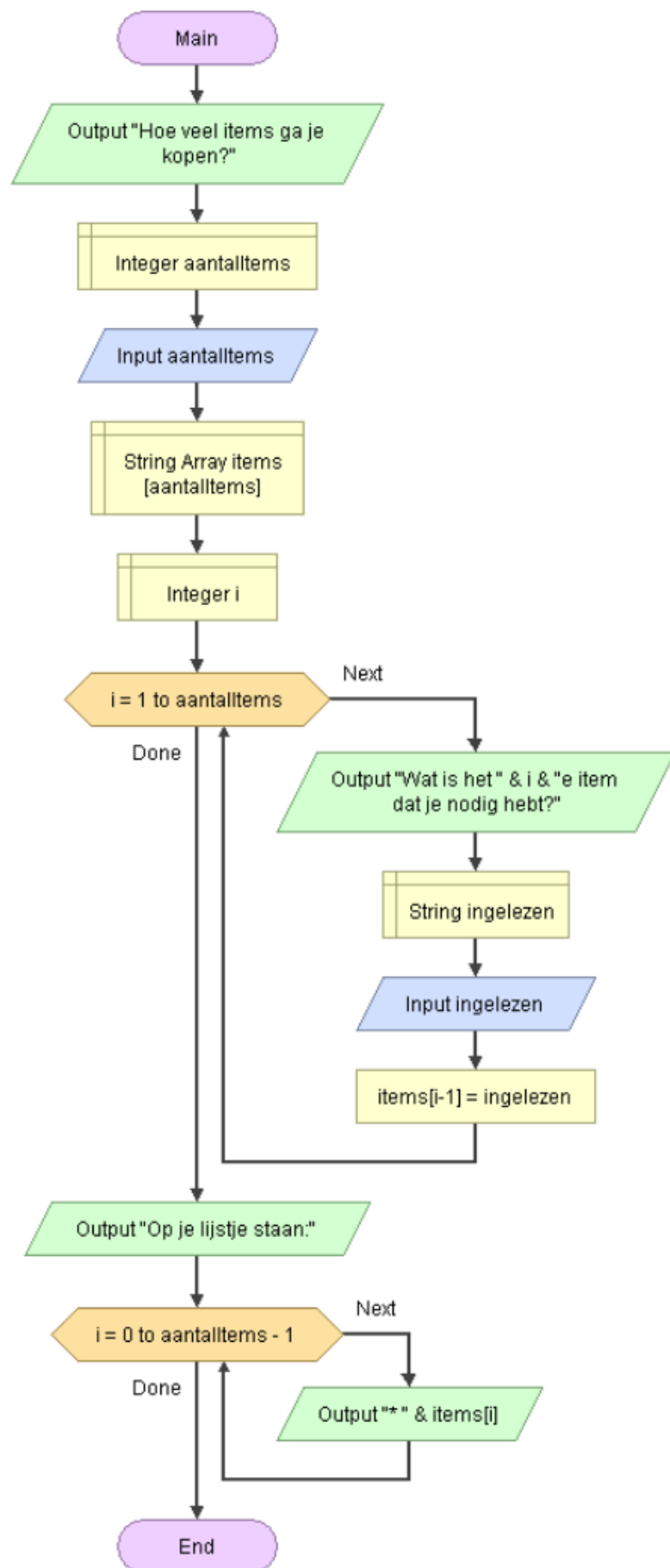
```
Console.WriteLine("Hoe veel items ga je kopen?");
int aantalItems = Convert.ToInt32(Console.ReadLine());
string lijstje = "";
for (int i = 1; i <= aantalItems; i++) {
    Console.WriteLine($"Wat is het {i}e item dat je nodig hebt?");
    lijstje = $"{lijstje} {Console.ReadLine()}";
}
```

Hiermee kan je elk gewenst aantal items toevoegen, maar de voorstelling is erg onhandig. Als je bijvoorbeeld item zeven op je lijstje wil zoeken, moet je plustekens eerst zes plustekens tellen. Als je het lijstje op een andere manier wil tonen aan de gebruiker, bijvoorbeeld met één item per regel, is dat lastig. Als je wil bijhouden of iets gekocht is, kan je daar geen boolean voor gebruiken, maar moet je bijvoorbeeld een vinkje vlak voor het item plaatsen. Dat is allemaal vrij ingewikkelde code voor een vrij eenvoudige taak.

Arrays bieden een oplossing voor deze problemen. Arrays zijn reeksen van waarden, maar ze zijn zelf ook waarden. Met andere woorden, ze maken het mogelijk één variabele te maken die verschillende waarden voorstelt. Hierdoor wordt je code leesbaarder en eenvoudiger in onderhoud. Arrays zijn een zeer krachtig hulpmiddel, maar er zitten wel enkele addertjes onder het gras.

