

```
int GrootsteGemeneDeler(int getal1, int getal2)
{
    while (getal2 != 0)
    {
        int rest = getal1 % getal2;
        getal1 = getal2;
        getal2 = rest;
    }

    return getal1;
}

static void Main(string[] args)
{
    int getal1 = Int32.Parse(Console.ReadLine());
    int getal2 = Int32.Parse(Console.ReadLine());
    int ggd = GrootsteGemeneDeler(getal1, getal2);
    Console.WriteLine("{0} is de GGD van {1} en {2}.",
        ggd, getal1, getal2);
    Console.ReadKey();
}
```

CO
- ORIGIN
'warn'.

monitor

thing 2
duties 3
picture

Programmeren 1 (C#)

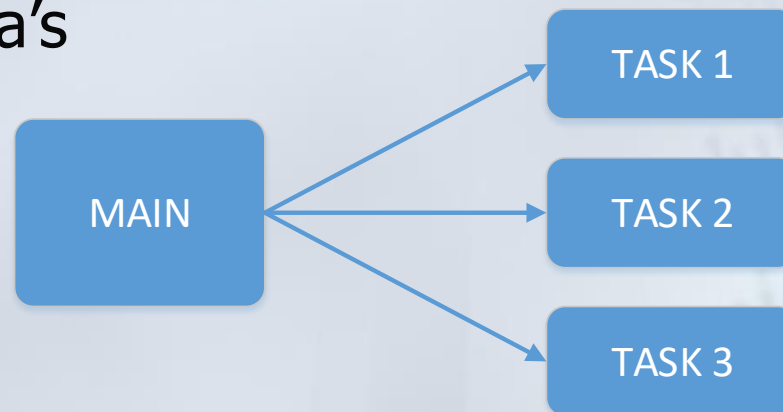
Gerwin van Dijken (gerwin.vandijken@inholland.nl)

Programma periode 1.1 (Programmeren 1)

01 (wk-36)	Inleiding / Visual Studio 2017/2019
02 (wk-37)	Sequentie
03 (wk-38)	Selectie
04 (wk-39)	Iteratie
05 (wk-40)	Array's
06 (wk-41)	Methoden
07 (wk-42)	Herhaling / oefententamen
08 (wk-43)	<i>roostervrije week</i>
09 (wk-44)	praktijktentamen (<i>computer opdrachten</i>)
10 (wk-45)	-

Subprogramma's

- Procedures (Pascal)
 - Subroutines (Visual Basic)
 - Functies (C, C++)
 - **Methoden** (Java, C#)
-
- Bouwstenen voor een specifieke taak
 - Eén programma met meerdere taken → meerdere subprogramma's



Voordelen van subprogramma's

- Voorkomen van redundantie
 - code hergebruik
 - subprogramma meerdere keren gebruiken
- Opdelen van een complex programma in minder complexe (sub)programma's
- Leesbaarder maken van een programma

Procedure/functie vs C# methoden

- Algemeen
 - procedure: geeft geen waarde/resultaat terug
 - functie: geeft wel een waarde/resultaat terug
- C#
 - methode: geeft wel of niet een waarde terug

Procedure

(geeft geen waarde terug)

main program

read number

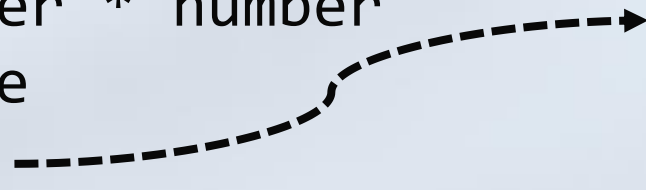
square = number * number

display square

WaitForUser()

WaitForUser()

read character



C# methode aanroep/method call

```
class Program
{
    // main program
    static void Main(string[] args)
    {
        Console.WriteLine("Enter an integer number: ");
        string input = Console.ReadLine();
        int number = int.Parse(input);
        int square = number * number;
        Console.WriteLine("The square of {0} is {1}.", number, square);
        WaitForUser();
    }

    // subprogram
    static void WaitForUser()
    {
        Console.ReadKey();
    }
}
```

A dashed black arrow originates from the `WaitForUser();` line in the `Main` method and points to the `WaitForUser()` method definition. A green circle highlights the `void` keyword in the `WaitForUser()` signature, with a green arrow pointing to the text *void (leeg): geen return waarde*.

