

IUT de Lannion
Dpt Informatique
1^{ère} année



Langage C

Cours et référence

Pierre NERZIC
mars 2003

Table des matières

A - Présentation	1
1 - Historique du langage C	1
2 - Exemple d'un programme simple	1
B - Apprentissage du langage	2
1 - Le premier programme C	2
2 - Variables simples	2
3 - Entrées/sorties	3
4 - Instructions conditionnelles	4
5 - Boucles itératives	5
6 - Boucles répétitives	6
7 - Fonctions et procédures	7
8 - Tableaux	7
9 - Chaînes de caractères	8
a) Constantes chaînes	8
b) Variables chaînes	9
10 - Types composites : structures	10
11 - Variables globales	10
12 - Pointeurs	11
a) Paramètres de sortie	11
b) Pointeurs et tableaux	12
C - Structure d'un programme C	14
1 - Éléments d'un programme	14
2 - Compilation d'un programme C	15
a) Compilation	15
b) Édition des liens	15
D - Variables et types de données	16
1 - Définition d'une variable	16
a) Syntaxe	16
b) Initialisation	16
2 - Types simples	16
a) Entiers	16
b) Booléens	17
c) Caractères	17
d) Réels	18
e) Énumérations	18
3 - Enregistrements	19
a) Définition d'un type enregistrement	19
b) Définition d'une variable structurée	19
c) Initialisation d'une variable structurée	19
4 - Tableaux	20
a) Définition d'une variable tableau	20
b) Initialisation des éléments	20
c) Définition d'un type tableau	21
d) Chaînes de caractères	21
E - Opérateurs et expressions	23
1 - Opérateurs numériques	23
a) Entiers	23
b) Réels	23
c) Opérateurs binaires	23
2 - Opérateurs logiques	23
3 - Opérateurs divers	24
a) Accès à un élément d'un tableau	24
b) Accès à un champ d'un struct	24
c) Taille d'une variable	24
d) Conversions de type des valeurs scalaires	24
4 - Adresses des variables	25
a) Obtenir l'adresse d'une variable	25
b) Mémoriser une adresse dans une variable	25
c) Accès à une donnée par son adresse	25
d) Tableaux et adresses	26
F - Instructions et structures de contrôle	27
1 - Instructions simples	27
a) Affectations	27
b) Autres instructions	27
c) Comment rendre une instruction illisible	28
2 - Blocs et sous-blocs	28
a) Définitions de variables locales	29
b) Instructions	29
3 - Instructions conditionnelles	29
a) Conditionnelles simples	29
b) Conditionnelles multiples	29
4 - Boucles	30
a) Boucles 'Tant que'	30
b) Boucles 'Jusqu'à'	30
c) Boucles 'Pour'	31
d) Contrôle des boucles	31
5 - Indentation	32
G - Fonctions et procédures	32
1 - Définition d'une fonction	32
a) Définition	32
b) Appel	33
c) Rappels sur la visibilité d'un symbole	33
d) Prototypes et prédéclarations	33
e) Passage par valeur	34
f) Passage par variable	34
g) Cas des paramètres de type tableau	35
2 - Définition d'une procédure	35
3 - Procédures et fonctions standard	36
a) Lecture du clavier	36
b) Affichage sur écran	37
c) Fichiers	38
4 - Librairie mathématique	39
a) Déclaration des fonctions	39
b) Utilisation des fonctions	40
c) Compilation	40
5 - Arguments d'un programme	40
H - Préprocesseur C	41
1 - Inclusion de fichiers	41
a) Présentation	41
b) Directives #include	42
2 - Définition de macros	43
a) Constantes	43
b) Macrofonctions (macros)	43
3 - Compilation conditionnelle	44

A - PRESENTATION

1 - HISTORIQUE DU LANGAGE C

Dennis Ritchie (laboratoires Bell - US) a défini ce langage évolué afin de pouvoir implanter le système Unix sur différents types de machines. Auparavant, les systèmes d'exploitation étaient écrits entièrement en assembleur spécifique à la machine visée. Écrite en langage C, la plus grande partie du système Unix (noyau, bibliothèques et commandes) peut être mise en œuvre sur toute machine possédant un compilateur C. Depuis, ce langage a conquis le monde des entreprises et les universités malgré ses quelques défauts (syntaxe parfois illisible, pas assez de contrôles sémantiques...).

On peut citer deux versions et une extension de ce langage :

- version originale non normalisée : C standard ancienne et amenée à disparaître peu à peu.
- version normalisée : C ansi.
- version objet : C++ (étudiée en deuxième année)

2 - EXEMPLE D'UN PROGRAMME SIMPLE

Les exemples de ce polycopié sont toujours fournis avec deux versions, à gauche un programme C, à droite le programme équivalent en Pascal.

Langage C – facto.c	Langage Pascal – facto.pas
<pre> /* un programme simple */ #define N_MAX 10 int fact(int n) { int i,r; if (n > N_MAX) return -1; r = 1; for (i=1; i<=n; i++) { r = r * i; } return r; } int Nombre; main() { scanf("%d", &Nombre); printf("fact(%d) = %d\n",Nombre, fact(Nombre)); } </pre>	<pre> { un programme simple } program simple; const N_max = 3; function fact(n:integer):integer; var i,r: integer; begin if n > N_max then fact := -1 else begin r := 1; for i:=1 to n do begin r := r * i end; fact := r end end; var Nombre: integer; begin read(Nombre); writeln('fact(',Nombre,',') = ', fact(Nombre)) end. </pre>

Saisir ce programme dans le fichier facto.c puis le compiler par la commande :

```
cc -Wall -o facto facto.c
```

B - APPRENTISSAGE DU LANGAGE

Ce polycopié présente les principes de programmation en langage C en comparant sa syntaxe à celle du langage Pascal sur des exemples simples. Ce polycopié contient deux parties : un cours pour apprendre le langage et une description du langage élément par élément.

Les exemples sont inspirés d'un cours écrit par Kernighan et Ritchie. Ils peuvent être compilés et essayés.

1 - LE PREMIER PROGRAMME C

Voici un programme très simple :

Langage C – <code>salut.c</code>	commentaire
<pre>main() { printf("bonjour à tous\n"); }</pre>	<p>nom de la fonction début de bloc = groupe d'instructions corps du programme, instructions fin du bloc</p>

Comme un programme Pascal, un source C contient des définitions de données : constantes, types, variables et des instructions : fonctions et procédures. Le langage C ne fait pas de différence entre fonctions et procédures, les procédures sont des fonctions dont on n'utilise pas le résultat.

Tout programme exécutable doit contenir une et une seule fonction appelée `main`, cette fonction est le point d'entrée du programme et peut se trouver n'importe où dans le source.

La fonction `printf` permet d'afficher des données mises en page (*print formatted*). Son mode d'emploi est `printf(format, données...)`, voir page 3. Le format est une chaîne de caractères indiquant comment il faut afficher les données. Dans cet exemple, on affiche seulement le format, il n'y a pas de données ; `\n` envoie un retour à la ligne. Il n'y a pas comme en Pascal deux fonctions, l'une pour écrire un retour à la ligne et une autre pour ne pas en écrire – on met ou pas ce `\n` :

Langage C	Langage Pascal
<pre>printf("bonjour à tous\n"); printf("salut");</pre>	<pre>writeln('bonjour à tous'); write('salut');</pre>

Noter les points-virgule obligatoires sur chaque ligne pour indiquer la fin des instructions.

2 - VARIABLES SIMPLES

Langage C – <code>somme.c</code>	commentaire
<pre>main() { int a, b, c, somme; a = 1; b = 2; c = 3; somme = a + b + c; printf("somme=%d\n", somme); }</pre>	<p>définition (création) des variables affectation de valeurs calcul affichage de la variable somme</p>

Remarquer les différences avec le langage Pascal en ce qui concerne les définitions des variables locales : les définitions sont à l'intérieur du bloc, les noms des types sont un peu différents et précèdent la liste des noms

des variables. Remarquer aussi l'affectation, c'est opérateur = (attention à ne pas confondre avec l'opérateur == qui signifie égalité).

