Telerik Academy

Linear Data Structures

Arrays, Lists, Stacks, Queues
Static and Dynamic Implementation

Data Structures and Algorithms

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

Lists

Static and Dynamic Implementations

The List ADT

- What is "list"?
  - A data structure (container) that contains a sequence of elements
    - Can have variable size
    - Elements are arranged linearly, in sequence
  - Can be implemented in several ways
    - Statically (using array → fixed size)
    - Dynamically (linked implementation)
    - Using resizable array (the `List<T>` class)

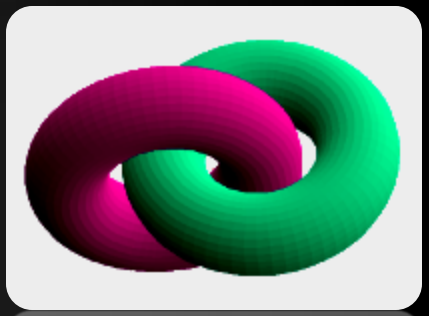
Static List

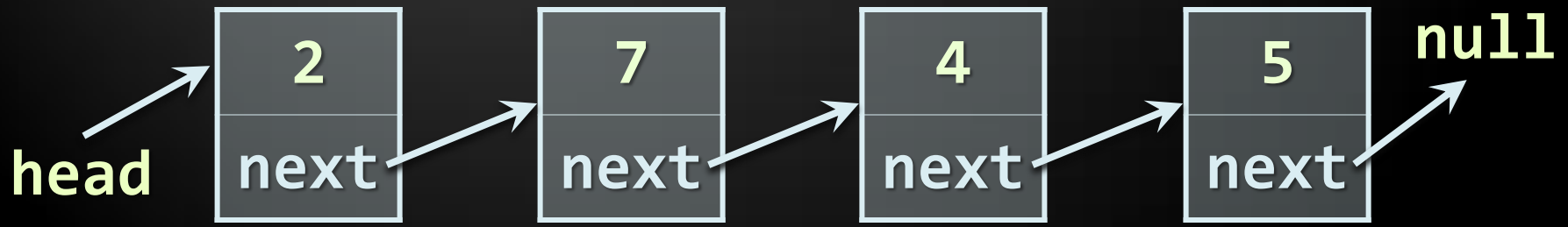
- Implemented by an array
  - Provides direct access by index
  - Has fixed capacity
  - Insertion, deletion and resizing are slow operations

	0	1	2	3	4	5	6	7
L	2	18	7	12	3	6	11	9

Linked List

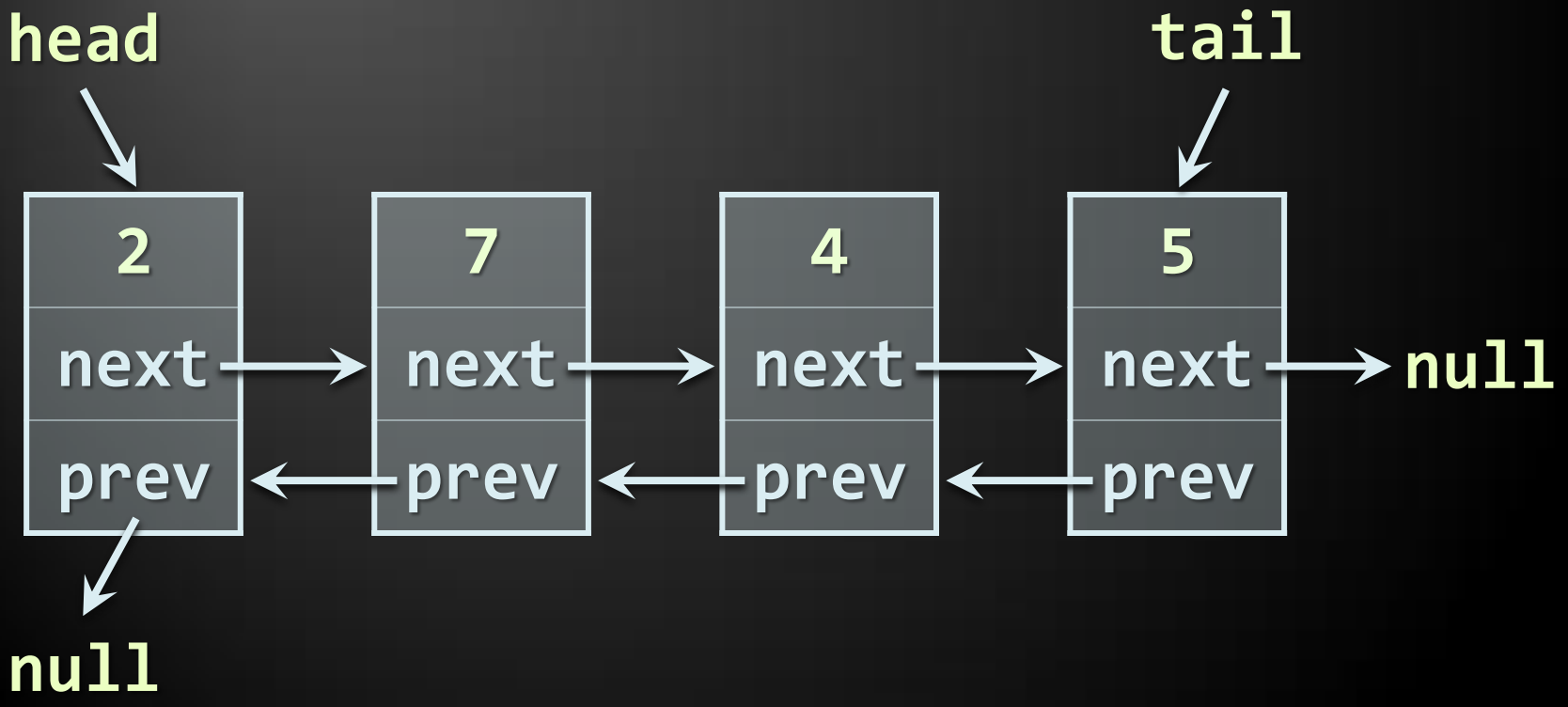
- Dynamic (pointer-based) implementation
- Different forms
  - Singly-linked and doubly-linked
  - Sorted and unsorted
- Singly-linked list
  - Each item has 2 fields: `value` and `next`





