Table: Language for Tables

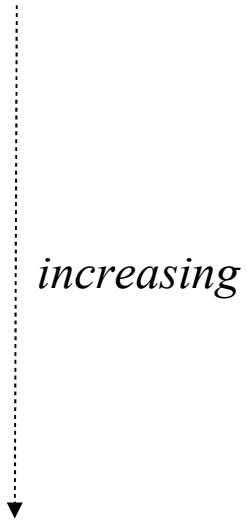
- Hybrid paradigm: imperative + relational
- Data types: atomic (**`integer`**, **`string`**, **`boolean`**) and tables (**`table`**)
- Program = **`program`** [*statements*] **`end`**
  - Definition of variables
  - Assignments (**id** = *expr*)
  - Control statements: **`if-then`**, **`if-then-else`**, **`while-do`**
  - Input of variables and output of expressions: **`read`**, **`write`**
- Expressions:
  - Logical (**`and`**, **`or`**, **`not`**): in short circuit
  - Comparison (**`==`**, **`!=`**, **`>`**, **`>=`**, **`<`**, **`<=`**)
  - Arithmetic (**`+`**, **`-`**, **`*`**, **`/`**)
  - Relational (**`project`**, **`select`**, **`exists`**, **`all`**, **`join`**, **`update`**, **`extend`**, **`rename`**)

Relational Expressions

Operator	*Example*
project	**project** [a,b] R
select	**select** [a==b **and** c != 25] R
exists	**exists** [a!=b **or** c] R
all	**all** [a **and** c] R
join	R **join** [a==b **and** c==d] S
update	**update** [a = b+c] R
extend	**extend** [**integer** d = b-c] R
rename	**rename** [x,y,z] R

Precedence, Associativity, Order of Evaluation

Operator	*Type*	*Associativity*
`and, or`	binary	left
`==, !=, >, >=, <, <=,`	binary	nonassoc
`+, -`	binary	left
`*, /, join`	binary	left
`-, not, project, select, exists, all, update, extend, rename`	unary	right



- Evaluation order of operands: from left to right

Conditional Statement

- **if** *expr* **then** *stat-list* [**else** *stat-list*] **end**

```
integer a, b, c;
table(integer x, string y) t, r;
...
if a==b then
  t = select [x>0] r
else
  a = b+c
end;
...
if t != r then
  write "Different tables!"
end;
...
```

Loop

- **`while`** *expr* **`do`** *stat-list* **`end`**

```
integer n, fact;
read n;
fact = 1;
while n > 1 do
  fact = fact * n;
  n = n -1
end;
write fact;
```

Input

- **read id**
- **read [** *filename* **] id**

```
integer i, j;
table(integer a, string b) t, r, z;
...
read t;
r = select [a == i] t;
read ["t.dat"] t;
z = select [a > j] t;
```

Output

- **write** *expr*
- **write [** *filename* **]** *expr*

```
integer i, j;
string name;
table(integer a, string b) t;
...
write select [a==i] t;
read name;
write [name] select [a>j] t;
```

BNF

program → **program** *stat-list* **end**
stat-list → *stat* **;** *stat-list* | *stat*
stat → *def-stat* | *assign-stat* | *if-stat* | *while-stat* | *read-stat* | *write-stat*
def-stat → *type id-list*
id-list → **id ,** *id-list* | **id**
type → *atomic-type* | *table-type*
atomic-type → **integer** | **string** | **boolean**
table-type → **table (** *attr-list* **)**
attr-list → *attr-decl* **,** *attr-list* | *attr-decl*
attr-decl → *atomic-type* **id**
assign-stat → **id =** *expr*
expr → *expr bool-op bool-term* | *bool-term*
bool-op → **and** | **or**
bool-term → *comp-term comp-op comp-term* | *comp-term*
comp-op → == | **!=** | > | >= | < | <=
comp-term → *comp-term low-bin-op low-term* | *low-term*
low-bin-op → + | -
low-term → *low-term high-bin-op factor* | *factor*
high-bin-op → * | / | *join-op*
factor → *unary-op factor* | **(** *expr* **)** | **id** | *constant*
unary-op → **-** | **not** | *project-op* | *select-op* | *exists-op* | *all-op* | *update-op* | *extend-op* | *rename-op*

join-op → **join [** *expr* **]**
project-op → **project [** *id-list* **]**
select-op → **select [** *expr* **]**
exists-op → **exists [** *expr* **]**
all-op → **all [** *expr* **]**
extend-op → **extend [** *atomic-type* **id =** *expr* **]**
update-op → **update [id =** *expr* **]**
rename-op → **rename [** *id-list* **]**
constant → *atomic-const* | *table-const*
atomic-const → **intconst** | **strconst** | **boolconst**
table-const → **{** *table-instance* **}**
table-instance → *tuple-list* | *atomic-type-list*
tuple-list → *tuple-const* **,** *tuple-list* | *tuple-const*
tuple-const → **(** *atomic-const-list* **)**
atomic-const-list → *atomic-const* **,** *atomic-const-list* | *atomic-const*
atomic-type-list → *atomic-type* **,** *atomic-type-list* | *atomic-type*
if-stat → **if** *expr* **then** *stat-list else-part* **end**
else-part → **else** *stat-list* | **ε**
while-stat → **while** *expr* **do** *stat-list* **end**
read-stat → **read** *specifier* **id**
specifier → **[** *expr* **]** | **ε**
write-stat → **write** *specifier expr*

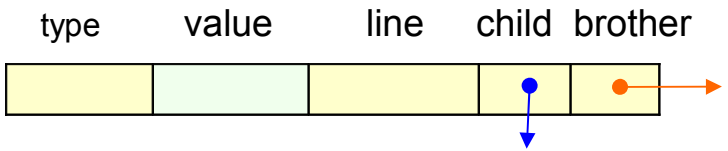
EBNF

program → **program** *stat-list* **end**
stat-list → *stat* { **;** *stat* }
stat → *def-stat* | *assign-stat* | *if-stat* | *while-stat* | *read-stat* | *write-stat*
def-stat → *type id-list*
id-list → **id** { **,** **id** }
type → *atomic-type* | *table-type*
atomic-type → **integer** | **string** | **boolean**
table-type → **table** **(** *attr-list* **)**
attr-list → *attr-decl* { **,** *attr-decl* }
attr-decl → *atomic-type* **id**
assign-stat → **id** = *expr*
expr → *bool-term* { (**and** | **or**) *bool-term* }
bool-term → *comp-term* [(== | **!=** | > | >= | < | <=) *comp-term*]
comp-term → *low-term* { (+ | -) *low-term* }
low-term → *factor* { (* | / | *join-op*) *factor* }
factor → *unary-op* *factor* | **(** *expr* **)** | **id** | *constant*
unary-op → **-** | **not** | *project-op* | *select-op* | *exists-op* | *all-op* | *update-op* | *extend-op* | *rename-op*