Le langage C contient trois types de données fondamentaux :

- char : entier sur 1 octet, contient le code ascii d'un caractère, on peut lui affecter un code ascii ou un caractère. `char c; c = 48; c = 'A'; c = '\n';`
- int : entier sur 4 octets. `int n; n = -15;`
- float : réel IEEE sur 4 octets. `float r; r = 3.14; r = 1e-4;`

Au contraire du Pascal, les variables ne sont jamais initialisées. Une syntaxe particulière permet de le faire au moment de la définition :

```
int nb = 32;
```

La manipulation des chaînes de caractères en C est assez compliquée car il s'agit de tableaux qu'on doit gérer « à la main », voir page 8. Les chaînes sont délimitées par des guillemets "...".

Les opérateurs de calcul sont presque les mêmes qu'en Pascal, + - * / (au lieu de div) et % (mod).

Langage C – nbblocs.c	commentaire
<pre>main() { int taille, taillebloc=15, nbblocs; taille = 36; nbblocs = ((taille-1)%taillebloc)+1; printf("nbblocs=%d\n", nbblocs); }</pre>	définition (création) des variables affectation de valeurs calcul affichage de la variable

Les fonctions mathématiques (sqrt, sin, log...) ne sont pas intégrées au langage, il faut faire appel à des bibliothèques externes (voir G - 4 - page 39).

3 - ENTREES/SORTIES

Langage C – somme2.c	commentaire
<pre>main() { int a=4, b=9; printf("%d + %d = %d\n", a,b, a+b); }</pre>	définition et initialisation de deux variables affichage avec un format

Le premier paramètre de la fonction `printf` est appelé format, c'est une chaîne qui contient du texte normal et des jokers (%d), les jokers sont remplacés par les valeurs fournies, dans l'ordre où elles se trouvent. Il existe différents jokers pour chaque type de données : %c pour un char, %f pour un float.

Pour entrer un entier dans une variable voici comment on fait, le & est indispensable, voir pages 34 et 36 :

Langage C – double.c	commentaire
<pre>main() { int a; scanf("%d", &a); printf("double (%d)=%d\n", a, 2*a); }</pre>	read(a), NB : mettre un & devant la variable à lire !!!

Voici un programme C peu utile qui récrit un caractère qu'on tape :

Langage C – repete.c	commentaire
<pre>main() { char c; printf("taper une touche puis entrée:"); c = getchar(); printf("code de %c = %d\n", c, c); putchar(c); putchar('\n'); }</pre>	<p>définition (création) de la variable</p> <p>read(c) affichage de c et de son code réaffichage de c par putchar affichage d'un retour à la ligne</p>

La fonction `getchar` n'a pas de paramètre mais les parenthèses sont obligatoires. Elle attend qu'on tape une ligne au clavier (stockée dans un tampon) et renvoie le code ascii du premier caractère tapé (ou le caractère suivant si on rappelle cette fonction, ainsi de suite jusqu'à la fin de la ligne).

On fournit un code ascii ou un caractère à la fonction `putchar` et elle affiche le caractère en question.

4 - INSTRUCTIONS CONDITIONNELLES

Langage C – signe.c	commentaire
<pre>main() { float nb; printf("entrer un nombre :"); scanf("%f", &nb); if (nb >= 0.0) printf("positif\n"); else printf("négatif\n"); }</pre>	<p>définition (création) de la variable</p> <p>read(nb)</p> <p>test sur nb</p>

Noter que les chaînes sont mises entre guillemets (double quotes), ex : "oui" et les caractères simples sont mis entre apostrophes, ex : 'c' ; les confondre peut provoquer des erreurs difficile à déceler.

Les conditions sont écrites comme en C-Shell et Awk. Lorsqu'on doit effectuer plusieurs instructions c'est à dire un bloc, on les encadre par des accolades {...} qui jouent le rôle de begin...end.

Langage C – signe2.c	commentaire
<pre>main() { float nb; printf("entrer un nombre :"); scanf("%f", &nb); if (nb >= 0.0) { if (nb > 0.0) { printf("positif\n"); } else { printf("nul\n"); } } else printf("négatif\n"); }</pre>	<p>{ signifie début de bloc (plusieurs instructions ensemble)</p> <p>} signifie fin de bloc</p>

5 - BOUCLES ITERATIVES

Voici un programme C qui applique la méthode dite de Jules César sur la ligne de texte tapée :

Langage C – cesar.c	commentaire
<pre>main() { char c; c = getchar(); while (c != '\n') { putchar((c + 1) % 26 + 'a'); c = getchar(); } putchar(c); }</pre>	<p>Taper toute une ligne à l'avance, retour à la fin.</p> <p>c = code ascii du caractère lu (c+1) modulo 26 = nbre entre 0 et 25 'a' vaut 97 = code de a minuscule, on affiche ce code final et on continue avec le caractère suivant</p> <p>écrire le retour à la ligne</p>

Le traitement indiqué entre accolades est répété tant que la condition est vraie.

Voici un programme C qui compte les lettres, chiffres et autres caractères de la ligne tapée.

Langage C – compte.c	commentaire
<pre>main() { int lettres, chiffres, autres, c; lettres = 0; chiffres = 0; autres = 0; c = getchar(); while (c != '\n') { if (('A'<=c && c<='Z') ('a'<=c && c<='z')) { lettres = lettres + 1; } else if ('0'<=c && c<='9') { chiffres = chiffres + 1; } else autres = autres + 1; c = getchar(); } printf("%d lettres, %d chiffres, %d autres\n", lettres, chiffres, autres); }</pre>	<p>définition des variables initialisations lire le 1^{er} caractère tq pas fin de ligne si c'est une lettre</p> <p>incrémenter lettres sinon c'est un chiffre ? incrémenter les compteurs concernés</p> <p>passer au caractère suivant fin du bloc à répéter affichage final : format valeurs</p>

Noter que c est défini en tant qu'entier. Cela ne pose aucun problème puisqu'un entier 32 bits peut contenir un code ascii sur 8 bits, cela consomme à peine plus de mémoire et ralentit très légèrement le programme.

Ce même programme peut être rendu illisible à l'aide de pseudo-optimisations, pseudo car le gain de vitesse est complètement illusoire, l'optimiseur de code des compilateurs font un travail largement plus efficace que celui qu'on peut faire comme cela. Voici un exemple de ce qui est déconseillé :

Langage C – compte_sale.c	commentaire
<pre>main() { char l, c, a, x; l = c = a = 0; while ((x = getchar()) != '\n') if (('A'<=x && x<='Z') ('a'<=x && x<='z')) ++l; else ('0'<=x && x<='9')?++c:++a; printf("%d lettres, %d chiffres, %d autres\n", l, c, a); }</pre>	<p>définition des variables initialisation en une instruction lecture et test ! pas de {} pour 1 instr. test et incréments pas de {} car une seule instruction !</p>

Certains programmeurs raffolent de cette syntaxe à écriture seule et c'est ce qui contribue à la réputation de difficulté du langage C. Voici quelques explications :

- une affectation peut servir de valeur pour une comparaison, ainsi `(c = getchar())` a pour valeur le code ascii du caractère lu, il est comparé à la constante `'\n'` : code du retour à la ligne.
- `++var` est à peu près identique à `(var = var + 1)`. Ne pas l'employer dans des expressions plus complexe sous peine de ne pas pouvoir prévoir les résultats. On l'appelle opérateur de préincrément.
- il est inutile de mettre des `{}` lorsqu'il n'y a qu'une instruction. Une structure de contrôle est considérée comme une seule instruction. Un point-virgule achève l'instruction en cours.
- la structure `condition ? action1 : action2` permet de spécifier de petites conditionnelles lors d'affectations.