Linked List (2)

- Doubly-linked List
  - Each item has 3 fields: `value`, `next` and `prev`



Telerik Academy

The List<T> Class

Auto-Resizable Indexed Lists

Telerik Academy

The List<T> Class

- Implements the abstract data structure list using an array
  - All elements are of the same type T
  - T can be any type, e.g. `List<int>`, `List<string>`, `List<DateTime>`
  - Size is dynamically increased as needed
- Basic functionality:
  - `Count` – returns the number of elements
  - `Add(T)` – appends given element at the end

Telerik Academy

List<T> – Simple Example

```
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);
    }

    // Result:
    //    C#
    //    Java
    //    SQL
    //    Python
}
```

Inline initialization: the compiler adds specified elements to the list.

Telerik Academy

List<T> – Simple Example

Live Demo

Telerik Academy

List<T> – Functionality

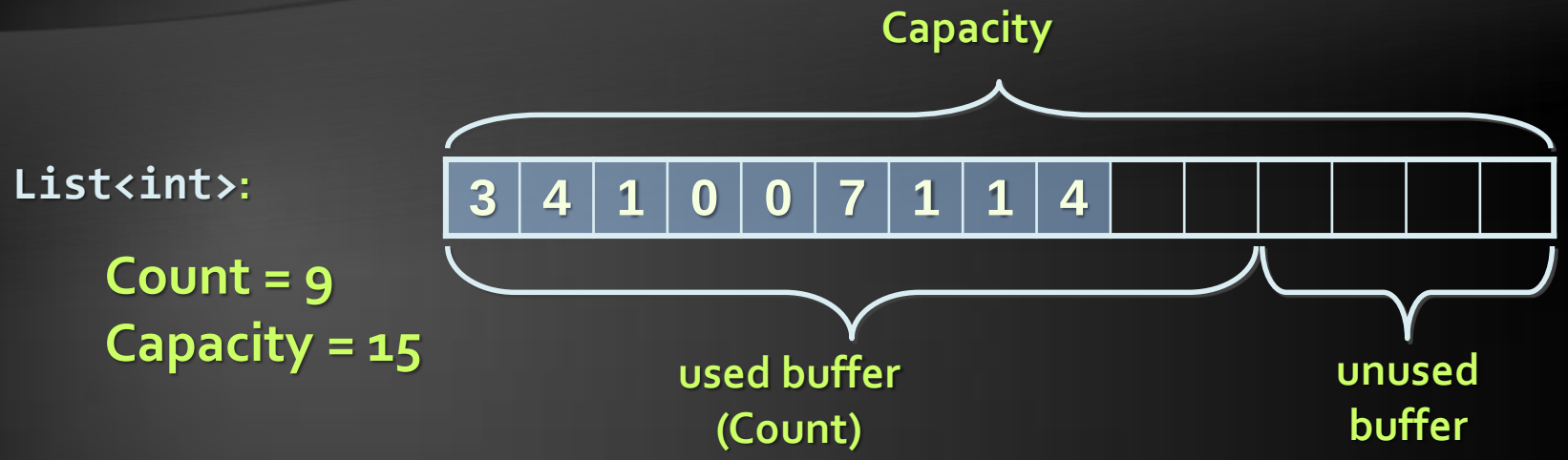
- `list[index]` – access element by index
- `Insert(index, T)` – inserts given element to the list at a specified position
- `Remove(T)` – removes the first occurrence of given element
- `RemoveAt(index)` – removes the element at the specified position
- `Clear()` – removes all elements
- `Contains(T)` – determines whether an element is part of the list

Telerik Academy

List<T> – Functionality (2)

- `IndexOf()` – returns the index of the first occurrence of a value in the list (zero-based)
- `Reverse()` – reverses the order of the elements in the list or a portion of it
- `Sort()` – sorts the elements in the list or a portion of it
- `ToArray()` – converts the elements of the list to an array
- `TrimExcess()` – sets the capacity to the actual number of elements

List<T>: How It Works?



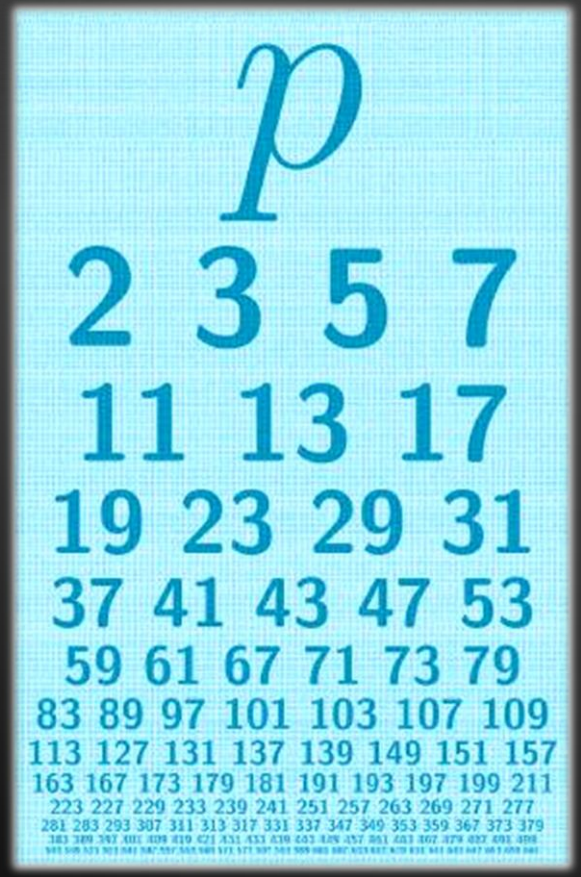
- `List<T>` keeps a buffer memory, allocated in advance, to allow fast `Add(T)`
  - Most operations use the buffer memory and do not allocate new objects
  - Occasionally the capacity grows (doubles)