join-op → **join** **[** *expr* **]**
project-op → **project** **[** *id-list* **]**
select-op → **select** **[** *expr* **]**
exists-op → **exists** **[** *expr* **]**
all-op → **all** **[** *expr* **]**
extend-op → **extend** **[** *atomic-type* **id** = *expr* **]**
update-op → **update** **[** **id** = *expr* **]**
rename-op → **rename** **[** *id-list* **]**
constant → *atomic-const* | *table-const*
atomic-const → **intconst** | **strconst** | **boolconst**
table-const → **{** *tuple-list* | *atomic-type-list* **}**
tuple-list → *tuple-const* { **,** *tuple-const* }
atomic-type-list → *atomic-type* { **,** *atomic-type* }
tuple-const → **(** *atomic-const* { **,** *atomic-const* } **)**
if-stat → **if** *expr* **then** *stat-list* [**else** *stat*-list] **end**
while-stat → **while** *expr* **do** *stat-list* **end**
read-stat → **read** *specifier* **id**
specifier → **[** *expr* **]** | **ε**
write-stat → **write** *specifier expr*

Abstract Tree

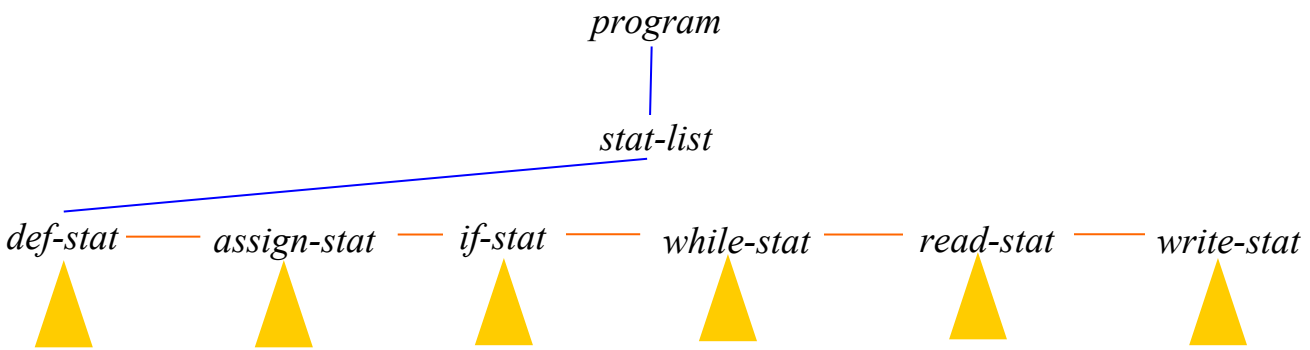
- Node structure:



(value: `ival`, `sval`)

- type ∈ { `N_ASSIGN_STAT, N_ATOMIC_TYPE, N_ATTR_DECL, N_BOOLCONST, N_COMP_EXPR, N_DEF_STAT, N_EXTEND_EXPR, N_ID, N_IF_STAT, N_INTCONST, N_JOIN_EXPR, N_LOGIC_EXPR, N_MATH_EXPR, N_NEG_EXPR, N_PROGRAM, N_PROJECT_EXPR, N_READ_STAT, N_RENAME_EXPR, N_SELECT_EXPR, N_SPECIFIER, N_STAT_LIST, N_STRCONST, N_TABLE_CONST, N_TABLE_TYPE, N_TUPLE_CONST, N_TYPE, N_UPDATE_EXPR, N_WHILE_STAT, N_WRITE_STAT` }
- type = `N_ATOMIC_TYPE` → `value.ival` ∈ { `INTEGER, STRING, BOOLEAN` }
- type = `N_COMP_EXPR` → `value.ival` ∈ { `EQ, NE, '>', GE, '<', LE` }
- type = `N_LOGIC_EXPR` → `value.ival` ∈ { `AND, OR` }
- type = `N_MATH_EXPR` → `value.ival` ∈ { `'+', '-', '*', '/'` }
- type = `N_NEG_EXPR` → `value.ival` ∈ { `'-', NOT` }
- type = `N_SELECT_EXPR` → `value.ival` ∈ { `SELECT, EXISTS, ALL` }

Abstract Tree (ii)



Type Checking

- Traversing of the tree starting from the root

- **Checks**:
  1. Uniqueness of names within environments (attributes in table, variables in blocks)
  2. Visibility within the **environment hierarchy** of a referenced identifier:

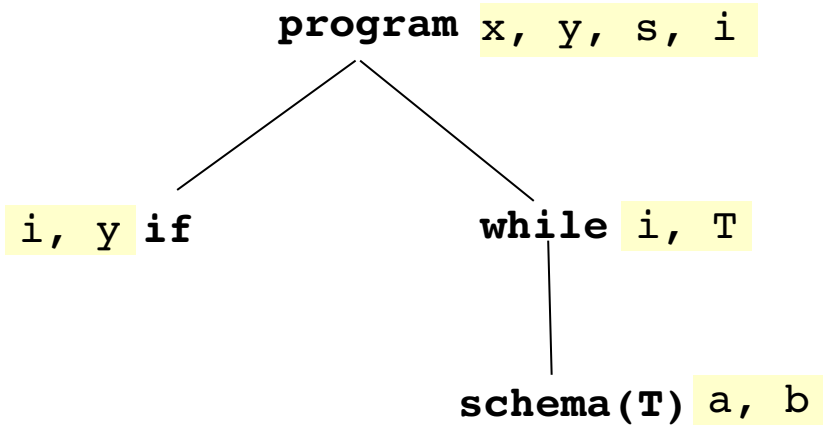
     environment hierarchy = **context hierarchy** + **block hierarchy**
  3. Compatibility of an operator with its operands
  4. Compatibility of the identifier with the assignment expression
  5. Compatibility of expressions with the relevant statements

