

C# FUNDAMENTALS

LINEAIRE DATASTRUCTUREN
LISTS, STACKS, QUEUES, DICTIONARIES

Inhoud data structuren

1. Lijsten – `List<T>` Class
2. Stapels – `Stack<T>` Class
3. Wachtlijen – `Queue<T>` Class
4. Dictionaries – `Dictionary<TKey, TValue>` Class



Basis Data Structuren

- Lineaire structuren
 - **Lists**: vaste en variabele lengte
 - **Stacks: LIFO (Last In First Out)** structuur
 - **Queues: FIFO (First In First Out)** structuur
- Dictionaries (maps)
 - Bevatten paren (key, value)
 - Hash tabellen: gebruiken hash functies om elementen te zoeken/ toe te voegen

Lijsten



Lijst

- Data structuur (container) dat een sequentie (reeks) van elementen bevat
 - Kan vaste of variabele lengte hebben
 - Elementen lineair bijgehouden, in sequentie

List<T> Class

Auto-Resizable geïndexeerde Lijsten



List<T> Class

- Implementeert de abstracte data structuur `list` door gebruik van een array
 - Alle elementen zijn van hetzelfde type `T`
 - `T` kan om het even welk datatype zijn, bv. `List<int>`, `List<string>`, `List<DateTime>`
 - Lengte (Size) wordt dynamisch vergroot wanneer nodig
- Basis-functionaliteit:
 - `Count` – geeft het aantal elementen terug
 - `Add(T)` – voegt een element toe aan het einde van de lijst

List<T> – Eenvoudig voorbeeld

```
..using System.Collections.Generic;

...
static void Main()
{
    List<string> list = new List<string>() { "C#", "Java" };
    list.Add("SQL");
    list.Add("Python");
    foreach (string item in list)
    {
        Console.WriteLine(item);
    }
    // Resultaat:
    //     C#
    //     Java
    //     SQL
    //     Python
}
```

Inline initialisatie:
de compiler voegt
de elementen toe
aan de lijst

List<T> – Eenvoudig voorbeeld



Demo

List<T> – Functionaliteiten

- `list[index]` – toegang tot element via index
- `Insert(index, T)` – voegt een bepaald element toe aan de lijst op een bepaalde positie
- `Remove(T)` – verwijdert het eerste voorkomen van een element uit de lijst
- `RemoveAt(index)` – verwijdert het element op de gespecificeerde positie
- `Clear()` – verwijdert alle elementen
- `Contains(T)` – Bepaalt of een element al dan niet aanwezig is in de lijst

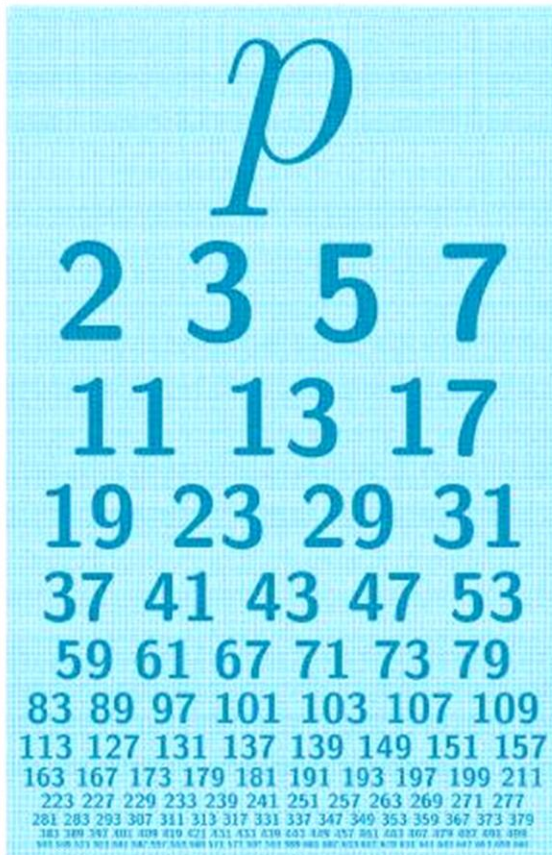
List<T> – Functionaliteiten (2)

- `IndexOf()` – Geeft de index terug van het eerste voorkomen van een waarde in de lijst (start bij index 0)
- `Reverse()` – keert de volgorde van de elementen om in de lijst of een gedeelte ervan
- `Sort()` – sorteert de elementen in de lijst of een gedeelte ervan
- `ToArray()` – converteert de elementen van de lijst naar een array
- `TrimExcess()` – zet de capaciteit op het actuele aantal elementen in de lijst

