An extensible programming language

translation

Chapter 1. Introduction to XL

is

while

1.1. Two basic examples

1.1.1. Hello World

Hello World

```
use XL.CONSOLE.TEXT_IO
print "Hello World"
```

XL.CONSOLE.TEXT_IO print

use

print

```
                                                                    ->
        is
```

```
color "white"
milkyway 10000
rotatez -23
earth 400
hello_world 440

milkyway R ->
// ----------------------------------------------------------------------------
//    Draw the Milky Way
// ----------------------------------------------------------------------------
    locally
        texture_wrap true, true
        texture_transform {scale 5, 5, 5}
        texture "milkyway.jpg"
        rotatey 0.02 * page_time + 100
        scale 1, -1, 1
        sphere R


earth R ->
// ----------------------------------------------------------------------------
//    Draw Earth
// ----------------------------------------------------------------------------
    locally
        texture "earth.bmp"
        texture_wrap true, true
        rotatey 5 * page_time + 250
        sphere 0, 0, 0, R


hello_world R ->
// ----------------------------------------------------------------------------
//    Draw "hello world" text
// ----------------------------------------------------------------------------
    locally
        frame_texture 1900, 600,
            color 1, 1, 1, 1
            reset_transform
            // If font Arial Unicode installed, it will be used.
            // Otherwise, unifont will be used (unifont is packaged
            // with Tao presentations).
            font "Arial Unicode MS", "unifont", 72
            move_to -800, -9, 0
            text "Hello World! or Καλημέρα κ□σμε; or □□□□□ □□"
        rotatey -11 * page_time + 180
        color 20% , 20% , 20% , 70%
        sphere 0, 0, 0, R - 30
        color 100% , 90% , 20% , 90%
        sphere 0, 0, 0, R
```

1.1.2. Factorial

```
use IO = XL.CONSOLE.TEXT_IO

0! is 1
N! is N * (N-1)!

for I in 1..5 loop
    IO.print "The factorial of ", I, " is ", I!
```

use IO print IO.print

0

```
0! is 1
```

(N-1)!

```
N! is N * (N-1)!
```

-3!

```
0!             is 1
N!  when N > 0  is N * (N-1)!
```

-3!

1.2. One operator to rule them all

is

is

Pattern is Implementation Pattern X+Y Implementation

Add X, Y

```
pi              is      3.1415926
```

```
funny_words     is      "xylophage", "zygomatic", "barfitude"
identity_matrix is
    [ [1, 0, 0],
      [0, 1, 0],
      [0, 0, 1] ]
```

```
abs X:number    is      if X < 0 then -X else X
```

```
X ≠ Y           is      (not X = Y)
```

```
0!              is      1
N!  when N > 0  is      N * (N-1)!
```

```
A in B..C       is      A >= B and A <= C
```

```
X * 1           is      X
X + 0           is      X
```

```
loop Body       is      { Body; loop Body }     // Define an infnite loop
```

```
complex         is      polar or cartesian
cartesian       is      type cartesian(re:number, im:number)
polar           is      type polar(mod:number, arg:number)
```

cartesian

cartesian cartesian(1,5)

```
adder N         is      { lambda X is N + X }
add3            is      ( adder 3 )

 // This will compute 8
 add3 5
```

lambda X \X X is 0

X \X is 0

35 "ABC"

lambda X is Y

adder N is (X is N + X)

lambda

```
my_map is
    0 is 4
    1 is 0
    8 is "World"
    27 is 32
    lambda N when N < 45 is N + 1

// The following is "World"
my_map 8

// The following is 32
my_map[27]

// The following is 45
my_map (44)
```

std::map

```
// An (inefficient) implementation of a generic 1-based array type
array[1] of T is type
    Value : T
    1 is Value
array[N] of T when N > 1 is type
    Head  : array[N-1] of T
    Tail  : T
    lambda I when I<N is Head[I]
    lambda I when I=N is Tail

A : array[5] of integer
for I in 1..5 loop
    A[I] := I * I
```

```
min X, Y    is { Z is min Y; if X < Z then X else Z }
min X       is X

// Computes 4
min 7, 42, 20, 8, 4, 5, 30
```

is

1.3. The standard library

1.3.1. Usual programming features

```
if [[true]]  then TrueClause else FalseClause   is TrueClause   ①
if [[false]] then TrueClause else FalseClause   is FalseClause
if [[true]]  then TrueClause                    is TrueClause
if [[false]] then TrueClause                    is false
```

① [[true]] [[false]]

foo true

is ... true foo [[true]] is ...

true

while

```
while Condition loop Body is
    if Condition then
        Body
        while Condition loop Body
```

if while

```
while N <> 1 loop
    if N mod 2 = 0 then
        N /= 2
    else
        N := N * 3 + 1
    print N
```

1.3.2. The next natural evolutionary step

PRINT WriteLn printf

pthread

x+1 x/3

X+1

<=>

<=> -1 0 1

```
syntax { INFIX 290 <=> }
X <=> Y     when X < Y  is -1
X <=> Y     when X = Y  is  0
X <=> Y     when X > Y  is  1
```

X*0 X*1 X+0

```
X*0     is 0
X*1     is X
X+0     is X
```

X*Y+Z

```
X*Y+Z   is FusedMultiplyAdd(X,Y,Z)
```

1.3.3. Benefits of moving features to a library

XL

use XL.MATH.COMPLEX

use

1.3.4. The case of text input / output operations

printf

print
...

```
write X:text            as mayfail      is ... ①
write X:integer         as mayfail      is ...
write X:real            as mayfail      is ...
write X:character       as mayfail      is ...
write [[true]]          as mayfail      is { write "true"  } ②
write [[false]]         as mayfail      is { write "false" }
write Head, Rest        as mayfail      is { write Head; write Rest }

print                   as mayfail      is { write SOME_NEWLINE_CHARACTER }
print Items             as mayfail      is { write Items; print }
```

① nil or error

② [[true]]
true

print
printf
print

```
print "The value of X is ", X, " and the value of Y is ", Y
```

print Items

```
Items   is "The value of X is ", X, " and the value of Y is ", Y`
```

write write Head, Rest

```
Head    is "The value of X is "
Rest    is X, " and the value of Y is ", Y
```

write Head write X:text
write Rest write Head, Rest

```
Head    is X
Rest    is " and the value of Y is ", Y
```

write Head write
X X write X:integer

write Rest

```
Head    is " and the value of Y is "
Rest    is Y
```

write Head write X:text write Rest
write Y

X Y integer

```
print "The sum is ", X+Y, " and the difference is ", X-Y
```

```
print A:text, B:integer, C:text, D:integer is
    write A, B, C, D
    print

write A:text, B:integer, C:text, D:integer is
    write A
    write B, C, D

write B:integer, C:text, D:integer is
    write B
    write C, D

write C:text, D:integer is
    write C
    write D
```

print

write

write write
Items complex
write

```
write Z:complex     is write "(", Z.Re, ";", Z.Im, ")"
```

iostream

print

single_thread

print

```
locked_print Items is
    single_thread
         print Items
```

iostream

1.4. Efficient translation

while

```
while Condition loop Body is
    if Condition then
        Body
        while Condition loop Body
```

```
while N <> 1 loop
    if N mod 2 = 0 then N /= 2 else N := N * 3 + 1
```

`while` `while Condition loop Body`

```
Conditions is N <> 1
Body is
   if N mod 2 = 0 then N /= 2 else N := N * 3 + 1
```

`while Condition loop Body`

```
    if Condition then
        Body
        while Condition loop Body
```

`while`

`builtin`

```
X:integer + Y:integer as integer    is builtin Add
```

`is` `X+Y` `X` `Y` `integer` `integer` `as integer` `builtin Add` `Add`

1.5. Adding complex features

1.5.1. Reactive programming in Tao3D

```
locally
    rotate_z -6 * minutes
    rectangle 0, 100, 15, 250

locally
    rotate_z -30 * hours
    rectangle 0, 50, 15, 150

locally
    color "red"
    rotate_z -6 * seconds
    rectangle 0, 80, 10, 200
```

