

# Programmeren 1 C#

METHODEN

## **1. Methods - vervolg**

- **Params keyword**
- **Scope van variabelen**
- **ByRef en ByVal params**
- **Out en ref params**



# Variabel aantal Parameters



# Variabel aantal Parameters

- Een method in C# kan een variabel aantal parameters aanvaarden bij het gebruik van keyword **params** keyword

```
static long CalcSum(params int[] elements)
{
    long sum = 0;
    foreach (int element in elements)
    {
        sum += element;
    }
    return sum;
}
static void Main()
{
    Console.WriteLine(CalcSum(2, 5));
    Console.WriteLine(CalcSum(4, 0, -2, 12));
    Console.WriteLine(CalcSum());
    Console.ReadKey();
}
```

# Scope van variabelen

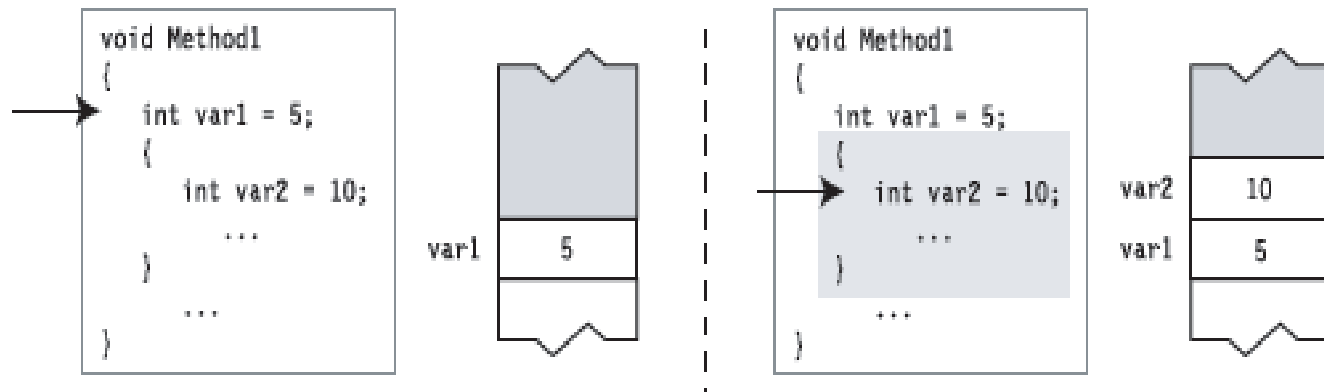
Variabelen gedeclareerd in functie => enkel gekend binnen deze functie

```
void Functie1()  
{  
    int i_test = 33;  
}  
void Functie2()  
{  
    int i_test = 44;  
}
```



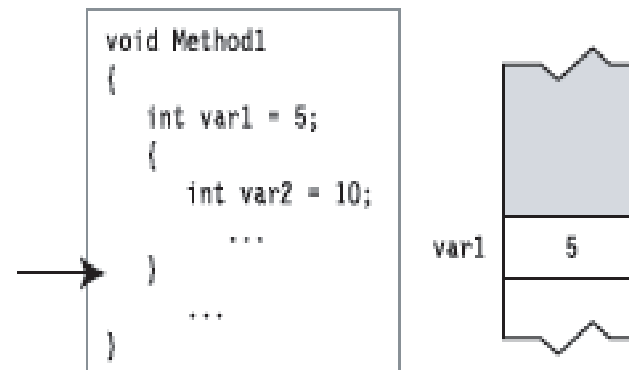
Beide i\_test variabelen  
weten niet van elkaar  
dat ze bestaan.