Laat ons beginnen bij het begin: een array is een reeks waarden van **één type** en heeft een bepaalde lengte. Bijvoorbeeld een reeks van 10 `int`s of 5 `string`s. Je kan dus geen reeks hebben met een mengeling van `int` en `string`, of van `boolean` en `double`, of wat dan ook. Je kan een array toekennen aan een variabele, maar je kan ook individuele waarden uit de reeks halen door middel van een `index`. Dit is een getal dat een positie in de reeks aanduidt, zoals je dat ook kent van de `substring` methode.

We zullen het eerst even voordoen in Flowgorithm.





boodschappenlijstje-met-array.fprg 2KB

Binary

## Basisgebruik arrays

### Arrays declareren

Net als andere datatypes moet je aangeven dat een variabele van een bepaald array type is. Je moet dus de compiler informeren dat er een array bestaat met een bepaalde naam en een bepaald type. Dit is bijna hetzelfde als het declareren van een variabele van een van de types die je al kent, maar je geeft aan dat het om een reeks gaat door middel van rechte haken.

We geven hier enkele voorbeelden:

```
string[] items; // een reeks boodschappen voorgesteld als tekst
double[] metingen; // metingen van de neerslag als kommagetallen
decimal[] aandeelWaarden; // kommagetallen met hoge precisie
```

De rechte haken betekenen dus "een reeks van" en een declaratie van een reeks kan een heleboel declaraties van individuele variabelen vervangen. Met deze voorstelling heb je bijvoorbeeld `item1` en `item2` en `item3` niet meer nodig.

Als we naar de Flowgorithm code van het boodschappenlijstje kijken, zien we dit ook:

```
string[] items = new string[aantalItems];
```

Merk wel op: hier vindt ook meteen een **initialisatie** plaats.

Een initialisatie vertelt de compiler niet alleen dat deze variabele bestaat, maar ook wat zijn eerste waarde is. Een waarde van een array-type stelt eigenlijk een hoeveelheid ruimte in het geheugen voor. Ruimte voor een bepaald aantal strings, voor een bepaald aantal getallen, maakt niet uit. Om technische redenen die we later in meer detail bekijken, moet je deze ruimte reserveren met het keyword `new`. Je moet ook zeggen hoe veel ruimte je nodig hebt. Bijvoorbeeld, met de declaraties van hierboven:

```
items = new string[10]; // ruimte om 10 items op het lijstje bij te houden
metingen = new double[365]; // ruimte om een jaar aan metingen te voorzien
aandelen = new decimal[365*10]; // ruimte om de koers over (ongeveer) 10 jaar bij te houden
```

Zoals je merkt uit het laatste voorbeeld, mag je deze waarde ook berekenen. Ook het Flowgorithm voorbeeld toont dit, want daar wordt `aantalItems` ingelezen.

### Arrays opvullen

Nadat de nodige ruimte voorzien is, kan je deze gebruiken. Dit doe je door te zeggen dat een bepaalde waarde op een bepaalde positie in de reeks terechtkomt. Deze positie noemen we de **index** van het element. **Het eerste element krijgt index 0, niet 1. Het laatste krijgt een index gelijk aan de lengte van de array verminderd met 1.**

We geven even een voorbeeld rechtstreeks in C#:

```
items[0] = "brood"; // het eerste element heeft index 0
items[1] = "thee";
items[2] = "fruit";
// nog wat boodschappen
items[9] = "krant";
// items[10] kan je niet gebruiken want de laatste index is 10-1
```

Dit hoeven geen vaste waarden te zijn! Je kan ook dit doen:

```
Console.WriteLine(); // geef 10 items die je nodig hebt
items[0] = Console.ReadLine(); // het eerste element heeft index 0
items[1] = Console.ReadLine();
items[2] = Console.ReadLine();
// nog wat boodschappen
items[9] = Console.ReadLine();
// items[10] kan je niet gebruiken want de laatste index is 10-1
```

Dit zorgt dat je code kan uitsparen. Met 10 individuele variabelen, dus **zonder array**, zou je bovenstaande code ongeveer zo moeten schrijven:

```
Console.WriteLine(); // geef 10 items die je nodig hebt
item1 = Console.ReadLine(); // het eerste element heeft index 0
item2 = Console.ReadLine();
item3 = Console.ReadLine();
// nog wat boodschappen
item10 = Console.ReadLine();
```

Maar **met een array** heb je ook deze optie:

```
Console.WriteLine(); // geef 10 items die je nodig hebt
for(int i = 0; i <= 9; i++) {
    items[i] = Console.ReadLine();
}
```

En als je je lijstje wil uitbreiden tot 1000 items, moet je alleen de 10 door 1000 vervangen. We zullen nog vaak gebruik maken van arrays en lussen om herhaalde code te vermijden!

