
# Programming 2

# Program term 1.2

| | |
|---|---|
| 01 (wk-46) | enumerations / structures / classes |
| 02 (wk-47) | 2-dim arrays / flow control |
| 03 (wk-48) | lists / dictionaries |
| 04 (wk-49) | file I/O / error handling |
| 05 (wk-50) | program structure |
| 06 (wk-51) | program structure |
| 07 (wk-52) | Christmas holiday |
| 08 (wk-53) | Christmas holiday |

---

| | |
|---|---|
| 09 (wk-01) | practice exam |
| 10 (wk-02) | *exams* |
| 11 (wk-03) | *retake exams* |
| 12 (wk-04) | *retake exams* |

# 2-dimensional arrays

# 1-dimensional arrays

- one index value: 0 .. Length - 1

```
static void Main(string[] args)
{
    int[] numbers;
    numbers = new int[20];

    Random rnd = new Random();
    for (int i = 0; i < 20; i++)
    {
        numbers[i] = rnd.Next(1, 101);
    }
}
```

Array with integer values.

Array with float values (floating-point values) and array with boolean values (true/false).

Only one index value is needed.

```
static void Main(string[] args)
{
    float[] monthValues = new float[12];
    bool[] bombField = new bool[20];

    // ...
}
```

# 1-dimensional arrays

- Length (property)

```
static void Main(string[] args)
{
    int[] numbers;
    numbers = new int[20];

    Random rnd = new Random();
    for (int i = 0; i < numbers.Length; i++)
    {
        numbers[i] = rnd.Next(1, 101);
    }
}
```

With property 'Length' we can determine the number of elements in an array.

# 2-dimensional arrays

- two index values: row and column

```
static void Main(string[] args)
{
    int[,] matrix = new int[5, 5];  // [rows, columns]
    Random rnd = new Random();

    // process all rows
    for (int row = 0; row < 5; row++)
    {
        // (within each row) process all columns
        for (int col = 0; col < 5; col++)
        {
            matrix[row, col] = rnd.Next(1, 101);
        }
    }
}
```

2 index values, row and column.

*processing order*

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | *r=0* |
| 6 | 7 | 8 | 9 | 10 | *r=1* |
| 11 | 12 | 13 | 14 | 15 | *r=2* |
| 16 | 17 | 18 | 19 | 20 | *r=3* |
| 21 | 22 | 23 | 24 | 25 | *r=4* |
| *c=0* | *c=1* | *c=2* | *c=3* | *c=4* | |

# 2-dimensional arrays

- GetLength (method)

```
static void Main(string[] args)
{
    int[,] matrix = new int[5, 5];  // [rows, columns]
    Random rnd = new Random();

    // process all rows
    for (int row = 0; row < matrix.GetLength(0); row++)
    {
        // (within each row) process all columns
        for (int col = 0; col < matrix.GetLength(1); col++)
        {
            matrix[row, col] = rnd.Next(1, 101);
        }
    }
}
```

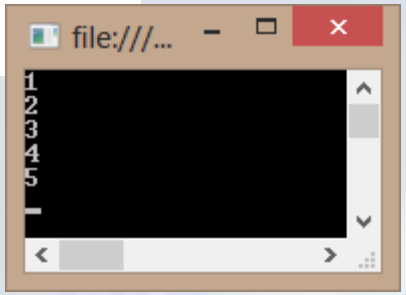
With method GetLength(…) we can determine the size of each dimension (1st, 2nd, …).

# Flow Control

# break – break out of loop

This will stop the while-loop.

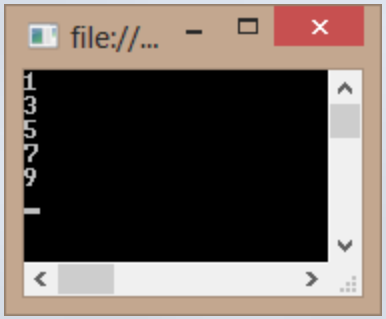
```
// break
int i = 1;
while (i <= 10)
{
  if (i == 6)
    break;
  Console.WriteLine("{0}", i++);
}
Console.WriteLine();
```



# continue – jump to start of loop

```
// continue
for (i = 1; i <= 10; i++)
{
  if ((i % 2) == 0)
    continue;
  Console.WriteLine(i);
}
Console.WriteLine();
```

This stops the current iteration and starts the next one. So the loop continues.