# Scope van variabelen



1. Variable `var1` is declared before the nested block, and space is allocated for it on the stack.

2. Variable `var2` is declared within the nested block, and space is allocated for it on the stack.

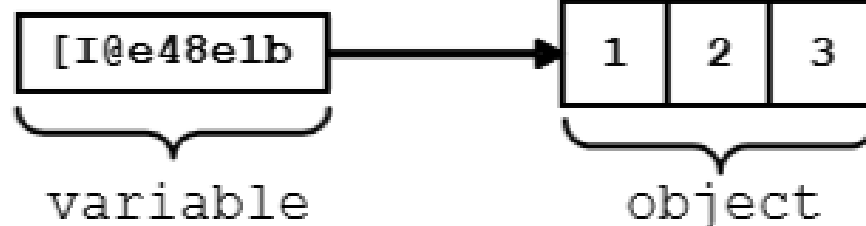


3. When execution passes out of the nested block, `var2` is popped from the stack.

# Parameters by Value en by Reference

- By Default: argumenten “passed by Value” bij primitieve typen (int, float, char,...)
  - In de methode wordt met een kopie gewerkt
  - Buiten de methode zal het oorspronkelijk argument waarmee de methode werd aangeroepen niet veranderen
- Reference types (bv arrays) Passed by Reference

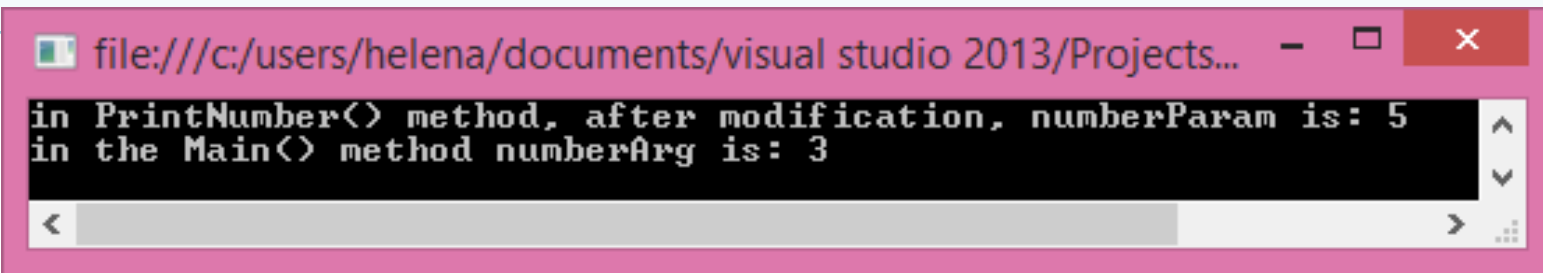
`arrArg: int[]`



# By Default: Parameters passed by Value

```
static void PrintNumber(int numberParam)
{
    // Modifying the primitive-type parameter
    numberParam = 5;
    Console.WriteLine("in PrintNumber() method, after " + "modification,
numberParam is: {0}", numberParam);
}

static void Main()
{
    int numberArg = 3;
    // Copying the value 3 of the argument numberArg to the
    // parameter numberParam
    PrintNumber(numberArg);
    Console.WriteLine("in the Main() method numberArg is: " + numberArg);
    Console.ReadKey();
}
```



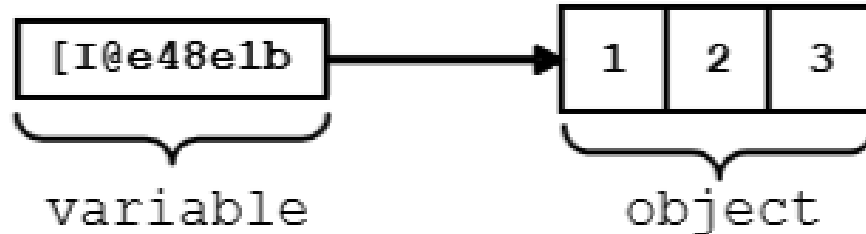


# Passing argument of Reference Type

- Bv Reference type = array

```
int[] arrArg = new int[] { 1, 2, 3 };
```

arrArg: int[]



- Bij doorgave argument wordt een kopie gemaakt van de reference naar de array
  - Binnen de method zal wijzigen van de array-elementen ook effect hebben buiten de method, waar deze werd aangeroepen

# Voorbeeld – passing argument of reference type

```
static void ModifyArray(int[] arrParam)
{
    arrParam[0] = 5;
    Console.WriteLine("In ModifyArray() the param is: ");
    PrintArray(arrParam);
}

static void PrintArray(int[] arrParam)
{
    Console.WriteLine("[");
    int length = arrParam.Length;
    if (length > 0)
    {
        Console.WriteLine(arrParam[0].ToString());
        for (int i = 1; i < length; i++)
        {
            Console.WriteLine(", {0}", arrParam[i]);
        }
    }
    Console.WriteLine("]");
}
```