locally minutes
hours seconds
hours

locally

locally

locally

seconds
seconds locally

locally

1.5.2. Declarative programming in Tao3D

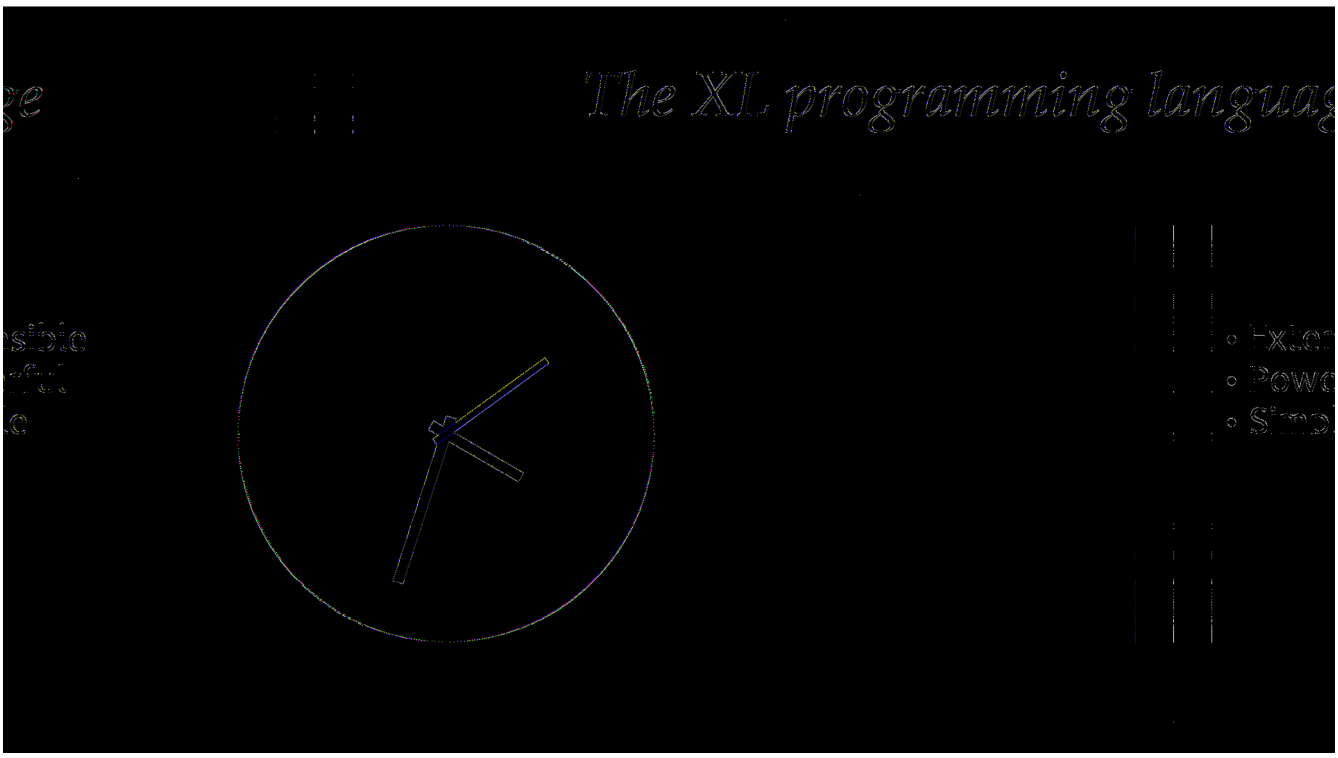
`import` `use`

```
import Slides

slide "The XL programming language",
    * "Extensible"
    * "Powerful"
    * "Simple"
```

`clock`

```
import Slides

clock is
    locally
        line_color "blue"
        color "lightgray"
        circle 0, 0, 300

    locally
        rotate_z -6 * minutes
        rectangle 0, 100, 15, 250

    locally
        rotate_z -30 * hours
        rectangle 0, 50, 15, 150

    locally
        color "red"
        rotate_z -6 * seconds
        rectangle 0, 80, 10, 200

slide "The XL programming language",
    * "Extensible"
    * "Powerful"
    * "Simple"
    anchor
        translate_x 600
        clock
```

```
theme_font Theme, Master, "title" is font "Palatino", 80, italic
```



1.5.3. Distributed programming with ELFE

tell

ask

invoke

reply

reply

invoke

```
WORKER_1 is "pi2.local"
WORKER_2 is "pi.local"

invoke WORKER_1,
   every 1.1s,
        rasp1_temp is
            ask WORKER_2,
                temperature
        send_temps rasp1_temp, temperature

   send_temps T1:real, T2:real is
       if abs(T1-T2) > 2.0 then
           reply
               show_temps T1, T2

show_temps T1:real, T2:real is
    print "Temperature on pi is ", T1, " and on pi2 ", T2, ". "
    if T1>T2 then
        print "Pi is hotter by ", T1-T2, " degrees"
    else
        print "Pi2 is hotter by ", T2-T1, " degrees"
```

WORKER_1 WORKER_2

- T1-T2 send_temps WORKER_1
- T1 WORKER_2 ask rasp1_temp
- reply T1 T2 WORKER_1
- reply show_temps

WORKER_1 WORKER_2

temperature

```
invoke "pi.local",
    min   is 100.0
    max   is 0.0
    sum   is 0.0
    count is 0

    compute_stats T:real is
        min   := min(T, min)
        max   := max(T, max)
        sum   := sum + T
        count := count + 1
        reply
            report_stats count, T, min, max, sum/count

    every 2.5s,
        compute_stats temperature

report_stats Count, T, Min, Max, Avg is
    print "Sample ", Count, " T=", T, " ",
          "Min=", Min, " Max=", Max, " Avg=", Avg
```

min max sum count

min : real :=

100 min is 100.0 min is 100.0

pi.local

pi2.local -remote

```
% xl -remote
```

```
% xl 7-two-hops.xl
```

Chapter 2. XL syntax

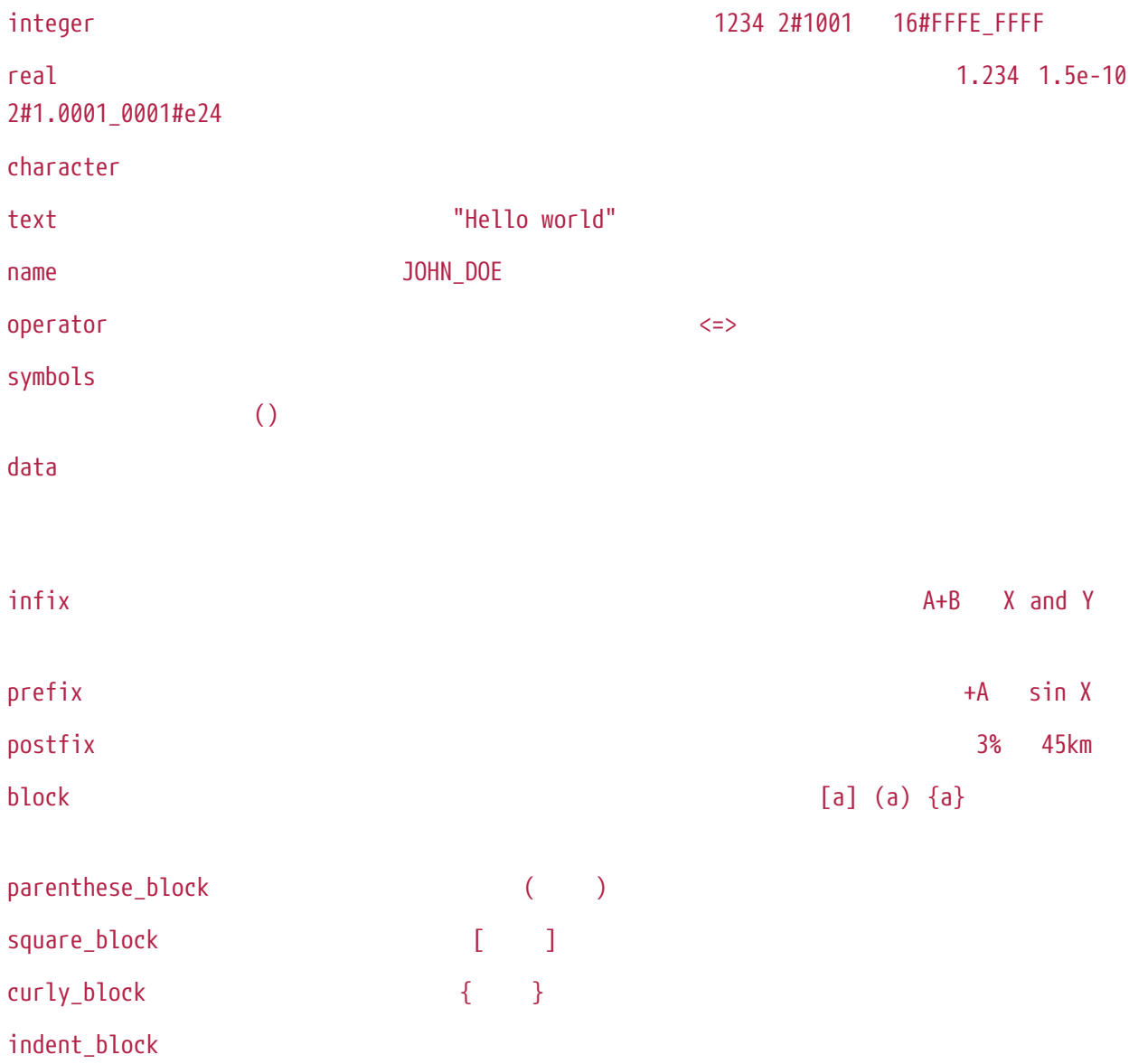
2.1. Homoiconic representation of programs

2.1.1. Why Lisp remains so strong to this day

`1+2`

`(+ 1 2)`

2.1.2. The XL parse tree



```
if X < 0 then
   print "The value of ", X, " is negative"
   X := -X
```

program.xl

```
% xl -parse program.xl -style debug -show
(infixthen
 (prefix
  if
  (infix<
   X
   0))
 (block indent
  (infix CR
   (prefix
    print
    (infix,
     "The value of "
     (infix,
      X
      " is negative"
     )))
   (infix:=
    X
    (prefix
     -
     X
    )))))
```

integer

real

bits

```
bits 16#FF_00_FF_00_FF_FF_00_FF_00
bits "image.png"`
```

()

[A,B,C]

2.2. Leaf nodes

42 3.5 "ABC"
'a' ABC ->

2.2.1. Numbers

0123456789
0 42

integer

real

_ 1_000_000

_1 2_

3__0

1_000_000 04_92_98_05_55

8#76 62

A Z a z A f Z

16#FF 16#ff

255

bits 64#SGVsbG8h

Hello!

.

0.2 2.0 .2 .