Priemgetallen binnen een interval

```
...using System.Collections.Generic;
...
static List<int> VindPriemGetallen(int start, int end)
{
    List<int> lijstPriemGetallen = new List<int>();
    for (int num = start; num <= end; num++)
    {
        bool isPriem = true;
        for (int div = 2; div <= Math.Sqrt(num); div++)
        {
            if (num % div == 0){
                isPriem = false;
                break;
            }
        }
        if (isPriem){
            lijstPriemGetallen.Add(num);
        }
    }
    return lijstPriemGetallen;
}
```

Priemgetallen binnen een interval



p

2	3	5	7
11	13	17	
19	23	29	31
37	41	43	47
53	59	61	67
71	73	79	
83	89	97	101
103	107	109	
113	127	131	137
139	149	151	157
163	167	173	179
181	191	193	197
199	211		
223	227	229	233
239	241	251	257
263	269	271	277
281	283	293	307
311	313	317	331
337	347	349	353
359	367	373	379
383	389	397	401
409	419	421	431
433	439	443	449
457	461	463	467
479	487	491	499
503	509	521	523
527	539	541	547
557	563	569	571
577	587	593	599
601	607	613	617
619	627	631	637
641	643	647	653
659	661	667	671
673	677	683	687
691	697	701	709
713	719	727	733
737	743	751	757
761	769	773	787
791	797	809	811
821	827	833	839
841	853	857	863
869	877	881	887
893	899	907	911
913	919	929	937
941	947	953	967
971	977	983	991

Demo

Stapels (Stacks)



Stack

- LIFO (Last In First Out) structuur
- Elementen worden toegevoegd (push) aan de “top”
- Elementen worden verwijderd (pop) van de “top”
- Wordt in verschillende situaties gebruikt
 - Bv. de execution stack van het program

Stack<T> Class



Stack<T> Class

- Implementeert de `stack` data structuur door gebruik van een array
 - Elementen zijn van hetzelfde type `T`
 - `T` kan om het even welk type zijn, bv `Stack<int>`
 - Grootte wordt dynamisch vergroot wanneer nodig
- Basis functionaliteit:
 - `Push(T)` – voegt elementen toe aan de stack
 - `Pop()` – verwijderent en geeft het top element van de Stack terug

Stack<T> Class (2)

- Basis functionaliteit:
 - `Peek()` – geeft het top-element van de stack, zonder het te verwijderen
 - `Count` – geeft het aantal elementen terug
 - `Clear()` – verwijdert alle elementen
 - `Contains(T)` – bepaalt of een element zich in de stack bevindt
 - `ToArray()` – converteert de stack naar een array
 - `TrimExcess()` – zet de capaciteit van de stack op het huidige aantal elementen

Stack<T> – Voorbeeld

- Using `Push()`, `Pop()` and `Peek()` methods

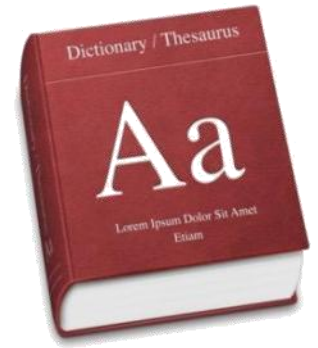
```
static void Main()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("1. Jan");
    stack.Push("2. Piet");
    stack.Push("3. Joris");
    stack.Push("4. Korneel");
    Console.WriteLine("Top = {0}", stack.Peek());
    while (stack.Count > 0)
    {
        string personName = stack.Pop();
        Console.WriteLine(personName);
    }
}
```

Stack<T>



Demo

Oefening - Stack



Vraag in een lus aan de gebruiker om een boodschap in te geven totdat de gebruiker een lege string ingeeft (op enter drukt). Zet elke boodschap in een stack.

Haal één voor één in een 2de lus de boodschappen van de stack en toon ze op de console. In welke volgorde worden de de boodschappen afgedrukt?

Queues

Statische en Dynamische Implementatie



Queue ADT

- FIFO (First In First Out) structuur
- Elementen worden achteraan toegevoegd (Enqueue)
- Elementen worden vooraan verwijderd (Dequeue)
- Wordt gebruikt voor verschillende doeleinden:
 - Print queues, message queues, ...

