

LANGAGE DE PROGRAMMATION C

Généralités

2

- Le langage **C** a été mis au point par **Ritchie** et **Kernighan** au début des années 70. Leur but était de permettre de développer un langage qui permettrait d'obtenir le système d'exploitation **UNIX**.
- Un des principaux intérêts du **C** est que c'est un langage très portable: Un programme écrit en C en respectant la norme ANSI est portable sans modifications sur n'importe quel système d'exploitation disposant d'un compilateur C : Windows, UNIX...

Structure d'un programme C

3

- Le fichier source d'un programme en langage C est un simple fichier texte dont l'extension est par convention «.c ».
- Un programme C peut être composé de plusieurs fonctions en plus de la fonction principale **main()**.

```
#include <stdio.h>      /*Entête*/
#include <... . h>
#define                ...
main(){                  /* La fonction principale*/
    type1      var1;
    ...                      /* Déclarations des variables*/
    typen      varn;
    Instruction 1
    ...
    Instruction n
}
```

Les variables

4

- **Rappel :** La mémoire est l'endroit où s'exécute les programmes, donc toutes les instructions et les données du programme se trouvent en mémoire.
- La mémoire est organisé sous la forme suivante :
 - ▣ **Les adresses :** une adresse est un nombre qui permet de se repérer dans la mémoire vive, la première adresse est 0.
 - ▣ **Les valeurs :** A chaque adresse, on peut stocker une valeur (un nombre). On ne peut stocker qu'un nombre par adresse !

Adresse		valeur
0		0.321456
1		7658
2		23
3		9.536883
4		547652
...		...
9 123 432 567		1.000028
...		...

Les variables

5

□ Définition d'une variable:

Une variable est un emplacement mémoire nommé qui peut contenir une valeur pouvant être modifiées pendant l'exécution du programme.

□ *Déclaration*

`type nomVariable ;`

Exemple :

`int n;`

`float a, b, c;`

`double x, y, z = 0.5;`

Les Types en Langage C

6

Type	En C	Nombre d'octets	Format
Caractère	char	1	%c
chaîne de caractère	char *	-	%s
entier court	short int	2	%d ou %i
Entier	int	4	
	long int	4	
entier non signé	unsigned int	4	%u
Réel simple	float	4	%f, %e, E, %g
réel double	double	8	
	long double	12	

Les codes des formats

7

- **x, o** : Afficher un entier dans le format hexadécimal respectivement octal
- **f** : Afficher un float ou un double en notation décimale
- **e, E** : Afficher un float ou un double en notation scientifique avec un petit **e** (respectivement **E**)
- **g, G** : Afficher un float ou un double (utilise le format le plus adapté)
- Les codes format utilisés dans **scanf** (*Voir paragraphe lecture et écriture*) sont à peu près les mêmes que dans **printf**, sauf pour les flottants notamment.

Code format	Utilisation
f, e, g	float
lf, le, lg	double
Lf, Le, Lg	long double

Les Constantes

8

En pratique, nous utilisons souvent **des valeurs constantes**:

- pour calculer,
- pour initialiser des variables,
- pour les comparer aux variables, etc.

Dans ces cas, *l'ordinateur doit attribuer un type numérique* aux valeurs constantes.

Pour **pouvoir prévoir le résultat et le type exact des calculs**, il est important pour le programmeur de **connaître les règles selon lesquelles l'ordinateur choisit les types pour les constantes**.

Initialisation des variables

9

Initialisation

En C, il est possible **d'initialiser les variables lors de leur déclaration**:

```
int MAX = 1023;  
char TAB = '\t';  
float X = 1.05e-4;
```

const

En utilisant l'attribut **const**, nous pouvons indiquer que la valeur d'une variable ne change pas au cours d'un programme:

```
const int MAX = 767;  
const double e = 2.71828182845905;  
const char NEWLINE = '\n';
```

Les fonctions arithmétiques standard

10

- ❑ Les fonctions suivantes sont prédéfinies dans la **bibliothèque** standard **<math>**.
- ❑ Pour pouvoir les utiliser, le programme doit contenir la ligne:

#include <math.h>

Type des données

Les arguments et les résultats des fonctions arithmétiques sont du type **double**.