Functie *(geeft waarde terug)*

main program

```
fiveSquared = CalcFiveSquared()  
display fiveSquared
```

CalcFiveSquared()

```
result = 5 * 5  
return result
```

- Hoofdprogramma vangt de return-waarde op

C# method aanroep/method call

```
class Program
{
    // main program
    static void Main(string[] args)
    {
        int fiveSquared = CalcFiveSquared();
        Console.WriteLine("Result is {0}.", fiveSquared);
        Console.ReadKey();
    }

    // subprogram
    static int CalcFiveSquared()
    {
        int result = 5 * 5;
        return result;
    }
}
```

C# method aanroep/method call

```
class Program
{
    // main program
    static void Main(string[] args)
    {
        int fiveSquared = CalcFiveSquared();
        Console.WriteLine("Result is {0}.", fiveSquared);
        Console.ReadKey();
    }

    // subprogram
    static int CalcFiveSquared()
    {
        int result = 5 * 5;
        return result;
    }
}
```

- Return value is een integer value.

Formele/actuele parameters

main program

read number1

read number2

product = CalcProduct(number1, number2)

display product

actuele parameters
(argumenten)

CalcProduct(num1, num2)

result = num1 * num2

return result

formele parameters

C# forme/actuele parameters

```
class program
{
    // main program
    static void Main(string[] args)
    {
        int number1 = 8, number2 = 4;
        int product = CalcProduct(number1, number2);
        Console.WriteLine("Product is {0}.", product); // 32
        Console.ReadKey();
    }

    // subprogram
    static int CalcProduct(int num1, int num2)
    {
        int result = num1 * num2;
        return result;
    }
}
```

The diagram illustrates the flow of data between the `Main` method and the `CalcProduct` subprogram. In the `Main` method, the variables `number1` and `number2` are assigned the values 8 and 4 respectively. These values are then passed to the `CalcProduct` method as arguments `num1` and `num2`. The `CalcProduct` method calculates the product of `num1` and `num2` and returns the result, which is stored in the `product` variable in the `Main` method. The diagram uses colored circles and arrows to highlight these relationships: yellow circles around `product` and `result`, green circles around `number1` and `num1`, blue circles around `number2` and `num2`, a dashed green arrow from `number1` to `num1`, a dashed blue arrow from `number2` to `num2`, and a dashed yellow arrow from `result` to `product`.

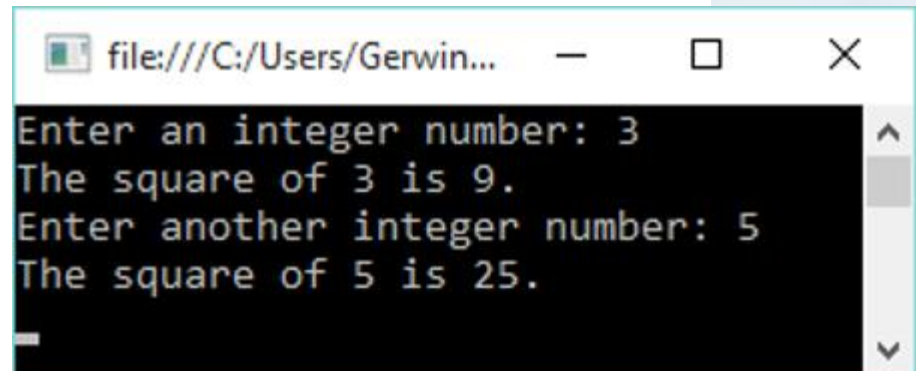
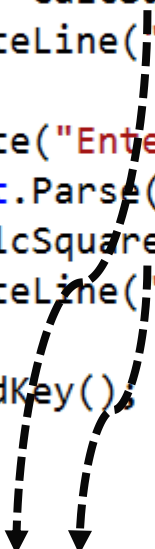
Hergebruik / reusability

```
class Program
{
    // main program
    static void Main(string[] args)
    {
        Console.Write("Enter an integer number: ");
        int number = int.Parse(Console.ReadLine());
        int square = CalcSquare(number);
        Console.WriteLine("The square of {0} is {1}.", number, square);

        Console.Write("Enter another integer number: ");
        number = int.Parse(Console.ReadLine());
        square = CalcSquare(number);
        Console.WriteLine("The square of {0} is {1}.", number, square);

        Console.ReadKey();
    }

    // subprogram
    static int CalcSquare(int num)
    {
        return num * num;
    }
}
```