# Voorbeeld – passing argument of reference type

```
static void Main()
{
    int[] arrArg = new int[] { 1, 2, 3 };
    Console.WriteLine("voor ModifyArray() is de array: ");
    PrintArray(arrArg);
    // Modifying the array's argument
    ModifyArray(arrArg);
    Console.WriteLine("na ModifyArray() is de array: ");
    PrintArray(arrArg);
    Console.ReadKey();
}
```

file:///c:/users/helena/documents/visual studio...

```
Before ModifyArray() the argument is: [1, 2, 3]
In ModifyArray() the param is: [5, 2, 3]
After ModifyArray() the argument is: [5, 2, 3]
```

# Methode ref Parameters

- Syntax
  - In de declaratie van de method voeg keyword **ref** toe:  
`<returnType> <methodname>(ref <type> parameter)`
  - Bij de aanroep van de method, voeg ook keyword **ref** toe:  
`<methodname>(ref argument)`
- de waarde van een variabele dat als argument wordt meegegeven kan gewijzigd worden door de method
- De variabele moet geïnitialiseerd zijn vóór de aanroep van de method

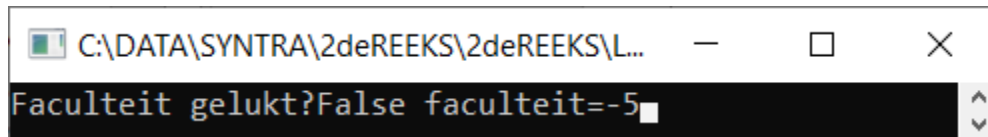
# Oefening ref Parameter

1. Schrijf een bool method `OefeningMethodRefParam` die als input parameter een `ref int` neemt, deze wordt vervangen in de methode door zijn faculteit.
  - Bv. Input parameter = 5. Deze zal worden vervangen in de methode door  $1*2*3*4*5$
  - Geef als returnwaarde false indien de input parameter negatief is, anders true

```
int facult=-5;
```

```
bool facultOK = OefeningMethodRefParam(ref facult);
```

```
Console.WriteLine($"Faculteit gelukt?{facultOK} faculteit={facult}");
```



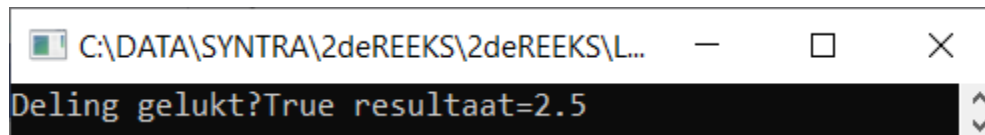
# Method out parameters

- Syntax
  - In de declaratie van de method voeg keyword **out** toe:  
`<returnType> <methodname>(out <type> parameter)`
  - Bij de aanroep van de method, voeg ook keyword **out** toe:  
`<methodname>(out argument)`
- Idem zoals ref parameters: de waarde van een variabele dat als argument wordt meegegeven kan gewijzigd worden door de method
- Variabele moet niet geïnitialiseerd zijn vóór de aanroep
- Compiler verplicht om waarde toe te kennen binnen de method

# Oefening out Parameter

- Schrijf een methode `OefeningMethodOutParam` die als terugkeerwaarde (return) Boolean geeft en die 2 doubles als input parameters neem en 1 output parameter type double waarin het resultaat van de deling van de 2 andere komt. Geef als return waarde false, indien de 2de input param = 0 en de output param wordt in dit geval op 0.0 gezet.

```
double resultaat;  
bool delingOK = OefeningMethodOutParam( 5, 2, out resultaat);  
Console.WriteLine($"Deling gelukt?{delingOK}resultaat={resultaat}");  
int facult=-5;
```



A screenshot of a Windows console window. The title bar shows the file path "C:\DATA\SYNTRA\2deREEKS\2deREEKS\L...". The console output is "Deling gelukt?True resultaat=2.5".

# Vragen?





# Referenties

<http://learncs.org/>

Fundamentals of computer programming with c#  
© svetlin nakov & co.