- **Type inference**: computation of result schema in each operation

Environment Hierarchy

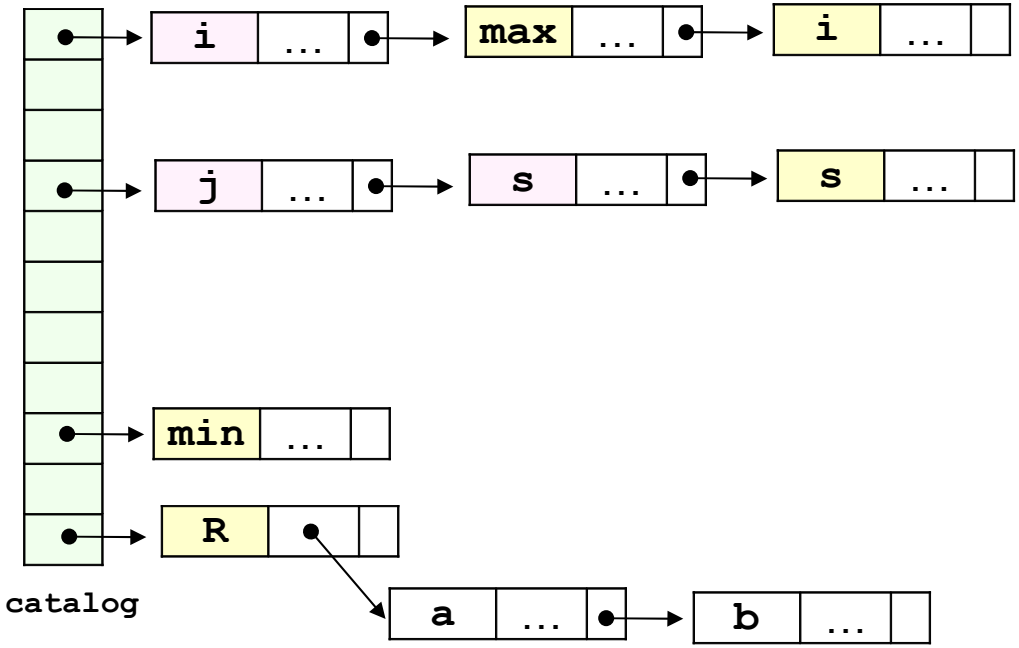
- Problem: binding of a name with the relevant definition
- Search within the environment hierarchy (static scope)

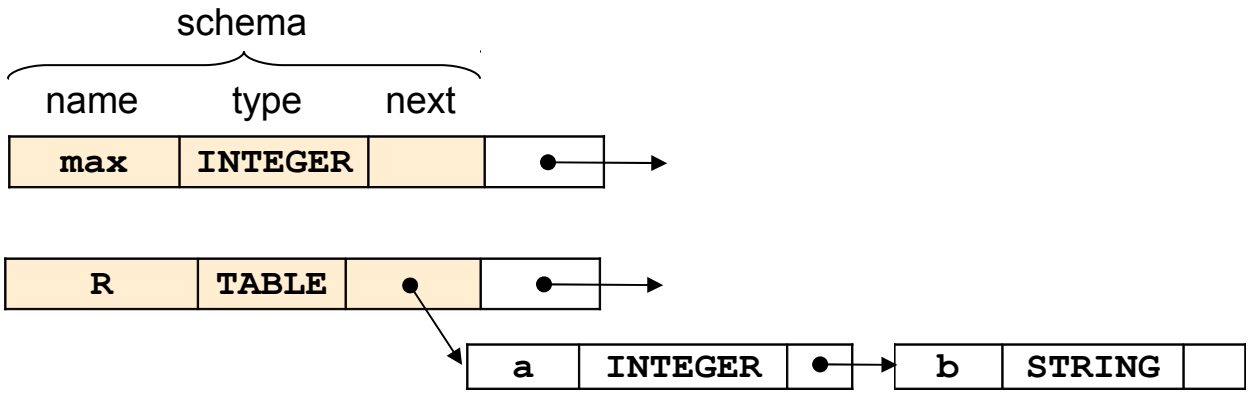
```
program
  real x, y;
  string s;
  integer i;
  ...
  if(x == y) then
    integer i;
    real y;
    ...
  end;
  ...
  while i > j do
    integer i;
    table(integer a, string b) T;
    ...
    write select [ a > i and  b == s ] T
  end
end
```



Symbol Table

```
program
  integer min, max, i;
  string s;
  table(integer a, string b) R;
  ...
  if min == max then
    table(string c, boolean d) T;
    read T;
    write project [a, d]
          R join [ b == c ] T;
  else
    while(min != max) do
      integer i, j, s;
      ...
      min = i + j – max;
      ...
      boolean ok;
      ...
    end
  end;
  ...
end
```





Context Hierarchy

- Contextualizing operators: `select`, `exists`, `all`, `join`, `update`, `extend`

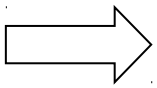
```
table(int a, string x) A;
table(int b, bool y) B, C;
...
C = select [ exists [ a > b ] A ] B
```

A

a	x
3	alpha
8	beta
5	gamma
2	delta
6	epsilon

B

b	y
13	alpha
4	beta
15	gamma
6	delta
24	epsilon
3	zeta



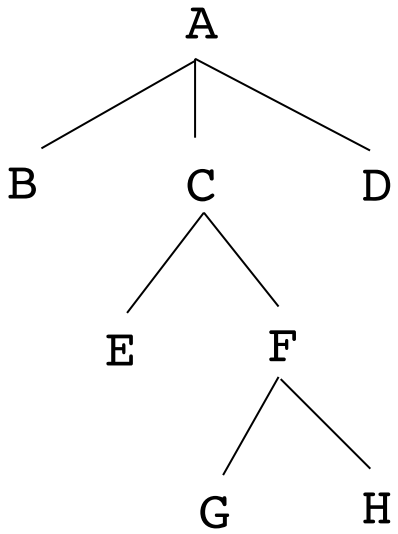
C

b	y
4	beta
6	delta
3	zeta

Context Hierarchy (ii)

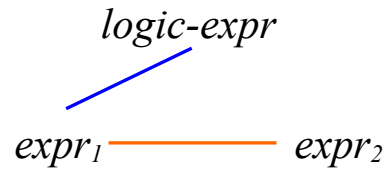
- Search for the name by ascending the context hierarchy

```
[
   [ ] B
   [
      [ ] E
      [
         [ ] G
         [ ] H
      ] F
   ] C
   [] D
] A
```



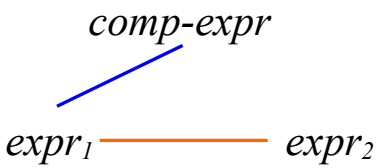
- Type checking: keeps a **context stack** for searching identifiers

Logical Expressions (*logic-expr*)



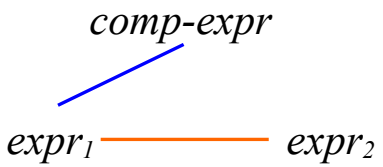
- Qualifier: `AND`, `OR`
- Constraint: type(*expr*$_1$) = **`boolean`**, type(*expr*$_2$) = **`boolean`**
- Result schema: type(*logic-expr*) = **`boolean`**

Comparison Expressions (*comp-expr*)



- Qualifier: `EQ`, `NE`
- Constraint: type(*expr*$_1$) = type(*expr*$_2$)
- Result schema: type(*comp-expr*) = **`boolean`**