Telerik Academy

Primes in an Interval – Example

```
static List<int> FindPrimes(int start, int end)
{
    List<int> primesList = new List<int>();
    for (int num = start; num <= end; num++)
    {
        bool prime = true;
        for (int div = 2; div <= Math.Sqrt(num); div++)
        {
            if (num % div == 0)
            {
                prime = false;
                break;
            }
        }
        if (prime)
        {
            primesList.Add(num);
        }
    }
    return primesList;
}
```

Telerik Academy



Primes in an Interval

Live Demo

Telerik Academy

Union and Intersection – Example

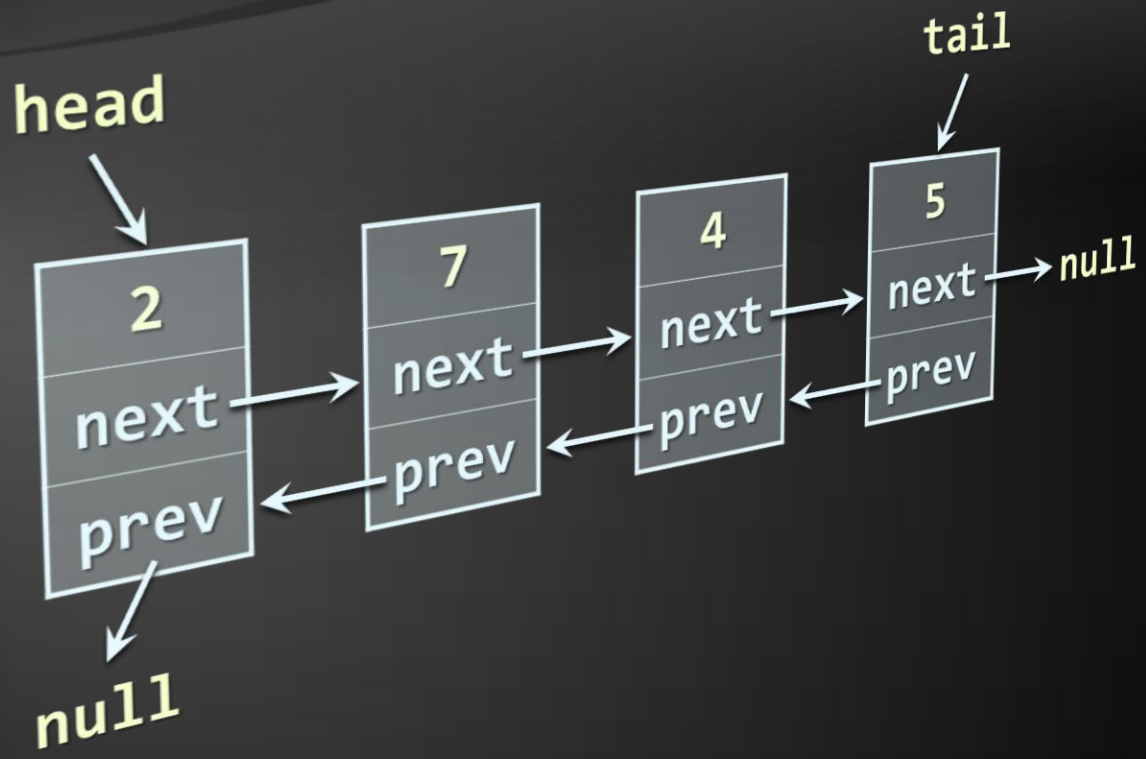
```
int[] Union(int[] firstArr, int[] secondArr)
{
   List<int> union = new List<int>();
   union.AddRange(firstArray);
   foreach (int item in secondArray)
      if (! union.Contains(item))
         union.Add(item);
   return union.ToArray();
}
int[] Intersection(int[] firstArr, int[] secondArr)
{
   List<int> intersect = new List<int>();
   foreach (int item in firstArray)
      if (Array.IndexOf(secondArray, item) != -1)
         intersect.Add(item);
   return intersect.ToArray();
}
```

Telerik Academy

Union and Intersection

Live Demo

Telerik Academy



The LinkedList<T> Class

Dynamic Linked List in .NET

Telerik Academy

The LinkedList<T> Class

- Implements the abstract data structure list using a doubly-linked dynamic structure
  - All elements are of the same type T
  - T can be any type, e.g. `LinkedList<int>`, `LinkedList<string>`, etc.
  - Elements can be added at both sides
- Basic `LinkedList<T>` functionality:
  - `AddFirst(T)`, `AddLast(T)`, `AddBefore(T)`, `AddAfter(T)`, `RemoveFirst(T)`, `RemoveLast(T)`, `Count`