Queue<T> Class



Queue<T> Class

- Implementeert de queue data structuur door gebruik te maken van een circulairr resizable array
 - Elementen zijn van hetzelfde type `T`
 - `T` kan elk type zijn, bv `Stack<int>`
 - De grootte wordt dynamisch vergroot wanneer nodig
- Basisch functionaliteit:
 - `Enqueue(T)` – voegt een element toe aan het einde van de queue
 - `Dequeue()` – verwijdert het element aan het begin van de queue en geeft deze terug

Queue<T> Class (2)

- Basisch functionaliteit:
 - `Peek()` – geeft het element aan het begin van de queue, zonder het te verwijderen
 - `Count` – geeft het aantal elementen terug
 - `Clear()` – verwijdert alle elementen
 - `Contains(T)` – bepaalt op een bepaald element zich in de queue bevindt
 - `ToArray()` – converteert de queue naar een array
 - `TrimExcess()` – zet de capaciteit op het huidige aantal elementen in de queue

Queue<T> – Voorbeeld

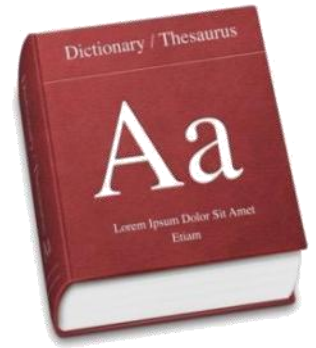
```
static void Main()
{
    Queue<string> queue = new Queue<string>();
    queue.Enqueue("één");
    queue.Enqueue("twee");
    queue.Enqueue("drie");
    queue.Enqueue("vier");
    while (queue.Count > 0)
    {
        string message = queue.Dequeue();
        Console.WriteLine(message);
    }
}
```

Queue<T> Class



Demo

Oefening - Queue



Vraag in een lus aan de gebruiker om een boodschap in te geven totdat de gebruiker een lege string ingeeft (op enter drukt). Zet elke boodschap in een queue.

Haal één voor één in een 2de lus de boodschappen van de queue en toon ze op de console. In welke volgorde worden de de boodschappen afgedrukt?



Dictionaries

The Dictionary (Map) ADT

- Het "**Dictionary**" mapt sleutels op waarden
 - Is een collectie van (key, value) paren
 - Waarden kunnen supersnel worden opgevraagd via hun sleutel
 - Sleutels moeten **uniek** zijn



Dictionary<TKey, TValue>

Operaties

- `Add(key, value)` – element toevoegen met de gespecificeerde key en value
- `Remove(key)` – verwijdert het element met sleutel key
- `this[key]` – toegang tot element met sleutel key
- `Clear()` – verwijdert alle elementen uit dictionary
- `Count` – geeft het aantal elementen terug
- `Keys` – geeft een collectie van sleutels terug
- `Values` – geeft een collectie van waarden terug

Dictionary<TKey, TValue> (2)

operaties:

- `ContainsKey(key)` – geeft true/false indien een element met sleutel key aanwezig is in dictionary
- `ContainsValue(value)` – geeft true/false indien een element met waarde value aanwezig is in dictionary
 - Opgelet: trage operatie bij dictionaries met veel elementen!
- `TryGetValue(key, out value)`
 - Geeft waarde in `value` van element indien sleutel key bestaat en geeft true terug
 - anders geeft `false` terug

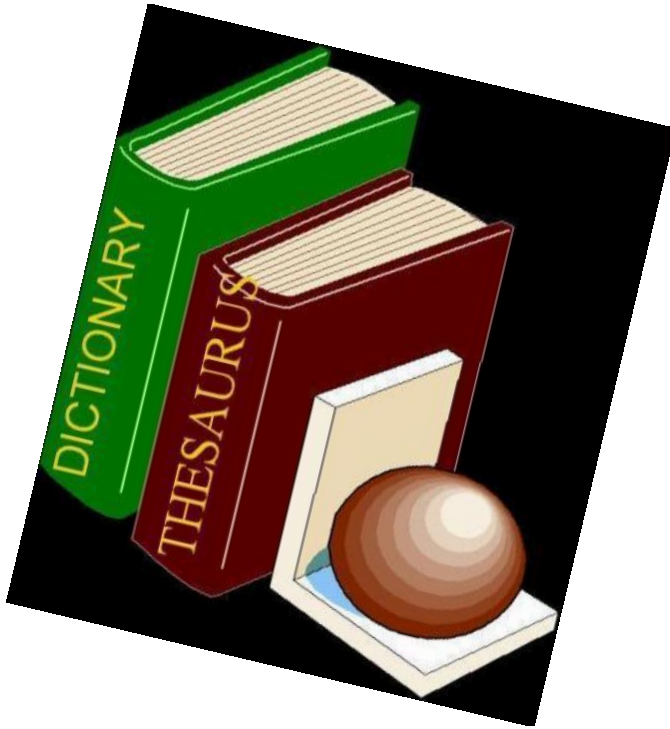
ADT Dictionary – Voorbeeld

Key	Value
1	Brussel
2	Kortrijk
3	Gent
...	...