.. 2..3 .. 2 3

e E

1e3 1e-3

0.001

2#1e8

e E 16#FF#e2

9.99e99

+ -

-2 - 2

16#FF_FF 65535

2.3.1 . 2.3 1

2.30.1

integer real

real

real

2.2.2. Symbols

MyName A22

_

A_2 A__2 _A

_2 étalon

⇒A2

JOE_DALTON JoeDalton

V v

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

<= <=> 1 <=> 2

(1 <= (> 2))

syntax

2.2.3. Text

"Hello World" 'ABC'

"He said ""Hello""" He said "Hello"

<< >>

```
MyLongText is <<
   This is a multi-line text
   that contains several lines
>>
```

HTML END_HTML

```
MyHTML is HTML
    <p>This is some HTML text here</p>
END_HTML
```

RATIONALE

2.3. Inner nodes

2.3.1. Indentation and off-side rule

begin end { } { }

```
loop { Eat; Pray; Love }
loop
    Eat
    Pray
    Love
```

A;B

; A B

{A} A

-show

```
% xl -parse loop.xl -style debug -show
(prefix
 loop
 (block indent
  (infix CR
   Eat
   (infix CR
    Pray
    Love
   ))))
```

Pray

```
loop
    Eat
   Pray
    Love
```

2.3.2. Operator precedence and associativity

INFIX PREFIX POSTFIX

INFIX

* / + - X+Y*Z

X+(Y*Z)

```
        21      -> is has
        310     + -
        320     * / mod rem
```

+ * X * Y * Z (X * Y) * Z
is X is Y is
Z X is (Y is Z)

INFIX PREFIX POSTFIX BLOCK

COMMENT TEXT

SYNTAX

BINARY bits

bits 16#000102030405060708090A0B0C0D0E0F bits
"image.png"

NEWLINE

STATEMENT

if loop
+ *

DEFAULT

FUNCTION

sin X

FUNCTION

2.3.3. Delimiters

/* */ //

<< >>

INDENT UNINDENT

{ }
( ) [ ]

2.3.4. Child syntax

C extern
;

```
extern real sqrt(real);
```

2.3.5. Extending the syntax

syntax

<=>
<=

```
syntax
    INFIX 290 <=>
```

use

2.4. Making the syntax easy for humans

2.4.1. Expression vs. statement

```
print sin X, cos Y
```

print (sin(X),cos(Y)) print
sin X cos Y

print sin
print(sin(X, cos(Y))

sin X

print
{ } () []

is

```
debug X     is write "X=", X
expm1 X     is exp X - 1
double X    is X; X
```

(exp X) -1 exp (X-1)
write ("X=", X)
double X X ;
is

is

```
debug X     is { write "X=", X } ①
expm1 X     is ( exp X - 1 )  ②
double X    is { X; X } ③
```

① write

② exp

③ double X

is type X
foo(A,B,C) foo A-1

2.4.2. infix vs. prefix

```
write -A    // write (-A)
B - A       // (B - A)
```

Chapter 3. XL program evaluation

3.1. Execution phases

3.1.1. Execution context

CONTEXT0 CONTEXT1

HIDDEN0 HIDDEN1

HIDDEN
CONTEXT0 HIDDEN0

3.1.2. Parsing phase

syntax syntax

use syntax
use

extern

bits BINARY

use
use

RATIONALE use
syntax use
XL.CONSOLE.TEXT_IO "xl/console/text_io.xs"
use
use "path"

3.1.3. Sequences

```
loop { print "Hello World" }
```

NEWLINE ; ,

```
print "One"; print "Two"
print "Three"
```

use

```
use XL.MATH.COMPLEX
```

syntax

```
syntax { INFIX 290 <=> }
```

is : as
:= :

```
pi is 3.1415                 // Definition of 'pi'
e as real is 2.71828         // Typed definition of 'e'
Count : integer              // Variable declaration of 'Count'
byte_size X as integer       // Function declaration of 'byte_size X'
Remaining : integer := 100   // Variable initialization of 'Remaining'
```

```
print "This is a statement"
```

```
pi is 3.14
circumference 5.3
circumference Radius:real is 2 * pi * Radius
```

`pi` `circumference` `Radius:real` `circumference 5.3`

`Radius`

`Radius` `real` `Radius`

`circumference 5.3`

3.1.4. Declaration phase

```
CONTEXT1 is
    pi is 3.14
    circumference Radius:real is 2 * pi * Radius
    CONTEXT0
    HIDDEN0
```

`pi` `circumference` `Radius` `:=`

circumference

is as :

: as

is

3.1.5. Evaluation phase

circumference 5.3 pi

circumference Radius:real 5.3

circumference Radius:real

```
CONTEXT2 is
    Radius:real := 5.3
    CONTEXT1
    HIDDEN1
HIDDEN1 is CONTEXT1
```

Radius

Redius:real Radius := "Hello"

circumference "Hello" real

:=

is X : T

Radius :=

2 * pi * Radius

circumference 5.3 2 * pi * Radius

Radius is 5.3

3.2. Expression evaluation

circumference Radius:real
2 * pi * Radius circumference 5.3

2 * pi * Radius
X * Y * Z
real
X:real * Y:real as real integer
X:integer * Y:integer
...

```
X:integer * Y:integer   as integer  is ...
X:real * Y:real         as real     is ...
```

* 2 * pi * Radius (2 * pi) * Radius
X 2 * pi Y Radius
integer real

2 * pi
pi 3.14 pi
2 * 3.14

2 * 3.14 X:real * Y:real 2 integer real
X:integer * Y:integer 3.14 real integer

```
X:integer as real     is builtin IntegerToReal
```

integer 2 real

real integer 2

X 2

```
CONTEXT3 is
    X:integer := 2
    CONTEXT2
    HIDDEN2
HIDDEN2 is CONTEXT2
```

2.0 X:real * Y:real as real real X

```
CONTEXT4 is
    X:real := 2.0
    Y:real := 3.14
    CONTEXT2
    HIDDEN2
```

real 6.28 Radius

```
CONTEXT5 is
    X:real := 6.28 // from 2 * pi
    Y:real :=5.3  // from Radius
    CONTEXT2
    HIDDEN2
```

real 33.284 circumference 5.3

real integer real

```
X:integer as real when X >= -2^53 and X < 2^53 is ...
```

3.3. Pattern matching

```
log X:real when X > 0.0 is ...
```

log 1.25

log 1.25 log

1.25 X real 1.25

X:real

X > 0.0 X is 1.25

3.3.1. Name definitions

Declaration	Matched by	Not matched by
pi is 3.14	pi	ip 3.14

+

3.3.2. Wildcards

New

Old New=Old

true

Declaration	Matched by	Not matched by
X+Y	2+"A"	2-3 +3 3+
N+N	3+3 A+B A=B	3-3 3+4

lambda

Declaration	Matched by	Not matched by
\N		

+

3.3.3. Type annotations

: as

Top-level pattern	Matched by	Not matched by
X:integer	X	2 'X'
seconds as integer	seconds	2 "seconds"

Parameter pattern	Matched by	Not matched by
X:integer	42	X integer
seconds as integer	42	X integer

: as

is

```
X:real + Y:real as real is ...
```

```
circle (Radius:real, CenterX:real, CenterY:real) as circle
C : circle := circle(Radius := 3.5, CenterX := 6.5, CenterY := 3.3)

picture is type picture
    Width  : size
    Height : size
    Buffer : buffer
P : picture is picture
    Width  is 640
    Height is 480
    Buffer is my_buffer
```

3.3.4. Function (prefix) definitions

```
sin X
```

Pattern	Matched by	Not matched by
`sin X`	`sin (2.27 + A)`	`cos 3.27`
`+X:real`	`+2.27`	`+"A" -3.1 1+1`

3.3.5. Postfix definitions

```
X%
```

Pattern	Matched by	Not matched by
`X%`	`2.27% "A"%`	`%3 3%2`
`X km`	`2.27 km`	`km 3 1 km 3`

3.3.6. Infix definitions

Pattern	Matched by	Not matched by
X:real+Y:real	3.5+2.9	3+2 3.5-2.9
X and Y	N and 3	N or 3

3.3.7. Argument splitting

Pattern	Matched by	Not matched by
write X,Y	write Items Items is "A","B"	write Items Items is "A"+"B" wrote 0,1
write X%	write Items Items is 2%	write Items Items is 2!
write -X	write Items Items is -2	write Items Items is +2

,

```
write Head, Tail is write Head; write Tail
```

write 1, 2, 3 Head is 1 Tail is 2,3
write Tail
Tail Head is 2
Tail is 3

3.3.8. Conditional patterns

Pattern when Condition

true

Pattern	Matched by	Not matched by
log X when X > 0	log 3.5	log(-3.5)

true

```
log X when X > 0 is ...
log "Logging an error"        // Will not match the definition above
```

3.3.9. Literal constants

integer 0 real 3.5 text "ABC"

Pattern = Value Pattern

Value

Pattern	Matched by	Not matched by
0!	N! N=0	N! N<>0

0! is 1

```
digits is
    0 is "Zero"
    1 is "One"
```

3.3.10. Metabox values

[[true]]

Pattern = Value Pattern

Value

Pattern	Matched by	Not matched by
[[true]]	true not false	"true" 1

and

```
[[true]]  and X   is X
[[false]] and X   is false
```

true false A and B

```
true  and X       is X
false and X       is false
```

3.3.11. Blocks

Definition	Matched by	Not matched by
(X+Y)*(X-Y) is X^2-Y^2	[A+3]*[A-3]	(A+3)*(A-4)

