# How to compile AST application?

Compilation is done thanks to the following command:

compile –o loo\_test loo\_test

# How to run AST application ?

To run AST application, please launch loo\_test.

Once the application is launched, the user must precise the test he wants to launch thanks to an integer value between 1 and 15 (see below to choose tests). Then, the test statement is displayed and evaluated.

ares ~/TPEiffel/loo-ast-schittly $ ./loo\_test

Nb of test you choose (between 1 and 15) [see documentation for more information]: 5

let

a := 5

b := 7

in

if ((a+b)<15) then let

a := 1

b := 2

in

a := (a\*b)

a

end

else 0

end

Result: 2

# How to test AST application ?

As you can see, when you want to start AST application, you must indicate an integer value. This corresponds to the number of test you want to activate.

12 developed were programmed as follow (1 to 5 are correct tests whereas 6 to 12 are incorrect ones):

## Correct tests

### *Test n°1*

This first test is used to check a basic statement, a binary operation between two statements.

(3 – 10) \* 5

Value: -35

### *Test n°2*

We check thanks to this second test a basic if statement.

if (12 < 10 + 3)

then

2

else

8

Value: 2

### *Test n°3*

We check here that substitution of a variable by its integer value functions correctly.

let

a := 5

in

a

end

Value: 5

### *Test n°4*

We check here that environment was well designed.

If several let statements have same variables that are declared, these are only used in their respective dictionary.

let

a := 3

b := 2

c := 4

in

let

a := 0

b := 1

in

a + b

end

a

end

Value: 3

### *Test n°5*

This test is done on if statement.

An integer is returned by both then and else conditions.

So we check here that let statement returns the good integer following if condition.

let

a := 5

b := 7

in

if (a + b < 15)

then

let

a := 1

b := 2

in

a := a \* b

a

end

else

0

end

Value: 2

### *Test n°6*

This test is done on assignment.

We check here that assignment functions with any statement that returns an integer.

let

a := 6

b := 2

in

a := let

a := b

c := 3

in

if (a = c - 1)

then

a

else

c

end

a

end

Value: 2

### *Test n°7*

This test is done on a complex binary statement.

We check here that this one functions with any statement: so it's possible to have an statement composed by several ones.

Indeed, {exp1 op exp2} is an statement.

let

a := 3

b := 2

c := 4

d := 11

in

let

a := c + 2

b := d

in

a + b

end

\*

if (a = 3)

then

let

c := (a - b) / d

in

c

end

else

a

+ d

end

Value: 11

## Incorrect tests

### *Test n°8*

This test is done in case of a non declared variable through a basic binary operation.

a + 1 => *a isn’t declared*

### *Test n°9*

This test is done on variables' visibility, also in case of an undeclared variable.

If a variable is defined in a new dictionary, it can't be accessed when the dictionary is removed (example is given as follow when several let statements are used).

let

a := 3

in

let

b := a

in

b

end

\* (b + a) => *b can't be used any more*

end

### *Test n°10*

This test is done on an incorrect program which returns a void value.

Indeed, the fact that a program returns an integer is mandatory.

let

a := 10

in

if (a < 5)

then => *This program is wrong as it doesn't return an integer*

a := a - 2

else

a := a / 2

end

### *Test n°11*

This test is done on binary statement.

A binary statement can't be evaluated if at least one of both nodes returns void.

let

a := 3

b := 5

in

a := b + if (a = 3) then a := b else a := 0 => *Wrong as a void value is returned*

a

end

### *Test n°12*

This test is done on the assignment.

An assignment can't be evaluated if statement isn't an integer.

let

a := 100

b := 50

c := 4

in

a := b := c + 3 => *Impossible as {b := c + 3} is an assignment too (void is returned)*

end

### *Test n°13*

This test is done on the environment.

A variable can't be declared more than one time in the same dictionary.

let

a := 2

b := 3

in

let

a := b + 1 => *Wrong as "a" is declared twice in the same dictionary*

a := 3

in

a

end

end

### *Test n°14*

This test is done on declarations.

let

a := b => *Impossible because "b" isn't defined in the current dictionary*

in

let

b := 2

in

b

end

end

### *Test n°15*

This test is done on if statement.

Both statements that make then and else conditions must have the same type (void or integer).

let

c := 3

d := 5

in

if (c = d) => *This if statement is wrong because then condition returns an integer*

then *whereas else condition returns a void value*

d

else

d := d / c

end