Les Fonctions arithmétiques

11

<i>Fonction</i>	<i>Explication</i>
exp(X)	fonction exponentielle
log(X)	logarithme naturel (Népérien)
log10(X)	logarithme à base 10
pow(X,Y)	X exposant Y
sqrt(X)	racine carrée de X
fabs(X)	valeur absolue de X
floor(X)	arrondir en moins
ceil(X)	arrondir en plus
fmod(X,Y)	reste rationnel de X/Y (même signe que X)

Les Fonctions arithmétiques (suite)

12

$\sin(X)$, $\cos(X)$, $\tan(X)$	sinus, cosinus, tangente de X
$\operatorname{asin}(X)$, $\operatorname{acos}(X)$, $\operatorname{atan}(X)$	$\arcsin(X)$, $\arccos(X)$, $\arctan(X)$
$\sinh(X)$, $\cosh(X)$, $\tanh(X)$	sinus, cosinus, tangente hyperboliques de X

Remarque

- ❑ La liste des fonctions ne cite que les fonctions les plus courantes.
- ❑ Pour la liste complète et les constantes prédéfinies voir [<math.h>](#).

Les opérateurs

13

- Les opérateurs **arithmétiques** courants: $+$, $-$, $*$, $/$, $\%$ (modulo)
- Les opérateurs de **comparaison** : $<$, $>$, $<=$, $>=$, $==$ (égalité), $!=$ (différence)
- Les opérateurs **logiques** : $\&$, $\&\&$ (et) , $|$, $||$ (ou) , $!$ (not)
- Les opérateurs d'**affectation** : $=$, $+=$, $-=$, $*=$, $/=$, ...
 - ▣ $A = A + 5;$ \rightarrow $A += 5;$
 - ▣ $B = B * 3;$ \rightarrow $B *= 3;$
 - ▣ $A = A / 3;$ \rightarrow $A /= 3;$
 - ▣ $B = B - 5;$ \rightarrow $B -= 5;$
- Les opérateurs d'**incrément** et de **décrément** : $++$ et $--$
 - ▣ post incrément : $k = i++;$ est équivalent à : $k = i; i = i+1;$
 - ▣ pré incrément : $k = ++i;$ est équivalent à : $i = i+1; k = i;$

Les opérateurs

14

Classes de priorité

Priorité 1 (la plus forte):	()
Priorité 2:	! ++ --
Priorité 3:	* / %
Priorité 4:	+ -
Priorité 5:	< <= > >=
Priorité 6:	== !=
Priorité 7:	&&
Priorité 8:	
Priorité 9 (la plus faible):	= += -= *= /= %=

Instruction d'écriture

15

- La bibliothèque standard `<stdio.h>` contient un ensemble de fonctions qui assurent la communication de la machine avec le monde extérieur.
- **Syntaxe :** `printf("Format", Expr1, Expr2,...)`
 - ▣ **Format :** est une chaîne de caractères dans laquelle sont insérés les caractères représentant la ou les variables à écrire.
 - ▣ Pour chaque variable, un type de conversion est spécifié. Ce type de conversion est décrit par les caractères qui suivent le caractère “%” (voir paragraphe type de base).
- **Exemple :**

```
int x ;  
X=10  
printf("la valeur de x est :%d", x);
```

Instruction d'écriture

16

Spécificateurs de format pour printf

SYMBOLE	TYPE	IMPRESSION COMME
%d ou %i	int	entier relatif
%u	int	entier naturel (unsigned)
%o	int	entier exprimé en octal
%x	int	entier exprimé en hexadécimal
%c	int	caractère
%f	double	rationnel en notation décimale
%e	double	rationnel en notation scientifique
%s	char*	chaîne de caractères

Instruction d'écriture

17

Largeur minimale et précision pour les rationnels

`printf("%4d", 123);` `==>` `_123`

`printf("%4d", 1234);` `==>` `1234`

`printf("%4d", 12345);` `==>` `12345`

`printf("%4u", 0);` `==>` `__0`

`printf("%4X", 123);` `==>` `__7B`

`printf("%4x", 123);` `==>` `__7b`

Instruction d'écriture

18

Largeur minimale et précision pour les rationnels

Pour les rationnels, nous pouvons indiquer la *largeur minimale* de la valeur à afficher et la *précision* du nombre à afficher.

