Objekt-Orienteret Programmering

Datastrukturer

Aslak Johansen asjo@mmmi.sdu.dk
Peter Nellemann pmn@mmmi.sdu.dk

September 18, 2024

SDU

Part 0:
Resumé

Type	*Størrelse*	*Beskrivelse*	*Default Værdi*
`boolean`	? bit	Sandhedsværdi	false
`byte`	8 bit	Heltal	0
`short`	16 bit	Heltal	0
`int`	32 bit	Heltal	0
`long`	64 bit	Heltal	0
`float`	32 bit	Kommatal	0.0
`double`	64 bit	Kommatal	0.0
`char`	16 bit	Tegn	\u0000

Resumé ▷ Udvalgte Primitive Typer

En primitiv datatype indeholder en enkelt værdi.

Resumé ▷ Udvalgte Primitive Typer

En primitiv datatype indeholder en enkelt værdi.

Exampler på heltal: `1`, `2`, `100`, `-1`

Resumé ▷ Udvalgte Primitive Typer

En primitiv datatype indeholder en enkelt værdi.

Exampler på heltal: `1`, `2`, `100`, `-1`

Exampler på kommatal: `1.0`, `2.7`, `3.14`, `1.4e-3`

Resumé ▷ Udvalgte Primitive Typer

En primitiv datatype indeholder en enkelt værdi.

Exampler på heltal: `1`, `2`, `100`, `-1`

Exampler på kommatal: `1.0`, `2.7`, `3.14`, `1.4e-3`

Exampler på booleans: `true`, `false`

Resumé ▷ Udvalgte Primitive Typer

En primitiv datatype indeholder en enkelt værdi.

Exampler på heltal: `1`, `2`, `100`, `-1`

Exampler på kommatal: `1.0`, `2.7`, `3.14`, `1.4e-3`

Exampler på booleans: `true`, `false`

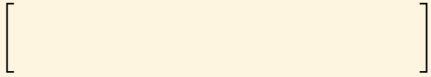
Det giver ikke meget mening at dele disse op i mindre dele.

Resumé ▷ Variable

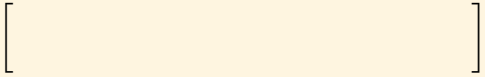
Variable er navne i vores kildekode som vi bruger til at referere til værdier.

```
1    int lewis_carroll;
2    int douglas_adams = 42;
3    lewis_carroll = 10;
```

Namespace:



Bindinger:



Resumé ▷ Variable

Variable er navne i vores kildekode som vi bruger til at referere til værdier.

```
1    int lewis_carroll; ⇐
2    int douglas_adams = 42;
3    lewis_carroll = 10;
```

Namespace:

$$\left[\begin{array}{c} \\ \texttt{lewis\_carroll} \end{array}\right]$$

Bindinger:

$$\left[\begin{array}{c} \texttt{lewis\_carroll} \mapsto 0 \\ \\ \end{array}\right]$$

Resumé ▷ Variable

Variable er navne i vores kildekode som vi bruger til at referere til værdier.

```
1    int lewis_carroll;
2    int douglas_adams = 42; ⇐
3    lewis_carroll = 10;
```

Namespace:	Bindinger:
`douglas_adams`	`lewis_carroll` ↦ 0
`lewis_carroll`	`douglas_adams` ↦ 42

Resumé ▷ Variable

Variable er navne i vores kildekode som vi bruger til at referere til værdier.

```
1    int lewis_carroll;
2    int douglas_adams = 42;
3    lewis_carroll = 10; ⇐
```

Namespace:	Bindinger:
`douglas_adams`	`lewis_carroll` ↦ 10
`lewis_carroll`	`douglas_adams` ↦ 42

Part 1:
Arrays

En kompleks type kan indeholde flere værdier.

En kompleks type kan indeholde flere værdier.

Af og til har vi brug for typer som kan indeholde mere end én værdi.

- For at samle sekvenser af data af en bestemt type i én struktur.
  - Fx heltal.
- For at skabe typer som er beskrevet ved mere end én (type af) dataværdi.
  - Fx kunne en `Person` bestå af en `int` for fødselsår og en `string` for navn.

En kompleks type kan indeholde flere værdier.

Af og til har vi brug for typer som kan indeholde mere end én værdi.

- For at samle sekvenser af data af en bestemt type i én struktur.
  - Fx heltal.
- For at skabe typer som er beskrevet ved mere end én (type af) dataværdi.
  - Fx kunne en `Person` bestå af en `int` for fødselsår og en `string` for navn.

Sådanne typer kalder vi under ét for *komplekse typer*.

En kompleks type kan indeholde flere værdier.

Af og til har vi brug for typer som kan indeholde mere end én værdi.

- For at samle sekvenser af data af en bestemt type i én struktur.
  - Fx heltal.
- For at skabe typer som er beskrevet ved mere end én (type af) dataværdi.
  - Fx kunne en `Person` bestå af en `int` for fødselsår og en `string` for navn.

Sådanne typer kalder vi under ét for *komplekse typer*.

I dag skal vi se på en datatype der repræsenterer sekvenser af data, der har samme type, i én struktur. Den datatype hedder "array".

Arrays ▷ Komplekse Typer i Hukommelsen

En computer kan kun arbejde på én værdi af gangen*.

Arrays ▷ Komplekse Typer i Hukommelsen

En computer kan kun arbejde på én værdi af gangen*.

Derfor refererer en variabel af en kompleks type til den adresse i hukommelsen hvor *samlingen* af værdier ligger.