6 - BOUCLES REPETITIVES

La boucle *pour* est une boucle *tantque* déguisée, il n'y a pas de vraie boucle *pour* en langage C :

Langage C – <code>facto5.c</code>	commentaire
<pre>main() { int v, factoV, i; v = 5; factoV = 1; for (i=1 ; i<=5; i=i+1) { factoV = factoV * i; } printf("%d! = %d\n", v, factoV); }</pre>	<pre>for (init ; condition ; progression) { corps ; }</pre>

Une boucle `for` contient 4 éléments : *initialisation*, *condition*, *progression* et *corps*. L'exécution commence par l'*initialisation*. Les instructions du *corps* sont faites tantque la *condition* est vraie. C'est une condition de continuation et non une condition d'arrêt. Après avoir exécuté le *corps*, le langage C effectue l'affectation contenue dans *progression* et reprend au niveau du *tantque*.

Voici l'équivalence entre les deux types de boucles :

structure for	équivalence while
<pre>for (init ; cond ; svt) { instructions ; }</pre>	<pre>init ; while (cond) { instructions ; svt ; }</pre>

Voici comment on fait les boucles les plus fréquentes en Pascal :

Langage C	Langage Pascal
<code>for (i=3 ; i<=8 ; i=i+1) xxx;</code>	<code>for i := 3 to 8 do xxx ;</code>
<code>for (i=4 ; i>=0 ; i=i-1) yyy ;</code>	<code>for i := 4 downto 0 do yyy ;</code>

Evidemment, s'il y a plusieurs instructions dans le corps de la boucle, il faut les encadrer par des `{}`.



7 - FONCTIONS ET PROCEDURES

La syntaxe de définition d'une fonction est plus compacte que celle du Pascal :

Langage C – plancher.c	commentaire
<pre>int max(int a, int b) { if (a > b) return a; return b; } main() { int i; scanf("%d", &i); i = max(i,0); printf("%d", i); }</pre>	<p>type du résultat, nom, paramètres</p> <p>return provoque une sortie immédiate de la fonction donc pas besoin de else après un return</p> <p>lecture d'un nombre entier appel de la fonction max, max(v,0)=0 si v<0 affichage</p>

Noter que les paramètres sont passés par valeur. L'instruction return provoque le retour immédiat de la valeur de la fonction. Les instructions suivantes ne sont pas effectuées.

Voici maintenant comment on définit une procédure : on met void pour indiquer qu'il n'y a pas de résultat :

Langage C	commentaire
<pre>void affiche(int nb) { printf("nb=%d\n", nb); }</pre>	<p>pour une procédure on met void (vide)</p> <p>pas de return ou bien return sans valeur</p>

Voici maintenant comment on appelle cette procédure :

Langage C	commentaire
<pre>main() { int i; scanf("%d", &i); affiche(max(i,0)); }</pre>	<p>on appelle la procédure</p>

Voici maintenant quelques cas d'erreurs difficiles à déceler à cause des possibilités du langage C :

Erreurs de programmation	commentaire
<pre>c = getchar; max(i, j);</pre>	<p>sans (), le C n'appelle pas la fonction mais affecte l'adresse du code machine on peut appeler une fonction sans utiliser son résultat</p>

8 - TABLEAUX

Le langage C en a une approche très bas niveau, proche du langage d'assemblage. Quand on manipule un tableau en langage C, il faut constamment imaginer les opérations effectuées par l'unité centrale : des adressages indexés (voir le cours d'architecture). Un tableau C est une zone mémoire contenant plusieurs cases successives, on les manipule grâce à l'adresse de la première qui est représentée par le nom du tableau, et un indice. Pour créer un tableau, le langage C ne demande que la taille (le nombre d'éléments). On ne peut pas choisir l'étendue des indices comme en Pascal, en C les indices vont obligatoirement de 0 à la taille - 1.

Voici un exemple :

```
int t[10] ;      crée un tableau de 10 cases, la première est t[0], la dernière est t[9]
```

Ces cases peuvent être manipulées par des indices appliqués à `t` : les éléments de `t` sont `t[0]`, `t[1]`, `t[2]`...`t[9]`. Bien noter que les indices commencent obligatoirement à 0. Ils vont donc jusqu'au nombre d'éléments – 1.

Langage C – <code>factos.c</code>	commentaire
<pre>main() { int f[10], i, facti; facti = 1; for (i=1; i<=10; i=i+1) { t[i-1] = facti; facti = facti * (i + 1); } }</pre>	<p>création d'un tableau de 10 entiers (non initialisés)</p> <p>la boucle va de <code>i=1</code> à <code>i=10</code> on place <code>i!</code> dans chaque case, noter la translation des indices pour aller de 0 à 9 dans le tableau calcul de <code>i!</code> pour le <code>i</code> suivant</p>

Attention, les tableaux ne peuvent pas être, comme en pascal, affectés l'un à l'autre et comparés. Il faut écrire soi-même des fonctions et des procédures pour faire ces traitements, voir page 12.

9 - CHAINES DE CARACTERES

Les chaînes sont un cas particulier de tableaux, ce sont de simples tableaux d'octets qui sont les codes ascii des caractères. Perdre de vue cette réalité ferait commettre de graves erreurs de programmation. En effet le C n'est capable ni d'affecter un tableau à un autre, ni de comparer deux tableaux¹, il faut obligatoirement employer des fonctions spécifiques : `strcpy` et `strcmp`.

Une chaîne est une zone mémoire d'une certaine taille (au moins aussi grande que ce qu'on veut y mettre). Cette zone mémoire contient des codes ascii : ceux des caractères de la chaîne. Le langage C oblige à mettre le code ascii zéro tout à la fin : `\0`. Attention le code ascii de `'0'` est 48 et celui de `'\0'` est zéro. Voici un exemple de chaîne (en réalité, au lieu des caractères, on trouve les codes ascii dans les cases) :

...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
	D	e	m	e	s	m	a	e	k	e	r	\0	A	r	s	è	n	e	\0	L	

Cette zone mémoire contient 20 emplacements, on y a placé un texte « Demesmaeker » puis le code ascii nul (noté aussi `\0`) ; les cases suivantes contiennent du texte qui n'est pas pris en compte et qui résulte probablement d'opérations précédentes sur cette chaîne.

Ne pas oublier que le tableau fait partie de la mémoire, il y a d'autres octets avant et d'autres octets après la zone mémoire attribuée à cette chaîne. Si on affecte les cases trop loin (indice > 19), on risque de modifier d'autres données, d'autres variables situées au delà.

a) Constantes chaînes

La syntaxe `"texte"` crée un tableau sans nom (ce n'est pas une variable affectable), contenant les codes ascii des caractères indiqués suivis d'un code nul. Ainsi `"bonjour"` réserve 8 octets en mémoire.

Attention, selon les options de compilation et les compilateurs, deux chaînes constantes contenant la même chose peuvent être placées à des adresses différentes ou pas. Elles seront donc considérées comme

¹ Plus exactement, le C peut affecter un tableau à un autre ou comparer deux tableaux, mais il ne s'occupe que des adresses de début, pas du contenu : deux tableaux sont donc égaux seulement s'ils sont à la même adresse, ce qui est plus restrictif. En pascal, deux tableaux sont égaux s'ils ont le même contenu.

différentes ou pas selon le compilateur. Pour comparer deux chaînes par leur contenu, il faut obligatoirement utiliser la fonction `strcmp` ci-dessous.

b) Variables chaînes

Les constantes chaînes ne peuvent pas être modifiées car elles n'ont pas de nom symbolique. Au contraire, avec les variables chaînes, on peut modifier le contenu : affecter, concaténer, tronquer, comparer...