### Arrays uitlezen

Je gebruikt de notatie met rechte haken ook om een array uit te lezen:

```

Console.WriteLine($"Het 1e item op je lijstje is {items[0]}");
Console.WriteLine($"Het 2e item op je lijstje is {items[1]}");
// herhaald
Console.WriteLine($"Het 10e item op je lijstje is {items[9]}");

```

Ook hier bespaar je vervelend werk met een lus:

```

for(int i = 0; i <= 9; i++) {
    Console.WriteLine($"Het {i+1}e item op je lijstje is {items[i]}");
}

```

Als je een element probeert uit te lezen dat nog geen waarde gekregen heeft, krijg je een defaultwaarde. Wat voor waarde dat is, hangt af van de array. Er zit een systeem achter, maar dat behandelen we pas later in de cursus. Voorlopig mag je dit onthouden: voor getallen krijg je 0. Voor booleans krijg je `false`. Voor strings krijg je een speciale waarde `null`, die verschillend is van `" "`. Wat je vooral moet weten over `null` is dat je er geen eigenschappen van kan opvragen of methodes op kan toepassen. Ik laat even zien wat ik bedoel.

```

string[] stukkenTekst;
Console.WriteLine(stukkenTekst[0].Length);

```

```

string[] stukkenTekst;
Console.WriteLine(stukkenTekst[0].ToUpper());

```

```

int[][] arrayVanArrays;
Console.WriteLine(arrayVanArrays.Length);

```

Dus als een variabele `null` bevat of kan bevatten, moet je syntax vermijden die hier een punt achter plaatst. Je kan dit op een eenvoudige manier doen met `if (mijnVariabele is null) { ... } else { ... }`.

De lengte van de array te weten komen

Soms kan het nodig zijn dat je in een later stadium van je programma de lengte van je array nodig hebt. De `Length` eigenschap geeft je deze lengte. Volgend voorbeeld toen dit:

```

string[] items = new string[10];
Console.WriteLine("Geef de 10 items die je wil kopen:");
for(int i = 0; i < items.Length; i++) {
    items[i] = Console.ReadLine();
}

```



Het voordeel hiervan is je maar één keer moet vastleggen hoe groot de array is en daarna in het algemeen over de grootte kan spreken. Zo kan je bij het aanpassen van je code niet vergeten **overall** de juiste aanpassing te doen. Let op: de lengte geeft niet aan hoe veel items je al hebt ingevuld! Ze geeft aan hoe veel plaats er maximaal is in de array.

# Alternatieve syntax

## ✓ Kennisclip

De reeds besproken manier om arrays te maken is veelzijdig en toont alle aspecten, maar vraagt vrij veel schrijfwerk. Er zijn nog manieren om arrays aan te maken, maar deze veronderstellen dat je de array in één keer kan opvullen.

### Manier 1: met `new`

Ook op deze manier moet je array gedeclareerd worden als volgt:

```
type[] arraynaam; // type vervang je door het type in kwestie
```

Dit is hetzelfde als eerder. Je kan bijvoorbeeld volgende concrete code hebben:

```
string[] items;
```

Nu kan je deze array meteen initialiseren op een andere manier dan eerder. Je kan kiezen de grootte niet in te vullen en de gebruikte waarden tussen accolades te zetten, bijvoorbeeld:

```
string[] items;  
items = new string[] {"brood", "koffie", "fruit", "thee", "yoghurt"};
```

Er staan vijf elementen tussen de accolades, dus de compiler kan achterhalen dat je een nieuwe array van vijf elementen wil maken, zonder dat je dat uitdrukkelijk zegt.

### Manier 2: verkorte declaratie en initialisatie

Indien je direct waarden wilt toekennen (initialiseren) tijdens het aanmaken van de array zelf dan mag dit ook. We noemen dit de **array literal syntax**:

```
string[] items = {"brood", "koffie", "fruit", "thee", "yoghurt"};
```

Hier zal je array dus een vaste lengte van 5 elementen hebben. Merk op dat deze manier dus enkel werkt indien je al weet hoe groot je array moet zijn voor je je programma opstart. De andere opties bieden in dat opzicht meer flexibiliteit ten koste van meer schrijfwerk.

