# Llenguatge Inventat

Marc Llobera Villalonga

*Compiladors 2024-2025*

## Definició del llenguatge

Una regla important del llenguatge que es compleix sempre és que tota línia de codi ha de acabar amb ";" excepte els inici i final de les declaracions de funcions i operacions com *if*, *while*, etc. També, per posar comentaris s'empra /// y tot el que vengui darrera serà un comentari fins trobar un salt de línia (\n).

### Tipus de dades implementades

- **Nombres enters**: aquests es declaren amb la paraula reservada ***integer*** i pot ser un nombre enter qualsevol en base 10.
- **Valors booleans**: aquests es declaren amb la paraula reservada ***logical*** i poden tenir el valor ***TRUE*** o ***FALSE***, sempre han de estar en majúscula.

### Declaració de variables i constants

Les variables i constants sempre que es creen per primer pic s'ha de indicar de quin tipus son.

- **Variables**: aquestes s'indiquen amb la paraula clau ***val*** i després del nom de la variable sempre es posa "::", el tipus de la variable i finalment s'assigna el valor. Per assignar un altre valor a una variable ja declarada simplement es fa amb ***nomVariable = nouValor;***. Es pot assignar un valor a una variable a partir d'una operació aritmètica o lògica però sempre ha d'anar entre "()".

```
val x::integer = 3;  // x = 2;

val faSol::logical = TRUE;

val y::logical = (3 === 3);
```

- **Constants**: s'indiquen amb la paraula clau ***con***. Es declaren igual que una variable però no es pot modificar el valor una vegada declarats.

```
con max::integer = 10;
```

### Tipus definits

Hi ha un tipus especial que és la tupla que funciona com una variable la qual esta definida per dues variables diferents que poden ser de qualsevol tipus. La tupla es declara amb la paraula reservada ***tuple*** i sempre s'ha de indicar el tipus de les seves dues variables dins "{}" i separades per ",". En aquest moment la tupla es buida així que s'ha de afegir el seu valor. Per fer la referència a les variables de la tupla es fa amb el nom de la tupla y l'indicador [posició variable]. D'aquesta manera només has de posar la posició per

obtenir el valor, només es por emprar 0 o 1 per l'índex, emprar un altre valor no està permès.

```
tuple persona = {integer, logical};

persona[0] = 2;

persona[1] = TRUE;

val x::integer = persona[0];
```

## Expressions aritmètiques i lògiques

- **Suma**: la suma s'indica amb el símbol **+** .

```
val x::integer = (3 + 4);
```

- **Resta**: la resta s'indica amb el símbol **-** .

```
val x::integer = (3 – 4);
```

- **Igual**: l'expressió lògica "igual a" s'indica amb el símbol **===** .

```
4 === 4
```

- **Diferent**: l'expressió lògica "diferent a" s'indica amb el símbol **/=** .

```
3 /= 4
```

- **AND**: l'expressió lògica "AND" o "i" s'indica amb el símbol ***&&*** .

```
(x === 3) && (esHome /= FALSE)
```

- **OR**: l'expressió lògica "OR" o "o" s'indica amb el símbol **|| .**

```
(x  /= 3) || (x /= 4)
```

Precedència d'operadors (de major a menor prioritat):

- **Parèntesis**: ( EXP )
- **Aritmètics**: +, - (avaluats de esquerra a dreta)
- **Relacionals/Lògics**: ===, /=, &&, || (avaluats de esquerra a dreta)

## Operacions

- **Assignació**: com ja s'ha vist abans, l'assignació es realitza amb el símbol **=** .

```
val z::logical = FALSE;
```

- **Condicional**: per al condicional s'empra ***if():***, ***else:*** i ***endif*** . No és possible fer ***else if*** .

```
if(x === 12):

/// Codi
```

```
else:
    /// Codi
endif
```

- **Bucle while**: es poden fer bucles de la forma ***while():*** i ***endwhile*** .

```
while(x === 12):
    /// Codi
endwhile
```

Tant al ***while*** com a l'***if*** les condicions han de anar sempre entre "()".

- **Bucle for**: també hi ha els bucles for que es poden fer amb ***for _ to _ : endfor*** . Aquest bucle només serveix per variables que son nombres enters y es sumarà 1 al final de la iteració fins que el comparador vegi que la variable es igual al nombre al que es vol arribar, en tal cas no executarà el codi intern sinó que sortirà directament, així per el primer exemple mostrat *x* iterarà amb els valors: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (10 pics). Hi ha la possibilitat de usar variables darrera el ***for*** tant declarades al moment del bucle com a fora del bucle però sempre han d'anar entre "()", darrera el ***to*** s'aplica el mateix però també es poden emprar constants.

```
for (val x::integer = 0) to 10:
    /// Codi
endfor
////////////////////////////////////
for 1 to 10:
    /// Codi
endfor
////////////////////////////////////
for 33 to (con y::integer = 35):
    /// Codi
endfor
////////////////////////////////////
for (x) to (y):
    /// Codi
endfor
```

## Operacions d'entrada i sortida

- **Entrada**: es pot entrar valors (enters i booleans, sempre indicats entre "()") des de teclat amb l'expressió ***in(tipusValor)***. Es pot assignar a una variable o retornar des d'una funció. El funcionament es com si sigues una funció on el que entres per teclat (si es vàlid) serà retornat. A la pantalla apareix aquest missatge: "Enter: ".

```
val x::integer = in(integer);

con y::logical = in(logical);
```

  Pensa que si li has indicat un integer l'únic que pots escriure per teclat es un nombre enter i si has indicat booleà només pots entrar *FALSE* o *TRUE*.