- Værdien af en komplekst typed variabel er derfor en adresse, **ikke** den logiske samling af værdier.

Arrays ▷ Komplekse Typer i Hukommelsen

En computer kan kun arbejde på én værdi af gangen*.

Derfor refererer en variabel af en kompleks type til den adresse i hukommelsen hvor *samlingen* af værdier ligger.

- Værdien af en komplekst typed variabel er derfor en adresse, **ikke** den logiske samling af værdier.

$a \mapsto 42$ (primitiv type: `int`)
$b \mapsto 273150$ (kompleks type: 3 `double` værdier)

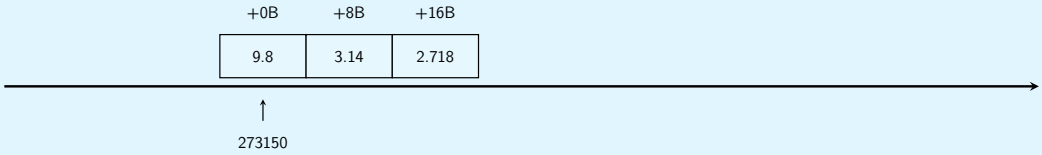
Arrays ▷ Komplekse Typer i Hukommelsen

En computer kan kun arbejde på én værdi af gangen*.

Derfor refererer en variabel af en kompleks type til den adresse i hukommelsen hvor *samlingen* af værdier ligger.

- Værdien af en komplekst typed variabel er derfor en adresse, **ikke** den logiske samling af værdier.

$a \mapsto 42$ (primitiv type: `int`)
$b \mapsto 273150$ (kompleks type: 3 `double` værdier)

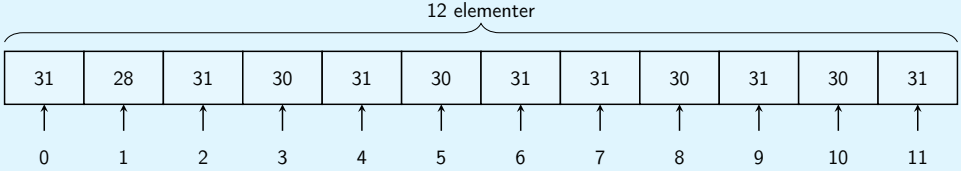


Arrays ▷ Definition

*Et array er en **data struktur** som indeholder en sekvens af andre stykker data, **alle af den samme bestemte type**.*

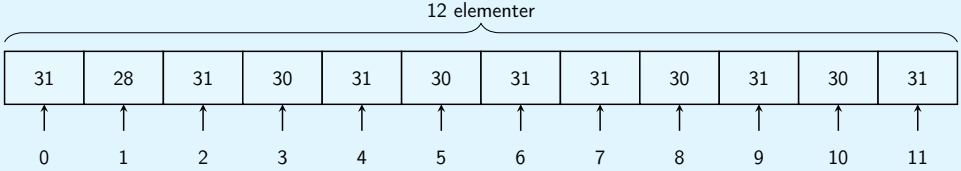
Arrays ▷ Definition

*Et array er en **data struktur** som indeholder en sekvens af andre stykker data, **alle af den samme bestemte type**.*



Arrays ▷ Definition

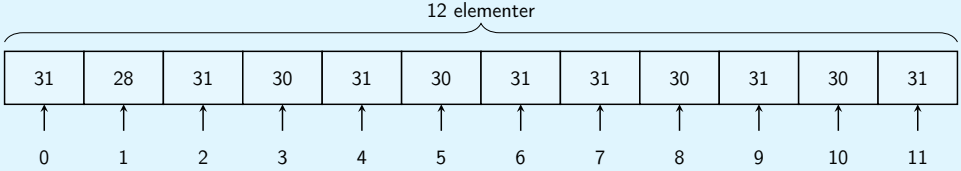
*Et array er en **data struktur** som indeholder en sekvens af andre stykker data, **alle af den samme bestemte type**.*



Elementerne er lagt fortløbende ud i hukommelsen.

Arrays ▷ Definition

*Et array er en **data struktur** som indeholder en sekvens af andre stykker data, **alle af den samme bestemte type**.*

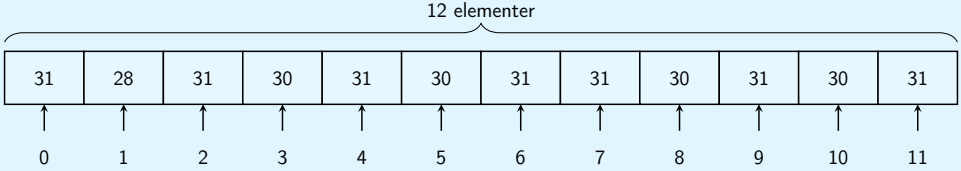


Elementerne er lagt fortløbende ud i hukommelsen.

Elementerne tilgås via et **indeks**.

Arrays ▷ Definition

*Et array er en **data struktur** som indeholder en sekvens af andre stykker data, **alle af den samme bestemte type**.*



Elementerne er lagt fortløbende ud i hukommelsen.

Elementerne tilgås via et **indeks**.

Arrays indekseres **startende fra 0!**

Arrays ▷ Problemer

Arrays ▷ Problemer

When you get the array index wrong

Arrays ▷ Hvorfor Arrays?

Hvorfor bruge arrays?

- Intet behov for separate variable for associerede værdier.
- Antallet af elementer kan være ukendt ved programmets start.
  - "Hvor mange studerende er der i lokalet lige nu?"
- Evnen til let at gentage handlinger over alle elementer med loops.