# Werken met arrays



Kennisclip

## Nuttige array methoden

Iedere array heeft een aantal methoden waar meteen gebruik van gemaakt kan worden. Al deze methoden hier beschrijven zou ons te ver nemen, maar enkele methoden zijn echter zeer handig om te gebruiken en hebben een vanzelfsprekende betekenis: `Max()`, `Min()`, `Sum()` en `Average()`.

Volgende code geeft bijvoorbeeld het grootste getal terug uit een array genaamd "leeftijden".

```
using System;
using System.Linq; // de methoden zijn afkomstig uit deze namespace
int[] leeftijden = {4, 9, 12, 38, 13, 7};
int oudsteleeftijd = leeftijden.Max();
```

Met de rest kan je het minimum, de som en het gemiddelde bepalen.

Daarnaast beschikken arrays ook over functionaliteit die niet specifiek gelinkt is aan getallen. We bekijken hier een paar mogelijkheden die heel vaak van pas komen. Voor deze methoden is de syntax, om redenen die we later zullen zien, wat anders: je schrijft eerst `Array.` en je geeft de array in kwestie mee tussen de haakjes.

## Sort: Arrays sorteren

Om arrays te sorteren roep je de `Sort()`-methode op als volgt, als parameter geef je de array mee die gesorteerd moet worden.

Volgende voorbeeld toont hier het gebruik van:

```
string[] items = {"brood", "koffie", "fruit"};
Array.Sort(items);

// Toon resultaat van sorteren
// De items zullen alfabetisch getoond worden
for (int i = 0; i < items.Length; i++)
{
    Console.WriteLine(items[i]);
}
```

Wanneer je de Sort-methode toepast op een array van strings dan zullen de elementen alfabetisch gerangschikt worden. Merk op dat je niet meteen iets merkt nadat je `Sort` hebt opgeroepen. Intern zijn er normaal elementen van plaats gewisseld, maar je ziet dit pas als je bijvoorbeeld de elementen afprint.

Reverse: Arrays omkeren

Met de `Array.Reverse()` -methode kunnen we dan weer de elementen van de array omkeren (dus het laatste element vooraan zetten en zo verder.

```
Array.Reverse(items);
```

Ook hier zie je pas iets als je de elementen toont.

IndexOf: Zoeken in arrays

De `IndexOf` -methode maakt het mogelijk om te zoeken naar de index van een gegeven element in een index. De methode heeft twee parameters, enerzijds de array in kwestie en anderzijds het element dat we zoeken. Als resultaat wordt de index van het gevonden element teruggegeven. Indien niets wordt gevonden zal het resultaat (in de praktijk) -1 zijn.

Volgende code zal bijvoorbeeld de index teruggeven van het item `"koffie"` indien dit in de array `items` staat:

```
Array.Sort(items);  
Array.BinarySearch(items, "koffie");
```

Volgend voorbeeld toont het gebruik van deze methode:

```
string[] items = {"brood", "koffie", "fruit"};  
  
Console.WriteLine("Wil je weten of iets al op je lijstje staat? Geef de naam van het item.")  
string item = Console.ReadLine();  
int index = Array.IndexOf(items, item);  
if(index >= 0) {  
    Console.WriteLine($"{item} staat op het lijstje met index {index}");  
}  
else {  
    Console.WriteLine("staat nog niet op het lijstje");  
}
```

`IndexOf` kan erg van pas komen om stukjes informatie aan elkaar te linken. Bijvoorbeeld: de prijs van de producten staat steeds op dezelfde index in de andere array:

```
string[] producten = { "appel", "peer", "citroen" };  
decimal[] prijzen = { 0.65d, 0.35d, 0.70d };
```

We weten nu genoeg om de gebruiker de prijs van elk product te kunnen tonen.

```
Console.WriteLine("Van welk product wil je de prijs weten?");  
string product = Console.ReadLine();
```

We tonen nu hoe we eerst het juiste product zoeken en dan vervolgens die index bewaren en gebruiken om de prijs te tonen:

```
string[] producten = {"appel", "peer", "citroen"};  
decimal[] prijzen = { 0.65m, 0.35m, 0.70m };  
System.Console.WriteLine("Van welk product wil je de prijs kennen?");  
string product = Console.ReadLine();  
int productIndex = Array.IndexOf(producten, product);  
if (productIndex < 0) {  
    System.Console.WriteLine("Het product is niet aanwezig.");  
}  
else {  
    System.Console.WriteLine(prijzen[productIndex]);  
}
```

# Defaultwaarden

Standaard weigert de compiler het gebruik van variabelen die niet geïnitialeerd zijn. Probeer volgende code maar eens in een testprogramma:

```
string naam;  
Console.WriteLine(naam);
```

Je krijgt een foutmelding en je programma voert niet uit. Lees de tekst van de foutmelding. Er wordt uitgelegd dat je `naam` probeert te gebruiken zonder er een waarde aan te geven. Een variabele moet dus geïnitialeerd zijn.

