# Trees and expressions

Dr. Giuseppe Maggiore

Hogeschool Rotterdam
Rotterdam, Netherlands

# Introduction

## Lecture topics

- Recursive definition of binary search trees
- Recursive traversal of trees with pattern matching
- Recursive definition of expressions
- Building an interpreter

```
type BinTree<'T> =
  | Leaf
  | Node of 'T * BinTree<'T> * BinTree<'T>
```

```
let rec add x t =
  match t with
  | Leaf -> Node(x, Leaf, Leaf)
  | Node(y,l,r) when x < y ->
    Node(y,add x l,r)
  | Node(y,l,r) when x > y ->
    Node(y,l,add x r)
  | _ -> t
```

```
let t = Leaf |> add 3 |> add 5 |> add 10 |> add 7 |>
    add 1 |> add 2
```

```
let (!) x = x, x.ToString()
let t = Leaf |> add !3 |> add !5 |> add !10 |> add !7
    |> add !1 |> add !2
```

```
let rec depth t =
  match t with
  | Leaf -> 0
  | Node(y,l,r) ->
    1 + max (depth l) (depth r)
```

```
let rec minElem t =
  match t with
  | Leaf -> None
  | Node(y,Leaf,r) -> Some y
  | Node(y,l,r) -> minElem l
```

```
let rec maxElem t =
  match t with
  | Leaf -> None
  | Node(y,l,Leaf) -> Some y
  | Node(y,l,r) -> maxElem r
```

```
let rec find k t =
  match t with
  | Leaf -> None
  | Node((y,z),l,r) when k < y ->
    find k l
  | Node((y,z),l,r) when k > y ->
    find k r
  | Node((y,z),l,r) -> Some z
```

```
let rec sum t =
  match t with
  | Leaf -> 0
  | Node(x,l,r) ->
    x + sum l + depth r
```

```
let rec interval m M t =
  seq{
    match t with
    | Leaf -> ()
    | Node(x,l,r) when x < m ->
      yield! interval m M r
    | Node(x,l,r) when x > M ->
      yield! interval m M l
    | Node(x,l,r) ->
      yield! interval m M l
      yield x
      yield! interval m M r
  }
```

# Introduction

## Representing expressions

- A special kind of trees is used to represent syntactic expressions
- By folding over the tree we can evaluate the expression tree

```
type Expr =
  | Const of int
  | Sum of Expr * Expr
  | Mul of Expr * Expr
```

```
let (!!) x = Const x
let (.+) e1 e2 = Sum(e1, e2)
let (.*) e1 e2 = Mul(e1, e2)
```

```
let e = !!1 .+ (!!3 .* !!4)
```

```
let rec eval e =
  match e with
  | Const c -> c
  | Sum(e1,e2) -> eval e1 + eval e2
  | Mul(e1,e2) -> eval e1 * eval e2
```

# Introduction

## Conclusions and assignment

- The assignments are on Natschool
- **Restore** the games to a working state
- Hand-in a **printed** report that only contains your **sources** and the associated **documentation**

# Introduction

## Conclusions and assignment

- Any book on the topic will do
- I did write my own (Friendly F#) that I will be loosely following for the course, **but it is absolutely not mandatory or necessary to pass the course**

# Dit is het

The best of luck, and thanks for the attention!