Telerik Academy

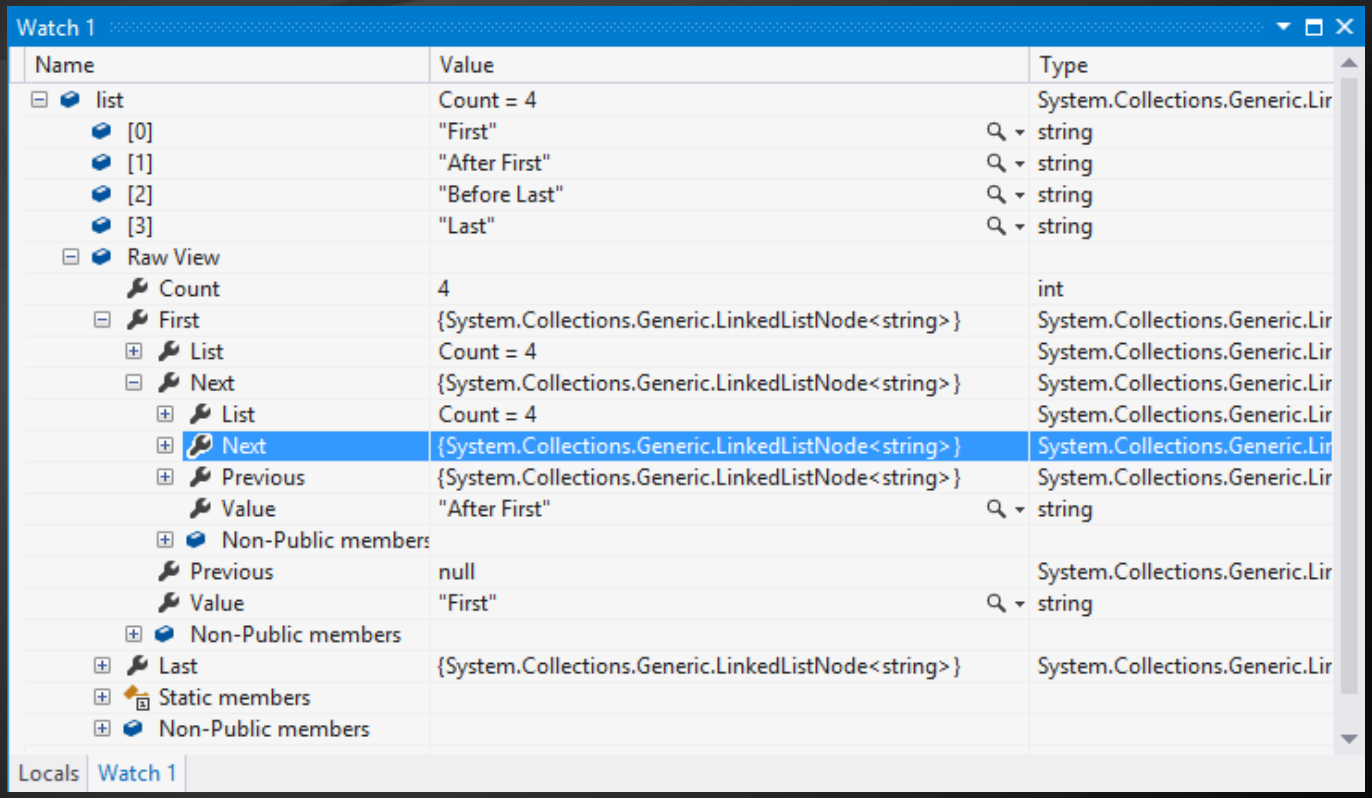
LinkedList<T> – Example

```
static void Main()
{
    LinkedList<string> list =
        new LinkedList<string>();
    list.AddFirst("First");
    list.AddLast("Last");
    list.AddAfter(list.First, "After First");
    list.AddBefore(list.Last, "Before Last");

    Console.WriteLine(String.Join(", ", list));

    // Result: First, After First, Before Last, Last
}
```

Telerik Academy



LinkedList<T>

Live Demo

Telerik Academy

Sorting Lists

Several Ways to Do It

Sorting Lists

```
List<DateTime> list = new List<DateTime>()
{
    new DateTime(2013, 4, 7),
    new DateTime(2002, 3, 12),
    new DateTime(2012, 1, 4),
    new DateTime(1980, 11, 11)
};

list.Sort();

list.Sort((d1, d2) => -d1.Year.CompareTo(d2.Year));

list.OrderBy(date => date.Month)));
```

Telerik Academy

Sorting Lists

Live Demo

Telerik Academy

Stacks

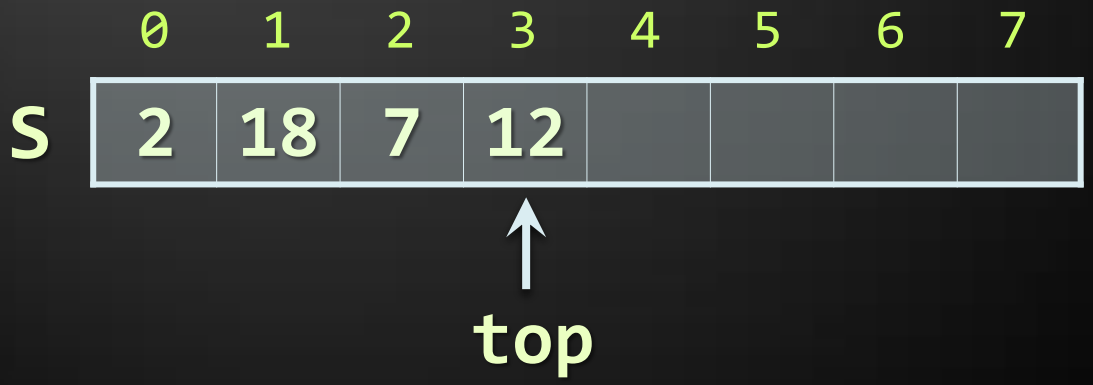
Static and Dynamic Implementation

The Stack ADT

- LIFO (Last In First Out) structure
- Elements inserted (push) at "top"
- Elements removed (pop) from "top"
- Useful in many situations
  - E.g. the execution stack of the program
- Can be implemented in several ways
  - Statically (using array)
  - Dynamically (linked implementation)
  - Using the `Stack<T>` class