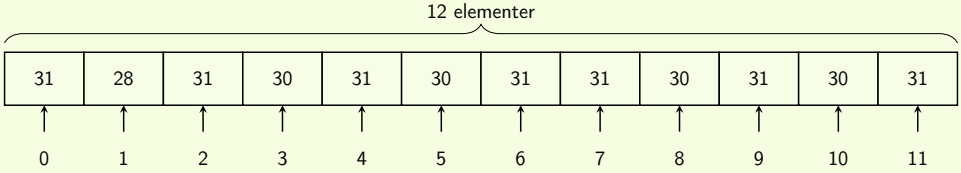
Part 2:
Anvendelse

Array Anvendelse ▷ Erklæring og Initialisering

Et array der indeholder en sekvens af `int` værdier kaldes et “int array”.

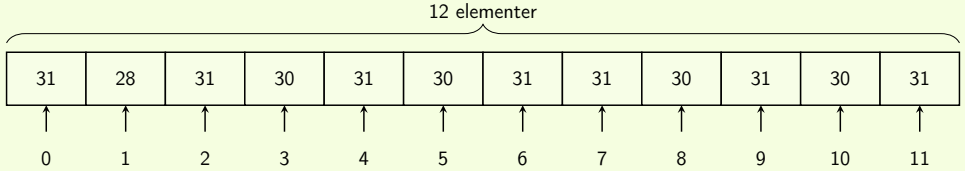
Array Anvendelse ▷ Erklæring og Initialisering

Et array der indeholder en sekvens af `int` værdier kaldes et "int array".



Array Anvendelse ▷ Erklæring og Initialisering

Et array der indeholder en sekvens af `int` værdier kaldes et "int array".

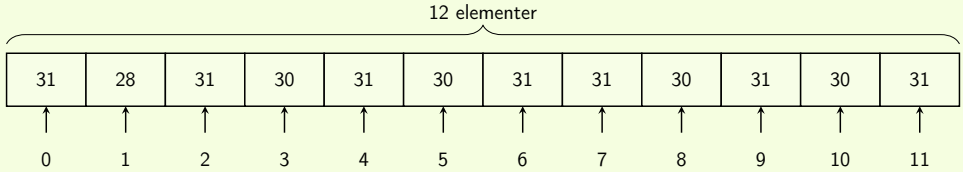


Erklæret ved hjælp af:

```
int[] months;
```

Array Anvendelse ▷ Erklæring og Initialisering

Et array der indeholder en sekvens af `int` værdier kaldes et "int array".



Erklæret ved hjælp af:

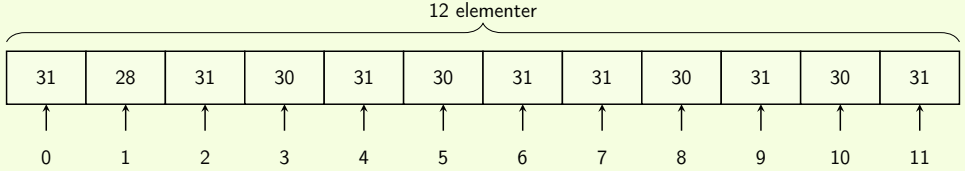
```
int[] months;
```

Initialiseret ved hjælp af:

```
months = new int[12];
```

Array Anvendelse ▷ Erklæring og Initialisering

Et array der indeholder en sekvens af `int` værdier kaldes et "int array".



Erklæret ved hjælp af:

```
int[] months;
```

Initialiseret ved hjælp af:

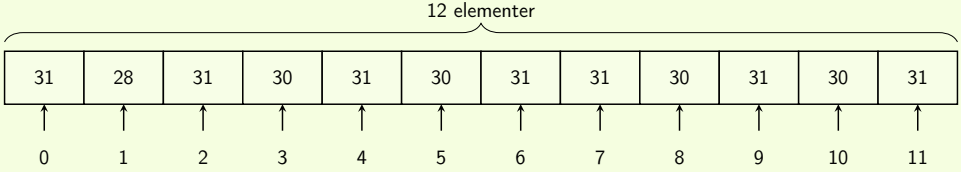
```
months = new int[12];
```

eller – hvis man ønsker selv at specificere startværdier – ved hjælp af:

```
months = [31,28,31,30,31,30,31,31,30,31,30,31];
```

Array Anvendelse ▷ Erklæring og Initialisering

Et array der indeholder en sekvens af `int` værdier kaldes et "int array".



Erklæret ved hjælp af:

```
int[] months;
```

Initialiseret ved hjælp af:

```
months = new int[12];
```

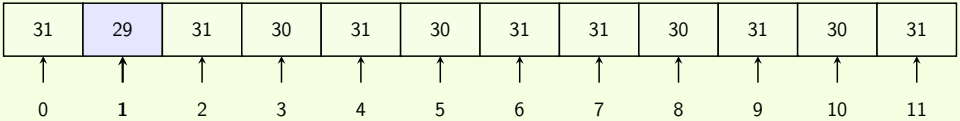
eller – hvis man ønsker selv at specificere startværdier – ved hjælp af:

```
months = [31,28,31,30,31,30,31,31,30,31,30,31];
```

Størrelsen af et array afgøres på initialiseringstidspunktet, og den kan ikke ændres efterfølgende.