We hebben eerder gezien dat elementen van een array ongeveer gebruikt kunnen worden als variabelen, bijvoorbeeld:

```
int[] getallen = new int[10];  
getallen[0] = 5;
```

Maar hier is een aandachtspuntje: `getallen[1]`, `getallen[2]`, enzovoort zijn nog steeds niet geïnitialeerd. Anderzijds is de grotere structuur `getallen` wel geïnitialeerd. Kunnen we ze dan wel al gebruiken?

Het antwoord is ja. De echte variabele `getallen` is geïnitialeerd. Je kan dit uittesten en er zal meteen iets opvallen:

```
int[] getallen = new int[10];  
getallen[0] = 5;  
for (int i = 0; i < getallen.Length; i++) {  
    // ToString heeft hier geen effect, maar we tonen verderop iets  
    Console.WriteLine(getallen[i].ToString());  
}
```

Alle niet-ingevulde getallen hebben een defaultwaarde gekregen, namelijk 0.

In het algemeen krijgen niet-geïnitialeerde waarden een defaultwaarde. Welke waarde dat is, hangt af van hun type. Er bestaan verschillende categorieën van types die deze waarde vastleggen, maar voorlopig mag je het volgende onthouden:

type(s)	defaultwaarde
elk getaltype	0
bool	false
string	null
elk type array	null
Random	null

`null` is een speciale waarde die eigenlijk betekent "er is geen waarde". Van een ontbrekende waarde kunnen we geen eigenschappen of methoden opvragen. Daarom crasht volgende code, die identiek gestructureerd is aan het vorige voorbeeld:

```
string[] teksten = new string[10];
teksten[0] = "hallo wereld";
for (int i = 0; i < teksten.Length; i++) {
    Console.WriteLine(teksten[i].ToString());
}
```

Het probleem is dat we `null.ToString()` oproepen. Let dus op: je mag geen eigenschap of methode gebruiken door een punt achter een waarde te zetten die `null` bevat. Je kan hier op controleren met `is null`:

```
string[] teksten = new string[10];
teksten[0] = "hallo wereld";
for (int i = 0; i < teksten.Length; i++) {
    if (!(teksten[i] is null)) {
        Console.WriteLine(teksten[i].ToString());
    }
}
```

# List<T>

Een `List<>` collectie is de meest standaard collectie die je kan beschouwen als een flexibelere variant op een een doodnormale array.

**i** De Generieke `List<>` klasse bevindt zich in de `System.Collections.Generic` namespace. Je dient deze namespace dus als `using` bovenaan toe te voegen wil je deze klasse kunnen gebruiken.

## List aanmaken

De klasse `List<T>` is een zogenaamde generieke klasse. Tussen de `< >` tekens plaatsen we het type dat de lijst zal moeten gaan bevatten. Vaak wordt dit genoteerd als `T` voor "type". Bijvoorbeeld:

- `List<int> alleGetallen= new List<int>();`
- `List<bool> binaryList = new List<bool>();`
- `List<Pokemon> pokeDex = new List<Pokemon>();`
- `List<string[]> listOfStringarrays = new List<string[]>();`

Zoals je ziet hoeven we bij het aanmaken van een `List` geen begingrootte mee te geven, wat we wel bij arrays moeten doen. Dit is een van de voordelen van `List`: ze groeien mee. Als we toch een begingrootte meegeven (zoals in de kennisclip even getoond wordt) is dat enkel om de performantie van de code wat te verhogen in bepaalde scenario's. Wij gaan dit nooit doen.

## Elementen toevoegen

Via de `Add()` methode kan je elementen toevoegen aan de lijst. Je dient als parameter aan de methode mee te geven wat je aan de lijst wenst toe te voegen. **Deze parameter moet uiteraard van het type zijn dat de `List` verwacht.**

In volgende voorbeeld maken we een `List` aan die objecten van het type `string` mag bevatten en vervolgens plaatsen we er twee elementen in.

```
List<String> myStringList = new List<String>();  
myStringList.Add("This is the first item in my list!");  
myStringList.Add("And another one!");
```

## Elementen indexeren



Het leuke van een List is dat je deze ook kan gebruiken als een gewone array, waarbij je met de indexer elementen kan aanspreken. Stel bijvoorbeeld dat we een lijst hebben met minstens 4 strings in. Volgende code toont hoe we de string op positie 3 kunnen uitlezen en hoe we die op positie 2 overschrijven:

```
Console.WriteLine(myStringList[3]);  
myStringList[2] = "andere zin";`
```

Ook de klassieke werking met `for` blijft gelden. De enige aanpassing is dat `List<T>` niet met `Length` werkt maar met `Count`.

```
for(int i = 0 ; i < myStringList.Count; i++)  
{  
    Console.WriteLine(myStringList[i])  
}
```

Wat kan een List nog?