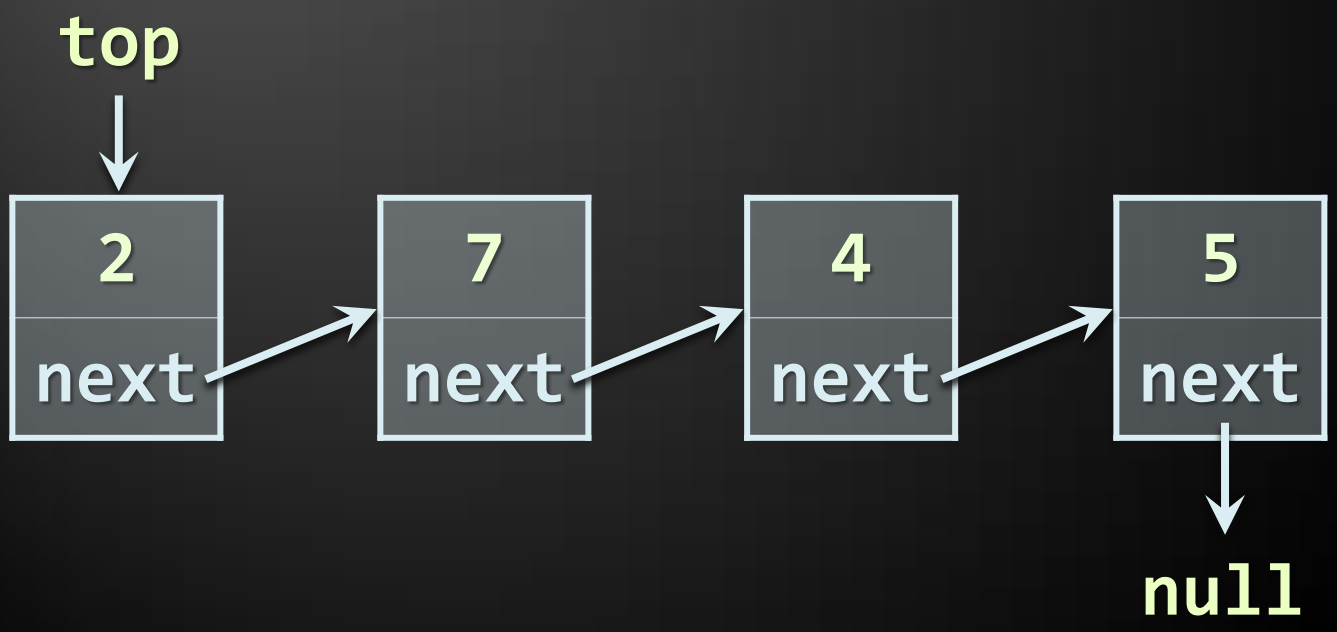
Static Stack

- Static (array-based) implementation
  - Has limited (fixed) capacity
  - The current index (`top`) moves left / right with each pop / push



Linked Stack

- Dynamic (pointer-based) implementation
  - Each item has 2 fields: `value` and `next`
  - Special pointer keeps the top element



Telerik Academy

The Stack<T> Class

The Standard Stack Implementation in .NET

Telerik Academy

The Stack<T> Class

- Implements the stack data structure using an array
  - Elements are from the same type `T`
  - `T` can be any type, e.g. `Queue<int>`
  - Size is dynamically increased as needed
- Basic functionality:
  - `Push(T)` – inserts elements to the stack
  - `Pop()` – removes and returns the top element from the stack

The Stack<T> Class (2)

- Basic functionality:
  - `Peek()` – returns the top element of the stack without removing it
  - `Count` – returns the number of elements
  - `Clear()` – removes all elements
  - `Contains(T)` – determines whether given element is in the stack
  - `ToArray()` – converts the stack to an array
  - `TrimExcess()` – sets the capacity to the actual number of elements

Stack<T> – Example

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

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

Stack<T>

Live Demo

Telerik Academy

Matching Brackets – Example

- We are given an arithmetical expression with brackets that can be nested
- Goal: extract all sub-expressions in brackets
- Example:
  - `1 + (2 - (2+3) * 4 / (3+1)) * 5`
- Result:
  - `(2+3) | (3+1) | (2 - (2+3) * 4 / (3+1))`
- Algorithm:
  - For each '(' push its index in a stack
  - For each ')' pop the corresponding start index

Telerik Academy

Matching Brackets – Solution

```
string expression = "1 + (2 - (2+3) * 4 / (3+1)) * 5";
Stack<int> stack = new Stack<int>();
for (int index = 0; index < expression.Length; index++)
{
    char ch = expression[index];
    if (ch == '(')
    {
        stack.Push(index);
    }
    else if (ch == ')')
    {
        int startIndex = stack.Pop();
        int length = index - startIndex + 1;
        string contents =
            expression.Substring(startIndex, length);
        Console.WriteLine(contents);
    }
}
```

Telerik Academy

Matching Brackets

Live Demo

Telerik Academy

Queues

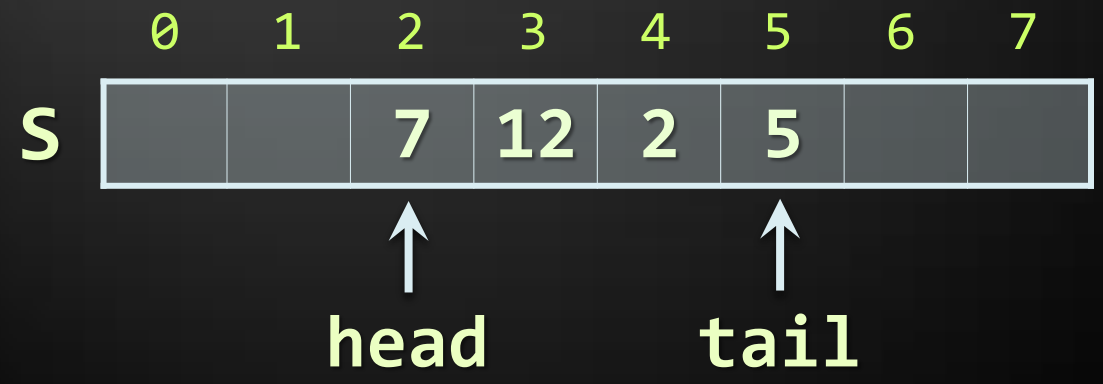
Static and Dynamic Implementation

The Queue ADT

- FIFO (First In First Out) structure
- Elements inserted at the tail (Enqueue)
- Elements removed from the head (Dequeue)
- Useful in many situations
  - Print queues, message queues, etc.
- Can be implemented in several ways
  - Statically (using array)
  - Dynamically (using pointers)
  - Using the `Queue<T>` class

