Lists

Lists

- The list is a fundamental data structure in functional programming
- A list having x_1, … x_n as elements is written List(x_1, … x_n)
- Examples:
    - ***val fruit: List[String] = List("apples", "oranges", "pears")***
    - ***val nums: List[Int] = List(1, 2, 3, 4)***
    - ***val id3: List[List[Int]] = List(List(1, 0, 0), List(0, 1, 0), List(0, 0, 1))***
    - ***val empty: List[Nothing] = List()***
- There are two important differences between lists and arrays:
    - Lists are **immutable** – the elements of a list cannot be changed
    - Lists are **recursive**, while arrays are flat
- Like arrays, list are homogeneous: the elements of a list must all have the same type
- The type of a list with elements of type T is written ***scala.List[T]*** or shorter just ***List[T]***

Constructors of Lists

All lists are constructed from:

- The empty list ***Nil***, and
- The construction operation **::** (*cons*): ***x :: xs*** gives a new list with the first element x followed by the elements of xs
- **Convention**: Operators ending in : associate to the right

Operations on Lists

- All operations on lists can be expressed in terms of the following three:
    - ***head*** – the first element of the list
    - ***tail*** – the list composed of all the elements except the first
    - ***isEmpty*** – 'true' if the list is empty, 'false' otherwise
- These operations are defined as methods of objects of type List
- It is also possible to decompose lists with pattern matching:
    - ***Nil*** – the Nil constant
    - ***p :: ps*** – a pattern that matches a list with a head matching p and tail matching ps
    - ***List(p₁, … pₙ)*** – same as p_1 :: … :: p_n :: Nil

Exercise: *Consider the pattern* ***x :: y :: List(xs, ys) :: zs****. What is the condition that describes most accurately the length L of the list it matches?*

L >= 3 *(because zs is a tail and can have none or more elements)*

Sorting Lists

- Suppose we want to sort a list of numbers in ascending order:
    - One way to sort the list List(7, 3, 9, 2) is to sort the tail List(3, 9, 2) to obtain List(2, 3, 9)
    - The next step is to insert the head 7 in the right place to obtain the result list List(2, 3, 7, 9)
- This idea describes Insertion Sort:

```
def isort(xs: List[Int]): List[Int] = xs match
   case List() => List()
   case y :: ys => insert(y, isort(ys))

def insert(x: Int, xs: List[Int]): List[Int] = xs match
   case List() => List(x)
   case y :: ys =>
        if x < y then x :: xs
        else y :: insert(x, ys)
```

Exercise*: What is the worst-case complexity of insertion sort relative to the length of the input list N? –* ***proportional to N * N***