Voici un exemple de création d'une chaîne :

`char nom[21]` ; réserve 20 octets pour stocker des codes ascii, c-à-d des caractères. Il faut en prévoir un de plus pour mettre le code nul.

Langage C – chaîne.c	commentaire
<pre>main() { char nom[31], prenom[21], mail[61]; printf("ton prénom :"); gets(prenom); if (strcmp(nom, "pierre") == 0) printf("ah, salut pierrot !\n"); printf("ton nom :"); gets(nom); strcpy(mail, nom); strcat(mail, "."); strcat(mail, prenom); strcat(mail, "@iut-lannion.fr"); puts(mail); }</pre>	<p>réserve des chaînes de 30, 20 et 60 caractères (1 de plus pour le \0) comparaison de chaînes</p> <p>mail := nom mail := mail + '.' mail := mail + prenom mail := mail + '@iut-lannion.fr' afficher la chaîne mail</p>

La procédure `strcpy(dst, src)` copie dans la chaîne *dest* ce qui se trouve dans la chaîne *src*.

La procédure `strcat(dst, src)` ajoute au bout de la chaîne *dest* ce qui se trouve dans la chaîne *src*.

La fonction `strcmp(ch1, ch2)` compare les contenus des deux chaînes et renvoie 0 si elle contiennent la même chose. `strcmp` renvoie un nombre non nul si elles ne contiennent pas la même chose.

La fonction `strlen(ch)` calcule la longueur de la chaîne, attention distinguer la longueur de la chaîne (quantité de mémoire utilisée, variable) de sa taille (quantité de mémoire allouée, constante). Ainsi la chaîne `nom` du haut peut contenir jusqu'à 20 caractères utiles mais occupe toujours 21 octets en mémoire.

Voici maintenant quelques exemples techniques :

Langage C	équivalence
<pre>char txt[5]; txt[0] = 'j'; txt[1] = 'e'; txt[2] = '\0';</pre>	<pre>char txt[5]; strcpy(txt, "je");</pre>

Les affectations de chaînes sont impossibles directement car le C ne sait pas recopier des tableaux, attention à réserver assez de place dans la chaîne destination, sans quoi on écrase la suite de la mémoire :

Impossible/interdit	correct
<pre>char txt1[5], txt2[5]; txt1 = "oui"; txt2 = txt1;</pre>	<pre>char txt1[5], txt2[5]; strcpy(txt1, "oui"); strcpy(txt2, txt1);</pre>

De la même façon, le C ne sait pas comparer des tableaux, donc :

Faux ami, mauvais résultat	correct
<pre>char txt1[5], txt2[5]; if (txt1 == txt2) ... if (txt1 == "tu") ...</pre>	<pre>char txt1[5], txt2[5]; if (strcmp(txt1,txt2) == 0) ... if (strcmp(txt1,"tu") == 0) ...</pre>

Il existe des procédures/fonctions d'entrées/sorties pour les chaînes :

Langage C	Langage Pascal
<pre>char texte[80]; gets (texte); puts (texte);</pre>	<pre>var texte : string(80); read(texte); writeln(texte);</pre>

La fonction `gets` remplit le tableau avec ce qu'on tape au clavier jusqu'à la fin de ligne et place le code de fin de chaîne à la fin ('\\0'). La fonction `puts` écrit la chaîne à l'écran.

Langage C	Langage Pascal
<pre>char texte[80]; scanf("%s",mot); printf("%s",mot);</pre>	<pre>var mot : string(80); read(mot); writeln(mot);</pre>

La fonction `scanf` remplit le tableau avec le premier mot lu sur le clavier et place le code de fin de chaîne à la fin ('\\0'). Le joker `%s` écrit la chaîne à l'écran.

10 - TYPES COMPOSITES : STRUCTURES

Langage C – <code>infos.c</code>	commentaire
<pre>struct client { char nom[20]; int age; };</pre>	définition d'un type structuré, la syntaxe est très différente de celle du Pascal
<pre>void afficher(struct client untel) { printf("%s, %d ans\\n", untel.nom, untel.age); }</pre>	définition d'une procédure d'affichage avec une structure en paramètre, remarquer le mot clé <code>struct</code> présent à chaque fois
<pre>main() { struct client cahier[5]; strcpy(cahier[0].nom, "Michel") ; cahier[0].age = 25; afficher(cahier[0]); }</pre>	définition d'une variable tableau de 5 structures affectation des champs de la première case appel de la fonction d'affichage

Noter que de cette façon le type ne s'appelle pas *client* seulement mais *struct client*. L'utilisation de ce type est identique à Pascal.

11 - VARIABLES GLOBALES

Langage C	commentaire
<pre>float TVA = 19.6; float prixttc(float prixht) { return prixht * TVA; }</pre>	variable globale préinitialisée TVA est prise en dehors de la fonction

Il s'agit de variables situées hors de toute fonction. On peut y accéder partout dans le programme. Lorsqu'une variable locale porte le même nom, c'est cette dernière qui est prise en compte.

12 - POINTEURS

Le langage C est très proche de la machine et n'offre pas des mécanismes très élaborés pour manipuler certaines structures de données dynamiques (tableaux dynamiques, listes, arbres, graphes). Pour les représenter en C, il faut manipuler directement les adresses des données et faire des indirections sur ces adresses (adressage indirect registre). En C, on appelle *pointeur* une variable qui contient une adresse.

Langage C	commentaire
<pre>int a,b,c; main() { int *adr; adr = &b; (*adr) = 50; }</pre>	<p>définition de trois variables globales</p> <p>adr : adresse d'une variable de type int adr = adresse de la variable b mettre 50 dans la case mémoire désignée par adr, c'à d b</p>

L'opérateur & donne l'adresse de son opérande : &a est l'adresse de a, &c est l'adresse de c.

L'opérateur * donne la valeur située à l'adresse indiquée par l'opérande : *15 donne le contenu de la case mémoire d'adresse 15. Attention, on ne doit donner que des adresses valides (situées dans le segment de données du programme et correspondant à des variables).

a) Paramètres de sortie

On se sert de ce mécanisme pour passer des paramètres de sortie :

Langage C – échange.c	commentaire
<pre>void echanger(int *adr1, int *adr2) { int inter; inter = (*adr1); (*adr1) = (*adr2); (*adr2) = inter; } main() { int a=14, b=23; printf("avant: a=%d b=%d\n", a,b); echanger(&a, &b); printf("apres: a=%d b=%d\n", a,b); }</pre>	<p>adr1 et adr2 sont deux adresses de variables int</p> <p>adr1 = adresse d'un entier *adr1 = cet entier</p> <p>&a = adresse de a placée dans adr1 &b = adresse de b placée dans adr2</p>

b) Pointeurs et tableaux

Le langage C entretient une relation très forte entre pointeurs et tableaux. Le nom d'un tableau est un pointeur non affectable sur son premier élément, d'indice 0. Ainsi $\&(\text{tab}[0]) = \text{tab}$ et $(*\text{tab}) = \text{tab}[0]$. D'autre part, on peut additionner une constante à un pointeur pour désigner une des cases suivantes (adressage basé avec un décalage). Ainsi $\text{tab}+i = \&\text{tab}[i]$ et $(*(\text{tab}+i)) = \text{tab}[i]$.

Langage C	commentaire
<pre>main() { int tab[4]; (*(tab+2)) = 4; echange(tab+1, &tab[3]); }</pre>	<p>tableau de quatre entiers, tab = adresse du 1^{er} élément tab+2 = adresse du 3^e entier $(*(\text{tab}+2)) = \text{tab}[2]$ tab+1 = $\&\text{tab}[1]$</p>

En Pascal, le nom d'un tableau représente toutes ses cases tandis qu'en C, le nom d'un tableau représente uniquement l'adresse du début du tableau. C'est pour cela qu'en C on ne peut pas affecter des tableaux l'un à l'autre et que les tableaux sont toujours passés par variable.