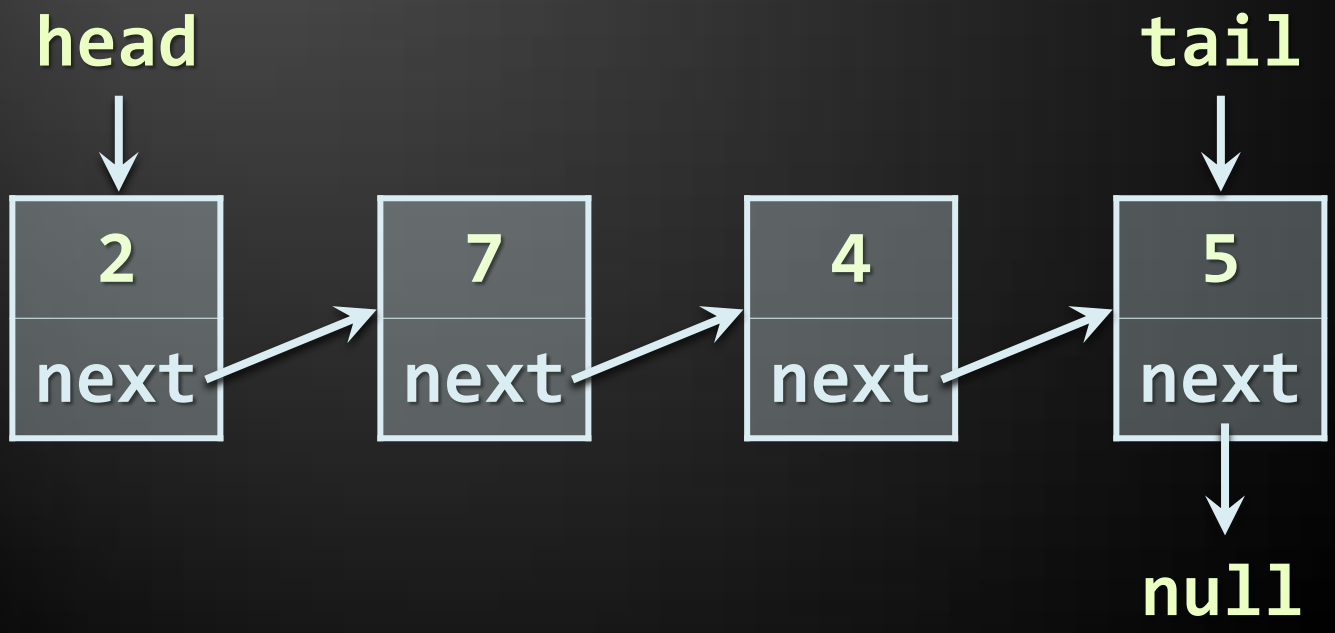
Static Queue

- Static (array-based) implementation
  - Has limited (fixed) capacity
  - Implement as a "circular array"
  - Has `head` and `tail` indices, pointing to the head and the tail of the cyclic queue



Linked Queue

- Dynamic (pointer-based) implementation
  - Each item has 2 fields: `value` and `next`
  - Dynamically create and delete objects



Telerik Academy

The Queue<T> Class

Standard Queue Implementation in .NET

The Queue<T> Class

- Implements the queue data structure using a circular resizable array
  - Elements are from the same type `T`
  - `T` can be any type, e.g. `Queue<int>`
  - Size is dynamically increased as needed
- Basic functionality:
  - `Enqueue(T)` – adds an element to the end of the queue
  - `Dequeue()` – removes and returns the element at the beginning of the queue

The Queue<T> Class (2)

- Basic functionality:
  - `Peek()` – returns the element at the beginning of the queue without removing it
  - `Count` – returns the number of elements
  - `Clear()` – removes all elements
  - `Contains(T)` – determines whether given element is in the queue
  - `ToArray()` – converts the queue to an array
  - `TrimExcess()` – sets the capacity to the actual number of elements in the queue

Queue<T> – Example

- Using Enqueue() and Dequeue() methods

```
static void Main()
{
    Queue<string> queue = new Queue<string>();
    queue.Enqueue("Message One");
    queue.Enqueue("Message Two");
    queue.Enqueue("Message Three");
    queue.Enqueue("Message Four");
    while (queue.Count > 0)
    {
        string message = queue.Dequeue();
        Console.WriteLine(message);
    }
}
```

Telerik Academy

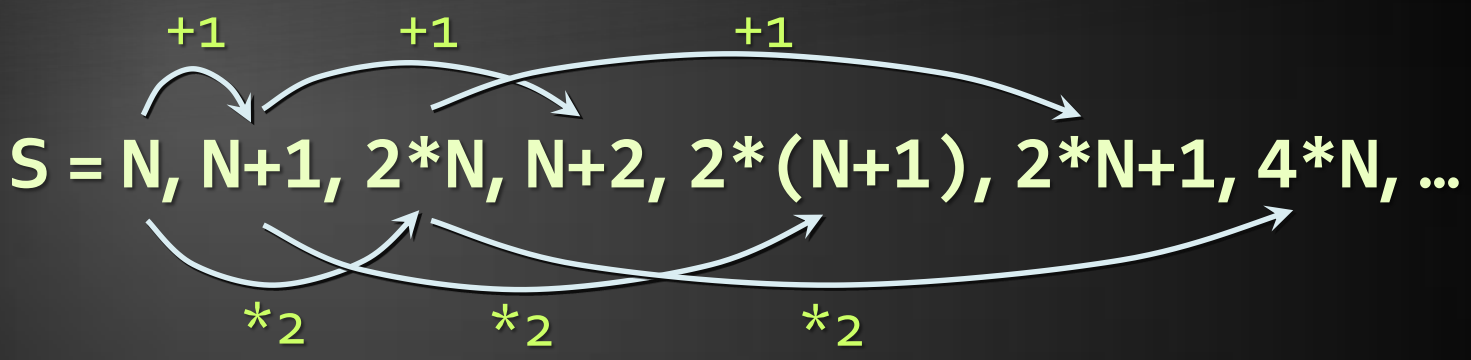
The Queue<T> Class

Live Demo

Telerik Academy

Sequence N, N+1, 2*N

- We are given the sequence:



