telerik

Linear Data Structures

Lists, Stacks, Queues

Svetlin Nakov

Telerik Corporation

www.telerik.com

telerik

Table of Contents

telerik

Abstract Data Types

Basic Data Structures

telerik

Abstract Data Types

- An Abstract Data Type (ADT) is a data type together with the operations, whose properties are specified independently of any particular implementation
  - ADT are set of definitions of operations (like the interfaces in C#)
  - Can have several different implementations
  - Different implementations can have different efficiency

telerik

Basic Data Structures

- Linear structures
  - Lists: fixed size and variable size
  - Stacks: LIFO (Last In First Out) structure
  - Queues: FIFO (First In First Out) structure
- Trees
  - Binary, ordered, balanced, etc.
- Dictionaries (maps)
  - Contain pairs (key, value)
  - Hash tables: use hash functions to search/insert

telerik

Lists

Static and Dynamic Implementations

telerik

The List ADT

- 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)

telerik

Static List

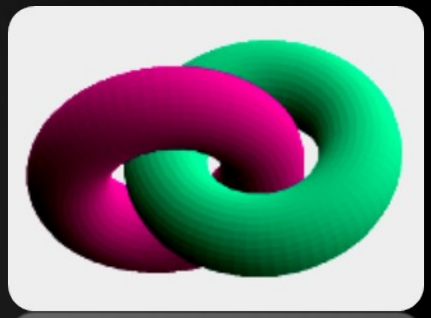
- Implemented by an array
  - Provides direct access by index
  - Has fixed capacity
  - Insertion, deletion and resizing are slow operations

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

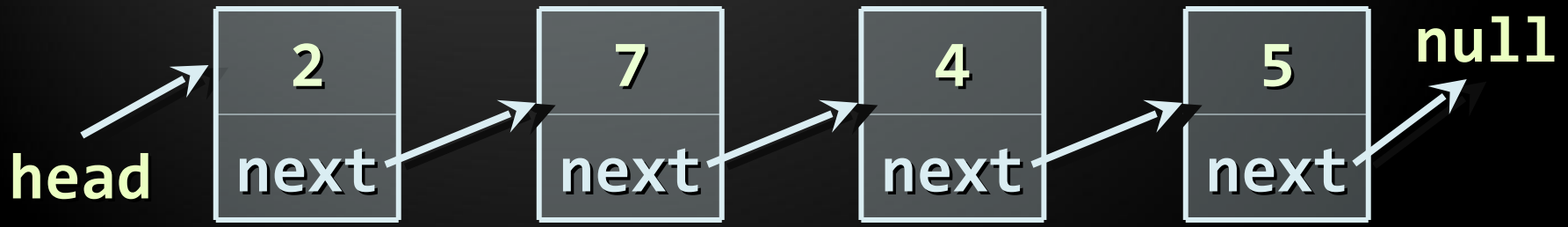
telerik

Linked List

- Dynamic (pointer-based) implementation
- Different forms
  - Singly-linked and doubly-linked
  - Sorted and unsorted