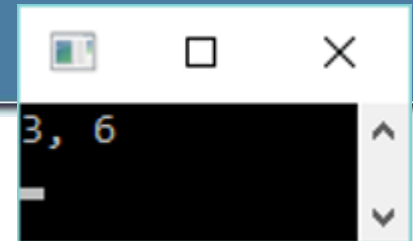
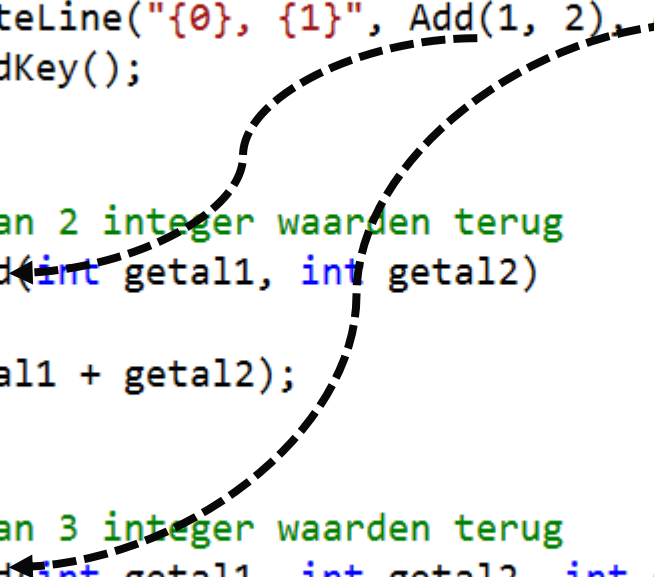
```
file:///C:/Users/Gerwin...
Enter an integer number: 3
The square of 3 is 9.
Enter another integer number: 5
The square of 5 is 25.
```

'Method overloading'

```
class Program
{
    // hoofdprogramma
    static void Main(string[] args)
    {
        Console.WriteLine("{0}, {1}", Add(1, 2), Add(1, 2, 3)); // ??
        Console.ReadKey();
    }

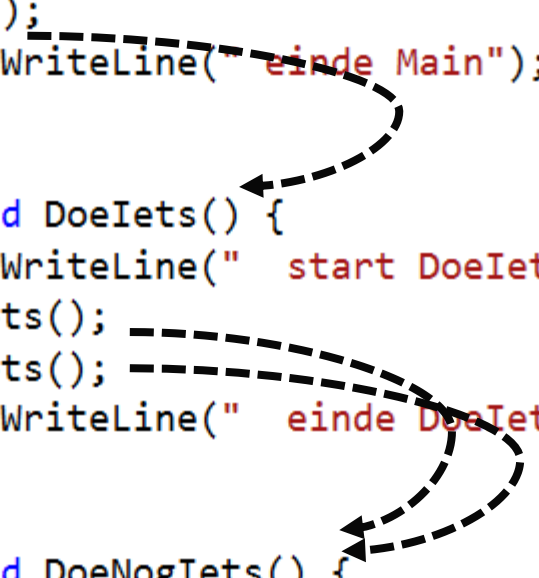
    // geef som van 2 integer waarden terug
    static int Add(int getal1, int getal2)
    {
        return (getal1 + getal2);
    }

    // geef som van 3 integer waarden terug
    static int Add(int getal1, int getal2, int getal3)
    {
        return (getal1 + getal2 + getal3);
    }
}
```