- Find the first index of given number P
- Example: N = 3, P = 16

  S = 3, 4, 6, 5, 8, 7, 12, 6, 10, 9, 16, 8, 14, …

  Index of P = 11

Telerik Academy

Sequence – Solution with a Queue

```
int n = 3, p = 16;

Queue<int> queue = new Queue<int>();
queue.Enqueue(n);
int index = 0;
while (queue.Count > 0)
{
    int current = queue.Dequeue();
    index++;
    if (current == p)
    {
        Console.WriteLine("Index = {0}", index);
        return;
    }
    queue.Enqueue(current+1);
    queue.Enqueue(2*current);
}
```

Telerik Academy

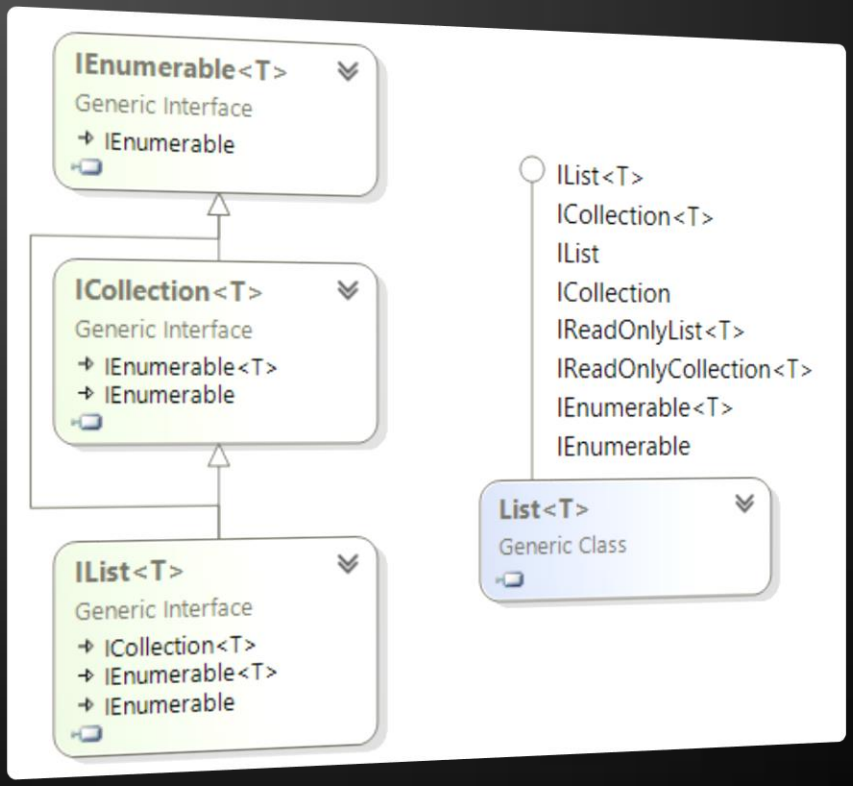
Sequence N, N+1, 2*N

Live Demo

Telerik Academy

List Interfaces in .NET

IEnumerable, ICollection, IList, …

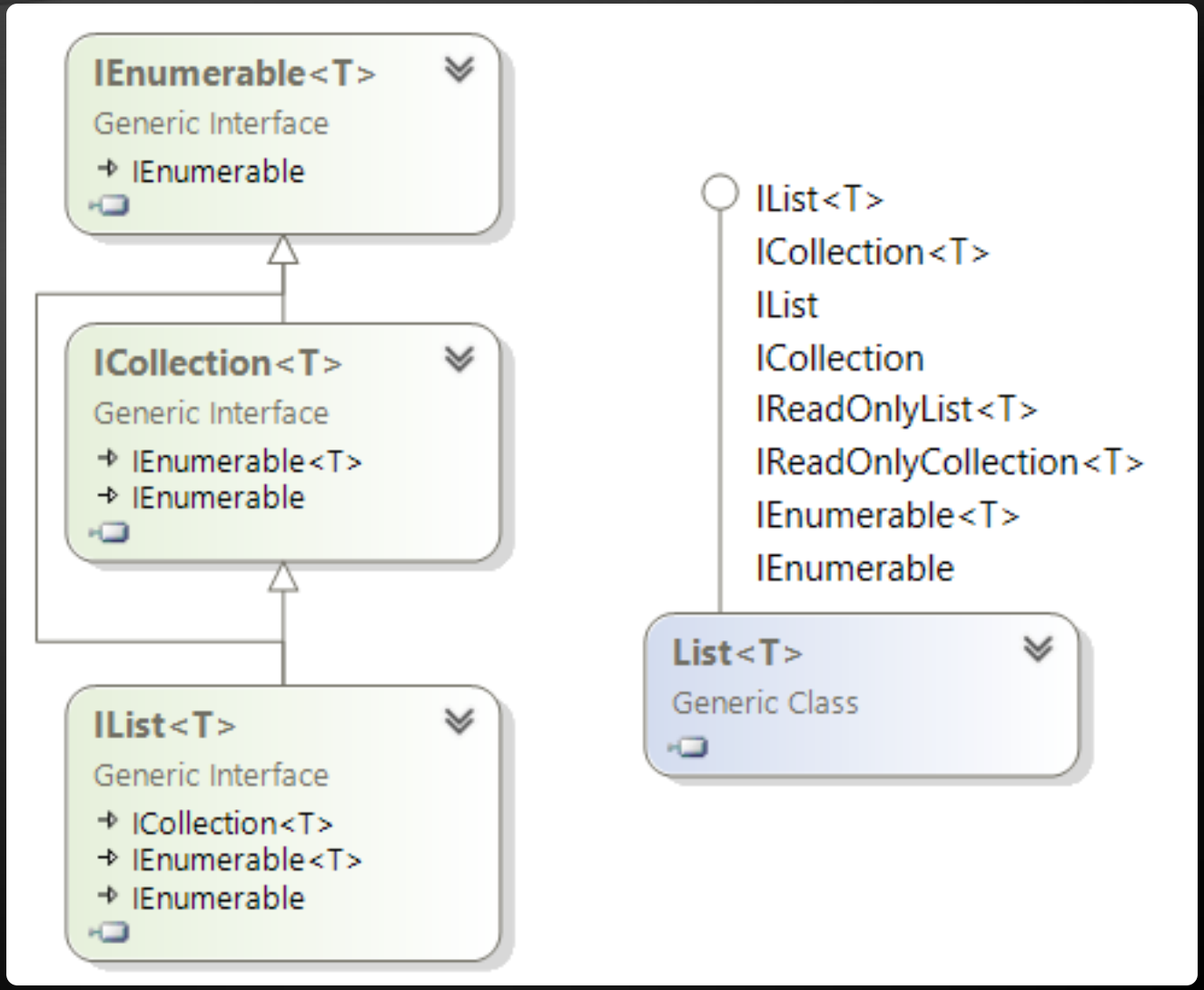


List Interfaces in .NET

- IEnumerable, IEnumerable<T>
  - GetEnumerator() → Current, MoveNext()
- ICollection, ICollection<T>
  - Inherits from IEnumerable<T>
  - Count, Add(…), Remove(…), Contains(…)
- IList, IList<T>
  - Inherits from ICollection<T>
  - Item / indexer [], Insert(…), RemoveAt(…)

Telerik Academy

List Interfaces Hierarchy

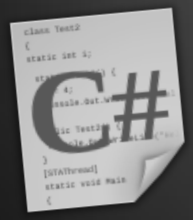


Summary

- The basic linear data structures in the computer programming are:
  - List (static, linked)
    - Implemented by the `List<T>` and `LinkedList<T>` classes in .NET
  - Stack (static, linked)
    - Implemented by the `Stack<T>` class in .NET
  - Queue (static, linked)
    - Implemented by the `Queue<T>` class in .NET

Telerik Academy

Linear Data Structures



Questions?

```
DFS(node)
{
  stack ← node
  visited[node] = true
  while stack not empty
    v ← stack
    print v
    for each child c of v
      if not visited[c]
        stack ← c
        visited[c] = true
}
```

```
DFS(node)
{
  for each child c of node
    DFS(c);
  print the current node;
}
```

http://academy.telerik.com

Exercises

1. Write a program that reads from the console a sequence of positive integer numbers. The sequence ends when empty line is entered. Calculate and print the sum and average of the elements of the sequence. Keep the sequence in `List<int>`.
2. Write a program that reads N integers from the console and reverses them using a stack. Use the `Stack<int>` class.
3. Write a program that reads a sequence of integers (`List<int>`) ending with an empty line and sorts them in an increasing order.

Exercises (2)

4. Write a method that finds the longest subsequence of equal numbers in given `List<int>` and returns the result as new `List<int>`. Write a program to test whether the method works correctly.