[A:integer] 2 (2) {2}

```
has_parentheses B:block when B.opening = "(" and B.closing = ")"  is true
has_parentheses B:block                                           is false
```

STYLE

```
circle CenterX:real, CenterY:real, Radius:real is ...
```

circle circle

```
circle (CenterX:real, CenterY:real, Radius:real) as circle is ...
C : circle := circle(0.3, 2.6, 4.0)
```

3.3.12. Scope pattern matching

```
circle(Radius:real, CenterX:real, CenterY:real) as circle ①
C1 : circle := circle(3.5, 2.6, 3.2) ②
C2 : circle := circle(CenterX is 0.0; CenterY is 1.5; Radius is 2.4) ③
C3 : circle := circle ④
    Radius  is 1.5
    CenterX is 3.5
    CenterY is 2.4
```

①

②

③

;

,

is

is

④

```
person is type person
    first_name : text
    middle_name: text
    last_name  : text
    birthdate  : date
    address    : address
JohnDoe : person := person
    last_name   is "Doe"
    first_name  is "John"
    middle_name is "W"
    birthdate   is date { Month is December; Day is 5; Year is 1968 }
    address     is address
        city    is "New-York"
        street  is "42nd"
        no      is 42
        zip     is 00002
```

3.3.13. Pattern-matching scope

```
foo T:text, A:real is
    print "T=", T, " A=", A
foo "Hello", 2.5
```

foo

```
CONTEXT1 is
    T : text is "Hello"
    A : real is 2.5
```

complex complex

```
complex is type complex(Re:real, Im:real) ①
Z1:complex + Z2.complex as complex is complex(Z1.Re+Z2.Re, Z1.Im+Z2.Im) ②
Z:complex := complex(1.3, 4.5) + complex(6.3, 2.5) ③
```

① complex
type

② Z1.Re Re Z1

③ complex Z1 Z2
Z1.Re Re
Z1.Re 1.3 Z2.Im 2.5

3.4. Overloading

X*Y
integer real

```
X:integer * Y:integer as integer        is ...
X:real    * Y:real    as real           is ...
```

2+3
5.5*6.4

X+1

```
X:integer + Y:integer
X:integer + 1
X:integer + Y:integer when Y > 0
X + Y
Infix:infix
```

X+1

```
foo X
+1
X * Y
```

integer

integer

0 N:integer when N mod 2 = 0

```
fib 0   is 0
fib 1   is 1
fib N   is (fib(N-1) + fib(N-2))
```

fib(N-1) fib(N-2)

fib (N-1)+fib(N-2)

fib(N-1) fib

N-1

0 1

print

```
print Items             is { write Items; print }
write Head, Rest        is { write Head; write Rest }
write Item:integer      is ...  // Implementation for integer
write Item:real         is ...  // implementation for real
```

print "Hello", "World"

```
CONTEXT1 is
    Items is "Hello", "World"
    CONTEXT0
```

write Items write write Head, Rest
Items

```
CONTEXT2 is
    Head is "Hello"
    Rest is "World"
    CONTEXT0
    HIDDEN1 is CONTEXT1
```

3.5. Dynamic dispatch

fib(N-1)
fib N

fib integer

shape
rectangle circle polygon
Draw
shape

```
draw R:rectangle    is ... // Implementation for rectangle
draw C:circle       is ... // Implementation for circle
draw P:polygon      is ... // Implementation for polygon
draw S:shape        is ... // Implementation for shape

draw Something      // Calls the right implementation based on type of Something
```

and

```
[[false]] and [[false]]     is false
[[false]] and [[true]]      is false
[[true]]  and [[false]]     is false
[[true]]  and [[true]]      is true
```

```
X:rectangle intersects Y:rectangle  as boolean  is ... // two rectangles
X:circle    intersects Y:circle     as boolean  is ... // two circles
X:circle    intersects Y:rectangle  as boolean  is ... // rectangle & circle
X:polygon   intersects Y:polygon    as boolean  is ... // two polygons
X:shape     intersects Y:shape      as boolean  is ... // general case

if shape1 intersects shape2 then    // selects the right combination
    print "The two shapes touch"
```

shape.Draw()

Draw(shape) intersects

```
theme_font "Christmas", "main",       "title"   is font "Times"
theme_font "Christmas", SlideMaster,  "code"    is font "Menlo"
theme_font "Christmas", SlideMaster,  SlideItem is font "Palatino"
theme_font SlideTheme,  SlideMaster,  SlideItem is font "Arial"
```

3.6. Immediate evaluation

circumference 2 * pi * Radius X * Y
2 * pi real integer

Radius

```
write X:infix   is  write X.left, " ", X.name, " ", X.right
write A+3
```

A+3 infix X
X:infix is A+3

(N-1)
0 0!

```
0! is 1
N! is N * (N-1)!
```

if-then-else
true false

```
if [[true]]  then TrueBody else FalseBody    is TrueBody
if [[false]] then TrueBody else FalseBody    is FalseBody
```

```
A - A    is 0
```

X 2 * Y X - 2 * Y

```
syracuse N when N mod 2 = 0  is N/2
syracuse N when N mod 2 = 1  is N * 3 + 1
syracuse X+5 // Must evaluate "X+5" for the conditional clause
```

3.7. Lazy evaluation

`while`

```
while Condition loop Body is
    if Condition then
        Body
        while Condition loop Body
```

`while`

```
while N <> 1 loop
    if N mod 2 = 0 then
        N /= 2
    else
        N := N * 3 + 1
    print N
```

`while` `Condition` `Body`

```
CONTEXT1 is
    Condition is N <> 1
    Body is
        if N mod 2 = 0 then
            N /= 2
        else
            N := N * 3 + 1
        print N
    CONTEXT0
```

while Condition
[[true]] [[false]] if-then-else while Body

Body Condition Body
Condition while Condition loop Body
N <>
1 N

and Condition false
and

```
[[true]]  and Condition is Condition
[[false]] and Condition is false
```

3.8. Closures

Condition Body
while

```
Condition is N > 1
while Condition loop N -= 1
```

```
CONTEXT2 is
    Condition is Condition
    Body is N -= 1
    CONTEXT0
```

Condition Condition

```
CONTEXT2 is
    Condition is CONTEXT1 { Condition }
    Body is CONTEXT1 { N-= 1 }
    CONTEXT0
```

CONTEXT1 { Condition } Condition CONTEXT1

CONTEXT1

Condition Body

Body N -= 1

N CONTEXT1 N > 1

Condition

```
adder N is { lambda X is X + N }
add3 is adder 3     // Creates a function that adds 3 to its input
add3 5              // Computes 8
```

add3 N is 3 N is 3 { N is 3 } M is "Hello" { N is 3; M is "Hello" }

adder N { lambda X is X + N } { N is 3 } { lambda X is X + N } X + N

N add3 add3 5 8 N is 3

CONTEXT EXPR CONTEXT EXPR CONTEXT EXPR

CONTEXT CONTEXT

write print

```
write Head, Tail        is { write Head; write Tail }
print Items             is { write Items; print }
```

{ X is 42 } { print "X=", X } Items { X is 42 }

```
CONTEXT1 is
    Items is { X is 42 } { "X=", X }
```

write Items Items { X is 42

} write

```
CONTEXT2 is
    Head is CONTEXT1 { "X=" }
    Tail is CONTEXT1 { X }
    CONTEXT1 is { X is 42 }
```

write write Tail X
X write Tail 42

3.9. Memoization

```
X + 0               is Case1(X)
X + Y when Y > 25   is Case2(X, Y)
X + Y * Z           is Case3(X,Y,Z)
```

A + foo B foo B
0 Y > 25
foo B

A + B * foo C B * foo C
0 Y > 25
B * foo C
B foo C B *
foo C foo C Z

```
do_not_chase is
    0 is 1
    1 is 2
    2 is 3
do_not_chase 0          // Returns 1, not 3
```

0 1

RATIONALE

fib(random(3..10))
fib 3 10
random
0 1
N
0 1

3.10. Self

self
self
true false

```
true    is self
false   is self
```

true true false false
true is true true

self

```
X, Y    is self
```

integer real text
{ Zero is 0; One is 1 }

3.11. Implicit result variable

result

```
factorial N:unsigned as unsigned is
    result := 1
    for I in 2..N loop
        result *= I
```

3.12. Returned value

`return` `return`

`error`

`result`

`result`

`return`

```
factorial_return N:unsigned as unsigned is
    if N = 0 then
        return 1
    return N * factorial_return(N-1)
```

```
factorial_last N:unsigned as unsigned is
    if N = 0 then
        1
    else
        N * factorial_last(N-1)
```

3.13. Nested declarations

```
count_vowels InputText is
    is_vowel C is
        Item in Head, Tail  is Item in Head or Item in Tail
        Item in RefItem     is Item = RefItem
        C in 'a', 'e', 'i', 'o', 'u', 'y', 'A', 'E', 'I', 'O', 'U', 'Y'

    Count : integer := 0
    for C in InputText loop
        if is_vowel C then
            Count += 1
    Count
count_vowels "Hello World" // Returns 3
```

is_vowel C C

is_vowel X

count_vowels T

in

```
C in 'a', 'e', 'i', 'o', 'u', 'y', 'A', 'E', 'I', 'O', 'U', 'Y'
```

count_vowels "Hello World"

```
CONTEXT1 is
    is_vowel C is ...
    Count:integer := 0
    InputText is "Hello World"
    CONTEXT0
```

is_vowel Char

```
CONTEXT2 is
    Item in Head, Tail is ...
    Item in RefItem is ...
    C is 'l'
    CONTEXT1
```

is_vowel Char Count InputText

```
count_vowels InputText is count C in InputText where C in
"aeiouyAEIOUY"
```

3.14. Scoping

.