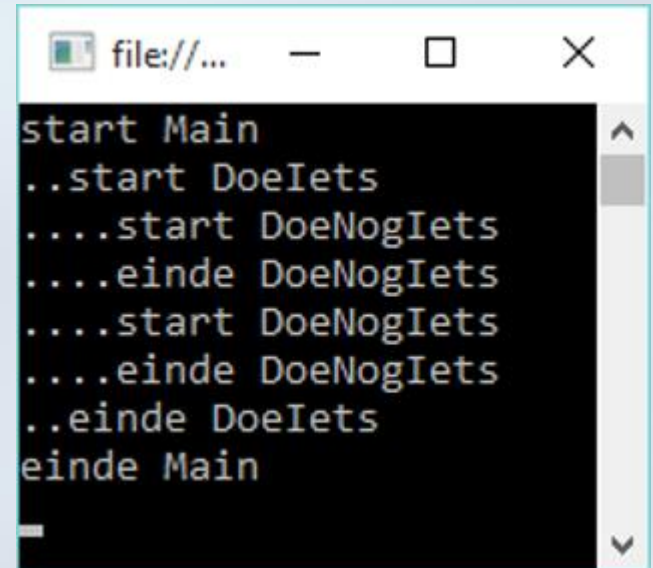


Geneste aanroep

```
class Program {  
  
    static void Main(string[] args) {  
        Console.WriteLine(" start Main");  
        DoeIets();  
        Console.WriteLine(" einde Main");  
    }  
  
    static void DoeIets() {  
        Console.WriteLine(" start DoeIets");  
        DoeNogIets();  
        DoeNogIets();  
        Console.WriteLine(" einde DoeIets");  
    }  
  
    static void DoeNogIets() {  
        Console.WriteLine(" start DoeNogIets");  
        // ...  
        Console.WriteLine(" einde DoeNogIets");  
    }  
}
```



The diagram illustrates the sequence of method calls using dashed arrows. An arrow points from the `DoeIets()` call inside `Main` to the `DoeIets()` method definition. Another arrow points from the `DoeNogIets()` call inside `DoeIets` to the `DoeNogIets()` method definition. A third arrow points from the `DoeNogIets()` call inside `DoeNogIets` back to the `DoeNogIets()` method definition, indicating a recursive call.



The screenshot shows a console window with the following output, which corresponds to the `Console.WriteLine` statements in the code:

```
start Main  
..start DoeIets  
....start DoeNogIets  
....einde DoeNogIets  
....start DoeNogIets  
....einde DoeNogIets  
..einde DoeIets  
einde Main
```

Scope / visibility

```
class Program
{
    // hoofdprogramma
    static void Main(string[] args)
    {
        int getal1 = 8, getal2 = 4;
        int produkt = GeefProdukt(getal1, getal2);
        Console.WriteLine("Produkt is {0}.", produkt); // 32
        Console.ReadKey();
    }

    // subprogramma
    static int GeefProdukt(int getal1, int getal2)
    {
        int resultaat = getal1 * getal2;
        return resultaat;
    }
}
```

*getal1, getal2, produkt zijn lokaal
(niet benaderbaar vanuit GeefProdukt)*

*resultaat is lokaal
(niet benaderbaar vanuit Main)*

Parameters doorggeven aan methoden

- Er zijn 3 manieren om parameters door te geven aan methoden:
 1. pass by value
 2. pass by reference
 3. pass by reference out

Parameter 'pass by value'

```
class Program
{
    // hoofdprogramma
    static void Main(string[] args)
    {
        int g1 = 8, g2 = 4;
        Console.WriteLine("[voor verwisselen] getal1: {0}, getal2: {1}", g1, g2);
        VerwisselWaarden(g1, g2);
        Console.WriteLine("[na verwisselen] getal1: {0}, getal2: {1}", g1, g2);
        Console.ReadKey();
    }

    // subprogramma
    static void VerwisselWaarden(int waarde1, int waarde2)
    {
        int temp = waarde1;
        waarde1 = waarde2;
        waarde2 = temp;
    }
}
```

↓ ↓

pass by value:

- *kopie van g1 en g2 worden meegegeven!*
(gekopieerde waarden worden verwisseld)
- *alleen invoer*

```
file:///C:/Users/Gerwin van Dij...
[voor verwisselen] getal1: 8, getal2: 4
[na verwisselen] getal1: 8, getal2: 4
```

