Fiche de traduction "notation algorithmique ↔ C Ansi"

Dans la colonne de gauche, la notation algorithmique, dans la colonne de droite, les traductions C proposées.

Constantes et macro-définitions

i : la constante 50 de type entier	`#define i 50`

Types simples

entier	`int, long, short`, ...
réel	`float, double`
caractère	`char`
booléen	pas d'équivalent direct. Une expression est vraie si elle est $\neq 0$ et fausse si $= 0$.

Types construits, déclaration de types

T : le type ...	`typedef ... T ;`
T : le type tableau sur [0..N-1] de X	`typedef X T[N] ;`
T : le type < A : un entier, B : un réel >	`typedef struct { int A ; float B ; } T;`
T : le type [A, B, ...]	`typedef enum A, B, ... T ;`
T : le type pointeur de Cel	`typedef Cel *T ;`

Déclaration de variables

X : un typeT	`typeT X ;`
ch : un tableau sur [0..N-1] de caractères	`char ch[N] ;`
a, b, c : des entiers	`int a, b, c ;`

Accès aux tableaux

Ch : le type tableau sur [0..N-1] de caractères	`typedef char Ch[N] ;`
C1, C2 : des Ch	`Ch C1, C2 ;`
i : un entier sur 0..N-1	`int i ;`
C1[i] ou $C1_i$	`C1[i]`

Accès aux structures

T : le type < A : un entier, B : un réel >	`typedef struct { int A ; float B ; } T;`
toto : un T	`T toto ;`
A de toto ← 1	`toto.A = 1 ;`

Manipulation de pointeurs

T : le type pointeur de Cel	`typedef Cel *T;`
p : un T	`T p ;`
p↑	`*p`
C : une Cel	`Cel C ;`
&C	`&C`

Opérateurs courants

←	`=`
= *(prédicat d'égalité)*	`==`
<, >, ≠	<, >, `!=`
etpuis, oualors, non	`&&, ||, !`
+, -, *, /, mod	`+, -, *, /, %`

Compositions d'actions

si C alors A1 sinon A2	`if (C) { A1 ; } else { A2 ; } ;`
i parcourant 1..N : A	`for (i = 1 ; i <= N ; i=i+1) { A ; } ;`
i parcourant 1..N en sens inverse : A	`for (i = N ; i >= 1 ; i=i-1) { A ; } ;`
tantque C : A	`while (C) { A ; } ;`
répéter A jusqu'à C	`do { A; } while (! C) ;`

Structure selon avec conditions de la forme selon X=*constante*

a : l'entier ...	`#define a ...`
b : l'entier ...	`#define b ...`
c : l'entier ...	`#define c ...`
X : un entier	`int X ;`
selon X :	`switch (X) {`
X = a : A1 ;	`case a : A1 ; break ;`
X = b : A2 ;	`case b : A2 ; break ;`
X = c : A3 ;	`case c : A3 ; break ;`
	`} ;`

Structure selon avec conditions générales

T1, T2 : des types	`T1 X ; T2 Y ;`
X : un T1, Y : un T2	
selon X, Y :	`if (C1 (X, Y)) { A1 ; }`
C1 (X, Y) : A1	`else if (C2 (X, Y)) { A2 ; }`
C2 (X, Y) : A2	`else if (C3 (X, Y)) { A3 ; } ;`
C3 (X, Y) : A3 ;	

Entrées et Sorties

x : un entier ; Lire(x)	`int x; scanf("%d",&x);`
Ecrire("coucou")	`printf("coucou");`
x : un entier ; Ecrire(x)	`printf("%d",x);`

Formats courants des printf, scanf

entier, réel	`%d`, `%lf`
caractère, chaîne de caractères	`%c`, `%s`

Fonctions et Actions

T, T', T" : des types	`typedef ... T ;`
	`typedef ... T' ;`
	`typedef ... T'' ;`

f : T, T' → T" f (x, y) : ...	`T'' f (T x, T' y)` `{ return ... ; }`

a : une action (
la donnée x : un T,
le **résultat** y : un T')
 	
lexique : z : un T" ;
a (x, y) :
z ← x
y ← x + z
y ← **y** + 1
	`void a (`
	`T x,`
	`T' *y )`
	`{`
	`T'' z ;`
	
	`z = x ;`
	`*y = x + z ;`
	`*y = *y + 1 ;`
	`}`

x : un T ; y : un T' ; z : un T"
a(x,**y**)
z = f(x,y)
	`T x; T' y; T'' z;`
	`a(x,&y);`
	`z = f(x,y);`