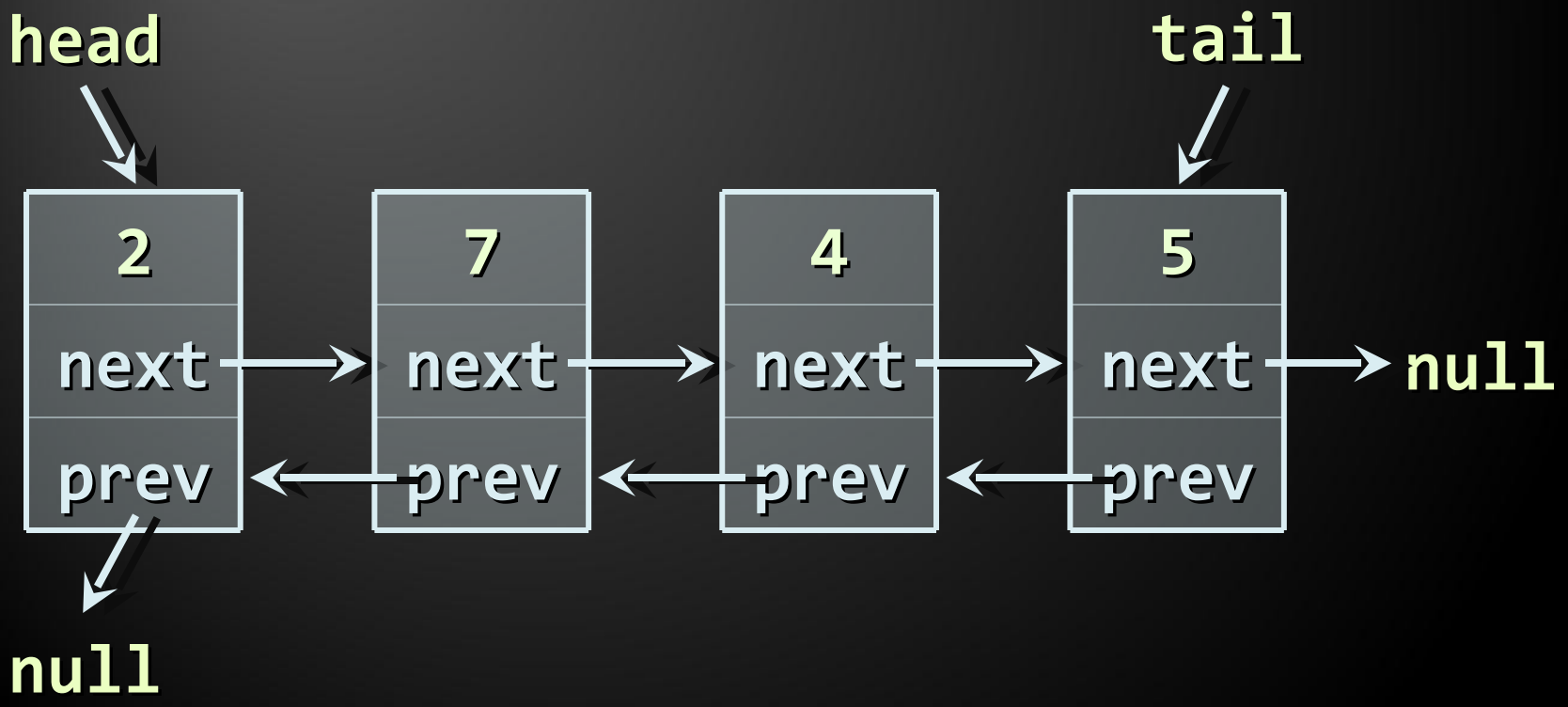
- Singly-linked list
  - Each item has 2 fields: `value` and `next`



telerik

Linked List (2)

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



telerik

The List<T> Class

Auto-Resizable Indexed Lists

telerik

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

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

List<T> – Simple Example

Live Demo

telerik

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

List<T> – Functionality (2)