Parameter 'pass by reference'

```
class Program
{
    // hoofdprogramma
    static void Main(string[] args)
    {
        int g1 = 8, g2 = 4;
        Console.WriteLine("[voor verwisselen] getal1: {0}, getal2: {1}", g1, g2);
        VerwisselWaarden(ref g1, ref g2);
        Console.WriteLine("[na verwisselen] getal1: {0}, getal2: {1}", g1, g2);
        Console.ReadKey();
    }
}
```

```
    // subprogramma
    static void VerwisselWaarden(ref int waarde1, ref int waarde2)
    {
        int temp = waarde1;
        waarde1 = waarde2;
        waarde2 = temp;
    }
}
```



pass by reference:

- referentie naar g1 en g2 worden meegegeven (g1 en g2 worden verwisseld)
- zowel invoer als uitvoer

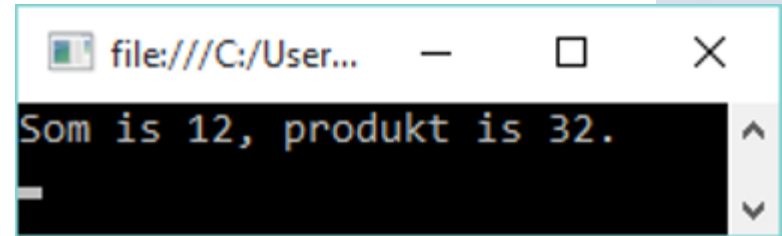
file:///C:/Users/Gerwin van Dij...

```
[voor verwisselen] getal1: 8, getal2: 4
[na verwisselen] getal1: 4, getal2: 8
```

Parameter 'pass by reference out'

```
class Program
{
    // hoofdprogramma
    static void Main(string[] args)
    {
        int getal1 = 8, getal2 = 4;
        int som, produkt;
        SomProdukt(getal1, getal2, out som, out produkt);
        Console.WriteLine("Som is {0}, produkt is {1}.", som, produkt);
        Console.ReadKey();
    }

    // subprogramma
    static void SomProdukt(int g1, int g2, out int som, out int produkt)
    {
        som = g1 + g2;
        produkt = g1 * g2;
    }
}
```



pass by output:

- waarde wordt teruggegeven
- *alleen uitvoer*

Subprogramma's

- **Strong cohesion** (hoge cohesie)

Sterke samenhang tussen elementen binnen in de subroutine

- **Weak coupling** (zwakke koppeling)

Elementen binnen in een subroutine hebben geen/weinig raakvlakken met daar buiten

Opdracht – Scrabble (Wordfeud)

- Lees herhaaldelijk een woord totdat de gebruiker 'stop' invoert. Bepaal van elke ingelezen woord (ongelijk aan 'stop') de Scrabble-score; doe dit via een aparte methode 'BepaalScore' die een string-parameter ontvangt en een int-waarde retourneert. Toon aan het einde de totaalscore.



Dutch

Letter Count Points Letter Count Points

A	7	1	N	11	1
B	2	4	O	6	1
C	2	5	P	2	4
D	5	2	Q	1	10
E	18	1	R	5	2
F	2	4	S	5	2
G	3	3	T	5	2
H	2	4	U	3	2
I	4	2	V	2	4
J	2	4	W	2	5
K	3	3	X	1	8
L	3	3	Y	1	8
M	3	3	Z	2	5

Opdracht – Scrabble (Wordfeud)

main program

```
totalScore = 0
```

```
read word
```

```
while word ≠ "stop"
```

```
    score = CalculateScore(word)
```

```
    totalScore = totalScore + score
```

```
    read word
```

```
display totalScore
```

Opdracht – Scrabble (Wordfeud)

CalculateScore(word)

totalScore = 0

alfabet = "abcdefghijklmnopqrstuvwxyz"

letterValues =

[1,4,5,2,1,4,3,4,2,4,3,3,3,1,1,4,10,2,2,2,2,4,5,8,8,5]

for i=0 to word.Length-1

 character = word[i]

 pos = alfabet.IndexOf(character)

 score = letterValues[pos]

 totalScore = totalScore + score

return totalScore

Huiswerk

- Lezen
 - Yellow Book → zie Moodle
- (praktijk) Programmeren 1 *(deze week)*
 - week 6 opdrachten