{ X is 40; Y is 2 } { X + Y }

42 X Y X + Y

digit_spelling

```
digit_spelling is
    0 is "zero"
    1 is "one"
    2 is "two"
    3 is "three"
    4 is "four"
    5 is "five"
    6 is "six"
    7 is "seven"
    8 is "eight"
    9 is "nine"
```

digit_spelling 3 "three"

digit_spelling[4]

digit_spelling[A+3] A is 2
A+3 digit_spelling

```
{ X:integer+Y:integer as integer is ... }
  { A is 2 }
    { 0 is "zero"; 1 is "one"; ... }
      [A+3]
```

0
A+3
0
A+3

X:integer+Y:integer 2+3 5
0 0=5

A+3 5 1 2
5 "five"

\N lambda
N

```
number_spelling is
    \N when N<10    is digit_spelling[N]
    11              is "eleven"
    12              is "twelve"
    13              is "thirteen"
    14              is "fourteen"
    15              is "fifteen"
    16              is "sixteen"
    17              is "seventeen"
    18              is "eighteen"
    19              is "nineteen"
    20              is "twenty"
    30              is "thirty"
    40              is "forty"
    50              is "fifty"
    60              is "sixty"
    70              is "seventy"
    80              is "eighty"
    90              is "ninety"
    \N when N<100   is (number_spelling[N/10*10] & " " &
                        digit_spelling[N mod 10])
    \N when N<1000  is (digit_spelling[N/100] & " hundred and " &
                        digit_spelling[N mod 100])
```

```
byte_magic_constants is
    num_bits    is 8
    min_value   is 0
    max_value   is 255
```

byte_magic_constants.num_bits 8

```
magic_constants(Bits) is
    num_bits    is Bits
    min_value   is 0
    max_value   is 2^Bits - 1
```

magic_constants(4).max_values 15

use use IO = XL.CONSOLE.TEXT_IO IO IO.write

3.15. Super lookup

super

```
X is 42
foo X:integer is X + super.X    // super.X refers to X above
foo 3                           // Returns 45
```

3.16. Assignments and moves

:=
+= -= *= /=

```
X : integer := 0    // Initialize X to 0
X := 5              // Now X contains value 5
X += 7              // Now X contains value 12
```

:=
:

```
X += Y      is      X := X + Y
X -= Y      is      X := X - Y
X *= Y      is      X := X * Y
X /= Y      is      X := X / Y
X &= Y      is      X := X & Y
X |= Y      is      X := X | Y
X ^= Y      is      X := X ^ Y
```

:+ :<

:+

+

:<

:=

:=

RATIONALE

spreadsheet graph picture

3.17. Functions as values

my_function

```
my_function X is X + 1
apply Function, Value is Function(Value)
apply my_function, 1        // Error: Nothing called 'my_function'
```

RATIONALE

`A in B..C` `in..`

`print` `Items` `write Items` `write Head, Tail`

`apply`

```
0!                      is 1
N!                      is N * (N-1)!
apply Function, Value   is Function(Value)

// Using an anonymous map to compute 3!
apply { \N is N! }, 3

// Using a named map to compute 5!
factorial   is { \N is N! }
apply factorial, 5
```

3.18. Error handling

`error` `error` `sqrt`

```
sqrt X:real as real     when X >= 0     is ...
print "Square root of 2 is ", sqrt 2        // OK
print "Square root of -1 is ", sqrt(-1)     // Error
```

```
Square root of 2 is 1.41421356237
Square root of -1 is Error: No form matches sqrt(-1)
```

```
sqrt X:real as real     when X >= 0     is ...
sqrt X:real as error    when X <  0     is error "Square root of negative real ", X
```

```
Square root of 2 is 1.41421356237
Square root of -1 is Error: Square root of negative real -1.0
```

3.18.1. Taking error parameters

`error`

```
sqrt X:real as real     when X >= 0     is ...
sqrt X:real as error    when X <  0     is error "Square root of negative real ", X
sqrt E:error as error                   is error "Square root of error: ", E
```

```
print "Double error is ", sqrt(sqrt(-1))
Double error is Error: Square root of error: Square root of negative real -1.0
```

`print` `write` `error`

3.18.2. Fallible types

`mayfail T` `T` `error` `mayfail` `mayfail nil`

mayfail T

value T
error

error error
nil

good true bad

bad not good

mayfail real 0.0 sqrt

```
sanitized_sqrt X:real as real is
    R : mayfail real := sqrt X
    if R.bad then
        print "Got an error in sqrt: ", R.error
        R := 0.0
    return R.value
```

3.18.3. Try-Catch

try Body catch Handler Body
Body error Handler catch caught

sanitized_sqrt

```
sanitized_sqrt X:real as real is
    try
        sqrt X
    catch
        print "Got an error in sqrt: ", caught
        0.0
```

error

error

3.18.4. Error statements

error

error

error

```
Thing *read_thing_from_file(const char *filename)
{
    FILE *file = fopen(filename, "r");
    if (file == NULL)
        return NULL;
    Thing *thing = malloc(sizeof(Thing))
    if (thing == NULL)
    {
        fclose(file);
        return NULL;
    }
    thing->header = malloc(sizeof(ThingHeader));
    if (thing->header == NULL)
    {
        free(thing);
        fclose(file);
        return NULL;
    }
    size_t header_read = fread(&thing->header, 1, sizeof(ThingHeader), file);
    if (header_read != sizeof(ThingHeader))
    {
        free (thing->header);
        free (thing);
        fclose(file);
        return NULL;
    }
    if (thing->header.size < MIN_SIZE)
    {
        log_error("Header size is too small: %u", thing->header.size);
        free(thing->header);
        free(thing);
        fclose(file);
        return NULL;
    }
    // ... possibly more of the same
    fclose(file);
    return thing;
}
```

error

```
read_thing_from_file FileName:text as mayfail own thing is
    F:file := file(FileName)            // May error out ①
    H:own thing_header := read(F)       // May error out (and close F) ②
    if H.size < MIN_SIZE then
        // Explicitly error out with custom message
        error "Header size %1 is too small", H.size ③
    T:own thing := thing(H)             // May error out, dispose H, close F ④
    // ... possibly more of the same
    T
```

①
file_error

②
thing_header

③
error

④ thing H storage_error
thing

own T

3.19. Interface and implementation

integer X

```
X : integer
```

X+X 2*X+1 X<0 X:=18 X
X
X

```
X : integer := 42
```

```
is_odd N:integer as boolean
```

```
for I in 1..100 loop
    if is_odd I then
        print I, " is odd"
    else
        print I, " is even"
```

is_odd

```
is_odd N:integer as boolean is N mod 2 <> 0
```

and

```
is_odd N:integer as boolean is N and 1 = 1
```

RATIONALE

Chapter 4. Types

A real A+A X:real+Y:real X:integer+Y:integer

type(Pattern) real type(A:real+B)

2+3*5 type(A+B*C) type(A:integer+B:integer) infix

X:integer X integer X 2 even_integer positive_integer type(2) 2

4.1. Type annotations

X:integer X integer

X:T X as T X T < integer boolean

```
X:integer < Y:integer as boolean
```

: as : as X:integer Y:integer X Y X < Y as boolean 3 < 5 boolean

T constant variable X:T X X as T X seconds : integer seconds as integer

4.2. Type definitions

type

int integer

```
int is integer
```

int integer

```
int is type(base:integer)
```

type

type(42)

42

type
positive

```
positive is type(X when X > 0)
```

positive

integer 27 real

3.14 X > 0

X > 0

integer and positive

struct record

complex

```
complex is type complex(Re:real, Im:real)
Z:complex := complex(4.3, 2.1)
```

complex complex

complex(Re:real, Im:real) complex(4.3, 2.1)

`complex` `real` `real`

```
complex[real:type] is type complex(Re:real, Im:real)
complex is complex[real]
Z:complex := complex(4.3, 2.1)
K:complex[real32] := complex(1.2, 3.4)
```

4.3. Shared type annotations

```
complex is type complex(Re:real, Im:real)
Z1:complex + Z2:complex as complex is ...
Z1:complex - Z2:complex as complex is ...
Z1:complex * Z2:complex as complex is ...
Z1:complex / Z2:complex as complex is ...
```

`Z1` `Z2` `complex` `with`

```
complex is type complex(Re:real, Im:real)
with
    Z1 : complex
    Z2 : complex
Z1 + Z2 as complex is ...
Z1 - Z2 as complex is ...
Z1 * Z2 as complex is ...
Z1 / Z2 as complex is ...
```

```
with
    N  :  unsigned
    N! as unsigned
0! is 1
N! is N * (N-1)!
```

with N
unsigned
with

```
N:unsigned! as unsigned
```

0! N:unsigned!

```
0:unsigned! as unsigned is 1
N:unsigned! as unsigned is N * (N-1)!
```