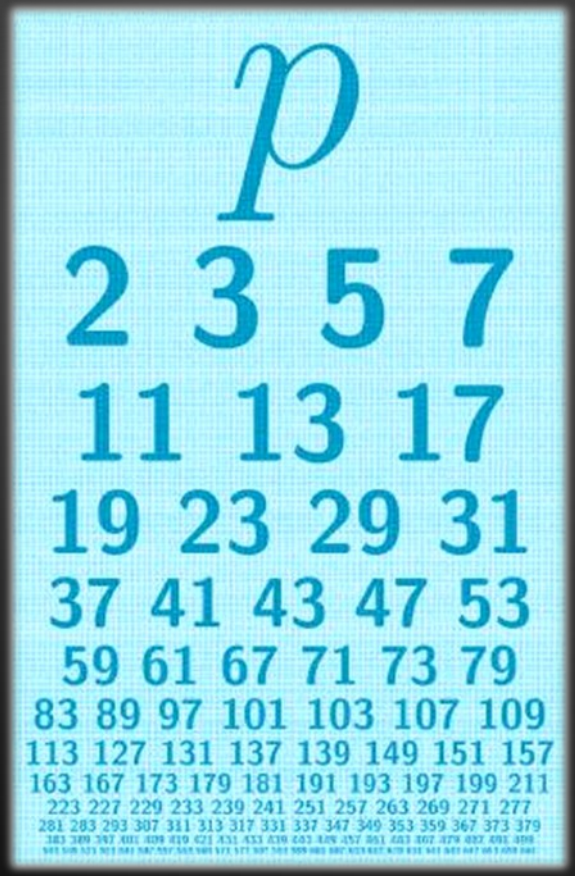
- `IndexOf()` – returns the index of the first occurrence of a value in the list (zero-based)
- `Reverse()` – reverses the order of the elements in the list or a portion of it
- `Sort()` – sorts the elements in the list or a portion of it
- `ToArray()` – converts the elements of the list to an array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

telerik

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);
        }
```

telerik



Primes in an Interval

Live Demo

telerik

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

Union and Intersection

Live Demo

telerik

Stacks

Static and Dynamic Implementation

telerik

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

telerik

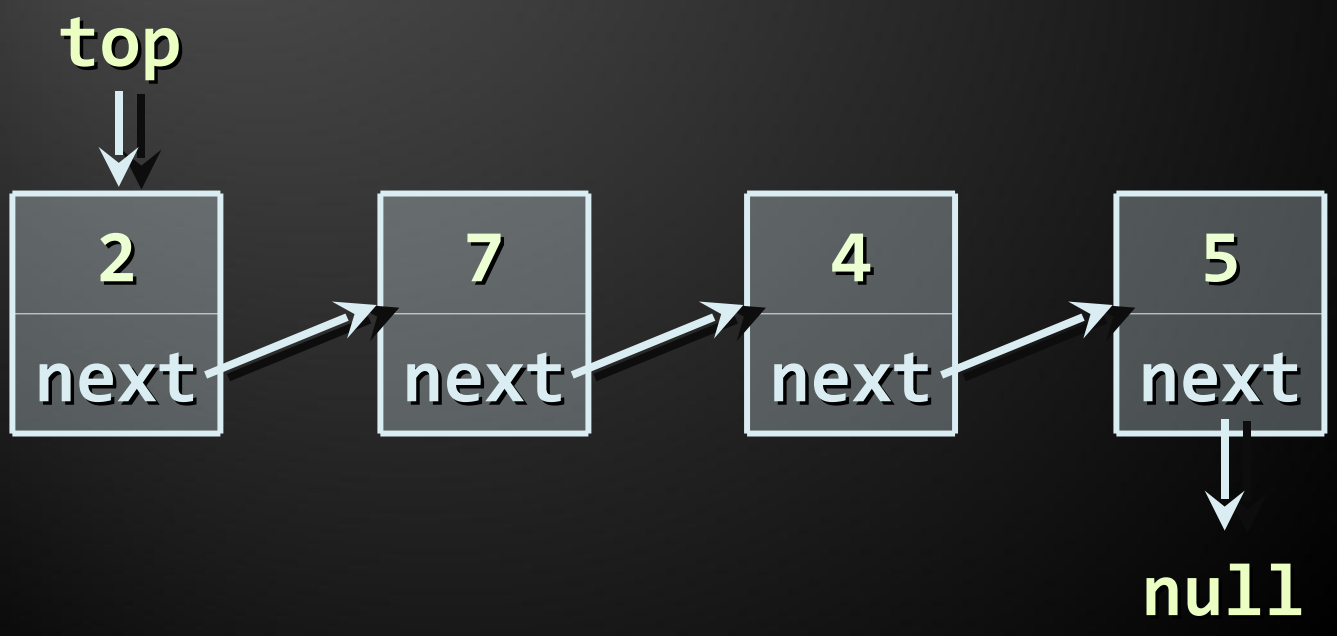
Static Stack

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

telerik

Linked Stack

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



telerik

The Stack<T> Class

The Standard Stack Implementation in .NET

telerik

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. `Stack<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

telerik

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

telerik

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);
    }
}
```