5. Write a program that removes from given sequence all negative numbers.
6. Write a program that removes from given sequence all numbers that occur odd number of times.
   Example:

   {4, 2, 2, 5, 2, 3, 2, 3, 1, 5, 2} → {5, 3, 3, 5}

Exercises (3)

7. Write a program that finds in given array of integers (all belonging to the range [0..1000]) how many times each of them occurs.

   Example: array = {3, 4, 4, 2, 3, 3, 4, 3, 2}

   2 → 2 times

   3 → 4 times

   4 → 3 times

8. * The majorant of an array of size N is a value that occurs in it at least N/2 + 1 times. Write a program to find the majorant of given array (if exists). Example:

   {2, 2, 3, 3, 2, 3, 4, 3, 3} → 3

Exercises (4)

9. We are given the following sequence:

$S_1 = N;$

$S_2 = S_1 + 1;$

$S_3 = 2*S_1 + 1;$

$S_4 = S_1 + 2;$

$S_5 = S_2 + 1;$

$S_6 = 2*S_2 + 1;$

$S_7 = S_2 + 2;$

...

Using the `Queue<T>` class write a program to print its first 50 members for given N.

Example: N=2 → 2, 3, 5, 4, 4, 7, 5, 6, 11, 7, 5, 9, 6, ...

Exercises (5)

10. * We are given numbers N and M and the following operations:
    a) N = N+1
    b) N = N+2
    c) N = N*2

    Write a program that finds the shortest sequence of operations from the list above that starts from N and finishes in M. Hint: use a queue.

    - Example: N = 5, M = 16
    - Sequence: 5 → 7 → 8 → 16

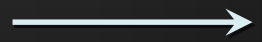
Exercises (6)

11. Implement the data structure linked list. Define a class `ListItem<T>` that has two fields: `value` (of type T) and `NextItem` (of type `ListItem<T>`). Define additionally a class `LinkedList<T>` with a single field `FirstElement` (of type `ListItem<T>`).
12. Implement the ADT stack as auto-resizable array. Resize the capacity on demand (when no space is available to add / insert a new element).
13. Implement the ADT queue as dynamic linked list. Use generics (LinkedQueue<T>) to allow storing different data types in the queue.

Exercises (7)

14. * We are given a labyrinth of size N x N. Some of its cells are empty (0) and some are full (x). We can move from an empty cell to another empty cell if they share common wall. Given a starting position (*) calculate and fill in the array the minimal distance from this position to any other cell in the array. Use "u" for all unreachable cells. Example:

0	0	0	x	0	x
0	x	0	x	0	x
0	*	x	0	x	0
0	x	0	0	0	0
0	0	0	x	x	0
0	0	0	x	0	x



3	4	5	x	u	x
2	x	6	x	u	x
1	*	x	8	x	10
2	x	6	7	8	9
3	4	5	x	x	10
4	5	6	x	u	x

Free Trainings @ Telerik Academy

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com