4.4. Standard types

4.4.1. Basic types

use

type

nil nil

integer
integer.min integer.max
integer

integer integer.min 12
-3

unsigned
integer unsigned
0 16#FF 42

size unsigned
unsigned

offset integer size unsigned

character

character 'A'

text

string of character text
"Hello World"

boolean true false
boolean

4.4.2. Sized data types

integer

integer64 integer

integer

unsigned

real

character

unsigned24

unsigned8 0 255
255 0 0 255

integer range 1..5 1 5
integer bits 24 integer

4.4.3. Category types

anything

number integer real complex

positive

ordered <

discrete integer character index

access

vector

number sort ordered

4.4.4. Generic containers

array

- array[5] of integer
- array['A'..'Z'] of boolean discrete
- array['A'..'H', 1..8] of chess_piece

string

- string of integer text string of character
- string[1000] of integer 1000 integer
- string[1..10] of real 10 1
- string[25,80] of character

list

- list of integer
- queue of integer
- xor_list of integer

stack push pop
string list queue list of T stack of T

map map[text] of real
text real

set
set of character character

4.4.5. True generic types

some_
array

```
some_array is type (array[index:array_index] of value:type)
```

array

```
sum A:some_array as A.value is
    result := 0
    for I in A loop
        result += I
```

complex real

```
some_complex is type complex[real:type like number]
```

RATIONALE

template

```
template <typename T>
T sum(const vector<T> &v)
{
    T s = 0;
    for (auto i : v)
        s += i;
    return s;
}
```

4.4.6. Other generic types

range of T T range of integer
1..5 1 5

own T T own
own

ref T T

in T T

out T T

in_out T T

inout T io T

any T T

slice of T array string
text

access T T own T ref T
slice of T

4.4.7. Mathematical types

real

complex[T] T
complex complex[real] complex
polar cartesian

quaternion[T]
quaternion
quaternion[real]

vector
vector[3] of real32 real32
vector[4] real vector array
vector

matrix
sparse dense matrix
array
vector
matrix matrix[4,3] of integer32
integer32 matrix[2,2]
real

4.4.8. Parse tree types

XL.PARSER

tree
integer 42
real 3.14
text "ABC"
character 'A'
name A
operator +
symbol
()
infix A+B
prefix +3
postfix 4%
block (A)

4.4.9. Program-related types

`lifetime`

`parser`

`evaluator`

`task`

4.4.10. Other common types

`XL.MEMORY`

- `byte`
- `address` `byte`

`XL.FILE`

- `file`
- `name` `text`
- `path` `name`
- `directory`
- `attributes`

4.5. Type-related concepts

4.5.1. Lifetime

X

lifetime X < lifetime Y true Y

X lifetime X < lifetime Y lifetime X > lifetime Y

lifetime

x+y a+b+c+d

((a+b)+c)+d a+b

(a+b)+c

```
use XL.CONSOLE.TEXT_IO ①
use XL.TEXT.FORMAT

print "Starting printing Fibonacci sequences" ②

fib 0 is 1 ③
fib 1 is 1
fib N is (fib(N-1) + fib(N-2)) ④

for I in 1..5 loop ⑤
    F is fib I ⑥
    print format("Fib(%1) is %2", I, F) ⑦
```

① use XL XL.CONSOLE XL.CONSOLE.TEXT_IO XL.TEXT XL.TEXT.FORMAT XL

② print XL.CONSOLE.TEXT_IO print

print

③ fib fib N fib I

④ (fib(N-1)+fib(N-2)) N-1 N-2

tmp1 tmp2 tmp3 tmp4 tmp5

```
tmp1 is N-1
tmp2 is fib(tmp1)
delete tmp1
tmp3 is N-2
tmp4 is fib(tmp3)
delete tmp3
tmp5 is tmp2+tmp4
delete tmp2
delete tmp4
tmp5
```

⑤ for I 1 2 3 4 5 I

I

```
tmpBody is
    F is fib I
    print format("Fib(%1) is %2", I, F)
{ I is 1 } ( tmpBody )
{ I is 2 } ( tmpBody )
{ I is 3 } ( tmpBody )
{ I is 4 } ( tmpBody )
{ I is 5 } ( tmpBody )
```

⑥ F is fib I

tmpBody delete F

```
tmpBody is
    F is fib I
    print format("Fib(%1) is %2", I, F)
    delete F
```

⑦ fib I F is fib I fib I fib I F F F+F

4.5.2. Creation

V create V

```
Add Z:complex is
   T:complex
   Z+T
```

```
Add Z:complex is
   T:complex
   (create T)
   Z+T
```

array[1..5] of complex complex

create out out

```
create Z:out complex is
    print "Creator called"
```

```
create Z:out complex is
   (create Z.Re)        // Implicit creation of complex fields
   (create Z.Im)
   print "Creator called"
```

create

type nil nil

integer unsigned size offset 0

character character 0

text ""

boolean false

create

out

create

```
Z:complex
create Z
```

```
Z:complex
(create Z)      // because of the declaration above
(delete Z)      // because Z is passed as out argument to `create`
create Z
```

complex

```
complex is type complex(Re:real, Im:real)
```

complex(2.3, 5.6) complex
complex
complex complex(1.3)

error
error

complex i
2+3i

```
i   is complex(0.0, 1.0)

syntax { POSTFIX 190 i }
Re:real + Im:real i                 is complex(Re, Im)      // Case 1
Re:real + Im:real * [[i]]           is complex(Re, Im)      // Case 2
Re:real + [[i]] * Im:real           is complex(Re, Im)      // Case 3
Re:real as complex                  is complex(Re, 0.0)     // Case 4
X:complex + Y:complex as complex    is ...

2 + 3i              // Calls case 1 (with explicit concersions to real)
2 + 3 * i           // Calls case 2 (with explicit conversions to real)
2 + i * 3           // Calls case 3
2 + 3i + 5.2        // Calls case 4 to convert 5.2 to complex(5.2, 0.0)
2 + 3i + 5          // Error: Two implicit conversions (exercise: fix it)
```

```
large is type (N when N > 42)
A:large := 44   // OK
B:large := 99.1 // OK
C:large         // Error
create V:out large is
    print "Creator called"
```

A:large B:large large C:large create create V.N integer 0

large

create

```
large is type (N when N > 42)
C:large         // OK
create V:out large is
    print "Creator called"
    V := 44     // Creator for a large value
```

file

```
MY_FILE as module with
    file as type
    open(Name:text) as file
    close F:io file

MY_FILE as module is
    file is type file(fd:integer)
    open(Name:text) as file is
        fd:integer := libc.open(Name, libc.O_RDONLY)
        file(fd)
    close F:inout file is
        if fd >= 0 then
            libc.close(F.fd)
            F.fd := -2
    delete F:inout file is close F    // Destruction, see below
```

create

file open file

RATIONALE

4.5.3. Destruction

V delete V
delete X:T
T
delete X:inout T
delete V delete V.X X V

```
delete Z:inout complex is
    print "Deleting complex ", Z
```

```
delete Z:inout complex is
    print "Deleting complex ", Z
    (delete Z.Im)
    (delete Z.Re)
```

```
delete Anything is nil
```

file MY_FILE

```
delete F:inout file when F.fd < 0  is ... // Invalid flie
delete F:inout file                is ... // Valid file
```

valid_file
F.fd<0 valid_file delete

```
positive is type (N when N > 0)
integers is string of integer
N:integers > 0 as boolean is
    for I in N loop
        if not (I > 0) then
            return false
    return true

delete N:inout positive is
    print "Deleting positive: ", N

delete N:integer is
    print "Deleting integer: ", N

delete N:integers is
    print "Deleting integers with size: ", size N

example is
   print "Beginning example"
   A:integers := string(1,8,4)
   B:integers := string(-1,0,5)
   print "End of example"
```

integers string of integers
N>0 string
integers positive example A positive B

integers delete

```
integers is type(base:string of integer)

delete P:inout positive is
    print "Deleting positive: ", P
    (delete P.N)        // Delete the N bound in `positive`

delete N:integer is
    print "Deleting integer: ", N
    (delete N.base)     // For `integer`, this is a no-op

delete S:integers is
    print "Deleting integers with size: ", size S
    (delete S.base)     // Delete the underlying 'string of integer'
```

```
Beginning example
End of example
Deleting integers with size 3 ①
Deleting positive: 5 ②
Deleting integer: 5 ③
Deleting integer: 0 ④
Deleting integer: -1
Deleting positive: string(1,8,4) ⑤
Deleting integers with size 3 ⑥
Deleting positive: 4
Deleting integer: 4
Deleting positive: 8
Deleting integer: 8
Deleting positive: 1
Deleting integer: 1
```

① B integers
positive -1

② string of integer B
string

string delete
delete string of integer
example 5 positive

③ integer P.N

④ 0 string of integer B positive
integer

⑤ A A

⑥ B integers B
P.N

```
show_destructors is
    delete Something is
        print "Deleted", Something
        super.delete Something
    X is 42
    Y is 57.2
    X + Y
```

```
Deleted 42.0
Deleted 57.2
Deleted 42
```

`X` `integer` `real`

```
delete Value
```

`Value` `delete` `Value`

```
for I in 1..LARGE_NUMBER loop
    delete Value
```

`out`

4.5.4. Errors

`error`

```
error as type is either
    error Message:text
    error Message:text, Payload
```

`format`

```
log X:real as error when X <= 0 is
    error "Logarithm of negative value %1", X
```

`T or error`

`mayfail T`

```
mayfail T:type as type is T or error
```

log

```
log X:real as mayfail real     is ... // May return real or error
```

log

```
log X:real as real  when X > 0.0    is ... // Always return a real
log X:real as error                 is ... // Always return an error
```

log X real X > 0.0 error
real or error mayfail real

```
if X > 0.0 then
    print format("Log(%1) is %2", X, log X)
```