- **Sortida**: pots fer sortir per consola qualsevol variable, constant o valor directament amb ***out()***. Només es pot imprimir un enter o un booleà però si indiques una variable o constant no fa falta indicar el tipus (no es pot declarar una variable o constant, aquestes ja han d'estar creades). No funciona com una funció per tant no es pot usar per assignar un valor a una variable ni dins un bucle ni res similar, s'ha de usar sempre en solitari, però al seu paràmetre de sortida si es poden usar constants, variables i funcions.

```
out(x);

out(FALSE);

out(10);
```

## Definició i crida de funcions

Per definir una funció emprem les paraules reservades ***fnct tipus ()*** i ***endfnct***, i es poden usar paràmetres sempre que es defineixin amb el seu tipus dins els "()". Es obligatori que la funció retorni un valor i s'ha de indicar el tipus quan es declara la funció. També està la capacitat de retornar valors també definint el tipus a la funció amb la paraula ***rtrn*** i sempre entre "()". Per cridar a la funció es simplement el nom amb els "()" i els paràmetres corresponents i sempre acabant amb ";". La darrera instrucció de la funció ha de ser un return per assegurar que retorna alguna cosa però es poden posar varis ***rtrn*** al cos. Exemple de definicions de funcions:

```
fnct integer suma(x::integer, y::integer):
        rtrn (x + y);
endfnct
//////////////////////////////////////
fnct logical imprimir1():
        out(1);
        rtrn (TRUE);
endfnct
//////////////////////////////////////
```

```
fnct logical sonIguals(x::integer, y::integer):
    con iguals::logical = (x === y);
    rtrn (iguals);
endfnct
////////////////////////////////////
imprimir1();
////////////////////////////////////
val x::integer = suma(1, 2);
```

## Extra

Es poden emprar funcions en lloc de variables o constants en qualsevol lloc on es pugui emprar una variable o constant.

Algunes de les coses que estan permeses son:

```
fnct integer retornaIn():
    rtrn (in(integer));
endfnct
////////////////////////////////////
out(suma(3, 5));
////////////////////////////////////
suma(in(integer), 2);
////////////////////////////////////
for (suma(3, 5)) to (in(integer)):
    /// Codi
endfor
////////////////////////////////////
tuple persona = {integer, logical};
persona[0] = 2;
persona[1] = TRUE;
val x::integer = persona[0];
val esAdult::logical = (persona[0] > 18) && (persona[1] === TRUE);
if (esAdult):
  out(TRUE);
```

```
        else:
            out(FALSE);
        endif
        ////////////////////////////////////
        val x::integer = persona[0];
        val esAdult::logical = (persona[0] > 18) && (persona[1] ===
TRUE);
        ////////////////////////////////////
        fnct logical majorDeEdad (edat::integer):
            if (edat < 18):
                rtrn (FALSE);
            endif
            rtrn (TRUE);
        endfnct

        out(majorDeEdad (20)); /// Imprimeix TRUE
        ////////////////////////////////////
```

## Gramàtica

S -> P

P -> DECL P | DECL

DECL -> VAR_DECL | CONS_DECL | TUPLA_DECL | FUNC_DECL | SENT

VAR_DECL -> val ID :: TIPO = EXP ;

CONS_DECL -> con ID :: TIPO = EXP ;

TUPLA_DECL -> tuple ID = { TIPO , TIPO };

FUNC_DECL -> fnct TIPO ID ( PARAM_LIST ) : COS FINAL_RTN endfnct

PARAM_LIST -> PARAM | PARAM , PARAM_LIST | ε

PARAM -> ID :: TIPO

COS -> SENT COS | SENT

FINAL_RTN -> rtrn ( EXP ) ;

SENT -> ASIG_SENT | IF_SENT | WHILE_SENT | FOR_SENT | TUPLA_ASSIGN | ENTR_SENT | SAL_SENT | CRID_FUNC ;

ASIG_SENT -> ID = EXP ;

IF_SENT -> if ( EXP ) : COS else : COS endif | if ( EXP ) : COS endif

WHILE_SENT -> while ( EXP ) : COS endwhile

FOR_SENT -> for FORINIT to FORLIMIT : COS endfor

FORINIT -> VAR_DECL | EXP

FORLIMIT -> CONS_DECL | EXP

TUPLA_ASSIGN -> TUPLA_ACCESS = EXP ;

ENTR_SENT -> in ( TIPO );

SAL_SENT -> out ( EXP );

CRID_FUNC -> ID ( ARGS ) | ID ()

ARGS -> EXP ARGS_LIST

ARGS_LIST -> , EXP ARGS_LIST | ε

TIPO -> integer | logical

EXP -> ARIT_TERM | LOG_TERM | CRID_FUNC | ID | TUPLA_ACCESS | LIT

ARIT_TERM -> ARIT_TERM + ARIT_TERM | ARIT_TERM - ARIT_TERM | ( EXP )

LOG_TERM -> EXP === EXP | EXP /= EXP | EXP && EXP | EXP || EXP

TUPLA_ACCESS -> ID [ ENT_LIT ]

LIT -> ENT_LIT | BOL_LIT

ENT_LIT -> - DIGIT | DIGIT

BOL_LIT -> TRUE | FALSE | true | false

DIGIT -> [0-9]+

ID -> [a-zA-Z][a-zA-Z0-9]*