Comparison Expressions (*comp-expr*) (ii)

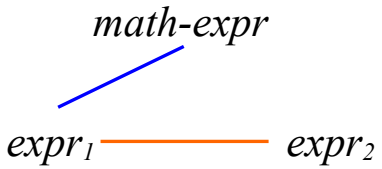


- Qualifier: `'>'`, `GE`, `'<'`, `LE`

- Constraint: (type(*expr*$_1$) = **`int`**, type(*expr*$_2$) = **`int`**) **`or`**
  (type(*expr*$_1$) = **`string`**, type(*expr*$_2$) = **`string`**)

- Result schema: type(*comp-expr*) = **`boolean`**

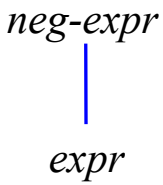
Arithmetic Expressions (*math-expr*)



- Qualifier: `'+'`, `'-'`, `'*'`, `'/'`
- Constraint: type(*expr*$_1$) = **`integer`**, type(*expr*$_2$) = **`integer`**
- Result schema: type(*math-expr*) = **`integer`**

Negation (*neg-expr*)

- Qualifier: `'-'`

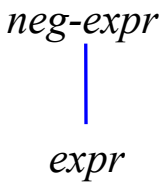


- Constraint: type(*expr*) = **`integer`**

- Result schema: type(*neg-expr*) = **`integer`**

Negation (*neg-expr*) (ii)

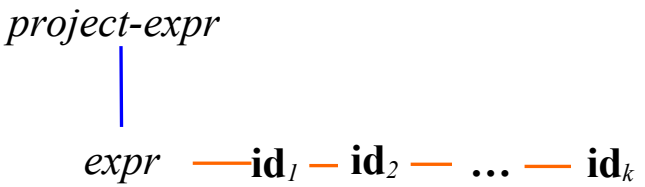
- Qualifier: `NOT`



- Constraint: type(*expr*) = **`boolean`**

- Result schema: type(*neg-expr*) = **`boolean`**

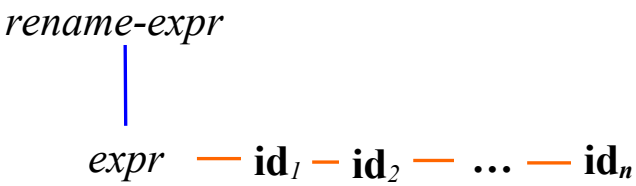
Projection (*project-expr*)



- Constraint: type(*expr*) = **`table`**,
  names(*id-list*) ⊂ names(attributes(type(*expr*))), nodup(names(*id-list*))

- Result schema: type(*project-expr*) =
  `table`(a | a ∈ attributes(type(*expr*)), name(a) ∈ names(*id-list*))

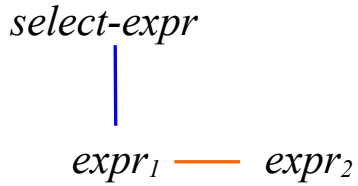
Rename (*rename-expr*)



- Constraint: type(*expr*) = **`table`**,
  | names(*id-list*) | = | attributes(type(*expr*)) | = *n*,
  nodup(*id-list*)

- Result schema: type(*rename-expr*) = **`table`** ⟨*id-list*[i]: type(*expr*)[i] | i ∈ [1..*n*]⟩

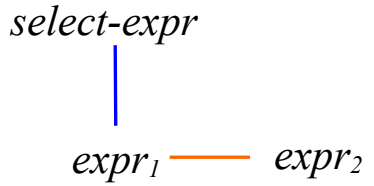
Selection (*select-expr*)



- Qualifier: `SELECT`
- Constraint: type($expr_2$) = **`table`**, type($expr_1$) = **`boolean`**
- Result schema: type(*select-expr*) = **`type`**($expr_2$)

Selection (*select-expr*) (ii)

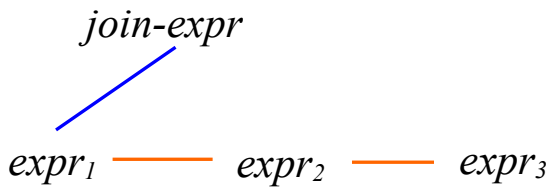
- Qualifier: `EXISTS`, `ALL`



- Constraint: type($expr_2$) = **`table`**, type($expr_1$) = **`boolean`**

- Result schema: type(*select-expr*) = **`boolean`**

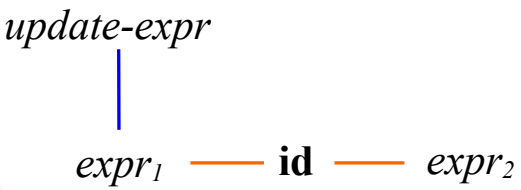
Join (*join-expr*)



- Constraint: type($expr_1$) = **`table`**, type($expr_3$) = **`table`**, type($expr_2$) = **`boolean`**,
  names(attributes(type($expr_1$))) ∩ names(attributes(type($expr_3$))) = ∅

- Result schema: type(*join-expr*) = type($expr_1$) ∪ type($expr_3$)

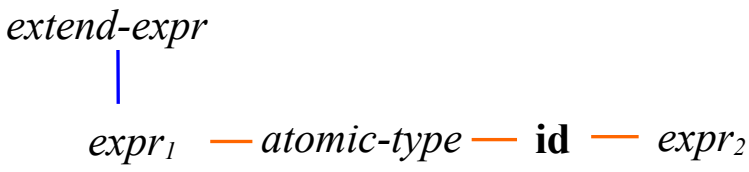
Update (*update-expr*)



- Constraint: type($expr_1$) = **`table`**,
  (name(**id**): domain) ∈ attributes(type($expr_1$)),
  type($expr_2$) = domain

- Result schema: type(*update-expr*) = type ($expr_1$)

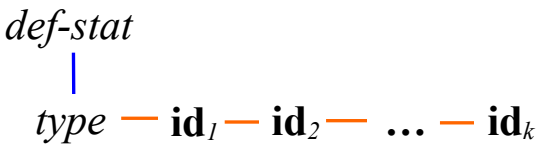
Extension (*extend-expr*)



- Constraint: type($expr_1$) = **`table`**,
  (name(**id**)) $\notin$ names(attributes(type($expr_1$))),
  type($expr_2$) = type(*atomic-type*)

- Result schema: type(*extend-expr*) = type($expr_1$) $\cup$ (name(**id**): type(*atomic-type*))

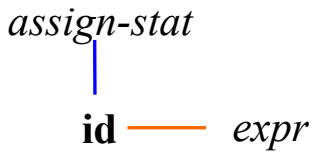
Definition of Variables (*def-stat*)