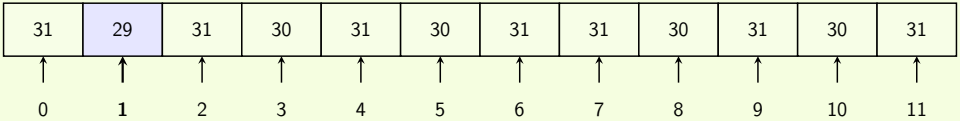
Array Anvendelse ▷ Manipulation

Elementerne af et array kan tilgås og manipuleres via det enkelte elements indeks i sekvensen.



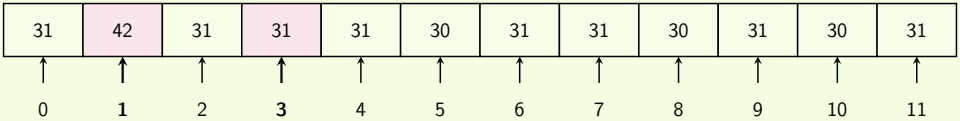
Array Anvendelse ▷ Manipulation

Elementerne af et array kan tilgås og manipuleres via det enkelte elements indeks i sekvensen.



Eksempler:

```
months[1] = 42;
months[3] = months[0];
```

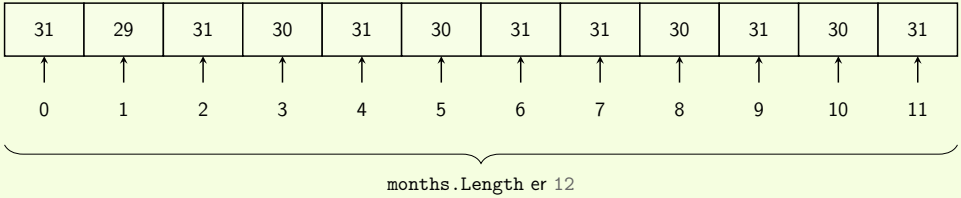


Array Anvendelse ▷ Array Længde

Længden af et array kan aflæses af arrayets **length** "attribut".

Array Anvendelse ▷ Array Længde

Længden af et array kan aflæses af arrayets **length** "attribut".




```
int arrayLength = months.Length;
Console.WriteLine(arrayLength);
```

```
12
```

Array Anvendelse ▷ Typer i Arrays

Arrays kan indeholde værdier af en vilkårlig type, men alle værdier i et specifikt array har den samme type.

Array Anvendelse ▷ Typer i Arrays

Arrays kan indeholde værdier af en vilkårlig type, men alle værdier i et specifikt array har den samme type.

Når et array initialiseres, allokeres der plads til samtlige elementer og disse vil blive tildelt en værdi der afhænger af element typen:

- ***int-array*** Alle elementer i arrayet starter med værdien `0`.
- ***string-array*** Alle elementer i arrayet starter med værdien `null`.
- ***bool-array*** Alle elementer starter med værdien `false`.

Array Anvendelse ▷ Typer i Arrays

Arrays kan indeholde værdier af en vilkårlig type, men alle værdier i et specifikt array har den samme type.

Når et array initialiseres, allokeres der plads til samtlige elementer og disse vil blive tildelt en værdi der afhænger af element typen:

- ***int-array*** Alle elementer i arrayet starter med værdien `0`.
- ***string-array*** Alle elementer i arrayet starter med værdien `null`.
- ***bool-array*** Alle elementer starter med værdien `false`.

Hvorfor?

Array Anvendelse ▷ Typer i Arrays

Arrays kan indeholde værdier af en vilkårlig type, men alle værdier i et specifikt array har den samme type.

Når et array initialiseres, allokeres der plads til samtlige elementer og disse vil blive tildelt en værdi der afhænger af element typen:

- ***int-array*** Alle elementer i arrayet starter med værdien `0`.
- ***string-array*** Alle elementer i arrayet starter med værdien `null`.
- ***bool-array*** Alle elementer starter med værdien `false`.

Hvorfor?

Fordi disse er *default* værdierne for typerne.

Part 3:
Traversering af Arrays

Traversering af Arrays ▷ Iteration over et String-Array

Eksempel:

```
string[] stringArray = [
  "This", "is", "an", "example", "of", "a", "String", "array"
];

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

Traversering af Arrays ▷ Iteration over et String-Array

Eksempel:

```
string[] stringArray = [
  "This", "is", "an", "example", "of", "a", "String", "array"
];

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

Output:

```
This is an example of a String array
```

Traversering af Arrays ▷ Iteration over et int-Array

Eksempel:

```
int[] months = [31,28,31,30,31,30,31,31,30,31,30,31];

for (int i=0 ; i<months.Length ; i++) {
  Console.WriteLine("Month "+(i+1)+" has "+months[i]+" days");
}
```

Traversering af Arrays ▷ Iteration over et int-Array

Eksempel:

```
int[] months = [31,28,31,30,31,30,31,31,30,31,30,31];

for (int i=0 ; i<months.Length ; i++) {
  Console.WriteLine("Month "+(i+1)+" has "+months[i]+" days");
}
```

Output:

```
Month 1 has 31 days
Month 2 has 28 days
Month 3 has 31 days
Month 4 has 30 days
...
Month 12 has 31 days
```

Traversering af Arrays ▷ Iteration med Foreach

Eksempel:

```
int[] months = [31,28,31,30,31,30,31,31,30,31,30,31];

foreach (int monthLength in months) {
  Console.WriteLine(monthLength + " days");
}
```

Traversering af Arrays ▷ Iteration med Foreach

Eksempel:

```
int[] months = [31,28,31,30,31,30,31,31,30,31,30,31];

foreach (int monthLength in months) {
  Console.WriteLine(monthLength + " days");
}
```