RATIONALE error

error

range_error

```
T:text[I:offset] as character or range_error is
    if I >= length T then
        range_error "Text index %2 is out of bounds for text %2", I, T
    else
        P : memory_address[character] := memory_address(T.first)
        P += I
        *P
```

logic_error
assert require ensure

```
if X > 0 then
    print "X is positive"
else if X < 0 then
    print "X is negative"
else
    logic_error "Some programmer forgot to consider this case"
```

storage_error own T

```
S : string of integer           // The string requires storage
loop
    V : own integer := 3        // This allocates an integer, freed each loop
    S &= V                      // Accumulate integers in an unbounded way
```

file_error

permission_error

compile_error

compile_error
compile_warning

```
// Emit a specific compile-time error if assigning text to an integer
X:integer := Y:text is
    compile_error "Cannot assign text %1 to integer %2", Y, X

// Emit a specific warning when writing a real into an integer
X:integer := Y:real is
    compile_warning "Assigning real to integer may lose data"
    T is integer Y
    if real T = Y then
        X := T
    else
        range_error "Assigned real value %1 is out of range for integer", Y
```

4.5.5. Mutability

X:T X T T

X X X as T
X T T
X

```
StartupMessage : text := "Hello World"  // Variable
Answer as integer is 42                 // Named constant
```

:=
+=

```
X : integer := 42       // Initialize with value 42
X := X or 1             // Binary or, X is now 43
X -= 1                  // Subtract 1 from X, now 42
```

text
character
text slice text text
text text
text

```
Greeting : text := "Hello"              // Variable text
Person as text is "John"                // Constant text
Greeting := Greeting & " " & Person     // (1) Greeting now "Hello John"
Greeting &= "!"                         // (2) Greeting now "Hello John!"
Greeting[0..4] := "Good m0rning"        // (3) Greeting now "Good m0rning John!"
Greeting[6] := 'o'                      // (4) Greeting now "Good morning John!"
```

Person
Person[3]:='a' Person

text
Greeting[0..4]
slice text Greeting
0..4 :=
text

K
L I

```
for J in 1..5 loop
    for I in 1..5 loop
        K is 2*I + 1
        L is 2*J + 1
        print "I=", I, " K=", K, " L=", L
```

RATIONALE

4.5.6. Compactness

integer real integer

text

"Hi" "There once was a time where text was represented in languages such as Pascal by fixed-size character array with a byte representing the length. This meant that you could not process text that was longer than, say, 255 characters. More modern languages have lifted this restriction."

4.5.7. Ownership

file MY_FILE

delete Value

own nil
own own nil nil own T or nil

array buffer string

- array
- buffer
- string

text character string of character

file

mutex

timer

thread

task

process

context

4.5.8. Access

T text A B integer T[A..B]
0 A B
T[A..B] T text
A B character
T

T[A..B]

A[I] *(A+I) 3[buffer]
buffer ptr[-1]
ptr

ref own

slice array buffer
string text string of character

reader writer file

lock mutex

timing dispatch timeout rendezvous timer thread task
context

in out inout

XL.SYSTEM.MEMORY.address

4.5.9. Inheritance

```
Derived:derived as base is ...
```

integer16 integer32
integer16 integer32

complex number

number complex[3.5] complex[real]

```
some_complex is type complex[real:number]
```

`type like base`

```
some_complex is type complex[real:type like number]
```

`type like base`

4.5.10. Subtypes

`integer` `month` `integer` `1` `12`

```
month is type(M:integer when M >= 1 and M <= 12)
```

4.5.10.1. Range subtypes

`range`

```
T:type range Low:T..High:T      is type(X:T when X in Low..High)
```

`month`

```
month is integer range 1..12
```

4.5.10.2. Size subtypes

`bits` `real` `integer` `character`

`integer8`

```
integer8 is integer bits 8
```

`range` `integer` `unsigned`

```
[[integer]]  bits N:unsigned    is integer  range -2^(N-1)..2^(N-1)-1
[[unsigned]] bits N:unsigned    is unsigned range  0..2^N-1
```

bits

integer

bits 22

4.5.10.3. Real subtypes

real range bits

real

digits

real digits 3

quantum

real quantum 0.25 0.25

exponent real exponent 100

1.0e100

base 2 10 16

2 10

16

real

exponent 0

range

quantum exponent

hundredth unsigned
0 100 real quantum

```
hundredth is real range 0.0..1.0 quantum 0.01
```

4.5.10.4. Saturating subtypes

saturating integer real

color_component 0.0 1.0

```
color_component is saturating real range 0.0..1.0 bits 16
Red : color_component := 0.5
Red += 0.75             // Red is now 1.0
```

4.5.10.5. Character subtypes

character range bits

```
letter is character range 'A'..'Z'
ASCII  is character bits 7
```

encoding character encoding
"ASCII"

locale character locale "fr_FR"

collation character
character collation "de_DE"

4.5.10.6. Text subtypes

text encoding locale collation
character

4.5.11. Type interface

storage_error

picture width height pixels
area pixels
pixels buffer

```
picture as type with
    width  : unsigned
    height : unsigned
    pixels : buffer[area] of unsigned8
    area as unsigned
```

text string of character

```
text as type like string of character with
    byte_count as size                  // Number of bytes used by characters
    as_number[T:type like number] as T  // Numerical conversion
```

MY_FILE file

```
file as type
```

picture
picture picture

```
picture(width:unsigned, height:unsigned) as picture
```

4.5.11.1. Information hiding

picture

is_square

```
is_square P:picture is P.width = P.height
```

P picture
P.width integer

4.5.11.2. Direct implementation

picture

```
picture is type picture
    pixels : buffer[area] of unsigned8
    width  : unsigned
    height : unsigned
```

P picture
P
width P.width
P.width
picture
P.width
picture

4.5.11.3. Indirect implementation

text

```
T:text as string of character is convert_to_string(T)
```

text string of character

4.5.11.4. Delegation

picture bitmap

```
bitmap as type with
    width  : unsigned16
    height : unsigned16
    buf    : array[width, height] of unsigned8
picture as type picture
    bits:bitmap
```

picture
bitmap
width height

```
picture is type picture
    bits:bitmap

    width  is bits.width
    height is bits.height
```

width height
picture
width as unsigned
P.width

4.5.11.5. Attributes implementation

width height

```
picture is type picture
    bits:bitmap

    width  is bits.width
    height is bits.height

    width  := W is bits.width := W
    height := H is bits.width := H
```

width width := W
[[width]] := W
width
width type(width) width

```
picture is type picture
    bits:bitmap

    width  is bits.width
    height is bits.height

    width:type(width)   := W is bits.width := W
    height:type(height) := H is bits.width := H
```

width height

```
picture is type picture
    bits:bitmap

    width       is bits.width
    height      is bits.height

    width W     is bits.width := W
    height H    is bits.height := H
```

bits.width
bits.height

```
picture is type picture
    bits:bitmap

    width       is bits.width
    height      is bits.height

    width W     is bits.width W
    height H    is bits.height H
```

4.5.11.6. Generic implementations

width 320
unsigned16 unsigned

unsigned16

unsigned16

P picture P.width 320 P.width 1_000_000
unsigned16
unsigned unsigned
unsigned16
unsigned unsigned16

4.5.11.7. Attribute error checking

```
picture is type picture
    bits:bitmap

    width       is bits.width
    height      is bits.height

    // Working case
    width  W:unsigned16 is bits.width W
    height H:unsigned16 is bits.height H

    // Error case: drop input, display message
    width  W:unsigned is
        print error("Invalid picture width %1", W)
        bits.width      // Return previous value
    height  H:unsigned is
        print error("Invalid picture height %1", H)
        bits.height     // Return previous value
```

```
picture as type with
    width  : mayfail unsigned
    height : mayfail unsigned
    pixels : buffer[area] of unsigned8
    area as unsigned
```

width error
width height

```
picture as type with
    width  as unsigned
    height as unsigned
    width W as mayfail unsigned
    height H as mayfail unsigned
    pixels : buffer[area] of unsigned8
    area as unsigned
```

```
picture as type with
    width  : unsigned16
    height : unsigned16
    pixels : buffer[area] of unsigned8
    area as unsigned
```

4.5.11.8. Exposed details

pixels
picture buffer
storage_error
buffer unsigned unsigned16
buffer bitmap
P.pixels picture
storage_error
P.pixels
P.pixels

P.pixels

4.5.12. Transfers

4.5.12.1. Assignments

:=

```
julia_depth(Z:complex, Mu:complex, Bound:real, Max:unsigned) as unsigned is
    while result < Max and Z.Re^2 + Z.Im^2 < Bound ^2 loop
        result := result + 1
        Z := Z^2 - Mu
```

result

Z result := result + 1 result

unsigned

result += 1

complex picture

```
Target:out complex := Source:complex    is Target :+ Source     // Copy
Target:out picture := Source:picture    is Target :< Source     // Move
```

4.5.12.2. Copy

copy Target :+ Source +

:+

```
Target:out T :+ Source:T as mayfail T
copy Source:T as mayfail T is result :+ Source
```

Target error

picture

pixels

pixels

tree

```
copy (Source:T, StopConditions) as mayfail T
```

node_filter

```
copy(Source:tree, Depth:unsigned) as mayfail T
copy(Source:tree, Keep:node_filter) as mayfail T
node_filter is type(N:node as boolean)
```