- Constraint: $\forall\ i \in [1\ ..\ k], \forall\ j \in [1\ ..\ k], i \neq j$, (name(**id**$_i$) ≠ (name(**id**$_j$)),
  $\forall\ i \in [1\ ..\ k]$ (name(**id**$_i$) $\notin$ Environment)

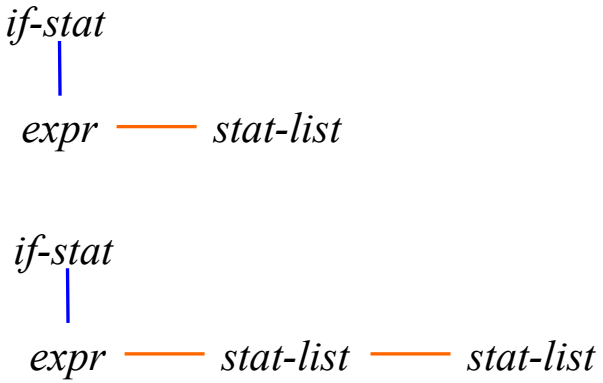
Assignment (*assign-stat*)

- Constraint: visible(name(**id**)),
  type(**id**) = type(*expr*)



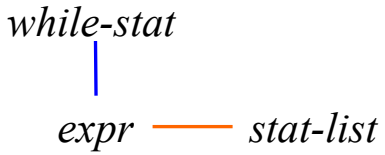
Conditional Statement (*if-stat*)

- Constraint: type(*expr*) = **`boolean`**



While Statement (*while-stat*)

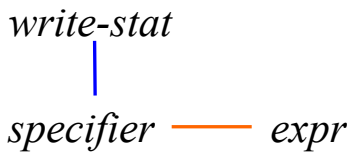
- Constraint: type(*expr*) = **`boolean`**



Reading a Variable (*read-stat*)

- Constraint: visible(name(**id**)),
  (type(*specifier*) = **nil** or type(*specifier*) = **`string`**)

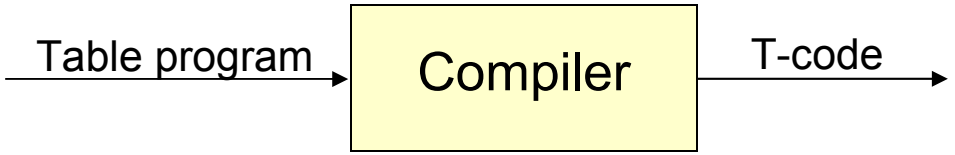
Writing a Value (*write-stat*)



- Constraint: (type(*specifier*) = **nil** or type(*specifier*) = **`string`**)

Intermediate Code Generation

```
program
  integer min, max, i;
  string s;
  table(integer a, string b) R;
  ...
  if min == max then
    table(string c, boolean d) T;
    read T;
    write project [a, d]
          R join [ b == c ] T;
  else
    while(min != max) do
      integer i, j, s;
      ...
      min = i + j – max;
      ...
      boolean ok;
      ...
    end
  end;
  ...
end
```

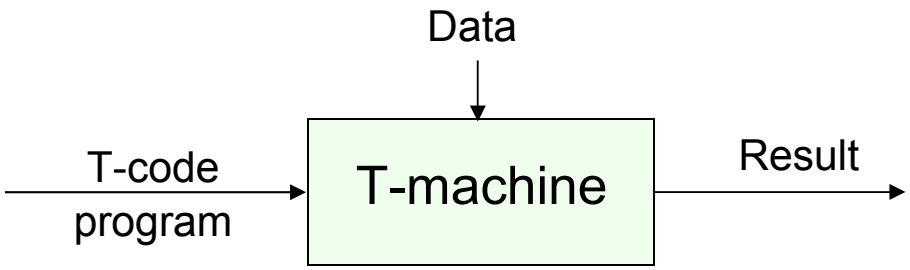


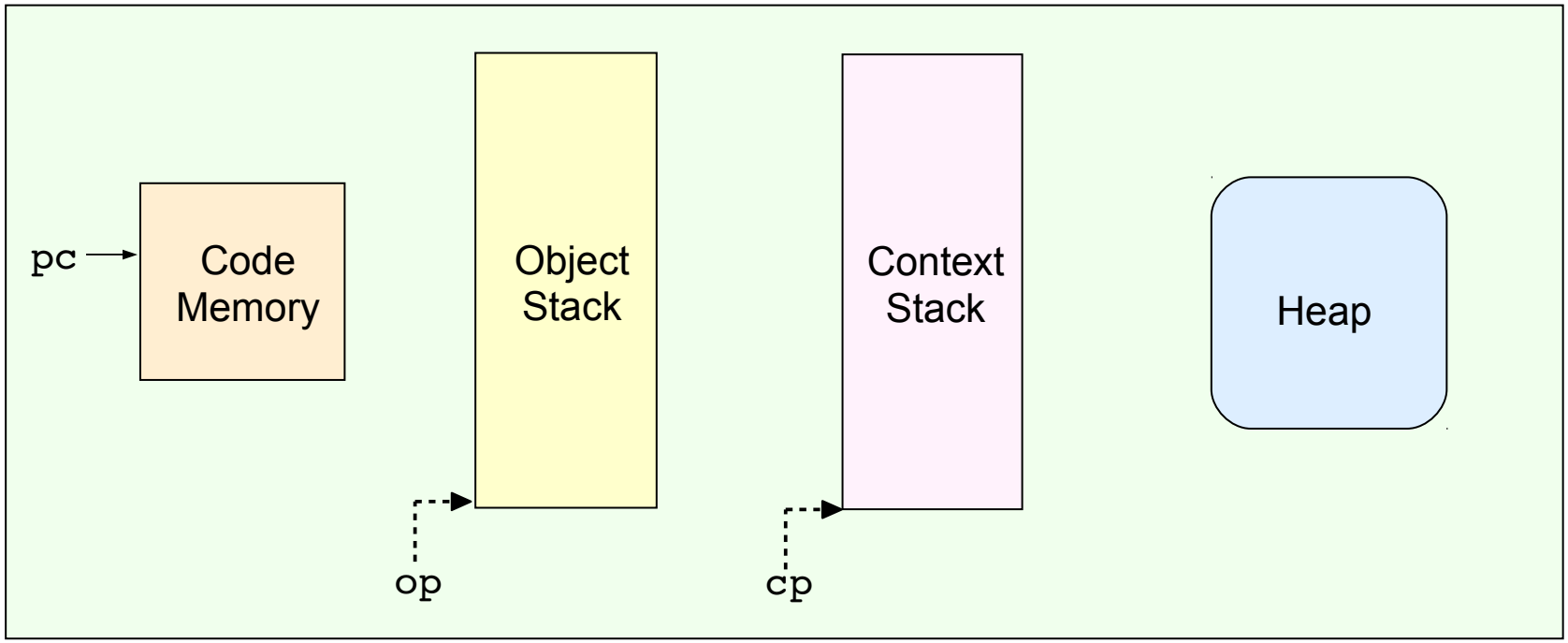
```
TCODE size
  Program
HALT
```

Design Choices

- Code directly addressable (without explicit labels)
- Address of a T-code stat = position of the stat within the generated code
- Variable descriptors allocated in the order in which variables are defined
- Variable identification: *oid* = position of the variable within the stack (0 .. *n*)
- Attribute identification: *context-offset, position-within-schema, attribute-size*
- Management of blocks by stack: exit from block $\rightarrow$ pop variables in block