La précision par défaut est fixée à six décimales. Les positions décimales sont arrondies à la valeur la plus proche.

Instruction d'écriture

19

Exemples

<code>printf("%f", 100.123);</code>	<code>==></code>	<code>100.123000</code>
<code>printf("%12f", 100.123);</code>	<code>==></code>	<code>__100.123000</code>
<code>printf("%.2f", 100.123);</code>	<code>==></code>	<code>100.12</code>
<code>printf("%5.0f", 100.123);</code>	<code>==></code>	<code>__100</code>
<code>printf("%10.3f", 100.123);</code>	<code>==></code>	<code>____100.123</code>
<code>printf("%.4f", 1.23456);</code>	<code>==></code>	<code>1.2346</code>

Instruction de lecture

20

□ Syntaxe :

scanf("Format", arg1, arg2, ...)

□ **Format** : est une chaîne de caractères qui décrit la ou les variables à lire.

□ Exemple :

```
int n , ref;  
char c[20] ;  
float p ;  
scanf ("%d", &n) ; /* Lecture d'une seule variable*/  
scanf ("%s%d%f", c, &ref, &p) ; /* Lecture de plusieurs variables à la fois*/
```

□ Remarque:

Le symbole **&** est obligatoire devant les identificateurs car **scanf** attend des adresses et non des valeurs, sauf devant un identificateur de chaîne de caractères qui est déjà une adresse.

Instruction de lecture

21

Indication de la largeur maximale

Pour tous les spécificateurs, nous pouvons indiquer la *largeur maximale* du champ à évaluer pour une donnée. Les chiffres qui passent au-delà du champ défini sont attribués à la prochaine variable qui sera lue !

Exemple

Soient les instructions:

```
int A,B;  
scanf("%4d %2d", &A, &B);
```

Si nous entrons le nombre **1234567**, nous obtiendrons les affectations suivantes:

A=1234

B=56

le chiffre 7 sera gardé pour la prochaine instruction de lecture.

Instruction de lecture

22

Les signes d'espacement

Lors de l'entrée des données, *une suite de signes d'espacement* (espaces, tabulateurs, interlignes) *est évaluée comme un seul espace*.

Dans la chaîne de format, les symboles `\t`, `\n`, `\r` ont le même effet qu'un simple espace.

Exemple

Pour la suite d'instructions

```
int JOUR, MOIS, ANNEE;  
scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
```

les entrées suivantes sont correctes et équivalentes:

12 4 1980

ou 12 004 1980

ou 12 4 1980

Instruction de lecture

23

Formats 'spéciaux'

Si la chaîne de format contient aussi d'autres caractères que des signes d'espacement, alors ces symboles doivent être introduits exactement dans l'ordre indiqué.

Exemple

La suite d'instructions

```
int JOUR, MOIS, ANNEE;  
scanf("%i/%i/%i", &JOUR, &MOIS, &ANNEE);
```

accepte les entrées:	rejette les entrées:
12/4/1980	12 4 1980
12/04/01980	12 /4 /1980

Instruction de lecture

24

Nombre de valeurs lues

Lors de l'évaluation des données, **scanf** s'arrête:

- si la chaîne de format a été travaillée jusqu'à la fin
- ou si une donnée ne correspond pas au format indiqué.

scanf retourne comme résultat *le nombre d'arguments correctement reçus et affectés.*

Instruction de lecture

25

Exemple

La suite d'instructions

```
int JOUR, MOIS, ANNEE, RECU;  
RECU = scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
```

réagit de la façon suivante (- valeur **indéfinie**):

Introduit:	RECU	JOUR	MOIS	ANNEE
12 4 1980	3	12	4	1980
12/4/1980	1	12	-	-
12.4 1980	1	12	-	-
12 4 19.80	3	12	4	19

Structures conditionnelles

26

□ **Forme 1 : La sélection simple**

□ **Syntaxe :**

```
if (condition){  
    Instructions  
}
```

□ **Exemple:**

```
if (age >= 18){  
    printf ("Vous êtes majeur !");  
}
```

Structures conditionnelles

27

□ **Forme 2 : La sélection alternative**

□ **Syntaxe :**

```
if (condition){  
    Bloc Instructions 1  
}  
else{  
    Instructions 2  
}
```

Exemple:

```
if (a==0){  
    printf("Pas de solution") ; }  
else{  
    printf("x=%f",-a/b)  
}
```

Structures conditionnelles

28

```
if (N>0)
if (A>B)
MAX=A;
else
MAX=B;
```

A quel **if** revient le **else** ?