# return – stop current method

```
// return
for (i = 1; i <= 10; i++)
{
  if ((i % 2) == 0)
    return;
  Console.WriteLine(i);
}
Console.WriteLine();
```

*return stops the execution of the method.*

# return – stop current method

```
bool IsPrimeNumber(int number)
{
    if (number < 2) return false;

    bool isPrime = true;
    int i = 2;
    while ((i < number) && (isPrime))
    {
        if ((number % i) == 0)
            isPrime = false;
        else
            i++;
    }
    return isPrime;
}
```

We can implement method IsPrimeNumber a bit more efficient.

```
bool IsPrimeNumber(int number)
{
    if (number < 2) return false;

    int i = 2;
    while (i < number)
    {
        if ((number % i) == 0)
            return false;
        i++;
    }
    return true;
}
```

But… don't use to many return-statements in one method!

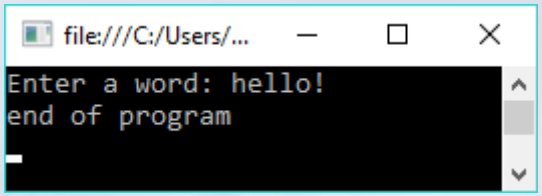
# return – stop current method

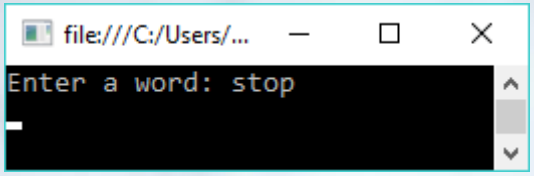
```
static void Main(string[] args)
{
    Console.Write("Enter a word: ");
    string input = Console.ReadLine();

    if (input == "stop")
    {
        return;
    }

    Console.WriteLine("end of program");
    Console.ReadKey();
}
```

A return-statement in the main-method stops the program.



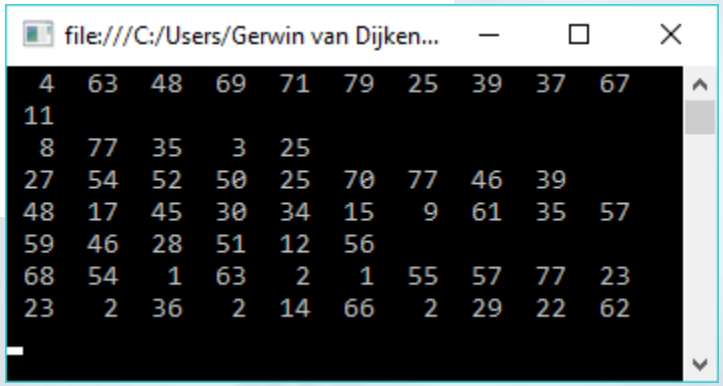
# break – stop enclosing loop

```
int[,] matrix = new int[8, 10]; // 8 rows, 10 columns
Random rnd = new Random();

for (int row = 0; row < matrix.GetLength(0); row++)
{
    for (int col = 0; col < matrix.GetLength(1); col++)
    {
        int number = rnd.Next(1, matrix.Length + 1);
        matrix[row, col] = number;
        Console.Write("{0,3} ", number);

        if (number == (row * 10 + col + 1))
            break;
    }
    Console.WriteLine();
}
```

This will only stop the 'current row'.




check the stop positions/numbers!!

# Stopping a nested loop

- When searching a value in a matrix (2-dimensional array), how can we stop <u>both inner and outer loop</u>, when the search value is found?

```
for row = 0 to number_of_rows - 1
        for col = 0 to number_of_columns – 1
                if matrix[row, col] = number
                        // found, stop both loops...
                end_if
        end_for
end_for
// we want to continue here...
```

# Bad use of break

```
bool IsNumberPresent(int number, int[] numbers)
{
    bool found = false;
    int i = 0;
    while (i < numbers.Length)
    {
        if (numbers[i] == number)
        {
            found = true;
            break;
        }
        else
        {
            i++;
        }
    }
    return found;
}
```

After a break an else-block is not needed…

# Bad use of continue

```
void PrintEvenNumbers(int[] numbers)
{
    for (int i = 0; i < numbers.Length; i++)
    {
        if (numbers[i] % 2 == 0)
        {
            Console.WriteLine("{0} ", numbers[i]);
        }
        else
        {
            continue;
        }
    }
}
```