Lorsqu'on passe un tableau en paramètre, on passe uniquement l'adresse de ce tableau, cette adresse permet d'accéder au contenu.

Langage C – zeros.c	commentaire
<pre>int nbnuls(int *tab, int taille) { int i, n=0; for (i=0 ; i<taille ; i=i+1) if (tab[i] == 0) n=n+1; return n; } main() { int tab[4] = { 2,0,8,0 }; printf("nuls=%d\n", nbnuls(tab, 4)); }</pre>	<p>le paramètre tab est un pointeur, adresse de la première case d'un tableau</p> <p>aucune différence d'emploi car $\text{tab}[i] = *(\text{tab}+i)$</p> <p>tableau de quatre entiers initialisés appel de la fonction adresse de tab et taille</p>

Les chaînes sont également manipulées de cette façon :

Langage C – zeros.c	commentaire
<pre>int longueurphrase(char *chaine) { int i; for (i=0 ; chaine[i] !='\0'; i=i+1) if (chaine[i] == '.') return i; return i; } main() { char texte[80] = "Nouvelles.Ras."; printf("lngphrase(%s)=%d\n", texte, longueurphrase(texte)); }</pre>	<p>le paramètre sera un tableau de caractères</p> <p>boucle tant qu'on reste dans la chaîne arrêt immédiat quand on trouve un point on renvoie son indice</p> <p>texte[0]='N'...texte[3]='v'...texte[6]='l'...texte[14]='\0'</p>

En C on ne peut pas affecter un tableau à un autre : il faut clairement indiquer si on souhaite affecter les adresses ou si on souhaite affecter le contenu :

affectation des adresses	affectation du contenu
<pre>int tab1[5] = { 1, 4, 2, 6, 7 }; int *tab2; { tab2 = tab1; }</pre>	<pre>int tab1[5] = { 1, 4, 2, 6, 7 }; int tab2[5]; { int i; for (i=0 ; i<5 ; i++) tab2[i] = tab1[i]; }</pre>

Les deux programmes créent tab1, tableau de 5 entiers et font en sorte que tab2 contiennent les mêmes valeurs que tab1. La différence est que dans le programme de gauche tab2 est à la même adresse que tab1, si on modifie tab2[3], on modifie aussi tab1[3], tandis que à droite, tab2 est un autre tableau dont les valeurs ont été initialisées avec celles de tab1, si on modifie tab2[3], cela ne change rien à tab1[3].

Bien noter qu'on peut affecter une valeur à un pointeur (comme tab2 à gauche) mais qu'on ne peut pas affecter un tableau (tab1 est en lecture seule). D'autre part, il faut veiller à réserver assez de place dans le tableau destination.

Les mêmes considérations s'appliquent aux chaînes de caractères. La fonction `strcpy` effectue la copie des octets d'une chaîne à l'autre.

C - STRUCTURE D'UN PROGRAMME C

1 - ÉLÉMENTS D'UN PROGRAMME

Un programme C est composé des éléments suivants : définitions des constantes, types, variables globales, procédures et fonctions et corps principal. Ces éléments sont disposés en vrac (sans structuration), la seule contrainte est qu'un élément doit être défini ou déclaré avant son utilisation : ce qui est défini ou déclaré à un endroit devient disponible pour la suite du programme.

Vocabulaire : *définir* = créer (réserver la place en mémoire à l'exécution) ; *déclarer* = prévenir le compilateur que la définition est faite plus loin. Voir aussi le paragraphe G - 4 - a) page 39.

Dans la plupart des cas, certaines définitions de constantes et types et déclarations de variables et de fonctions sont placées dans des fichiers inclus au programme. Ces fichiers portent l'extension .h.

- **Commentaires**

Un commentaire doit être encadré par /*... */. Il n'y a pas de limite à la longueur d'un commentaire. Un commentaire ne peut pas en contenir d'autres.

Langage C	Langage Pascal
/* ceci est un commentaire */	(* ceci est un commentaire *)
/* il n'y a qu'une possibilité */	{ il y a deux possibilités }

- **Fichiers inclus et librairies**

Lorsque le programme fait appel à des fonctions extérieures (par exemple, les fonctions mathématiques ou les fonctions système Unix), il est souvent nécessaire d'inclure des déclarations de constantes, types et prototypes de fonctions qui se trouvent dans certains fichiers appelés fichiers d'entêtes de librairies. Ces fichiers sont regroupés dans le répertoire /usr/include et sont inclus par la directive #include <fichier.h>. Voir aussi le §H - 1 - page 41.

Langage C
#include <stdio.h>

Lorsque le fichier à inclure se trouve dans le répertoire courant, des "" remplacent les <> :

Langage C
#include "donnees.h"

- **Définition des constantes**

Une constante est un identificateur simple pour représenter une valeur numérique compliquée.

Syntaxe : *#define* constante valeur

définit la valeur du symbole *constante*

Langage C	Langage Pascal
#define PI 3.1415	const PI = 3.1415;

Pièges : ne pas mettre de ; après la valeur. Cela provoque une erreur de syntaxe à toutes les mentions de la constante.

Conseil : en général, les constantes sont mises en majuscules afin de les localiser facilement.

#define permet également de définir des macro-fonctions. Il s'agit de pseudo-fonctions dont la définition remplace chaque appel. Il faut donc faire attention à ne pas les employer à mauvais escient. Voir page 43.

Langage C
#define dernierchar(S) (S[strlen(S)-1])

Cette macro donne le dernier caractère d'une chaîne non-vide.

- **Définition des types**

Voir le paragraphe D - 2 - page 16.

- **Définition des variables globales**

Une variable globale est définie de la même manière qu'une variable locale dans une fonction ou procédure. Se reporter au paragraphe D - 1 - page 16.

- **Déclaration des fonctions et procédures**

Dans certains cas, il faut indiquer au compilateur quels sont les types des paramètres et le type du résultat d'une fonction sans pouvoir fournir en même temps le corps de cette fonction (récursivité croisée, modularité ou simple confort du programmeur). Cela s'appelle une déclaration de prototypes et consiste à fournir l'entête d'une fonction sans son corps. Voir le paragraphe G - 1 - c) page 33.

- **Définition des fonctions et procédures**

Voir le paragraphe G - 1 - page 32.

- **Définition du point d'entrée : le corps du programme**

Dans le langage C, le point d'entrée est défini simplement à l'aide d'une procédure particulière qui s'appelle *main*. Le programmeur doit définir cette procédure qui sera automatiquement appelée lors de l'exécution.

2 - COMPILATION D'UN PROGRAMME C

Comme en Pascal, le but est d'obtenir un programme exécutable à partir du ou des programmes source. La construction de l'exécutable se déroule en deux étapes : compilation des fichiers sources et édition des liens.

a) Compilation

Cela consiste à traduire les instructions et définitions de données en langage machine. Le compilateur consulte le source .c et tous les fichiers .h qu'il mentionne. Il construit un fichier dit objet .o.

Il existe également des fichiers .a dits bibliothèques construits à partir des .o. Ces fichiers contiennent les corps des fonctions tandis que les fichiers .h contiennent les déclarations des fonctions.

Commande : `cc -Wall -c prog.c`

L'option -Wall affiche tous les messages d'avertissement. La présence d'avertissements est mauvais signe mais leur absence ne signifie pas pour autant que le programme est sain.

b) Édition des liens

Cela consiste à rassembler tous les fichiers .o et les bibliothèques de fonction en associant les appels de fonctions aux définitions. Voir le cours de production/maintenance.