Convention

En **C**, une partie **else** est toujours liée au dernier **if** qui ne possède pas de partie **else**.

Le bloc sera donc
interprété comme suit:

```
if (N>0)
    if (A>B)
        MAX=A;
    else
        MAX=B;
```

Ainsi, pour **N=0**, **A=1** et **B=2**, **MAX** reste inchangé.

Structures conditionnelles

29

□ **Forme 3 : Les opérateurs conditionnels**

Le langage **C** possède **une paire d'opérateurs** qui peut être utilisée comme alternative à **if - else** et qui a l'avantage de pouvoir être intégrée dans une expression:

<expr1> ? <expr2> : <expr3>

- Si **<expr1>** fournit une **valeur différente de zéro**,
→ la valeur de **<expr2>** est fournie comme résultat
- Si **<expr1>** fournit **la valeur zéro**,
→ la valeur de **<expr3>** est fournie comme résultat

Structures conditionnelles

30

□ Forme 3 : Les opérateurs conditionnels

Exemple

La suite d'instructions

```
if (A>B) MAX=A;  
else  MAX=B;
```

peut être remplacée par:

```
MAX = (A > B) ? A : B;
```

Structures conditionnelles

31

□ **Forme 4 : la sélection à choix multiples**

□ **Syntaxe :**

```
switch(variable){  
    case valeur1 : instruction1 ; break ;  
    case valeur2 : instruction2 ; break ;  
    .  
    .  
    case valeurN : instructionN ; break ;  
    :  
    default : instruction N+1  
}
```

□ **Exemple :** afficher le jour correspondant à un code (1 → Lundi, 2 → Mardi...)

```
switch(code){  
    case 1 : printf("Lundi"); break ;  
    case 2 : printf("Mardi"); break ;  
    case 3 : printf("Mercredi"); break ;  
    case 4 : printf("Jeudi"); break ;  
    Case 5: printf("Vendredi"); break ;  
    case 6 : printf("Samedi"); break ;  
    case 7 : printf("Dimanche"); break ;  
    default : printf("Erreur")  
}
```

Les Boucles

32

□ **Forme1 : la boucle while (tant que)**

□ **Syntaxe :**

```
while (condition){  
    Bloc d'instructions  
}
```

□ **Forme2 : la boucle do...while (répéter..jusqu'à)**

□ **Syntaxe :**

```
do{  
    Bloc d'instructions  
} while (condition)
```

□ **Exemple :** affichage les nombres de 1 à 99

```
i=1  
while(i<100){  
    printf("%d ",i);  
    i++;  
}
```

□ **Exemple :** Lecture et calcul de la somme des nombres saisis dont le dernier est nul

```
S=0;  
do{  
    scanf("%d",&n);  
    S+=n;  
} while(n!=0);  
printf("la somme est:",S);
```


Les Boucles

33

□ **Forme3 : la boucle for (pour)**

□ **Syntaxe :**

```
for (exp1 ;exp2 ;exp3) {  
    Bloc d'instructions  
}
```

Avec:

□ **exp1** : Initialisation du compteur

□ **exp2** : Condition de la boucle

□ **exp3** : Pas de variation

□ **Exemple** : calcul de la somme de n premiers nombres

```
S=0;  
for(i=0;i<=n;i++){  
    S=S+i;  
}  
printf("S=%d",S);
```

Remarque: En général, la boucle **for** est utilisée lorsque le nombre d'itérations à effectuer est connu.

Les Conversions de type

34

Possibilité en C de **mélanger des données de différents types** dans une expression.

Avant de pouvoir calculer, les données doivent être converties dans un même type.

La plupart de ces **conversions** se passent **automatiquement**, sans l'intervention du programmeur, qui doit quand même prévoir leur effet.

Parfois il est nécessaire de convertir une donnée dans un type différent de celui que choisirait la conversion automatique; dans ce cas, nous devons forcer la conversion à l'aide d'un opérateur spécial ("**cast**").