*continue at the end of a loop is useless!!*

# Debug - demo

- Breakpoint
- Step into
- Step over
- Watches

```
17          void Start()
18          {
19              int[,] matrix = new int[8, 10]; // 8 rows, 10 columns
20              Random rnd = new Random();
21
22              for (int row = 0; row < matrix.GetLength(0); row++)
23              {
24                  for (int col = 0; col < matrix.GetLength(1); col++)
25                  {
26                      int number = rnd.Next(1, 80);
27                      matrix[row, col] = number;
28                      Console.Write("{0,3} ", number);
29
30                      if (number == (row * 10 + col + 1))
31                          break;   ≤1ms elapsed
32                  }
33                  Console.WriteLine();
34              }
35
36              Console.ReadKey();
37          }
38      }
```

100 %

Locals

| Name | Value |
|---|---|
| matrix | {int[8, 10]} |
| rnd | {System.Random} |
| row | 4 |
| col | 6 |
| number | 47 |

# Command line arguments

# Command line arguments

- To make our programs more flexible, we can use "Command line arguments" to feed our program with input

- For example, we want our program to work with a 2 dimensional array with any number of rows and any number of columns. *(4 rows and 8 columns, or 9 rows and 6 columns, or …)*

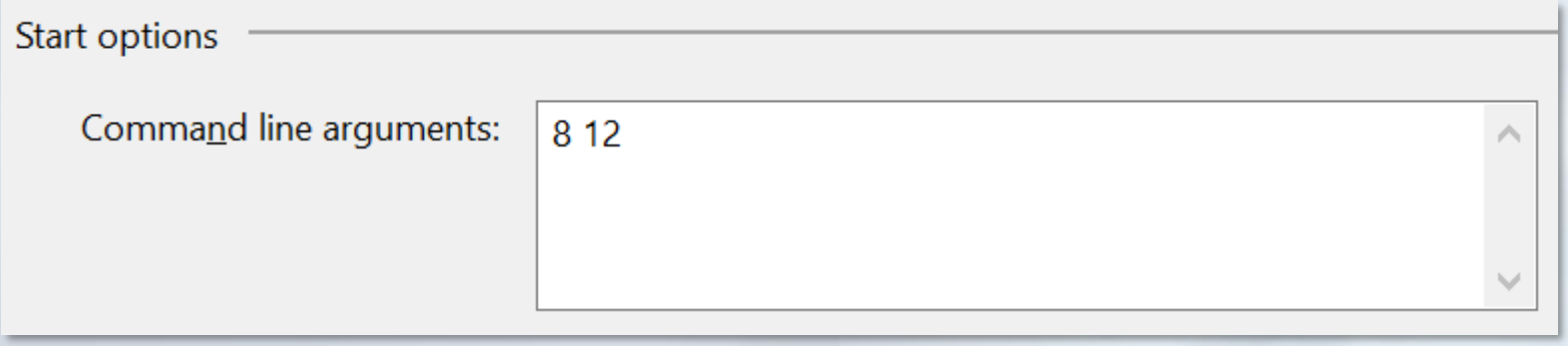
```
void Start()
{
    int[,] matrix = new int[8, 12];

    // ...
}
```

Everytime we want to use different number of rows or different number of columns, we have to change the code…

# Command line arguments

- In Visual Studio, we can give our program input using so called "command line arguments"
- Go to the properties of your program (right mouse click on your project | Properties) and select menu item Debug.
- Fill in the arguments (example below has 2 arguments: 8 and 12)



# Command line arguments

```
static void Main(string[] args)
{
    if (args.Length != 2)
    {
        Console.WriteLine("invalid number of arguments!");
        Console.WriteLine("usage: assignment <nrOfRows> <nrOfColumns>");
        return;
    }

    int nrOfRows = int.Parse(args[0]);
    int nrOfColumns = int.Parse(args[1]);

    Program myProgram = new Program();
    myProgram.Start(nrOfRows, nrOfColumns);
}

void Start(int nrOfRows, int nrOfColumns)
{
    int[,] matrix = new int[nrOfRows, nrOfColumns];

    // your code here...
}
```

This program expects 2 arguments.

We can pass the 2 command line arguments to the Start method, ...

... and use them for creating the 2-dimensional array.

# Homework

- Read paragraphs ‘Yellow Book’
  *(references can be found on Moodle)*

- Assignments week 2
  *(can be found on Moodle)*