Interessante methoden en properties voorts zijn:

- `Clear()` :methode die de volledige lijst leegmaakt
- `Insert()` : methode om element op specifieke plaats in lijst toe te voegen, bijvoorbeeld:  

```
myStringList.Insert(1,"A fourth sentence");
```

voegt de string toe op de tweede plek en schuift de rest naar achter
- `Contains()` : geef als parameter een specifiek object mee (van het type `T` dat de `List<T>` bevat) om te weten te komen of dat specifieke object in de `List<T>` terug te vinden is. Indien ja dan zal `true` worden teruggegeven.
- `IndexOf()` : geeft de index terug van het element item in de rij. Indien deze niet in de lijst aanwezig is dan wordt `-1` teruggegeven.
- `RemoveAt()` : verwijder een element op de index die je als parameter meegeeft.
- `Remove()` : verwijder het gegeven element



`Contains`, `Remove` en `IndexOf` zullen zich met jouw eigen klassen niet noodzakelijk gedragen zoals je verwacht. De verklaring hierachter komt later aan bod, wanneer we `Equals` en `GetHashCode` bekijken. Ze zullen wel werken zoals verwacht voor voorgedefinieerde types, inclusief `DateTime`.

## Foreach loops

Je kan met een eenvoudige `for` of `while`-loop over een lijst itereren, maar het gebruik van een `foreach`-loop is toch handiger.

Dit is dan ook de meestgebruikte operatie om eenvoudig en snel een bepaald stuk code toe te passen op ieder element van de lijst:

```
List<int> integerList=new List<int>();
integerList.Add(2);
integerList.Add(3);
integerList.Add(7);

foreach(int prime in integerList)
{
    Console.WriteLine(prime);
}
```

⚠ Er bestaat strikt gesproken geen meerdimensionale `List<T>` , maar een lijst kan wel andere lijsten bevatten. Je kan de functionaliteit ervan dus nabootsen.

# Oefeningen

## Oefeningen

### Oefening: H6-ArrayTrueFalse

#### Leerdoelen

- Declareren van arrays
- Initialiseren van arrays
- Opvullen van arrays
- Arrays gebruiken - afprinten

#### Functionele analyse

Maak een array gevuld met afwisselend true en false (de array is 30 lang)

Per array: output de array naar het scherm, maar ieder element naast elkaar met komma gescheiden. Dus niet:

1 true

2 false

3 true

4 \\etc

maar wel: true, false, true, ...

Implementeer onderstaande flowchart in C#.

#### Technische analyse



H6-ArrayTrueFalse.fprg 2KB

Binary

#### Voorbeeldinteractie

```
Maak een array gevuld met afwisselen true en false (lengte is 30)
True False True False True False True False True False True False True False True False
True False True False True False True False True False True False True False
```

## Oefening: H6-ArrayOefener1

### Leerdoelen

- Declareren van arrays
- Initialiseren van arrays
- Opvullen van arrays
- Arrays gebruiken

### Functionele analyse

Maak een programma dat aan de gebruiker vraagt om 10 waarden (int) in te voeren in een array. Vervolgens toont het programma de som, het gemiddelde en het grootste getal van deze 10.

Vervolgens vraagt het programma de gebruiker om een getal in te voeren. Het programma toont dan alle getallen die groter of gelijk zijn aan dit ingevoerde getal zijn die in de array aanwezig zijn. Indien geen getallen groter zijn dan verschijnt een bericht Niets is groter op het scherm.

Implementeer onderstaande flowchart in C#. Merk op: je kan in principe de statistieken al bijhouden terwijl de getallen worden ingevoerd, maar hier worden ze pas achteraf berekend. Doe het ook zo.

### Technische analyse



H6-ArrayOefener1.fprg 3KB  
Binary

### Voorbeeldinteractie

```
Voer 10 gehele getallen in
0
1
2
3
4
5
6
7
8
9
*****
Som is 45, Gemiddelde is 4,5, Grootste getal is 9
*****
Geef minimum getal in?
3
3 4 5 6 7 8 9
```

## Oefening: H6-Boodschappenlijst

### Leerdoelen

- Declareren van arrays
- Initialiseren van arrays
- Opvullen van arrays
- Arrays gebruiken

## Functionele analyse

Maak een programma dat de gebruiker een boodschappenlijstje laat samenstellen.

