II.3 Data types

A data type (or simply type) consists of a set of values and a set of operators:

Type = a set of values + a set of operators

Basic data types

This part explains all the basic data types available in SOFL.

The outline:

- The numeric types
- The character type
- The enumeration types
- The boolean type
- An example of using basic types

Numeric types

The numeric types include:

nat0 -- {0, 1, 2, 3, …}   naturals containing zero.
nat -- {1, 2, 3, …}   naturals
int -- {…, -2, -1, 0, 1, 2, …} integers
real -- {…, -2.5, -1.4, 0.0, 1.4, 2.5, …}
real numbers

The operations on the numeric types are given on the next slide.

Operator	Name	Type
- x	Unary minus	real --> real
abs(x)	Absolute value	real --> real
floor(x)	Floor	real --> int
x + y	Addition	real * real --> real
x - y	Subtraction	real * real --> real
x * y	Multiplication	real * real --> real
x / y	Division	real * real --> real
x div y	Integer division	int * int --> int
x rem y	Remainder	int * int --> nat0
x mod y	Modulus	nat0 * nat0 --> nat0
x ** y	Power	real * real --> real

Examples: let x = 9, y = 4.5, z = 3.14, a = - 4, b = 3.

Then

- z = - 3.14

abs(a) = 4

floor(y) = 4

x + z = 12.14

x - y = 4.5

a * b = - 12

x / y = 2.0

a div b = -2

a rem b = 2 (quotient = -2)

x mod b = 0

x ** b = 729

The relational operators on numeric types are:

Operator	Name	Type
x < y	Less than	real * real --> bool
x > y	Greater than	real * real --> bool
x <= y	Less or equal	real * real --> bool
x >= y	Greater or equal	real * real --> bool
x < y < z	Less-between	real * real * real -->bool
x <= y <= z	Less-equal-between	real * real * real --> bool
x >= y >= z	greater-equal-between	real * real * real --> bool
x = y	Equal	real * real --> bool
x <> y	Unequal	real * real --> bool

Character type

char

A value of char type: 'x'

Examples:

'a' 'B' '|' ')' ':' '@' '7'

All the characters:

English letters:

a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Other characters:

, . : ; * + - / _ ~| ¥ ( ) [ ] { } @ ^` ' & % $ # " ! < > = ?

Newline

White space

Two characters can only be compared to see if they are the same (=) or different (<>).

Enumeration types

An enumeration type is a finite set of special values, usually with the feature of describing a systematic phenomena.

For example:

Week = {<Monday>, <Tuesday>,
<Wednesday>, <Thursday>,
<Friday>, <Saturday>, <Sunday>}

Except that two values of an enumeration type can be compared to be the same (=) or different (<>), there is no other operator on the enumeration type.

If we declare a variable weekday with the type Week as

```
 weekday: Week;
```

then the variable can take any value of the type, that is, weekday can take <Monday>, <Tuesday>, <Wednesday>, and so on.

```
<Tuesday> = <Tuesday> <=> true
<Tuesday> <> < Wednesday > <=> true
```

Boolean type

bool = {true, false}

Operator	Name
and	and
or	or
not	not
=>	implies
<=>	is equivalent to

These operators also apply to undefined value nil.

An example of using basic types

A simple process telling fares of railway tickets
for different kinds of passengers:

```
process Tell_Fare(passenger: {<STUDENT>,
     <ORDINARY>, <PENSIONER>}) fare: real
ext rd normal_fare: real
post fare = case passenger of
 <STUDENT> —> normal_fare - 0.25 * normal_fare;
 <ORDINARY> —> normal_fare;
<PENSIONER> —> normal_fare - 0.30 * normal_fare
                  end_case
end_process;
```

Class exercise 4

Assume that the courses to teach on weekdays are: "Software Engineering" on Monday, "Program Design" on Tuesday, "Discrete Mathematics" on Wednesday, "Programming Language" on Thursday, and "Formal Engineering Methods" on Friday. Write a formal specification for the process that gives the corresponding course title for an input weekday.

(Hint: define a type Course as an enumeration type)