telerik

Stack<T>

Live Demo

telerik

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

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

Matching Brackets

Live Demo

telerik

Queues

Static and Dynamic Implementation

telerik

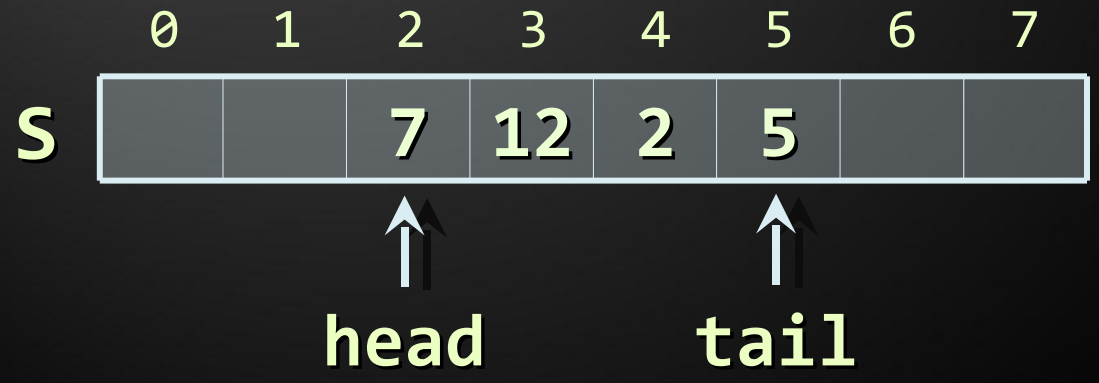
The Queue ADT

- 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

telerik

Static Queue

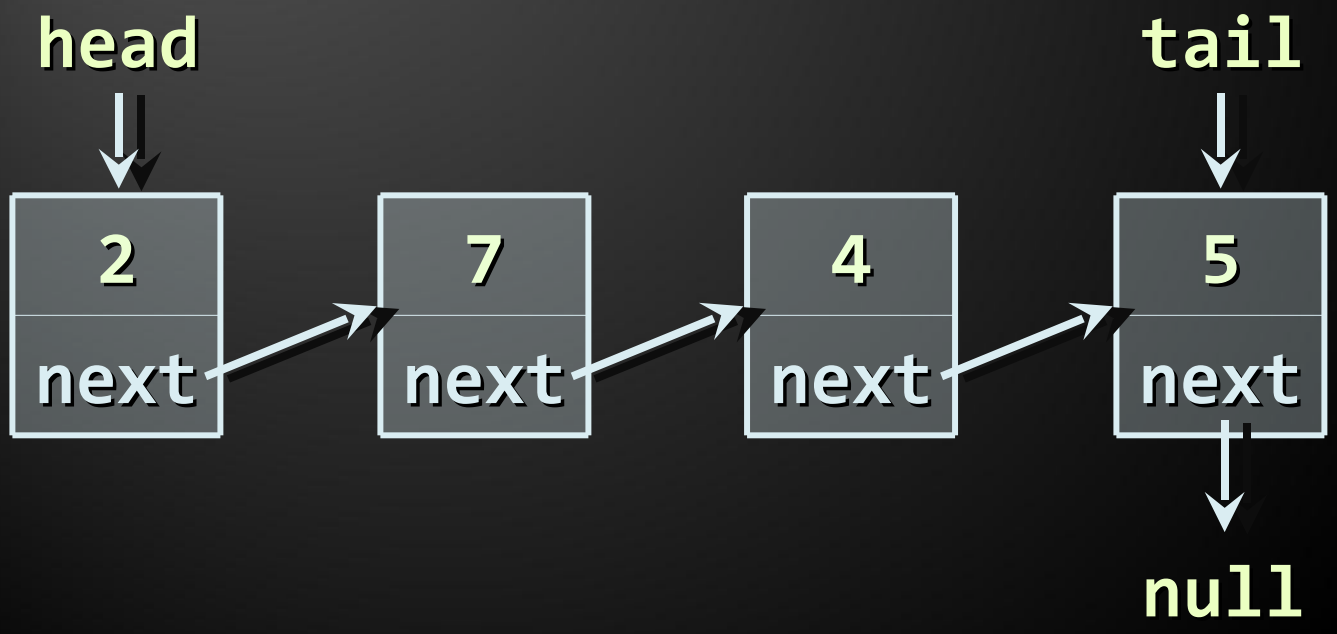
- Static (array-based) implementation
  - Has limited (fixed) capacity
  - Implement as a "circular array"
  - Has `head` and `tail` indices, pointing to the head and the tail of the cyclic queue