Abstract Machine





Definition of Variables

- Two types of objects { Atoms, Tables }

```
integer min, max, i;
string s;
boolean b;
table(integer a, string b) R;
```

⇨

```
NEWATOM |integer|
NEWATOM |integer|
NEWATOM |integer|
NEWATOM |string|
NEWATOM |integer|
NEWTAB ⟨|integer|+|string|⟩
```

- **Notes:**
  - atom (embedded instance): `NEWATOM` *object-size*
  - table (instance in heap): `NEWTAB` *tuple-size*

Reference to Atomic Constants

- Integer constant

```
y = 25;
```

```
LDINT 25
```

- String constant

```
s = "alpha";
```

```
LDSTR "alpha"
```

- Boolean constant

```
b = true;
```

```
LDINT 1
```

- **Note:**
  - Boolean values: **`true`**, **`false`** → surrogated by integers: 1, 0

Reference to Table Constants

```
table (integer i, string s, boolean b) t;

t = {(1, "alpha", true),
     (2, "beta", false),
     (3, "gamma, "true")};
```

```
LDTAB ⟨|integer|+|string|+|integer|⟩ 3
  IATTR 1
  SATTR "alpha"
  IATTR 1
  IATTR 2
  SATTR "beta"
  IATTR 0
  IATTR 3
  SATTR "gamma"
  IATTR 1
ENDTAB
```

```
table (integer i, string s, boolean b) t;

t = {integer, string, boolean};
```

```
LDTAB ⟨|integer|+|string|+|integer|⟩ 0
ENDTAB
```

- **Notes:**
  - *tuple-size* = size of tuples
  - *num-tuples* = number of tuples
  - ⟨*attribute loadings*⟩ = sequence of attribute-constant loadings

```
LDTAB tuple-size num-tuples
 ⟨attribute loadings⟩
ENDTAB
```

Reference to Identifiers

- Variable

```
x = y + 1;
```

⇨

```
LOB ^y
```

```
t1 = t2;
```

⇨

```
LOB ^t2
```

```
integer x, y;
table(integer a, boolean b) t1, t2;
table(integer c, string d) t3;
```

- Attribute within current context

```
write
select [d == "alpha"] t3;
```

⇨

```
LAT 0 &d |string|
```

- Attribute within external context

```
write
select [ exists [a > c] t1 ] t3;
```

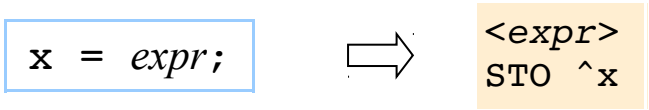
⇨

```
LAT 1 &c |integer|
```

- **Notes:**
  - Argument of `LOB` = *oid* (object identifier)
  - Arguments of `LAT` = *context-offset*, *attribute-offset*, *attribute-size*
  - For reference to attributes within predicate of `join`, *attribute-offset* in `LAT` refers to result schema (concatenation of operand schemas)

Assignment

- Computation of assignment expression + assignment statement (`STO`)



- **Note:**
  - Arguments of `STO` = *oid* (of assigned object)

Logical Operations (`and`, `or`)

logic-expr → *expr*$_1$ *expr*$_2$

and

⟨ *expr*$_1$ ⟩
SKIPF *offset*
⟨ *expr*$_2$ ⟩
SKIP 2
LDINT 0

or

⟨ *expr*$_1$ ⟩
SKIPF 3
LDINT 1
SKIP *exit*
⟨ *expr*$_2$ ⟩

```
boolean a, b, c;
...
a = (b and c) or g;
```

```
LOB ^b
SKIPF 3
LOB ^c
SKIP 2
LDINT 0
SKIPF 3
LDINT 1
SKIP 2
LOB ^g
STO ^a
```

- **Notes:**
  - Short circuit evaluation
  - `SKIP` = unconditional jump
  - `SKIPF` = conditional jump
  - Argument of `SKIP`, `SKIPF` = length of the jump (offset)

$$\begin{cases} exit = |\langle expr_2 \rangle| + 1 \\ offset = |\langle expr_2 \rangle| + 2 \end{cases}$$

Comparison Operations: ==, !=

$comp\text{-}expr \rightarrow expr_1 \ expr_2$

==	!=
⟨ $expr_1$ ⟩	⟨ $expr_1$ ⟩
⟨ $expr_2$ ⟩	⟨ $expr_2$ ⟩
EQU	NEQ

```
integer i, j;
boolean b;

b = (i == j or j != k);
```

```
LOB ^i
LOB ^j
EQU
SKIPF 3
LDINT 1
SKIP 4
LOB ^j
LOB ^k
NEQ
STO ^b
```

- **Note:**
  - EQU, NEQ: polymorphic for all types

Comparison Operations: >, >=, <, <=

comp-expr → *expr*$_1$ *expr*$_2$

	>	>=	<	<=
integer	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ IGT	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ IGE	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ ILT	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ ILE
string	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ SGT	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ SGE	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ SLT	⟨ *expr*$_1$⟩ ⟨ *expr*$_2$⟩ SLE

```
integer i, j;
string x, y;
boolean b;

b = (i > j or x < y);
```

```
LOB ^i
LOB ^j
IGT
SKIPF 3
LDINT 1
SKIP 4
LOB ^x
LOB ^y
SLT
STO ^b
```

Arithmetic Operations: +, –, *, /,

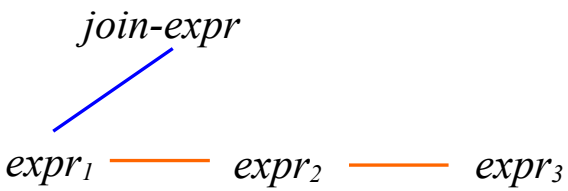
math-set-expr → $\langle expr_1 \rangle$ $\langle expr_2 \rangle$