Output:

```
31 days
28 days
31 days
30 days
...
31 days
```

Traversering af Arrays ▷ Større Eksempel

```
double[] doubleArray = new double[12];
double total = 0;
double average;
int i;

// insert code to fill up doubleArray

for (i = 0; i<doubleArray.Length; i++) {
  total += doubleArray[i];
}

average = total / doubleArray.Length;
```

Part 4:
Reference Manipulation

Reference Manipulation

Vi afgør dynamisk hvad en variabel refererer til.

```
int[] monthsNormal = [31,28,31,30,31,30,31,31,30,31,30,31];
int[] monthsLeap   = [31,29,31,30,31,30,31,31,30,31,30,31];

for (int i=0 ; i<2020 ; i++) {
  int[] months = monthsNormal;
  if (i%4==0){
    months = monthsLeap;
  }
  Console.WriteLine("In year "+i+" February is "+months[1]+
                    " days long");
}
```

Reference Manipulation

Vi afgør dynamisk hvad en variabel refererer til.

```
int[] monthsNormal = [31,28,31,30,31,30,31,31,30,31,30,31];
int[] monthsLeap   = [31,29,31,30,31,30,31,31,30,31,30,31];

for (int i=0 ; i<2020 ; i++) {
  int[] months = (i%4==0 ? monthsLeap : monthsNormal);
  Console.WriteLine("In year "+i+" February is "+months[1]+
                    " days long");
}
```

Part 5:
Arrays af Arrays

Arrays af Arrays ▷ Introduktion

Vi har set at man kan erklære arrays således:

```
int[]    ints    = [1,2,3,4];
double[] doubles = [1.0,2.0,3.0,4.0];
bool[]   bools   = [false,true,false,true];
```

Arrays af Arrays ▷ Introduktion

Vi har set at man kan erklære arrays således:

```
int[]    ints    = [1,2,3,4];
double[] doubles = [1.0,2.0,3.0,4.0];
bool[]   bools   = [false,true,false,true];
```

Hvis man kan lave et array af en vilkårlig type, kan man så også lave et array af arrays?

Arrays af Arrays ▷ Introduktion

Vi har set at man kan erklære arrays således:

```
int[]    ints    = [1,2,3,4];
double[] doubles = [1.0,2.0,3.0,4.0];
bool[]   bools   = [false,true,false,true];
```

Hvis man kan lave et array af en vilkårlig type, kan man så også lave et array af arrays?

Måske sådan her?

```
int[][] arrayofarray = [[1,2,3,4], [2,3,4,5], [3,4,5,6], [4,5,6,7]];
```

Arrays af Arrays ▷ Introduktion

Vi har set at man kan erklære arrays således:

```
int[]    ints    = [1,2,3,4];
double[] doubles = [1.0,2.0,3.0,4.0];
bool[]   bools   = [false,true,false,true];
```

Hvis man kan lave et array af en vilkårlig type, kan man så også lave et array af arrays?

Måske sådan her?

```
int[][] arrayofarray = [[1,2,3,4], [2,3,4,5], [3,4,5,6], [4,5,6,7]];
```

Og svaret er: Ja!

Arrays af Arrays ▷ Introduktion

Vi har set at man kan erklære arrays således:

```
int[]    ints    = [1,2,3,4];
double[] doubles = [1.0,2.0,3.0,4.0];
bool[]   bools   = [false,true,false,true];
```

Hvis man kan lave et array af en vilkårlig type, kan man så også lave et array af arrays?

Måske sådan her?

```
int[][] arrayofarray = [[1,2,3,4], [2,3,4,5], [3,4,5,6], [4,5,6,7]];
```

Og svaret er: Ja!

Et array er også en (kompleks) type, og arrays kan indeholde både simple og komplekse typer.

Arrays af Arrays ▷ Layout i Hukommelsen

Eksempel:

```
int[][] arrayofarray = [
  [1,2,3,4],
  [2,3,4,5],
  [3,4,5,6],
  [4,5,6,7]
];
```

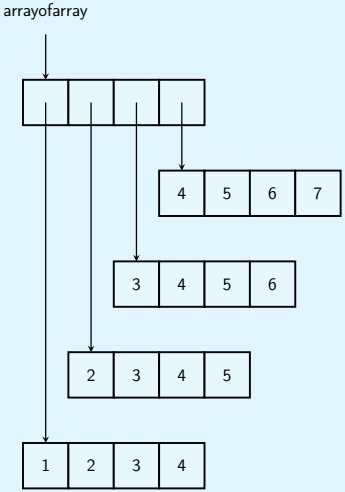
Arrays af Arrays ▷ Layout i Hukommelsen

Eksempel:

```
int[][] arrayofarray = [
  [1,2,3,4],
  [2,3,4,5],
  [3,4,5,6],
  [4,5,6,7]
];
```

For at manipulere en værdi skal vi først slå op i det yderste array, og dernæst det inderste.

```
Console.WriteLine(arrayofarray[2]);
Console.WriteLine(arrayofarray[2][3]);
```



Arrays af Arrays ▷ Tabeller

Arrays af arrays kan (ofte) ses som tabeller.

```
int[][] array = new int[4][];

// fill out array
array[3] = [43, -6, 17, 100, 1 -12];
```

Arrays af Arrays ▷ Tabeller

Arrays af arrays kan (ofte) ses som tabeller.

```
int[][] array = new int[4][];

// fill out array
array[3] = [43, -6, 17, 100, 1 -12];
```

Herefter er
`intArray[3][2]` lig
med `17`.