- Het programma vraagt eerst hoeveel items de boodschappenlijst moet bevatten en laat dan de lijst vullen.
- Vervolgens wordt een gesorteerde lijst van de items getoond.
- Daarna, in de winkel, kan de gebruiker aangeven welke items er gekocht worden. De gebruiker kan dit blijven doen zolang er 'ja' geantwoord wordt op de vraag 'Nog winkelen?'. Als de gebruiker een item intypt dat niet op de lijst staat, wordt er een bericht getoond.
- Na het winkelen toont het programma welke items van de lijst niet gekocht zijn.

Implementeer onderstaande flowchart in C#.

## Technische analyse



H6-Boodschappenlijst.fprg 8KB  
Binary

## Voorbeeldinteractie

```

We gaan de boodschappenlijst samenstellen. Hoeveel items wil je opschrijven?
3
Wat is item 1 op je lijst?
kaas
Wat is item 2 op je lijst?
eieren
Wat is item 3 op je lijst?
boter
Dit is je gesorteerde lijst:
1: boter
2: eieren
3: kaas
Op naar de winkel!
Welk item heb je gekocht?
kaas
Nog winkelen? <Ja of Nee>
nee
Naar huis met de boodschappen!
Volgende items van je lijst ben je vergeten te kopen:
boter eieren

```

## Oefening: H6-Kerstinkopen

### Leerdoelen

- Declareren van arrays
- Initialiseren van arrays
- Opvullen van arrays
- Arrays gebruiken
- Array methodes gebruiken

## Functionele analyse

Maak een programma dat kan gebruikt worden om kerstinkopen te doen, rekening houdend met een budget. Na de inkopen, wordt het totaal gespendeerd bedrag getoond en het hoogste, laagste en gemiddelde bedrag.

## Technische analyse

Probeer zelf een flowchart te maken en zet die dan om naar C#.

Voorbeeldinteractie

```
Wat is het budget voor je kerstinkopen?
500
Hoeveel cadeautjes wil je kopen?
2
Prijs van cadeau 1?
200
Prijs van cadeau 2?
330
Je bent al 30,0 euro over het budget!
Info over je aankopen:
Totaal bedrag: 530,0 euro.
Duurste cadeau: 330,0 euro.
Goedkoopste cadeau: 200,0 euro.
Gemiddelde prijs: 265,0 euro.
```


## Oefening: H6-Lotto

### Leerdoelen

- Arrays in flowcharts gebruiken
- Declareren van arrays
- Initialiseren van arrays
- Opvullen van arrays
- Opzoeken waarde in arrays

## Functionele analyse

Je vraagt de gebruiker zijn 6 lotto getallen (getal tussen 1 en 42) in te geven. Je hoeft geen controle te doen op de getallen die de gebruiker ingeeft. Je bewaart deze getallen in een array lottoFormulier.

Vervolgens simuleer je een trekking. Dat doe je door random 6 getallen te genereren (zie hiervoor  **Random**) en die ook in een array lottoTrekking te bewaren. Let op: hier moet je wel controleren of het inderdaad 6 verschillende getallen zijn (hoe kan je nagaan of iets aanwezig is in je array?).

Dan doe je een validatie. Heeft de gebruiker 3 juiste cijfers wint hij 10 €, bij 4 juiste cijfers 1000 €, bij 5 juiste cijfers 100.000 € en bij 6 juiste cijfers 10.000.000 €.

Zorg dat alles mooi wordt getoond aan de gebruiker. Lotto getallen in stijgende volgorde, Trekking getallen in stijgende volgorde, winst. Zie voorbeeldinteractie.

### Technische analyse

Maak een flowchart en zet die om naar C#

Wat betreft het random genereren van 6 getallen tussen 1 en 42. Daar krijg je het volgend stukje code voor (probeer te begrijpen hoe het werkt en zoek op internet het gebruik van Random eens op):

```
Random random = new Random();
int lottoGetal;

for (int i = 0; i < lottoTrekking.Length; i++) {
    do {
        lottoGetal = random.Next(42) + 1;
    }
    while (Array.IndexOf(lottoTrekking, lottoGetal) >= 0);
    lottoTrekking[i] = lottoGetal;
}
```

Voorbeeldinteractie

```
Geef je lotto getallen <getallen moeten tussen 1 en 42 liggen>
Geef lotto nummer 1
11
Geef lotto nummer 2
22
Geef lotto nummer 3
33
Geef lotto nummer 4
4
Geef lotto nummer 5
9
Geef lotto nummer 6
23
Je gekozen cijfers zijn:
4 9 11 22 23 33
De trekking cijfers zijn:
2 6 13 32 34 35
Je hebt 0 Euro gewonnen
```

Uitbreiding

Test de ingegeven lotto getallen en zorg er voor dat er geen getallen kleiner dan 1 of groter dan 42 ingegeven kunnen worden en dat er ook geen dubbels voorkomen..