+	-	*	/
$\langle expr_1 \rangle$	$\langle expr_1 \rangle$	$\langle expr_1 \rangle$	$\langle expr_1 \rangle$
$\langle expr_2 \rangle$	$\langle expr_2 \rangle$	$\langle expr_2 \rangle$	$\langle expr_2 \rangle$
PLUS	MINUS	TIMES	DIV

```
integer i, j, k;


i = (i + 5) * (j – k);
```

```
LOB ^i
LDINT 5
PLUS
LOB ^j
LOB ^k
MINUS
TIMES
STO ^i
```

Join



⇨

```
⟨ expr1 ⟩
⟨ expr3 ⟩
JOIN gap
   ⟨ expr2 ⟩
ENDJOIN gap
```

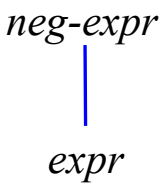
```
table(integer a, boolean b, integer c) r;
table(integer d, integer e) s;

w = r join [ c == a+d–5 and b ] s;
```

⇨

```
LOB ^r
LOB ^s
JOIN 11
  LAT 0 &c |integer|
  LAT 0 &a |integer|
  LAT 0 &d |integer|
  IPLUS
  LDINT 5
  IMINUS
  EQU
  SKIPF 3
  LAT 0 &b |integer|
  SKIP 2
  LDINT 0
ENDJOIN 11
STO ^w
```

- **Notes:**
  - $expr_2$ = predicate of join
  - gap = $|\langle expr_2 \rangle|$
  - Argument *attribute-offset of* `LAT`: computed on result (also for external contexts)
  - Argument of `ENDJOIN`: = implicit jump (backward) to predicate of `JOIN`

Change of Sign

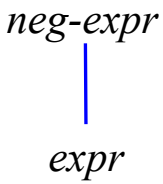


⟨ *expr*⟩
UMI

```
integer i, j;

j = -i + 10;
```

```
LOB ^i
UMI
LDINT 10
PLUS
STO ^j
```

Logical Negation

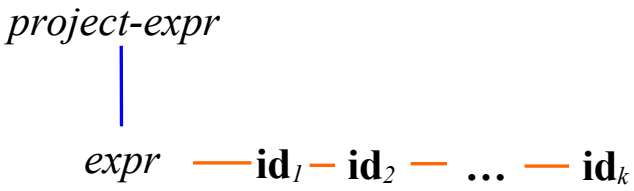


```
⟨ expr⟩
NEG
```

```
integer i, j, k;
boolean a, b;

b = i > j * k and not (a or j);
```

```
LOB ^i
LOB ^j
LOB ^k
TIMES
IGT
SKIPF 8
LOB ^a
SKIPF 3
LDINT 1
SKIP 2
LOB ^j
NEG
SKIP 2
LDINT 0
STO ^b
```

Projection



⇨

⟨ *expr* ⟩
PROJ *k*
  ATTR *attr-offset*$_1$ *size*$_1$
  ATTR *attr-offset*$_2$ *size*$_2$
  ...
  ATTR *attr-offset*$_k$ *size*$_n$
ENDPROJ
REMDUP

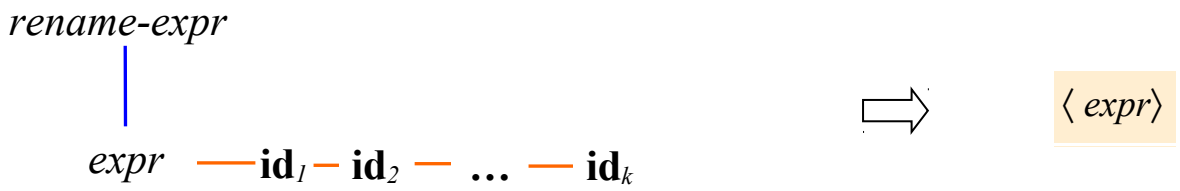
```
table(integer a, boolean b, string c) t;

w := project [ a, c ] t;
```

⇨

```
LOB ^t
PROJ 2
  ATTR &a |integer|
  ATTR &c |string|
ENDPROJ
REMDUP
STO ^w
```

- **Notes:**
  - Argument of PROJ (*k*) = | *id-list* |
  - Each projection attribute is specified by an ATTR
  - Need for duplicate removal by REMDUP

Rename



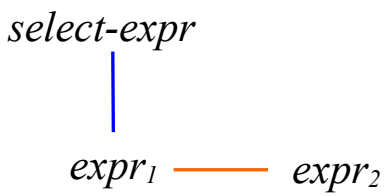

```
table(integer a, boolean b, string c) t;

w := rename [ x, y, z ] t;
```

⇨

```
LOB ^t
STO ^w
```

- **Note:**
  - Rename is relevant to semantic analysis only
  - Code of rename coincides with code of its operand

Selection Operations: SELECT, EXISTS, ALL



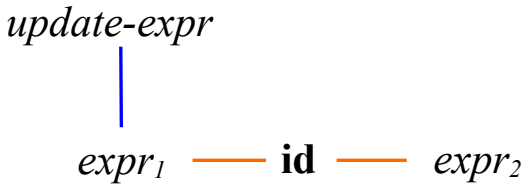
SELECT	EXISTS	ALL
⟨ *expr*$_2$⟩	⟨ *expr*$_2$⟩	⟨ *expr*$_2$⟩
SEL *gap*	EXS *gap*	ALL *gap*
⟨ *expr*$_1$⟩	⟨ *expr*$_1$⟩	⟨ *expr*$_1$⟩
ENDSEL *gap*	ENDEXS *gap*	ENDALL *gap*

```
table(integer a, boolean b) r;
table(integer c, string d) s;

w = select [ b and exists [ c == a ] s ] r;
```

```
LOB ^r
SEL 10
  LAT 0 &b |integer|
  SKIPF 8
  LOB ^s
  EXS 3
    LAT 0 &c |integer|
    LAT 1 &a |integer|
    EQU
  ENDEXS 4
  SKIP 2
  LDINT 0
ENDSEL 10
STO ^w
```

- **Notes:**
  - *expr*$_1$ = selection predicate
  - *gap* = |⟨ *expr*$_1$⟩|

Update



⇨

⟨ $expr_1$ ⟩
UPD *attr-offset size gap*
⟨ $expr_2$ ⟩
ENDUPD *gap*
REMDUP

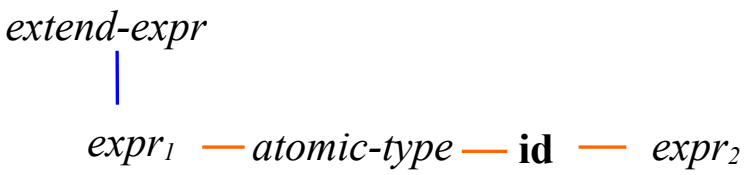
```
integer i;
table(integer a, string b) t;

w = update [ a = i + 8 ] t;
```

