Taula de traducció de llenguatge algorísmic a llenguatge C

boolea
cert, **fals**
:=
no, i, o, =, ≠, <, ≤, >, ≥
	`bool`
	`TRUE, FALSE`
	`=`
	`!, &&, ||, ==, !=, <, <=, >, >=`
	`!cal definir-lo amb anterioritat com`
	`typedef enum {FALSE, TRUE} bool;`

caracter
'a', 'W', '1'
:=
=, ≠, <, ≤, >, ≥
	`char`
	`'a', 'W', '1'`
	`=`
	`==, !=, <, <=, >, >=`

enter
3, 465434, -2134567
- (canvi de signe), +, -, *, div, mod, :=
=, ≠, <, ≤, >, ≥
	`int`
	`3, 465434, -2134567`
	`- (canvi de signe), +, -, *, /, %, =`
	`==, !=, <, <=, >, >=`

real
0.6, 5.0E-8, -49.22E+0.8, 1.0E5, 4.0
- (canvi de signe), +, -, *, /, :=
=, ≠, <, ≤, >, ≥
	`float`
	`0.6, 5.0E-8, -49.22E+0.8, 1.0E5, 4.0`
	`- (canvi de signe), +, -, *, /, =`
	`==, !=, <, <=, >, >=`

realAEnter(r) enterAReal(e) caracterACodi(c) codiACaracter(e)	`(int) r` `(float) e` `(int) c` `(char) e`

e := llegirEnter(); escriureEnter(e);
r := llegirReal() ; escriureReal(r);
c := llegirCaracter(); escriureCaracter(c);
escriureCaracter('a')
	`scanf("%d", &e);` `printf("%d ", e);`
	`scanf("%f", &r);` `printf("%f ", r);`
	`scanf("%c", &c);` `printf("%c ", c);`
	`printf("a");`
	`#include <stdio.h>` (a l'inici del programa)

const
nom: *tipus* = *valor*;
fconst
	`#define NOM valor`
	(o també)
	`const tipus nom = valor;`

tipus *nom* = {v_1, v_2, ... ,v_n}; **ftipus** **tipus** *nom* = *definició*; **ftipus**	`typedef enum { v1, v2, ... ,vn } nom ;` `typedef definició nom;`
var *nom*: *tipus*; **fvar** **var** n_1, n_2, n_3: *tipus*; **fvar**	`tipus nom;` `tipus n1, n2, n3;`

nom: **taula**[*mida*] **de** *tipus*;
nom: **taula**[$mida_1$, $mida_2$, ..., $mida_n$] **de** *tipus*;
nom: **taula**[$mida_1$][$mida_2$] ... [$mida_n$] **de** *tipus*;
índex de 1 a mida
	`tipus nom[mida];`
	`tipus nom[mida1][mida2] ... [midan];`
	`tipus nom[mida1][mida2] ... [midan];`
	`índex de 0 a mida-1 (MOLT IMPORTANT)`

nom: **tupla**
  $camp_1$: $tipus_1$;
  $camp_2$: $tipus_2$;
  ...
  ftupla
nom.camp
	`struct {`
	`  tipus1 camp1;`
	`  tipus2 camp2;`
	`  ...`
	`} nom;`
	
	`nom.camp`

si *expressió* **llavors**
  $acció_a$
sino
  $acció_b$
fsi
	`if (expressió) {`
	`  acció_a`
	`} else {`
	`  acció_b`
	`}`

mentre *expressió* **fer**
  acció
fmentre
	`while (expressió) {`
	`  acció`
	`}`

per *idx := v_ini* **fins** *v_fi* **fer**
  acció
fper
	`for (idx = v_ini; idx <= v_fi; idx++) {`
	`  acció`
	`}`

accio *nom*(pm_1, pm_2, ..., pm_n)
  ... cos de l'acció
faccio
nom(obj_1, obj_2, ..., obj_n);
	`void nom(pm1, pm2, ..., pmn)`
	`{`
	`  ... cos de l'acció`
	`}`
	`nom(obj1, obj2, ..., objn);`

funcio *nom*(pm_1, pm_2, ..., pm_n): *tipus*
  ... cos de la funció
  retorna *expressió;*
ffuncio
 	
nom(obj_1, obj_2, ..., obj_n)
	`tipus nom(pm1, pm2, ..., pmn)`
	`{`
	`  ... cos de la funció`
	`  return expressió;`
	`}`
	`nom(obj1, obj2, ..., objn)`

ent *nom: tipus* **sor** *nom: tipus* **entsor** *nom: tipus*	*tipus nom* *tipus *nom* (**nom* en el cos, *&nom* en la crida**) *tipus *nom* (**nom* en el cos, *&nom* en la crida**) ** **Mirar excepció al manual de C (pàg. 22)**

ent *nom: tipus_taula*
sor *nom: tipus_taula*
entsor *nom: tipus_taula*
	`const tipus_taula nom`
	`tipus_taula nom`
	`tipus_taula nom`

arrel(r)	`#include <math.h>` (a l'inici del programa) `sqrt(r)`

funcio *nom*(pm_1, ..., pm_n): *tipus_taula*
  ... cos de la funció
  retorna *nom_taula;*
ffuncio
	`void nom(pm1, ..., pmn, tipus_taula nom_taula)`
	`{`
	`  ... cos de la funció`
	`}`

accio *nom*(**entsor** *nom tipus_tupla*)
  nom.camp
faccio
	`void nom(tipus_tupla *nom)`
	`{`
	`  (*nom).camp`
	`o també`
	`  nom->camp`
	`}`