Waarom Array???

Stel dat we twee getallen van de invoer willen ontvangen en hun som willen berekenen en weergeven.

```
 int s1, s2, sum;
Console.Write("Numer 1 =");
s1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Numer 2 =");
s2 = Convert.ToInt32(Console.ReadLine());
sum = s1 + s2;
Console.WriteLine("Sum=" +sum);
```

Nu stel je dat we som van de honderd getallen willlen berekenen.

Dus soms moeten we grote hoeveelheden gelijkaardige gegevens bewaren in het geheugen van de computer, bijvoorbeeld een lijst met cijfers van studenten.
Het is onmogelijk om voor iedere cijfers van de studenten een nieuwe variabele te definiëren. In deze situatie kiezen we voor een array zodat we alle cijfers op één geheugenadres opslaan.

Array

Een array (Engels voor rij of reeks) is bij het programeren computers een datastructuur die bestaat uit een lijst van elementen. Ieder element heeft een unieke index waarmee dat element aangeduid kan worden.

index= 0	index=1	index= 2	index= 3	index= 4	index= 5

Hoewel een array een eenvoudige datastructuur is, kunnen er krachtige dingen mee gedaan worden.

De eenvoudigste implementatie van een array is een reeks opeenvolgende geheugencellen.

Een array is eigenlijk een tabel of matrix waarin we meerdere waarden van hetzelfde datatype kunnen opslaan. Aan iedere waarde in een array wordt een index toegekend.

A[0]	A[1]	A[2]	A[3]	A[4]

We moeten drie dingen weten:
1- Aantal array-elementen
2- Type waarden binnen de array
3- Geselecteerde naam voor de array

Er zijn drie manieren om een array te declareren.

Een manier om arrays aan te maken is de volgende, waarbij je aangeeft hoe groot de array moet zijn, zonder reeds effectief waarden toe te kennen:

Model 1:

De eenvoudigste variant is deze waarbij je een array variabele aanmaakt, maar deze nog niet initialiseert (i.e. je maakt enkel een identifier in aan). De syntax is als volgt:

```
Type[] ArrayName;
```

Type kan dus eender welk bestaand datatype zijn dat je reeds kent. De [] (square brackets) duiden aan dat het om een array gaat.

```
double[] A;
```

Andere voorbeelden van array declaraties kunnen dus bijvoorbeeld zijn:

```
int[] verkoopCijfers;

double[] gewichtHuisdieren;

bool[] examenAntwoorden;

ConsoleColor[] mijnKleuren;
```

Model 2:

Indien je ogenblikkelijk waarden wilt toekennen (initialiseren) tijdens het aanmaken van de array zelf dan mag dit ook als volgt:

```
Type[] ArrayName= { Waarden };
```

```
string[] myColors = {"red", "green", "yellow", "orange", "blue"};
```

Ook hier zal dus vanaf dit punt je array een vaste lengte van 5 elementen hebben.

Merk op dat deze manier dus enkel werkt indien je reeds weet welke waarden in de array moeten. In manier 1 kunnen we perfect een array aanmaken en pas veel later in het programma ook effectief waarden toekennen (bijvoorbeeld door ze stuk per stuk door een gebruiker te laten invoeren).

Model 3:

Nog een andere manier om arrays aan te maken is de volgende, waarbij je aangeeft hoe groot de array moet zijn, zonder reeds effectief waarden toe te kennen:

```
Type[] ArrayName;
ArrayName= new Type[ lengthOfNumbers];
```

Uiteraard kan dit ook in 1 stap:

```
Type[] ArrayName= new Type[ lengthOfNumbers];
```

example:

We geven hier aan dat de array bestaan 5 elementen kan bevatten.

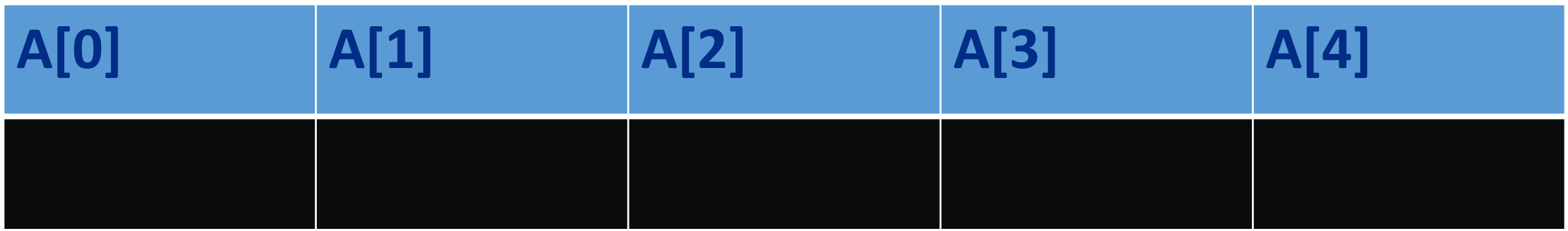
```
string[] myColors;
myColors = new string[5];
```

```
string[] myColors = new string[5];
```

De vierkante haken [ ] na het type geven aan dat de variabele een Array is.

```
double[] A = new double[5];
```

```
double[] A;
A = new double[5];
```