Commande : `cc prog.o -o prog`

compile le source `prog.c` et crée l'exécutable `prog`.

Dans le cas d'un programme simple, les deux étapes peuvent être rassemblées :

Commande : `cc -Wall prog.c -o prog`

D - VARIABLES ET TYPES DE DONNEES

1 - DEFINITION D'UNE VARIABLE

a) Syntaxe

Syntaxe1 : `type variable;`

Syntaxe2 : `type variable1, variable2, ...;`

définit la ou les variables : réserve la place pour une variable de ce type, la variable devient connue et utilisable dans la suite du programme si c'est une variable globale (c'est à dire définie en dehors d'une fonction) ou seulement dans le bloc d'instructions d'une fonction si c'est une variable locale (voir F - 2 - page 28). Les identificateurs autorisés sont les mêmes qu'en Pascal : lettre puis suite de lettres ou de chiffres.

b) Initialisation

Cela consiste à placer des valeurs dans les variables avant l'exécution du programme, au moment de leur définition. La ligne suivante définit la variable ainsi que sa valeur initiale.

Syntaxe : `type variable = valeurinitiale;`

2 - TYPES SIMPLES

a) Entiers

Le C offre plusieurs sortes d'entiers afin de s'adapter à toutes sortes de machines (micro-processeurs 8-16 bits ou 16-64 bits). Il existe donc des entiers courts, normaux ou longs qui sont mémorisés sur un nombre d'octets différents. Par exemple, en 2003 sur un PC linux, un entier court occupe 2 octets (16 bits), les normaux et les longs en occupent 4 (32 bits).

D'autre part, pour une même taille, il existe des entiers signés et des entiers non-signés. Cela change les valeurs entières minimum et maximum qui peuvent être représentées. Voici les différents types entiers :



	signé	non-signé
entier court	short	unsigned short
entier normal	int	unsigned int
entier long	long	unsigned long

Exemple :

Langage C	Langage Pascal
unsigned long nbatomes; short variation; int code;	var nbatomes: integer; var variation: integer; var code: integer;

Les constantes entières peuvent être données dans différentes bases :

base	syntaxe	exemple	valeur en base 10
10	comme en Pascal	45	45
8	faire précéder d'un 0	067	55
16	faire précéder de 0x	0x3E	62

Les entiers longs sont suffixés d'un L. Exemple : 125L est l'entier long qui vaut 125.

Langage C	Langage Pascal
nbatomes = 1868376613L; code = 0xFFE;	nbatomes := 1868376613; code := 4094;

b) Booléens

Il n'existe pas de type spécialisé pour gérer les booléens. Ils ont été intégrés à la syntaxe du langage C en tant qu'entiers avec la convention suivante :

0 : faux, ≠0 : vrai.

Tous les opérateurs booléens (voir E - 2 - page 23) respectent cette convention.

Langage C	Langage Pascal
int f; f = 1; f = 0;	var f: boolean; f := True; f := False;

Les fonctions d'entrées/sorties (voir G - 3 - page 36) ne sont pas capables de manipuler les valeurs booléennes. Si on veut, comme en Pascal, afficher ou lire True ou False, il faut construire des fonctions adaptées, exemple pour afficher un booléen :

Langage C	Langage Pascal
int f; printf("%s", f?"true":"false");	var f: boolean; write(f);

c) Caractères

Les caractères sont codés sur un octet en suivant le code ascii et sont gérés comme des entiers. Le nom du type caractère est char comme en Pascal.

Langage C	Langage Pascal
char c;	var c: char;

Il existe trois manières d'écrire une constante de type caractère :

- 'caractère' : équivalent au code ascii sur un octet du caractère ;
- '\lettre' : représente un caractère non-affichable (\n retour à la ligne, \t tab) ;

- `'\0nombre'` : nombre est le code ascii en base 8 (en octal) du caractère représenté.

Exemple :

Langage C	Langage Pascal
<code>c = 'A';</code>	<code>c := 'A';</code>
<code>c = '\n';</code>	<code>c := chr(10);</code>
<code>c = '\040';</code>	<code>c := chr(32);</code>

d) Réels

Comme pour les entiers, il existe deux sortes de nombres réels, les normaux (`float`) et les réels en double précision (`double`). Ces deux catégories sont gérées en virgule flottante, la norme dépend de la machine considérée (IEEE ou autre).

Une constante réelle est écrite en notation scientifique classique (par exemple $31.41e-1$).

Langage C	Langage Pascal
<code>float e;</code>	<code>var e: real;</code>
<code>double max;</code>	<code>var max: real;</code>
<code>e = 2.71;</code>	<code>e := 2.71;</code>
<code>max = 3e125;</code>	<code>max := 3e125;</code>

e) Énumérations

Les variables de ce type prennent une valeur parmi un ensemble. Par exemple pour un feu de circulation, sa couleur est vert, orange ou rouge. Le langage C propose deux syntaxes pour définir ces types :

Syntaxe n°1: `enum nouveautype { liste de symboles };`

définit un nouveau type qui s'appelle `enum nouveautype` (et non pas seulement `nouveautype`, voir la syntaxe n°2). Le domaine des variables de ce type est la liste des symboles fournis. Ces symboles sont équivalents à des constantes entières commençant à 0.

Langage C	Langage Pascal
<code>enum Jour { lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche };</code>	<code>type Jour = (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche);</code>
<code>enum Jour j;</code>	<code>var j: Jour;</code>
<code>j = jeudi;</code>	<code>j := jeudi;</code>

définit le type `enum Jour`, les variables de ce type peuvent prendre pour valeur `lundi`, `mardi`,... ou `dimanche`.

Syntaxe n°2: `typedef enum { liste de symboles } nouveautype;`

définit un nouveau type qui s'appelle `nouveautype` (sans `enum`).

Langage C	Langage Pascal
<code>typedef enum { nul, faible, moyen, fort, infini } Intensite;</code>	<code>type Intensite = (nul, faible, moyen, fort, infini);</code>
<code>Intensite son;</code>	<code>var son : Intensite;</code>
<code>son = moyen ;</code>	<code>son := moyen;</code>

3 - ENREGISTREMENTS

a) Définition d'un type enregistrement

Syntaxe n°1: `struct nouveautype { liste de champs };`

définit un nouveau type appelé `struct nouveautype` (et non pas seulement `nouveautype`, voir la syntaxe n°2). La liste des champs est composée de définitions identiques à des définitions de variables.

Langage C	Langage Pascal
<pre>struct point { float x,y; Intensite brillance; };</pre>	<pre>type point = record x,y : real; brillance : Intensite end;</pre>

définit le type `struct point` en tant qu'agrégat.

Syntaxe n°2: `typedef struct { liste de champs } nouveautype;`

définit un nouveau type appelé simplement `nouveautype`.

Langage C	Langage Pascal
<pre>typedef struct { struct point centre; int rayon; } Cercle;</pre>	<pre>type cercle = record centre: point; rayon : integer end;</pre>

NB : l'indentation est importante pour la lecture et la compréhension du programme.

b) Définition d'une variable structurée

La définition d'une variable de type structure se fait de l'une des manières suivantes suivant la syntaxe qui avait été employée lors de la définition du type :

Syntaxe n°1: `struct nomtype variable;`

Syntaxe n°2: `nom_typedef variable;`

créé la variable du type structuré indiqué.

Langage C	Langage Pascal
<pre>struct point P; Cercle C;</pre>	<pre>var P : point; var C: cercle;</pre>

c) Initialisation d'une variable structurée

Comme pour les variables normales, une variable de type struct peut être initialisée :

Syntaxe n°1: `struct nomtype variable = { valeurs_champs };`

Syntaxe n°1: `nom_typedef variable = { valeurs_champs };`

les valeurs de champs sont attribuées aux différents champs de la variable définie.