Les Conversions de type automatiques

35

Calculs et affectations

Si un opérateur a des opérandes de différents types, les valeurs des opérandes sont *converties automatiquement dans un type commun*.

Ces manipulations implicites convertissent en général des types plus 'petits' en des types plus 'larges'; de cette façon on ne perd pas en précision.

Lors d'une affectation, **la donnée à droite du signe d'égalité est convertie dans le type à gauche du signe d'égalité**. Dans ce cas, il peut y avoir perte de précision si le type de la destination est plus faible que celui de la source.

Exemple

Considérons le calcul suivant:

```
int I = 8;  
float X = 12.5;  
double Y;  
Y = I * X;
```

Pour pouvoir être multiplié avec **X**, la valeur de **I** est convertie en **float** (le type le plus large des deux). Le résultat de la multiplication est du type **float**, mais avant d'être affecté à **Y**, il est converti en **double**.

Nous obtenons comme résultat: **Y = 100.00**

Les Conversions de type automatiques

36

Appels de fonctions

Lors de l'appel d'une fonction, *les paramètres sont automatiquement convertis dans les types déclarés dans la définition de la fonction.*

Exemple

Au cours des expressions suivantes, nous assistons à trois conversions automatiques:

```
int A = 200;  
int RES;  
RES = pow(A, 2);
```

A l'appel de la fonction **pow**, la valeur de A et la constante 2 sont converties en **double**, parce que **pow** est définie pour des données de ce type.

Le résultat (type **double**) retourné par **pow** doit être converti en **int** avant d'être affecté à RES.

Les Conversions de type automatiques

37

Règles de conversion automatique

Conversions automatiques lors *d'une opération avec*,

(1) deux entiers:

D'abord, les types **char** et **short** sont convertis en **int**. Ensuite, l'ordinateur choisit le plus large des deux types dans l'échelle suivante: **int, unsigned int, long, unsigned long**

(2) un entier et un rationnel:

Le type entier est converti dans le type du rationnel.

(3) deux rationnels:

L'ordinateur choisit le plus large des deux types selon l'échelle suivante: **float, double, long double**

(4) affectations et opérateurs d'affectation:

Lors d'une affectation, le résultat est toujours converti dans le type de la destination. Si ce type est plus faible, il peut y avoir une perte de précision.

Les Conversions de type automatiques

38

Exemple

Observons les conversions nécessaires lors d'une simple division:

```
int X;  
float A=12.48;  
char B=4;  
X=A/B;
```

B est converti en **float** (règle 2).

Le résultat de la division est du type **float** (valeur 3.12) et sera converti en **int** avant d'être affecté à **X** (règle 4), ce qui conduit au résultat **X=3** .

Les Conversions de type automatiques

39

Phénomènes imprévus

Le mélange de différents types numériques dans un calcul peut inciter à ne pas tenir compte des phénomènes de conversion et conduit parfois à des résultats imprévus ...

Exemple

Dans cet exemple, nous divisons 3 par 4 à trois reprises et nous observons que le résultat ne dépend pas seulement du type de la destination, mais aussi du type des opérandes.

```
char A=3; int B=4; float C=4; float D,E; char F;  
D = A/C;  
E = A/B;  
F = A/C;
```

* Pour le calcul de **D**, A est converti en **float** (règle 2) et divisé par C. Le résultat (0.75) est affecté à D qui est aussi du type **float**. On obtient donc: **D=0.75**

* Pour le calcul de **E**, A est converti en **int** (règle 1) et divisé par B. Le résultat de la division (type **int**, valeur 0) est converti en **float** (règle 4). On obtient donc: **E=0.000**

* Pour le calcul de **F**, A est converti en **float** (règle 2) et divisé par C. Le résultat (0.75) est retraduit en **char** (règle 4). On obtient donc: **F=0**

Les Conversions de type forcées (casting)

40

Il est possible de convertir explicitement une valeur en un type quelconque en forçant la transformation à l'aide de la syntaxe:

Casting (conversion de type forcée)

(<Type> <Expression>

Exemple

Nous divisons deux variables du type entier. *Pour avoir plus de précision, nous voulons avoir un résultat de type rationnel.* Pour ce faire, nous convertissons l'une des deux opérandes en **float**. Automatiquement C convertira l'autre opérande en **float** et effectuera une division rationnelle:

```
char A=3; int B=4; float C;  
C = (float) A/B;
```

La valeur de *A* est *explicitement* convertie en **float**. La valeur de *B* est *automatiquement* convertie en **float** (règle 2). Le résultat de la division (type rationnel, valeur 0.75) est affecté à *C*.