Elementen van een array aanpassen en uitlezen

We plaatsen de naam van de array, gevolgd door brackets waarbinnen een getal , 2 in dit voorbeeld, aangeeft het index van element die we wensen te benaderen (lezen en/of schrijven). Deze nummering start vanaf 0.

```
myColors[2];
```

Bij het maken van een array is de lengte van een array gelijk aan het aantal elementen dat er in aanwezig is. Dus een array met 5 elementen heeft als lengte 5.

```
string[] myColors = {"red", "green", "yellow", "orange", "blue"};
```

0	1	2	3	4
red	green	yellow	orange	blue

Bij het schrijven en lezen van individuele elementen uit de array gebruiken we een indexering die start bij 0. Bijgevolg is 4 de index van het laatste element in een array met lengte 5.

0	1	2	3	4
red	green	yellow	orange	blue

Lezen

We weten nu hoe we individuele waarden in een array kunnen benaderen. Ze gebruiken is dus exact hetzelfde zoals we in het verleden al met eender welke andere variabele hebben gedaan.

Wanneer je dus het tweede element van een array wenst te gebruiken kan dit bijvoorbeeld als volgt:

Console.WriteLine(myColors[1]);

of ook

string kleurkeuze = myColors[1];

of zelfs

if(myColors[1] == "pink")

Een array proberen te tonen als volgt gaat niet:

Console.WriteLine(myColors);

De enige manier alle elementen van een array te tonen is door manueel ieder element individueel naar het scherm te sturen. Bijvoorbeeld:

```
for(int i = 0 ; i<myColors.Length;i++)
{
    Console.WriteLine($"{myColors[i]}");
}
```

Stel dat we een array van getallen hebben, dan kunnen we dus bijvoorbeeld 2 waarden uit die array optellen en opslaan in een andere variabele als volgt:

```
int[] numbers = {5, 10, 30, 45};
int som = numbers[0] + numbers[1];
```

De variabele som zal dan vervolgens de waarde 15 bevatten (5+10).

Stel dat we alle elementen uit de array numbers met 5 willen verhogen, dan kunnen we schrijven:

```
int[] numbers = {5, 10, 30, 45};
numbers[0] += 5;
numbers[1] += 5;
numbers[2] += 5;
numbers[3] += 5;
```

Maar eigenlijk zijn we dan het voordeel van arrays niet aan het gebruiken.

Met loops maken we bovenstaande oplossing beter zodat deze zal werken, ongeacht het aantal elementen in de array:

```
for(int teller = 0; teller < numbers.Length; teller++)
{
    numbers[teller] += 5;
}
```

Zoals je merkt zijn loops en arrays dikke vrienden.

Schrijven

Ook schrijven van waarden naar de array gebruikt dezelfde notatie. Enkel moet je dus deze keer de array accessor-notatie links van de toekenningsoperator(=) plaatsen.

Stel dat we bijvoorbeeld de waarde van het eerste element uit de myColors array willen veranderen van red naar indigo, dan gebruiken we volgende notatie:

myColors[0] = "indigo";

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 van iedere array geeft dit weer. Volgend voorbeeld toont dit:

```
string[] myColors1 = { "red", "green", "yellow", "orange", "blue" };
        foreach (string x in myColors1)

            Console.WriteLine(x);
```

Elementen benaderen buiten de range van een array geeft erg dikke errors.
Het jammerlijke is dat VS dit soort subtiele 'out of range' bugs niet kan detecteren tijdens het compileren.
Je zal ze pas ontdekken bij de uitvoer.

Volgende code zal perfect gecompileerd worden, maar bij de uitvoer zal er op lijn 2 een error verschijnen en het programma zal stoppen:

```
string[] myColors = { "red", "green", "yellow", "orange", "blue" };
Console.WriteLine(myColors[6]);
```

Het is hetzelfde als wanneer ik tegen m'n personeel zeg "ga jij de muur alvast metsen op de zesde verdieping" (etage[5]) terwijl we een flatgebouw met maar 4 verdiepingen hebben (.Length is dus 5).

Nu willen we een programma maken die getallen kunnen invoeren en som van de getallen kunnen berekenen en uitvoeren.

A[0]	A[1]	A[2]	A[3]	A[4]
12	20	18	19,5	13

```
double[] A = new double[5];
A[0] = 12;
A[1] = 20;
A[2] = 18;
A[3] = 19.5;
A[4] = 13;
Console.WriteLine("A[3]="+A[3]);
```

```
for(int i=0;i<=A..Length;i++)
    Console.WriteLine("A["+i+"]="+A[i]);
```

```
        double[] A;
        Console.Write(" Nummer van cijfers:" );
        int n=
Convert.ToInt32(Console.ReadLine());
        A = new double[n];
```

```
for(int i=0;i<n;i++)
    {
        Console.Write("A[" + i + "]=");
        A[i]= Convert.ToDouble(Console.ReadLine());

    }
```

```
Console.WriteLine(" Data in array :");
int j = 0;
foreach(double  m in A)
{
    Console.WriteLine("A[" + j + "]=" + m);
    j++;
}
```