Langage C	Langage Pascal
<pre>struct point P = {1.5,-2.3,nul}; Cercle C = { {1.0, 1.0, fort}, 3 };</pre>	<p><i>value n'est pas normalisé en Pascal</i></p>

Lorsque les champs sont eux-même des structures, on doit encadrer les valeurs par des {} (c'est le cas dans le deuxième exemple).

4 - TABLEAUX

a) Définition d'une variable tableau

Le langage C propose un type tableau qui est plus contraignant qu'en Pascal. En effet, les indices des cases ne peuvent commencer qu'à zéro et non par une valeur quelconque. D'autre part, les indices sont obligatoirement numériques ou de type énuméré. Enfin, la gestion de tableaux de plusieurs dimensions est délicate car très proche des mécanismes sous-jacents (modes d'adressage et allocation mémoire). La définition d'un tableau à une dimension suit la syntaxe suivante :

Syntaxe : `typedef nom_tableau[nombrecases];`

Cela définit un tableau (réserve la place en mémoire et publie son nom dans la suite du programme ou du bloc). Les cases sont numérotées de 0 jusqu'à *nombre_cases-1*. D'autre part, le langage n'assure aucun contrôle sur les indices employés, ils peuvent être négatifs ou supérieurs à la taille indiquée, le seul résultat à craindre est un bug voire un plantage du programme.

Langage C	Langage Pascal
<code>char nom[10];</code>	<code>var nom : array[1..10] of char;</code>
<code>int effectif[10];</code>	<code>var effectif : array [1993..2002] of integer;</code>
<code>struct point nuage[100];</code>	<code>var nuage : array [0..99] of point;</code>
<code>Cercle bulles[16];</code>	<code>var bulles : array [0..15] of cercle;</code>

Comme le langage C fait démarrer les indices à zéro, il est nécessaire de prévoir dans le programme une translation des cases. Pour le second exemple, il faut retirer 1993 aux indices du tableau `effectif` du programme C.

Pour les tableaux à deux dimensions, voici comment on définit une variable :

Syntaxe : `typedef nom_tableau[nblignes][nbcolonnes];`

définit un tableau contenant *nblignes* * *nbcolonnes* cases du type indiqué. Il faut remarquer que les deux nombres de cases sont donnés séparément. Dans un tel tableau, deux éléments consécutifs en mémoire sont dans deux colonnes consécutives et non pas dans deux lignes (au contraire du Fortran). Pour s'en souvenir, il suffit de penser que le tableau possède *nblignes* sous-tableaux de *nbcolonnes* cases.

Langage C	Langage Pascal
<code>char argv[10][80];</code>	<code>var argv: array[1..10,1..80] of char;</code>

Cette définition se lit : `argv` est un tableau contenant 10 tableaux de 80 caractères.

La définition d'un tableau ayant un plus grand nombre de dimensions suit le même principe. On trouve à gauche les dimensions écartant le plus en mémoire les éléments d'indices consécutifs.

b) Initialisation des éléments

L'initialisation suit une syntaxe identique à celle des enregistrements, on met des {} à chaque niveau :

Syntaxe : `type nom[nombre] = { v0, v1, v2, ...};`

Syntaxe : `type nom[nbl][nbc] = {{v00, v01...}, {v10, v11...}...};`

avec $vi j$ désignant une valeur à placer en ligne i et en colonne j .

Langage C	Langage Pascal
<pre>int tabs[3] = { 7, 14, 21 }; float pts[4][3] = { { 1.0, 0.0, 0.0 }, { 0.0, 2.0, 0.0 }, { 0.0, 0.0, 1.0 }, { 1.1, 1.1, 1.1 } };</pre>	<i>value</i> n'est pas normalisé en Pascal

c) Définition d'un type tableau

On peut définir un type tableau sans pour autant définir une variable par la syntaxe suivante :

Syntaxe : `typedef typecase nom_type[nombrecases];`

définit un type tableau sans définir de variable.

Langage C	Langage Pascal
<pre>typedef char tNom[20]; tNom n;</pre>	<pre>type tNom = array[1..20] of char; var n : tNom;</pre>

d) Chaînes de caractères

Contrairement au langage Pascal, le langage C n'offre pas un traitement confortable des chaînes de caractères. En effet, elles sont gérées sous la forme de tableaux de taille fixe, dans lesquels on ajoute une sentinelle au bout du contenu utilisable. Cette sentinelle est le caractère de code ascii nul (' $\backslash 0$ '). Elle est exploitée par toutes les fonctions de gestion des chaînes.

La définition d'une chaîne est identique à celle d'un tableau à une dimension. Il faut penser à réserver une case de plus que la longueur utile pour la sentinelle.

Syntaxe : `char nomchaîne[longueur_max_possible + 1];`

Exemple :

Langage C	Langage Pascal
<code>char ligne[81];</code>	<code>var ligne: string(80);</code>

En C, il faut prévoir explicitement une case de plus pour contenir la sentinelle (' $\backslash 0$ ') tandis qu'en Pascal, les procédures du langage gèrent la taille de manière transparente pour l'utilisateur.

Le langage C propose également la définition de chaînes constantes :

Syntaxe : `"contenu de la chaîne"`

définit un *nouveau* tableau préinitialisé au contenu indiqué. Attention, deux définitions identiques ne sont pas égales bien que leur contenu soit le même, puisque les tableaux sont à des adresses différentes. Certains compilateurs permettent de ne pas multiplier les copies des mêmes chaînes.

Ces chaînes constantes permettent d'initialiser des chaînes variables :

Syntaxe : `char nomchaîne[longueur_max_voulue + 1] = "contenu";`

Exemple :

Langage C	Langage Pascal
<pre>char ligne[81] = "bonjour"; char texte[81] = { 'b','o','n','j','o','u','r','\0' };</pre>	pas d'équivalent normalisé

Les deux initialisations sont équivalentes (une chaîne est un tableau de caractères).

Lorsqu'on manipule des chaînes de caractère, il est nécessaire de prendre en considération leur représentation en mémoire. En particulier, les comparaisons de chaînes peuvent sembler surprenantes mais il est nécessaire de comparer les contenus des chaînes et non pas les variables. Ainsi, dans l'exemple suivant, les chaînes Pascal sont égales tandis qu'elles sont différentes en C.

Langage C	Langage Pascal
<pre>char chaine1[81] = "bonjour"; char chaine2[81] = "bonjour";</pre>	<pre>var chaine1, chaine2: string(80); begin chaine1 := 'bonjour'; chaine2 := 'bonjour';</pre>
chaine1 ≠ chaine2	chaine1 = chaine2

Toutefois, dans les deux cas, les *contenus* des chaînes sont égaux. Le langage C établit donc une distinction entre contenu et contenant : comme pour les tableaux, le nom d'une chaîne indique seulement son emplacement en mémoire.

Le même problème surgit lorsqu'on affecte des chaînes. Le langage C interdit qu'on modifie le nom d'une variable chaîne :

Langage C	Langage Pascal
<pre>char chaine1[81] = "bonjour"; char chaine2[81]; { chaine2 = chaine1;</pre>	<pre>var chaine1, chaine2: string(80); begin chaine1 := 'bonjour'; chaine2 := chaine1;</pre>
erreur de compilation	pas de problème

Se reporter au paragraphe E - 2 - page 23 pour comparer deux chaînes et au paragraphe F - 1 - a) page 27 pour les affectations. Se reporter au paragraphe E - 4 - page 25 pour comprendre les mécanismes sous-jacents.

E - OPERATEURS ET EXPRESSIONS

1 - OPERATEURS NUMERIQUES

a) Entiers

Le langage C propose cinq opérateurs sur les entiers : +, -, *, / (division entière), % (modulo). Ils ont les mêmes priorités qu'en Pascal.