Arrays af Arrays ▷ Tabeller

Arrays af arrays kan (ofte) ses som tabeller.

```
int[][] array = new int[4][];

// fill out array
array[3] = [43, -6, 17, 100, 1 -12];
```

Herefter er `intArray[3][2]` lig med `17`.

	0	1	2	3	4	5
0	13	7	33	54	-5	-1
1	-3	0	8	42	18	0
2	44	78	90	79	-5	72
3	43	-6	17	100	1	-12

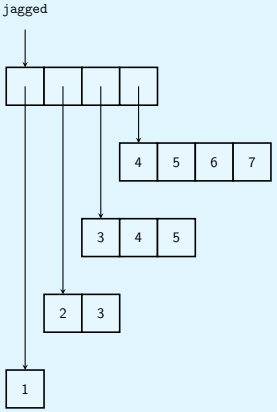
Arrays af Arrays ▷ Eksempel

```
double[][] profit = new double[25][];
double totalProfit = 0; // Company's total profit in 2014.
int store, month; // variables for looping through stores and months

// code for filling out profit

for ( store=0 ; store<25 ; store++ ) {
  for ( month=0 ; month<12 ; month++ ) {
    totalProfit += profit[store][month];
  }
}
```

Arrays af Arrays ▷ Jagged Arrays

Alle indre arrays behøver ikke at have samme længde: Pas på når du indekserer!

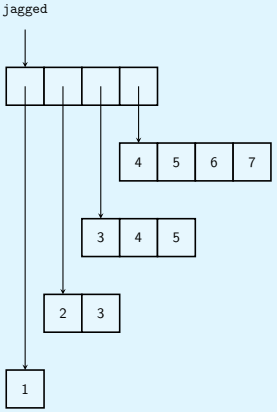
```
int[][] jagged = new int[4][]{
  new int[1]{1},
  new int[2]{2,3},
  new int[3]{3,4,5},
  new int[4]{4,5,6,7}
};

for (int y=0 ; y<jagged.Length ; y++) {
  for (int x=0 ; x<jagged[0].Length ; x++) {
    Console.WriteLine("jagged["+y+"]["+x+
                      "] = "+jagged[y][x]);
  }
}
```



Arrays af Arrays ▷ Jagged Arrays

Alle indre arrays behøver ikke at have samme længde: Pas på når du indekserer!

```
int[][] jagged = new int[4][]{
  new int[1]{1},
  new int[2]{2,3},
  new int[3]{3,4,5},
  new int[4]{4,5,6,7}
};

for (int y=0 ; y<jagged.Length ; y++) {
  for (int x=0 ; x<jagged[0].Length ; x++) {
    Console.WriteLine("jagged["+y+"]["+x+
                      "] = "+jagged[y][x]);
  }
}
```

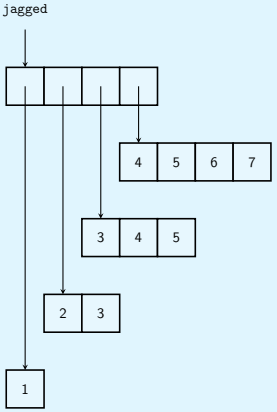


Nope!

Arrays af Arrays ▷ Jagged Arrays

Alle indre arrays behøver ikke at have samme længde: Pas på når du indekserer!

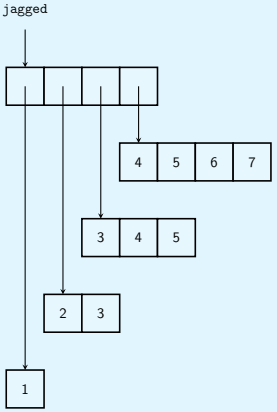
```
int[][] jagged = new int[4][]{
  new int[1]{1},
  new int[2]{2,3},
  new int[3]{3,4,5},
  new int[4]{4,5,6,7}
};

for (int y=0 ; y<jagged.Length ; y++) {
  for (int x=0 ; x<jagged[1].Length ; x++) {
    Console.WriteLine("jagged["+y+"]["+x+
                      "] = "+jagged[y][x]);
  }
}
```



Arrays af Arrays ▷ Jagged Arrays

Alle indre arrays behøver ikke at have samme længde: Pas på når du indekserer!

```
int[][] jagged = new int[4][]{
  new int[1]{1},
  new int[2]{2,3},
  new int[3]{3,4,5},
  new int[4]{4,5,6,7}
};

for (int y=0 ; y<jagged.Length ; y++) {
  for (int x=0 ; x<jagged[1].Length ; x++) {
    Console.WriteLine("jagged["+y+"]["+x+
                      "] = "+jagged[y][x]);
  }
}
```

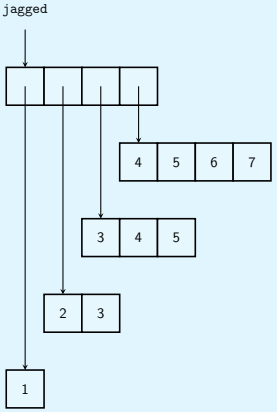


Nope!

Arrays af Arrays ▷ Jagged Arrays

Alle indre arrays behøver ikke at have samme længde: Pas på når du indekserer!