⇨

```
LOB ^t
UPD &a |integer| 3
  LOB ^i
  LDINT 8
  PLUS
ENDUPD 3
REMDUP
STO ^w
```

- **Notes:**
  - $expr_2$ = update expression
  - *gap* = |⟨ $expr_2$ ⟩|
  - need for duplicate removal (`REMDUP`)

Extend



⇨

⟨ *expr*$_1$ ⟩
EXT *size gap*
⟨ *expr*$_2$ ⟩
ENDEXT *gap*

```
integer i;
table(integer a, string b) t;

w = extend [ integer c = a - i ] t;
```

⇨

```
LOB ^t
EXT |integer| 3
  LAT 0 &a |integer|
  LOB ^i
  MINUS
ENDEXT 3
STO ^w
```

- **Notes:**
  - *expr*$_2$ = extension expression
  - *size* = size of the new attribute
  - *gap* = |⟨ *expr*$_2$ ⟩|

Read

```
GET  oid  format
```

null specifier

⟨ *specifier* ⟩
`FGET` *oid format*

instantiated specifier

```
boolean k;
string s;
table(integer a, string b) t;

  read k;
  read t;
  read [ s ] t;
```

```
GET ^k "b"
GET ^t "(a:i,b:s)"
LOB ^s
FGET ^t "(a:i,b:s)"
```

- **Notes:**
  - *oid* = object to be instantiated
  - *format* = string specifying the schema of the variable to instantiate

Write

⇨

```
⟨expr⟩
PRINT format
```

null specifier

```
⟨ expr⟩
⟨ specifier⟩
FPRINT format
```

instantiated specifier

```
integer i;
string s;
table(integer a, string b) t;

  write i + 25;
  write [ s ] select [ a >=  i ] t;
  write {(1, "alpha"),(2, "beta")};
```

⇨

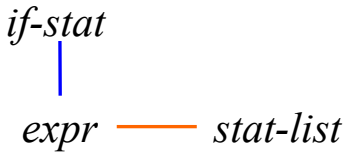
```
LOB ^i
LDINT 25
IPLUS
PRINT "i"
```

```
LOB ^t
SEL 3
  LAT 0 &a |integer|
  LOB ^i
  IGE
ENDSEL 3
LOB ^s
FPRINT "(a:i,b:s)"
```

```
LDTAB ⟨|integer|+|string|⟩ 2
  IATTR 1
  SATTR "alpha"
  IATTR 2
  SATTR "beta"
ENDTAB
PRINT "(?:i,?:s)"
```

- **Notes:**
  - *expr* = object to print
  - *specifier* (if instantiated) = file name where instance is printed
  - *format* = string specifying the schema of the object to instantiate
  - If attribute without name → ?

Conditional Statement: `if-then`



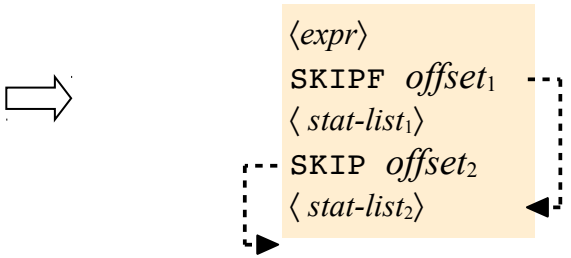
⟨*expr*⟩
SKIPF *offset*
⟨ *stat-list*⟩

```
integer i, j, k;
table(integer a, string b) t, r;

  if i == j then
    t = select [ a > 0 ] r
  end;
```

```
LOB ^i
LOB ^j
EQU
SKIPF 8
LOB ^r
SEL 3
  LAT 0 &a |integer|
  LDINT 0
  IGT
ENDSEL 3
STO ^t
```

- **Note:**
  - *offset* = |⟨*stat-list* ⟩| + 1

Conditional Statement: `if-then-else`

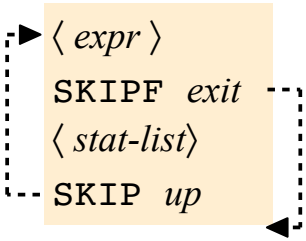



```
integer i, j, k;
table(integer a, string b) t, r;

  if i == j then
    t = select [ a > 0 ] r
  else
    k = i * j
  end;
```

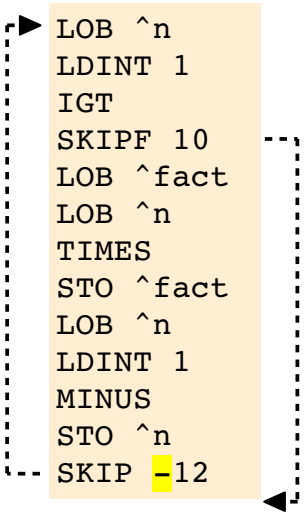
```
LOB ^i
LOB ^j
EQU
SKIPF 9
LOB ^r
SEL 3
  LAT 0 &a |integer|
  LDINT 0
  IGT
ENDSEL 3
STO ^t
SKIP 5
LOB ^i
LOB ^j
TIMES
STO ^k
```

- **Notes:**
  - $offset_1$ = $|\langle stat\text{-}list_1 \rangle|$ + 2
  - $offset_2$ = $|\langle stat\text{-}list_2 \rangle|$ + 1

Loop



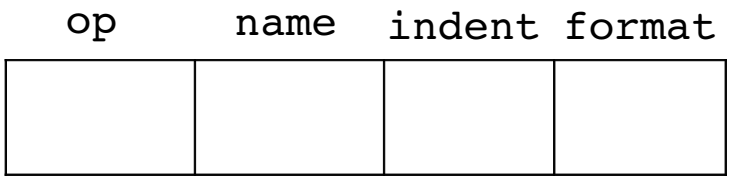

```
integer n, fact;
read n;
fact = 1;
while n > 1 do
  fact = fact * n;
  n = n - 1
end;
write fact;
```



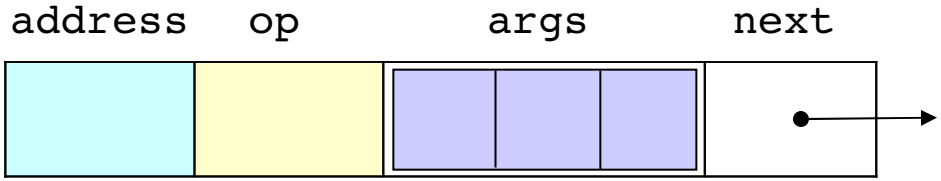
- **Notes:**
  - *exit* = |⟨*stat-list*⟩| + 2
  - *up* = –(|⟨*expr*⟩| + |⟨*stat-list*⟩| + 1)

Data Structures for Code Generation

- `Opdescr`:



- `Tstat`:



- `Code`:

- Representation of a code segment (list of T-code statements):