4.5.12.3. Move

4.5.12.4. Binding

4.5.12.5. Argument passing

4.5.12.6. Attributes

4.5.13. Atomicity

4.6. Type expressions

`type`

`complex`

`real`

`complex` `complex32` `real32`

```
type complex[real:type] is complex(Re:real, Im:real)
type complex is complex[real]
type complex32 is complex[real32]
```

`complex[real]`

`pointer to T`

4.7. Standard type expressions

`nil` `nil`

`T1 or T2` `T1` `T2`

`integer or real` `integer` `real`

`T1 or T2`

```
double X:(integer or real) is X + X
double 1      // returns 2 as an integer
double 3.5    // returns 7.0 as a real
```

T1 and T2 T1 T2 number and totally_ordered "ABC" totally_ordered number ieee754(2.5) number totally_ordered

another T T type distance is another real real distance real

```
type distance is another real
X:distance * Y:distance is compile_error "Cannot multiply distances"
X:real as distance is compile_error "Implicit distance from real"
syntax { POSTFIX 400 m cm mm km }
X:real m  is distance(X)
X:real cm is distance(X * 0.01)
X:real mm is distance(X * 0.001)
X:real km is distance(X * 1000.0)

D:distance is 3.2km
D + D     // OK: inherit X:distance+Y:distance from X:real+Y:real
D + 1.0   // Error: Implicit distance from real
D * D     // Error: Cannot multiply distances
```

distance X:integer as real D+1

optional T T or nil find

nil

mayfail T T or error nil error

array[N] of T N T N array[A..B] of T

A B array['A'..'Z'] of boolean boolean

string of T T string text string of character

either Patterns

```
type complex is either
    cartesian(Re:real, Im:real)
    polar(Mod:real, Arg:real)
```

variable T var T T constant T
T :=

X:integer

in T out T inout T

T in ValueList T
ValueList ValueList
A..B integer in 1..5,9,12..20
text in "One", "Two", "Three", "Four"

in T out T inout T

4.7.1. Copy or Move

4.7.2. Variant types

```
type picture with
    width  : unsigned
    height : unsigned
    format : either { RGB; GRAY }
    buffer : buffer
    size is width * height
    type grayscale is fixed_point range 0.0..1.0 bits 8
    type buffer is buffer[1..size] of pixel

    type pixel is pixel[format]
    type pixel[RGB] is rgb(red   : grayscale,
                           green : grayscale,
                           blue  : grayscale)
    type pixel[GRAY] is gray(gray: grayscale)
```

increment X:inout integer is X := X+1; print_A print_A is print "A=", A
A:integer := 45 increment A // Can print either "A=45" or "A=46" depending on copy or ref

inout T

copy_in T copy_out T copy_inout T

ref T ref T

ref T

```
// Increment in place
increment X:ref integer is X := X+1; print_A
print_A is print "A=", A
A:integer := 45
increment A   // Guaranteed to print "A=46", X is the same as A
```

4.8. Type hierarchy

4.8.1. MOSTLY JUNK BELOW, IGNORE (IDEAS SCRATCHPAD)

with

as

:

person

Name person Greeting

person person P person P.Name

P.Greeting person.Citizenship

Self

P.FullName person.FullName P

2 + 3

addition addition is type A+B infix

integer ARITHMETIC

constant integer integer

integer range 1..5

integer number
integer
number

type Pattern
type complex(Re:real, Im:real)
complex(2.0, 3.5)

T

```
Allocate[T:type] as pointer[T]
```

T.ByteSize

with
is

Z

type
integer
array[1..5] of real

access integer
Indirect integer
any integer
DynamicType
M:mammal A:any animal (any animal).Indirect = animal
(any animal).DynamicType any animal A
M mammal

DynamicSize

DynamicSize unsigned8

Mutable Constant

Chapter 5. Compiled XL

5.1. Compiled representations

`integer` `real`

5.2. Data

5.3. Lifetime

5.4. Closures

5.5. Compact vs. Packed

Chapter 6. Basic operations

Chapter 7. Modules

Chapter 8. Standard Library

8.1. Garbage collection

Chapter 9. History of XL

9.1. It started as an experimental language

-
- `mmap`
-
-
- `WriteLn`
-
- `printf`

{pragma}

9.2. LX, an extensible language

{annotations}

{annotation}

annotation

translation xl2/

{derivation}

{differentiation}

{derivation}

derivation

{derivation} d(X+sin(X))/dX

1 + cos(X)

9.3. LX, meet Xroma

9.4. XL moves to the off-side rule

`begin` `end`

`begin` `end`

9.5. Concept programming

9.6. Mozart and Moka: Adding Java support to XL

Performer

9.7. Innovations in 2000-vintage XL

begin end

X+Y*Z X in Y..Z

WriteLn

WriteLn

9.8. XL0 and XL2: Reinventing the parse tree

Notes

IfThenElse

Declaration

XL0

XL1

XL2

XL0

```
Integer                                 123    16#FFFF_FFFF

Real                                    123.456    2#1.001_001#e-3          Integer

Text                                "Hello"    'A'
Name                    ABC             <=
Infix                                                                  A+B    A and B
Prefix                                                                  sin X    -4
Postfix                                                                 3 km    5%
Block                                                                              [A] (3)
{write}
```

9.9. Bootstrapping XL

9.10. XL2 compiler plugins

```
function Differentiate (expr : PT.tree; dv : text) return PT.tree is
    translate expr
        when ('X' * 'Y') then
            dX : PT.tree := Differentiate(X, dv)
            dY : PT.tree := Differentiate(Y, dv)
            return parse_tree('dX' * 'Y' + 'X' * 'dY')
```

translate

X*Y then

parse_tree

when parse_tree

parse_tree(X)

X parse_tree('X') X

9.11. XL2 internal use of plugins: the `translation` extension

translation translate

translate

translation

translation X X

translation X translation Y

translation XLDeclaration

translation XLSemantics

9.12. Switching to dynamic code generation

9.13. Translating using only tree rewrites

is

X is 0 X 0

→

is →

```
x:integer - y:integer as integer    is opcode Sub
```

writeln

```
write x:text as boolean             is C xl_write_text
write x:integer as boolean          is C xl_write_integer
write x:real as boolean             is C xl_write_real
write x:character as boolean        is C xl_write_character
write A, B                          is write A; write B
writeln as boolean                  is C xl_write_cr
writeln X as boolean                is write X; writeln
```

; is

```
if true  then TrueBody else FalseBody   is TrueBody
if false then TrueBody else FalseBody   is FalseBody
if true  then TrueBody                  is TrueBody
if false then TrueBody                  is false
```

true

true [[true]]

```
while Condition loop Body is
    if Condition then
        Body
    while Condition loop Body

until Condition loop Body is while not Condition loop Body

loop Body is { Body; loop Body }

for Var in Low..High loop Body is
    Var := Low
    while Var < High loop
        Body
        Var := Var + 1
```

9.14. Tao3D, interactive 3D graphics with XL

Write -A (-A) X - Y Write

slide

```
import WhiteChristmasTheme
theme "WhiteChristmas"

slide "An example slide",
    * "Functional reactive programming is great"
    color_hsv mouse_x, 100%, 100%
    * "This text color changes with the mouse"
    color_hsv time * 20, 100%, 100%
    * "This text color changes with time"
```

`X := Y`

9.15. ELFE, distributed programming with XL

`→` `is`

```
invoke "pi2.local",
   every 1.1s,
        rasp1_temp ->
            ask "pi.local",
                temperature
        send_temps rasp1_temp, temperature

   send_temps T1:real, T2:real ->
       if abs(T1-T2) > 2.0 then
           reply
               show_temps T1, T2

show_temps T1:real, T2:real ->
    write "Temperature on pi is ", T1, " and on pi2 ", T2, ". "
    if T1>T2 then
        writeln "Pi is hotter by ", T1-T2, " degrees"
    else
        writeln "Pi2 is hotter by ", T2-T1, " degrees"
```

ask

invoke

reply invoke

```
invoke "pi2.local",
   every 1.1s,
        rasp1_temp ->
            ask "pi.local",
                temperature
        send_temps rasp1_temp, temperature

   send_temps T1:real, T2:real ->
       if abs(T1-T2) > 2.0 then
           reply
               show_temps T1, T2

show_temps T1:real, T2:real ->
    write "Temperature on pi is ", T1, " and on pi2 ", T2, ". "
    if T1>T2 then
        writeln "Pi is hotter by ", T1-T2, " degrees"
    else
        writeln "Pi2 is hotter by ", T2-T1, " degrees"
```

9.16. XL gets a type system

if-then-else

integer

real type(X+Y)

2+3*5 type(X+Y)

type(X:integer+Y:integer) infix

type(X:integer when X mod 2 = 0)

9.17. The LLVM catastrophy

9.18. Repairing and reconverging

`bigmerge`

→ is

FastCompiler

9.19. Language redefinition

if-then-else

if-then-else

```
if true  then TrueClause      is TrueClause
if false then TrueClause      is false
```

true

TrueClause

true

TrueClause

A - A is 0

translate [[X]]

X [[sqrt 2]]

true true [[true]] true

binary "image.png"

binary

16#FFFF_0000_FFFF_0000_FF00_00FF_FF00_00FF

lambda

(X is X + 1) (X is X + 1) 3 4

X lambda

lambda X is X + 1

9.20. Future work

native

1.0.1

Integer Real

( ) { }

```
binary 16#0001_0002_0003_0004_0005_0006_0007_0008_0009
binary "image.jpg"
```

make-it-quick recorder

Index

I

F

K

G

L

H

M

N

O

P

Q

R

S

U

V

W

Z

T