Résultat: C=0.75

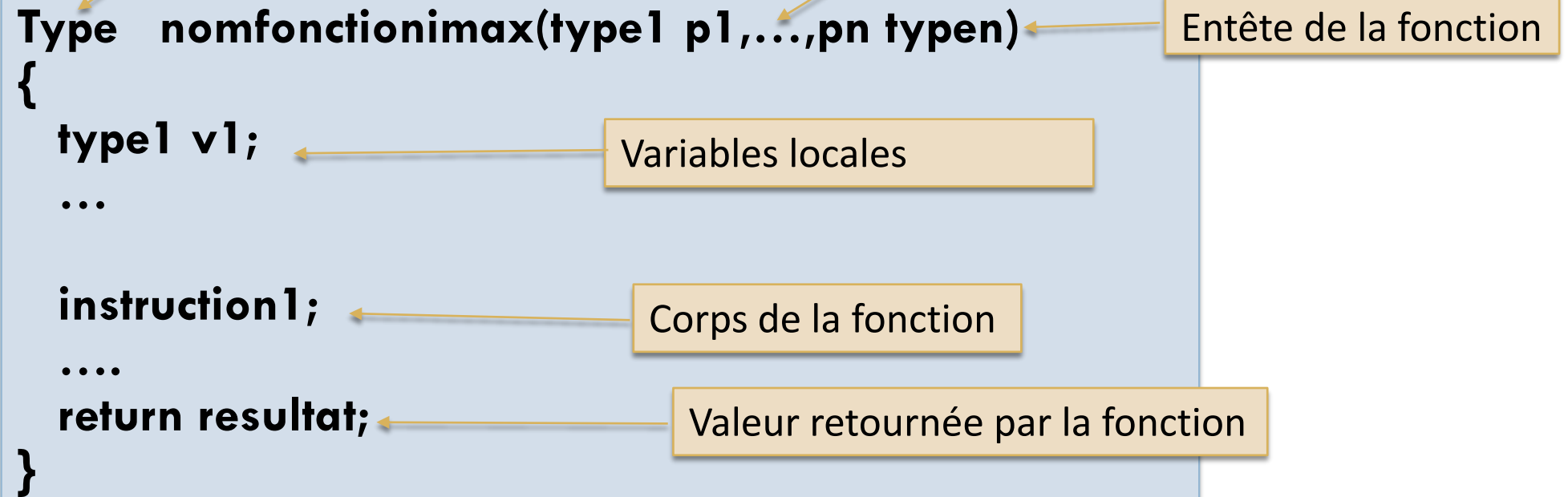
Attention !

Les contenus de *A* et de *B* restent inchangés; seulement les valeurs utilisées dans les calculs sont converties !

Les fonctions en C

41

□ Syntaxe



The diagram illustrates the syntax of a C function with the following components and annotations:

```
Type  nomfonctionimax(type1 p1,...,pn typen)  
{  
  type1 v1;  
  ...  
  
  instruction1;  
  ....  
  return resultat;  
}
```

- Type de retour**: Points to the **Type** at the beginning of the function signature.
- Paramètres de la fonction**: Points to the parameter list **(type1 p1,...,pn typen)**.
- Entête de la fonction**: Points to the entire function signature line.
- Variables locales**: Points to the local variable declaration **type1 v1;**.
- Corps de la fonction**: Points to the body of the function, starting from **instruction1;** and ending at **return resultat;**.
- Valeur retournée par la fonction**: Points to the **return resultat;** statement.

Les fonctions en C

42

- **Exemple:** une fonction qui retourne le minimum de deux entiers passés en paramètre

Définition de la fonction

```
int    maximum(int n,int m)
{
    int max;
    if (n>m)
        max = n;
    else
        max = m;
    return max;
}
```

Appel de la fonction

```
void    main()
{
    int a,b,p;
    scanf("%d",&a);
    scanf("%d",&b);
    p= maximum(a,b);
    printf("le max de a et b:%d",p);
}
```