ADT Dictionary – Voorbeeld –code

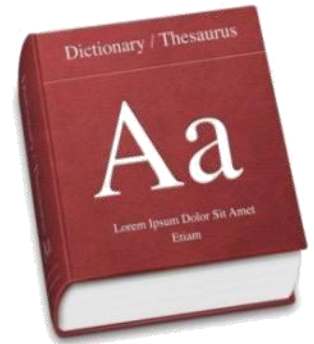
```
using System;
using System.Collections.Generic;

namespace Demo_Dictionaries
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Dictionary<int, string> dict_steden = new Dictionary<int, string>();
            dict_steden.Add(1, "Brussel");
            dict_steden.Add(2, "Kortrijk");
            dict_steden.Add(3, "Brugge");
            Console.WriteLine(dict_steden.ContainsKey(2));
            Console.WriteLine(dict_steden.ContainsValue("Brugge"));
            Console.WriteLine(dict_steden.GetValueOrDefault(3));
            string stad;
            if (dict_steden.TryGetValue(1, out stad)) Console.WriteLine(stad);
        }
    }
}
```



Demo

Oefeningen - Dictionary



- 1. Schrijf de code voor het voorbeeld 1 om een Dictionary<string,string> woordenBoek aan te maken. Zet de voorbeelden in de dictionary, test of een sleutel bestaat, of een waarde bestaat, vraag een waarde op aan de hand van zijn sleutel en schrijf alles naar de console
- 2. Schrijf een programma dat telt hoeveel keer een bepaalde waarde voorkomt in een reeks gehele getallen. De grootte van de dictionary wordt ingegeven via de console. Vul daarna de dictionary met random waarden tussen 1 en 6. Schrijf daarna het waarden naar de console en het aantal keer dat elke waarde voorkomt:

-3-4-4-2-3-3-4-3-2-

2 -> 2 keer

3 -> 4 keer

4 -> 3 keer

Oefeningen - Dictionary - vervolg

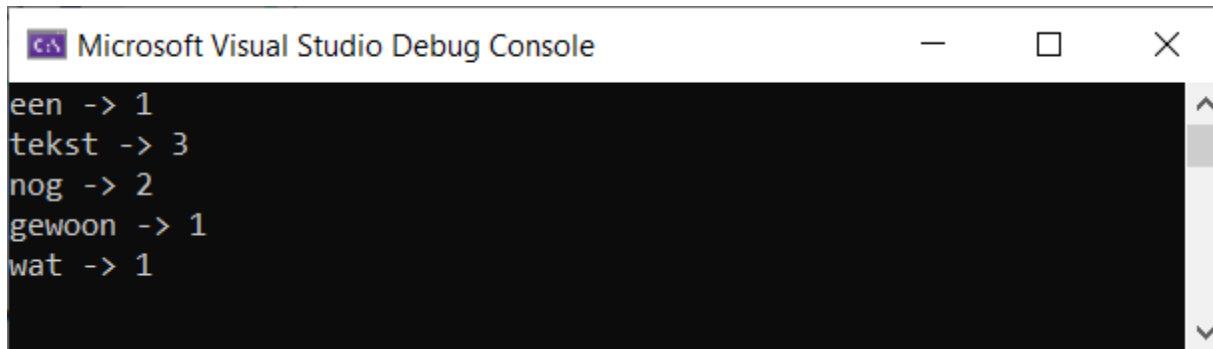
- 3. Schrijf code die het aantal keer dat elk woord voorkomt in een zin uitschrijft naar de console.

Vul de volgende code aan:

```
string text = "een tekst,nog tekst,gewoon nog wat tekst";
```

```
Dictionary<string, int> dict_woorden = new Dictionary<string, int>();
```

```
string[] woorden = text.Split(' ', ',', '.');
```



```
Microsoft Visual Studio Debug Console  
een -> 1  
tekst -> 3  
nog -> 2  
gewoon -> 1  
wat -> 1
```

Samenvatting

- De basis data structuren zijn:
 - List
 - `List<T>` class
 - Stack
 - `Stack<T>` class
 - Queue
 - `Queue<T>` class
 - Dictionary
 - `Dictionary<Tkey, TValue>` class

Data Structure

Vragen?



REFERENTIES

PRO C# 7 WITH .NET AND .NET CORE – ANDREW
TROELSEN – PHILIP JAPIKSE

[HTTPS://WWW.LEARNCS.ORG/](https://www.learncs.org/)

FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C#
© SVETLIN NAKOV & CO