### Oefening: H6-IndexOf

### Leerdoelen

- Arrays in flowcharts gebruiken
- Opzoeken waarde in integer arrays – zelf geschreven methode

## Functionele analyse

Probeer d.m.v. Flowgorithm de flowchart te gebruiken en zet dan om naar C#.

Technische analyse



H6-IntegerIndexOf.fprg 3KB  
Binary

Voorbeeldinteractie

```
Geef 10 willekeurige gehele getallen
10
11
12
13
14
15
16
17
18
19
Welke geheel getal wil je zoeken?
14
Je zocht 14, die is gevonden op index 4
```

```
Geef 10 willekeurige gehele getallen
10
11
12
13
14
15
16
17
18
19
Welke geheel getal wil je zoeken?
8
Je zocht 8, jammer die is niet gevonden
```

## Oefening: H6-BinarySearch

### Leerdoelen

- Arrays in flowcharts gebruiken
- De array gebruiken, in een array zoeken.

## Functionele analyse

Heel eenvoudig uitgelegd zal het algoritme de te zoeken waarde vergelijken met de middelste waarde (van een gesorteerde array). Als de waarde niet gelijk is wordt dat gedeelte van de array waar de waarde zich niet in kan bevinden weggegooid. De zoektocht gaat verder totdat de waarde is gevonden (of niet gevonden indien alle mogelijkheden zijn doorzocht)



Implementeer onderstaande flowchart in C#. Om twee strings te ordenen, gebruik je `string1.CompareTo(string2)`. Dit levert -1 als string1 voor string2 komt, 1 als string1 na string2 komt en 0 als ze op dezelfde plaats komen.

## Technische analyse



H6-BinarySearch.fprg 4KB

Binary

## Voorbeeldinteractie

```
Welke automerk wil U zoeken?  
Opel  
Je zocht Opel, die is gevonden op index 6
```

```
Welke automerk wil U zoeken?  
blabla  
Je zocht blabla, jammer die is niet gevonden
```

## Oefening: H6-Boodschappenlijst-list

### Leerdoelen

- Declareren van een `List`
- Initialiseren van een `List`
- `List` gebruiken

### Functionele analyse

Maak een alternatieve versie van H6-Boodschappenlijst. Maak ditmaal gebruik van `List<T>` in plaats van een array.

### Technische analyse

- Je hoeft de grootte van de lijst niet meer op voorhand te vragen.
- Een lijst `myList` sorteren doe je door `myList.sort()` op te roepen.
- Je mag elementen die gekocht zijn gewoonweg verwijderen uit de lijst.
- Op het eindtoon je de resterende items met een `foreach` lus.
- Je geeft een lege regel in om aan te geven dat je niets wil toevoegen aan je lijstje.
- Noem je methode `BoodschappenList`

## Voorbeeldinteractie

```
Wat is item 1 op je lijst?  
> kaas  
Wat is item 2 op je lijst?  
> eieren  
Wat is item 3 op je lijst?  
> boter  
Wat is item 4 op je lijst?  
>  
Dit is je gesorteerde lijst:  
1: boter  
2: eieren  
3: kaas  
Op naar de winkel!  
Welk item heb je gekocht?  
> kaas  
Nog winkelen? (Ja of Nee)  
> nee  
Naar huis met de boodschappen!  
Volgende items van je lijst ben je vergeten te kopen:  
boter eieren
```

## Oefening: H6-Kerstinkopen-list

### Leerdoelen

- Declareren van een `List`
- Initialiseren van een `List`
- `List` gebruiken

### Functionele analyse

Maak een alternatieve versie van H6-Kerstinkopen. Maak ditmaal gebruik van `List<T>` in plaats van een array.

### Technische analyse

- Je hoeft niet meer te vragen op voorhand hoe veel cadeautjes er gekocht zullen worden.
- Je mag stoppen bij een lege regel invoer.
- Je moet ook stoppen zodra je over budget gaat.
- Noem je methode `KerstinkopenList`

## Voorbeeldinteractie

Wat is het budget voor je kerstinkopen?

> 500

Prijs van cadeau 1?

> 200

Prijs van cadeau 2?

> 330

Je bent al 30.0 euro over het budget!

Info over je aankopen:

Totaal bedrag: 530.0 euro

Duurste cadeau: 330.0 euro

Goedkoopste cadeau: 200.0 euro

Gemiddelde prijs: 265.0 euro