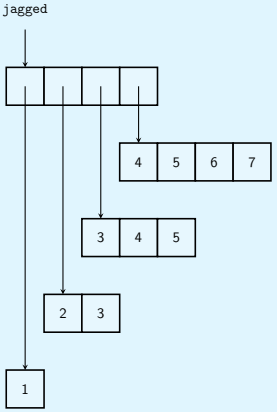
```
int[][] jagged = new int[4][]{
  new int[1]{1},
  new int[2]{2,3},
  new int[3]{3,4,5},
  new int[4]{4,5,6,7}
};

for (int y=0 ; y<jagged.Length ; y++) {
  for (int x=0 ; x<jagged[y].Length ; x++) {
    Console.WriteLine("jagged["+y+"]["+x+
                      "] = "+jagged[y][x]);
  }
}
```



Arrays af Arrays ▷ Jagged Arrays

Alle indre arrays behøver ikke at have samme længde: Pas på når du indekserer!

```
int[][] jagged = new int[4][]{
  new int[1]{1},
  new int[2]{2,3},
  new int[3]{3,4,5},
  new int[4]{4,5,6,7}
};

for (int y=0 ; y<jagged.Length ; y++) {
  for (int x=0 ; x<jagged[y].Length ; x++) {
    Console.WriteLine("jagged["+y+"]["+x+
                      "] = "+jagged[y][x]);
  }
}
```



Yay!

Part 6:
Multidimensional Arrays

Multidimensional Arrays ▷ Beskrivelse

I stedet for arrays af arrays kan man ofte anvende multidimensionelle arrays.

De har ikke et ydre array med referencer til indre arrays.

I stedet bestemmes størrelserne af dimensionerne på oversættelsestidspunktet, og dette gør at elementer kan slås direkte op.

Multidimensional Arrays ▷ Layout i Hukommelsen

Følgende resulterer i ens allokeringer:

```
int[,] marray = {
  {1,2,3,4},
  {2,3,4,5},
  {3,4,5,6},
  {4,5,6,7}
};

int[] array = {
  1,2,3,4,
  2,3,4,5,
  3,4,5,6,
  4,5,6,7
};
```

Multidimensional Arrays ▷ Layout i Hukommelsen

Følgende resulterer i ens allokeringer:

```
int[,] marray = {
  {1,2,3,4},
  {2,3,4,5},
  {3,4,5,6},
  {4,5,6,7}
};

int[] array = {
  1,2,3,4,
  2,3,4,5,
  3,4,5,6,
  4,5,6,7
};
```

Hvis i udpeget en poistion i `array` er det tilsvarende position i `marray`:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} i \bmod width \\ \lfloor i / width \rfloor \end{pmatrix}$$

Hvis x og y udpeget en position i `marray` er det tilsvarende position i `array` givet ved:

$$i = y \cdot width + x$$

Multidimensional Arrays ▷ Layout i Hukommelsen

Følgende resulterer i ens allokeringer:

```
int[,] marray = {
  {1,2,3,4},
  {2,3,4,5},
  {3,4,5,6},
  {4,5,6,7}
};

int[] array = {
  1,2,3,4,
  2,3,4,5,
  3,4,5,6,
  4,5,6,7
};
```

Hvis i udpeget en poistion i `array` er det tilsvarende position i `marray`:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} i \bmod width \\ \lfloor i / width \rfloor \end{pmatrix}$$

Hvis x og y udpeget en position i `marray` er det tilsvarende position i `array` givet ved:

$$i = y \cdot width + x$$

Bemærk: Dette kan kun lade sig gøre fordi bredden er både konstant og kendt.

Multidimensional Arrays ▷ vs Arrays af Arrays

Multidimensionelle arrays frem for arrays af arrays:

- Opslag "koster" kun ét opslag i hukommelsen og er derfor hurtigere.

Arrays af arrays frem for multidimensionelle arrays:

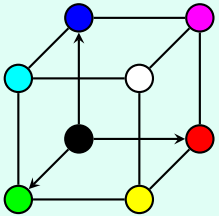
- Giver mulighed for 2d strukturer der ikke er rektangulære, og lignende for 2+ strukturer.
- Giver mulighed for at indices på første dimension refererer til samme indre array (og dermed spare plads).

Hvilken implementation der er den bedste afhænger i stor udstrækning af det problem man står overfor.

Ofte er det ikke vigtigt.

Part 7:
Structs

Structs ▷ Farver

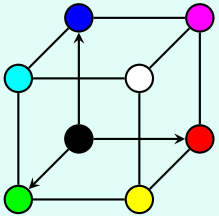


I en computer repræsenteres farver typisk som tre værdier:

1. Rød intensitet
2. Grøn intensitet
3. Blå intensitet

Ved at blande disse værdier kan en vilkårlig synlig farve kunne repræsenteres[†].

Structs ▷ Farver



I en computer repræsenteres farver typisk som tre værdier:

1. Rød intensitet
2. Grøn intensitet
3. Blå intensitet

Ved at blande disse værdier kan en vilkårlig synlig farve kunne repræsenteres[†].

Oftest inddeles hver af disse farvekomponenter i 256 niveauer. Det kan repræsenteres med en `byte`:

```
byte red;
byte green;
byte blue;
```

Structs ▷ Definition

Problem: Nogle gange hører værdier sammen, og vi ønsker at arbejde med dem som om det var en enkelt værdi.

En *struct* er en *struktur* af værdier, der ligger ved siden af hinanden i hukommelsen.

Erklæring:

```
struct Color {
  public byte red;
  public byte green;
  public byte blue;
}
```

C# har nogle designvalg der ikke er specielt pædagogiske[†]:

- For at vi kan bruge dem skal felterne erklæres `public`.
- Structs skal erklæres i bunden af fores fil.