telerik

Linked Queue

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



telerik

The Queue<T> Class

Standard Queue Implementation in .NET

telerik

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. `Stack<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

telerik

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

telerik

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

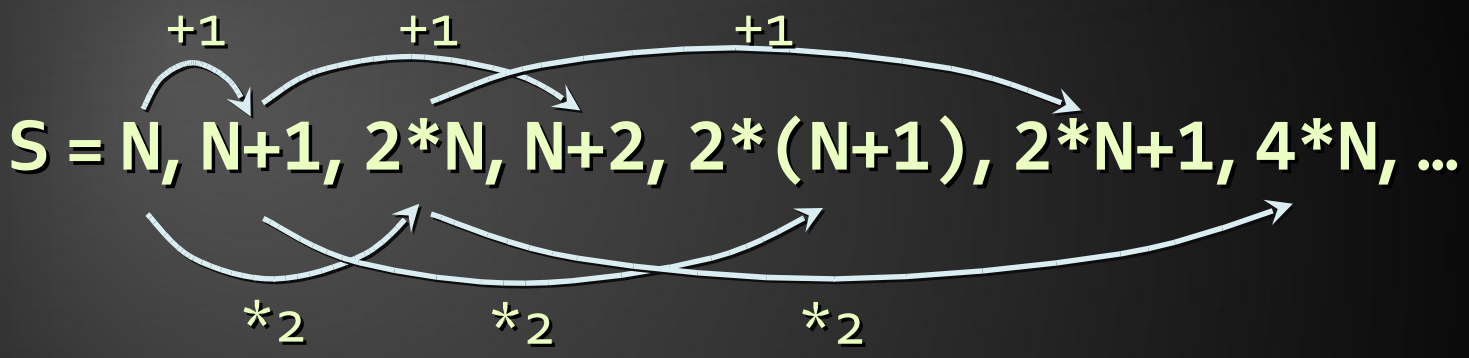
The Queue<T> Class

Live Demo

telerik

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

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);
}
```

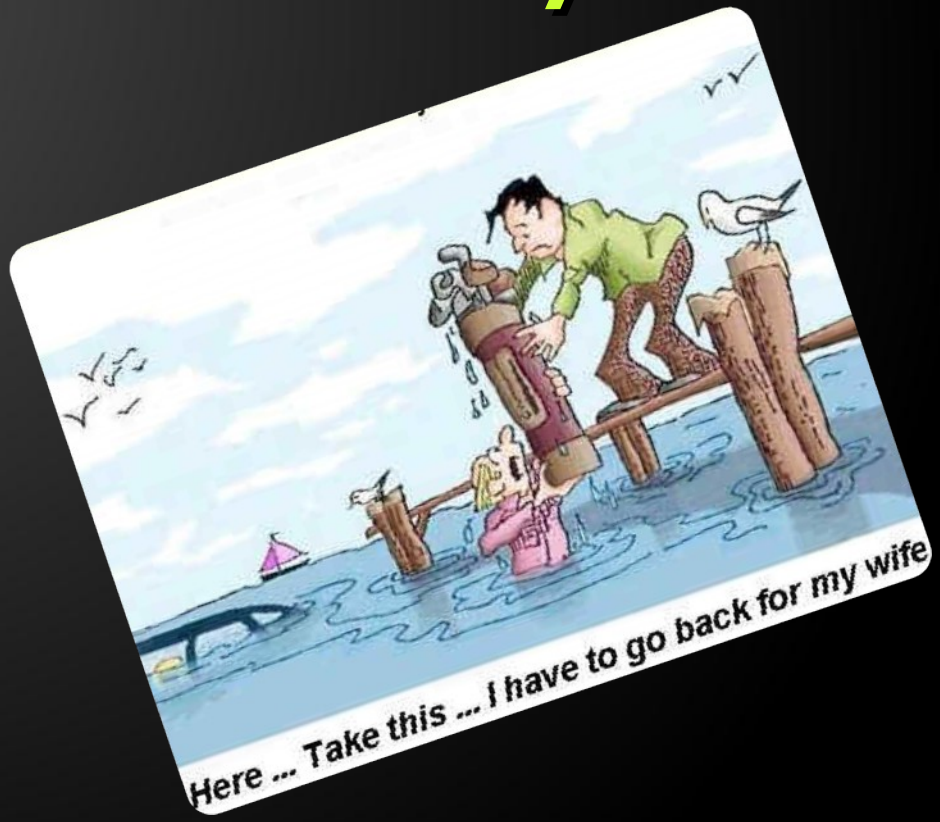
telerik

Sequence N, N+1, 2*N

Live Demo

telerik

Priority Queue



telerik

Priority Queue

- What is a Priority Queue
  - Data type to efficiently support finding the item with the highest priority
  - Basic operations
    - Enqueue(T element)
    - Dequeue
- There is no build-in Priority Queue in .NET
  - Can be easily implemented using `PowerCollections`

Priority Queue Implementation