Structs ▷ Eksempel

```
Color magenta = new Color { red=255 , green=0 , blue=255 };
Console.WriteLine("red="+magenta.red+" green="+magenta.green+"
↪  blue="+magenta.blue);

// darken color
magenta.red   /=2;
magenta.green /=2;
magenta.blue  /=2;
Console.WriteLine("red="+magenta.red+" green="+magenta.green+"
↪  blue="+magenta.blue);

struct Color {
  public byte red;
  public byte green;
  public byte blue;
}
```

Structs ▷ Eksempel

```
Color magenta = new Color { red=255 , green=0 , blue=255 };
Console.WriteLine("red="+magenta.red+" green="+magenta.green+"
↪  blue="+magenta.blue);

// darken color
magenta.red   /=2;
magenta.green /=2;
magenta.blue  /=2;
Console.WriteLine("red="+magenta.red+" green="+magenta.green+"
↪  blue="+magenta.blue);

struct Color {
  public byte red;
  public byte green;
  public byte blue;
}
```

```
red=255 green=0 blue=255
red=127 green=0 blue=127
```

Part 8:
Enums

Enums ▷ Definition

Lad os se på to primitive datatyper:

- En `bool` er type der kan repræsentere værdierne `true` og `false`.
- En `byte` er type der kan repræsentere værdierne 0, 1, 2, ... 255.

Med en `enum` kan vi definere vores egen type ved manuelt at liste samtlige mulige værdier.

Enums ▷ Definition

Lad os se på to primitive datatyper:

- En `bool` er type der kan repræsentere værdierne `true` og `false`.
- En `byte` er type der kan repræsentere værdierne 0, 1, 2, ... 255.

Med en `enum` kan vi definere vores egen type ved manuelt at liste samtlige mulige værdier.

Eksempel: Et spillekorts kulør:

```
enum Suit {
  Spade,
  Heart,
  Diamond,
  Club,
}
```

Enums ▷ Sammenligning

Enum værdier kan sammenlignes:

```
Suit suit = Suit.Heart;
if (suit == Suit.Diamond) {
  Console.WriteLine("Suit is diamond!");
} else {
  Console.WriteLine("Suit is not diamond :-(");
}
```

Enums ▷ Sammenligning

Enum værdier kan sammenlignes:

```
Suit suit = Suit.Heart;
if (suit == Suit.Diamond) {
  Console.WriteLine("Suit is diamond!");
} else {
  Console.WriteLine("Suit is not diamond :-(");
}
```

Udprint:

```
Suit is not diamond :-(
```

Enums ▷ Casting til Heltal

Enum værdier kan castes til et unikt heltal (nummereret fortløbende fra 0):

```
Suit suit = Suit.Heart;
int suitInt = (int) suit;
Console.WriteLine(suitInt);
```

Enums ▷ Casting til Heltal

Enum værdier kan castes til et unikt heltal (nummereret fortløbende fra 0):

```
Suit suit = Suit.Heart;
int suitInt = (int) suit;
Console.WriteLine(suitInt);
```

Udprint:

```
1
```

Enums ▷ Eksempel

```
Color[] colors = new Color[(int) ColorName.Count];
colors[(int) ColorName.Red]     = new Color { red=255 , green=  0 , blue=  0 };
colors[(int) ColorName.Green]   = new Color { red=  0 , green=255 , blue=  0 };
colors[(int) ColorName.Magenta] = new Color { red=255 , green=  0 , blue=255 };

ColorName c = ColorName.Magenta;
Color color = colors[(int) c];

Console.WriteLine("red="+color.red+" green="+color.green+" blue="+color.blue);

enum ColorName {
  Red,
  Green,
  Magenta,
  Count,
}

struct Color {
  public byte red;
  public byte green;
  public byte blue;
}
```

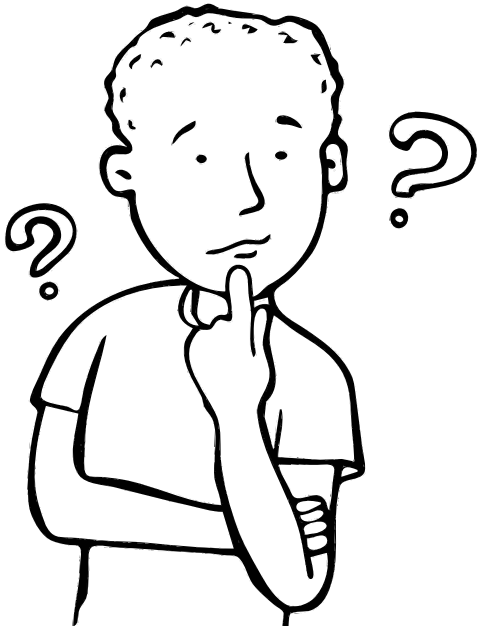
Enums ▷ Eksempel

```
Color[] colors = new Color[(int) ColorName.Count];
colors[(int) ColorName.Red]     = new Color { red=255 , green=  0 , blue=  0 };
colors[(int) ColorName.Green]   = new Color { red=  0 , green=255 , blue=  0 };
colors[(int) ColorName.Magenta] = new Color { red=255 , green=  0 , blue=255 };

ColorName c = ColorName.Magenta;
Color color = colors[(int) c];

Console.WriteLine("red="+color.red+" green="+color.green+" blue="+color.blue);

enum ColorName {
  Red,
  Green,
  Magenta,
  Count,
}

struct Color {
  public byte red;
  public byte green;
  public byte blue;
}
```

```
red=255 green=0 blue=255
```

Fristelsen

Questions?



https://openclipart.org/detail/236687/boy-thinking-of-question