```
class PriorityQueue<T> where T:IComparable<T>
{
    private OrderedBag<T> bag;
    public int Count
    {
        get { return bag.Count; }
        private set{ }
    }
    public PriorityQueue()
    {
        bag = new OrderedBag<T>();
    }
    public void Enqueue(T element)
    {
        bag.Add(element);
    }
    public T Dequeue()
    {
        var element = bag.GetFirst();
        bag.RemoveFirst();
        return element;
    }
}
```

Necessary to provide comparable elements

telerik

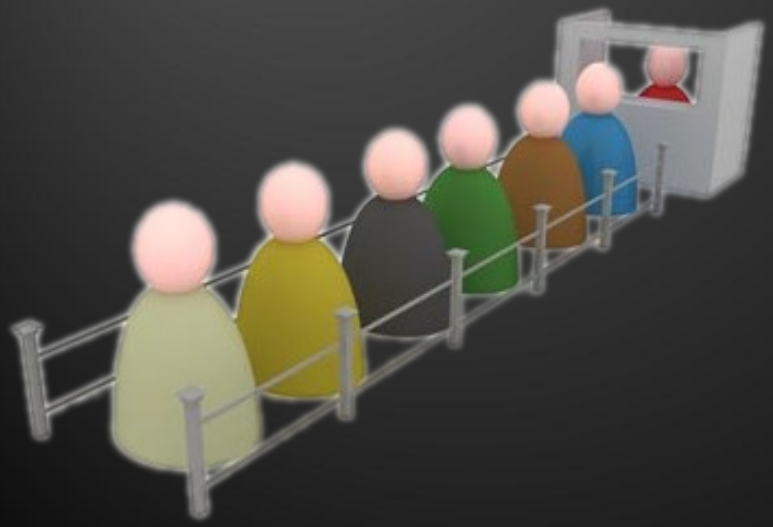
Priority Queue Additional Notes

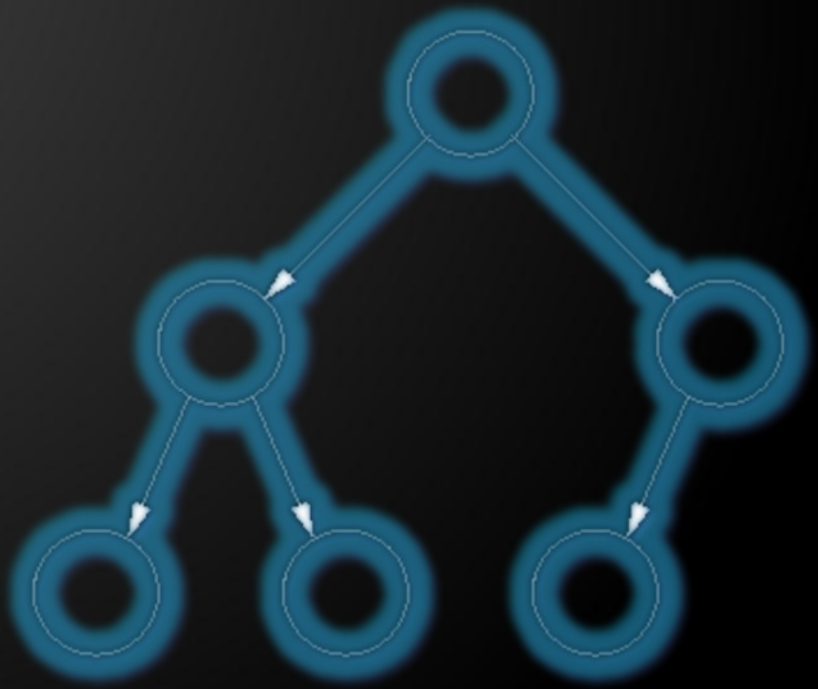
- The generic type is needed to implement `IComparable<T>`
- It is not necessary to use `OrderedBag`
  - Other Data Structures also can be used
- Adding and Removing Element in the Priority Queue is with complexity logN
- Keeps the elements Sorted
  - Always returns the best element that fulfills some condition
    - E.g. the smallest or the biggest element

telerik

Priority Queue

Live Demo





telerik

Summary

- ADT are defined by list of operations independent of their implementation
- The basic linear data structures in the computer programming are:
  - List (static, linked)
    - Implemented by the `List<T>` class in .NET
  - Stack (static, linked)
    - Implemented by the `Stack<T>` class in .NET
  - Queue (static, linked)
    - Implemented by the `Queue<T>` class in .NET
  - Priority Queue
    - Implemented by the `OrderedBag<T>` class

telerik

Linear Data Structures

Questions?

http://academy.telerik.com

telerik

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.

telerik

Exercises (2)

1. 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.
2. Write a program that removes from given sequence all negative numbers.
3. 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}

telerik

Exercises (3)

1. 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

2. * 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

telerik

Exercises (4)

1. 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, ...

telerik

Exercises (5)

1. 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

telerik

Exercises (6)

1. Write a class `Student`, that has three fields: `name` (String), `age`(Integer) and `paidSemesterOnline`(Boolean). When in a queue the students who paid online are with higher priority than those who are about to pay the semester. Write a program which with a given queue of student determine whose turn it is. Hint: use priority queue

telerik

Exercises (6)

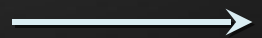
1. Implement the data structure linked list. Define a class `ListItem<T>` that has two fields: `value` (of type T) and `nextItem` (of type `ListItem<T>`). Define additionally a class `LinkedList<T>` with a single field `firstElement` (of type `ListItem<T>`).
2. Implement the ADT stack as auto-resizable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

telerik

Exercises (7)

1. * 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