Langage C	Langage Pascal
$a + b * (c - d \% 4) / 3$	$a + b * (c - d \text{ mod } 4) \text{ div } 3$

b) Réels

Le langage C ne propose que quatre opérateurs pour manipuler les réels : +, -, *, /. Les autres opérations font appel à des fonctions disponibles dans la librairie mathématique (voir page 39)

c) Opérateurs binaires

Le langage C offre des possibilités pour manipuler les données en base 2.

$n1 \ \& \ n2$	et bit à bit entre $n1$ et $n2$
$n1 \ \ n2$	ou bit à bit entre $n1$ et $n2$
$\sim n$	complément à 1 de n
$n \ \gg \ dec$	décalage à droite de n sur dec bits
$n \ \ll \ dec$	décalage à gauche de n sur dec bits

2 - OPERATEURS LOGIQUES

Le langage C offre des opérateurs de comparaison similaires à ceux du Pascal :

Langage C	Langage Pascal
$e1 < e2$	$e1 < e2$
$e1 \leq e2$	$e1 \leq e2$
$e1 == e2$	$e1 = e2$
$e1 \neq e2$	$e1 \neq e2$
$e1 \geq e2$	$e1 \geq e2$
$e1 > e2$	$e1 > e2$

Il faut faire attention à l'égalité (qui ne doit pas être confondue avec l'affectation) et à la différence. Ces opérateurs s'appliquent à tous les types scalaires.

Les opérateurs logiques diffèrent quelque peu, à la fois dans leur syntaxe et dans leur signification.

Langage C	Langage Pascal
$e1 \ \&\& \ e2$	$e1 \ \text{and} \ e2$
$e1 \ \ \ \ e2$	$e1 \ \text{or} \ e2$
$! \ e$	$\text{not} (e)$

Les deux premiers sont des opérateurs *paresseux* qui n'évaluent que le strict minimum en commençant par les expressions booléennes de gauche. Par exemple, dans `e1 && e2`, si `e1` est fausse, `e2` n'est pas calculée. On rappelle la convention : `0 = faux`, `≠0 = vrai`.

Dans le cas de la comparaison de chaînes de caractères, il faut faire appel à la fonction suivante :

Syntaxe : `strcmp(chaine1, chaine2)`

Elle renvoie 0 si les deux chaînes sont égales (cette fonction calcule la différence entre elles).

Langage C	Langage Pascal
<code>char nom[81];</code> <code>gets(nom);</code> <code>if (strcmp(nom, "toto") == 0) ok = 1;</code>	<code>var nom: string(80);</code> <code>readln(nom);</code> <code>if (nom = 'toto') then ok := true;</code>

3 - OPERATEURS DIVERS

a) Accès à un élément d'un tableau

L'opérateur `[indice]` permet de désigner comme en Pascal l'élément n°*indice* d'un tableau. En C, les bornes d'un tableau sont comprises entre 0 et (la taille du tableau - 1).

Langage C	Langage Pascal
<code>effectif[1]</code>	<code>effectif[1995]</code>

b) Accès à un champ d'un struct

L'opérateur `.` (point) permet d'accéder comme en Pascal à un champ d'un struct.

Langage C	Langage Pascal
<code>C.centre.x</code> <code>C.rayon</code> <code>nuage[10].x</code>	<code>C.centre.x</code> <code>C.rayon</code> <code>nuage[10].x</code>

c) Taille d'une variable

Il existe un moyen pour connaître l'encombrement en mémoire d'une variable ou d'un type.

Syntaxe : `sizeof(variable ou type)`

Cet opérateur renvoie le nombre d'octets occupés par la variable ou par le type. Attention, cet opérateur ne renvoie pas la taille d'une chaîne. Pour cela, il faut employer la fonction suivante :

Syntaxe : `strlen(chaîne)`

Cette fonction renvoie la taille de la chaîne.

Langage C	Langage Pascal
<code>char nom[81];</code> <code>gets(nom);</code> <code>printf("nbre = %d\n", strlen(nom));</code>	<code>var nom: string(80);</code> <code>readln(nom);</code> <code>writeln('nbre = ', length(nom));</code>

d) Conversions de type des valeurs scalaires

Pour convertir une valeur d'un type scalaire à un autre type, il faut employer la syntaxe suivante:

Syntaxe : `(typedestination) variable`

Convertit la valeur dans le type indiqué. En C, toutes les conversions sont possibles même si elles font perdre de la précision. Il existe des fonctions mathématiques pour ne pas perdre de précision (floor...). En Pascal, il faut passer par un ensemble de fonctions *ad-hoc*.

Langage C	Langage Pascal
(int)3.1415	round(3.1415)
(int)'A'	ord('A')
(char)10	chr(10)
(int)vert	ord(vert)

4 - ADRESSES DES VARIABLES

Proche de la machine, le langage C offre la possibilité de manipuler les adresses des données : effectuer des calculs, des indirections, etc. L'adresse d'une donnée est le nombre entier qui définit sa position en mémoire. Elle est également appelée pointeur.

a) Obtenir l'adresse d'une variable

Syntaxe : `&variable`

L'opérateur & appliqué à une variable renvoie l'adresse de cette variable, il s'agit d'un nombre entier qu'on peut afficher. Compte tenu de la segmentation de la mémoire, ce nombre indique généralement un déplacement dans le segment de données.

Langage C	Langage Pascal
float x; printf("adresse de x : %d\n",&x);	<i>pas d'équivalent normalisé</i>

Ce programme affiche l'adresse de la variable x (le type de la variable n'intervient pas).

NB: on ne peut pas appliquer cet opérateur aux constantes car elles n'ont pas d'adresse.

b) Mémoriser une adresse dans une variable

Lorsqu'on veut manipuler l'adresse d'une donnée, il est nécessaire de la placer dans une variable de type pointeur. Il s'agit simplement d'une variable destinée à contenir une adresse. La seule contrainte posée par le langage est qu'il faut donner un type au pointeur (en assembleur les adresses sont indifférenciées).

Syntaxe : `typedonnée *nom;`

Cette ligne définit une variable dont le contenu est une adresse sur une donnée du type indiqué.

Langage C	Langage Pascal
float *adr; float x; adr = &x;	<i>pas d'équivalent normalisé</i>

adr est une adresse de flottants, on lui affecte l'adresse de x.

NB: l'intérêt de cette manipulation apparaîtra dans les cours sur les structures de données.

c) Accès à une donnée par son adresse

Syntaxe : `*adresse`

L'opérateur * appliqué à une adresse (variable pointeur) renvoie la valeur stockée à cette adresse, il s'agit d'une donnée dont le type est celui du pointeur. On peut la consulter ou l'affecter.

Langage C	Langage Pascal
<pre>float *adr; adr = ...; printf("contenu de adr : %f\n", *adr); *adr = 3.14;</pre>	<i>pas d'équivalent normalisé</i>

adr est une adresse de flottants, on affiche le flottant qu'elle désigne puis on y met 3,14.

On peut courir le risque de placer n'importe quel entier dans un pointeur afin d'accéder à la mémoire. Cependant cela peut provoquer une erreur à l'exécution si elle se situe hors du segment de données (violation de segmentation).

d) Tableaux et adresses

Le langage C permet de manipuler les tableaux comme en langage d'assemblage, à l'aide de pointeurs. Le nom d'un tableau est considéré comme un pointeur sur son premier élément.

Langage C	Langage Pascal
<pre>float tab[10]; float *adr; adr = tab; printf("contenu de adr : %f\n", *adr);</pre>	<i>pas d'équivalent normalisé</i>

adr est une adresse de flottants, elle désigne le premier élément du tableau tab.

L'élément suivant se trouve à adr+1. Le type de l'élément (donc la place qu'il occupe en mémoire) est pris en compte dans cette incrémentation. Cette règle permet d'écrire les deux équations suivantes :

$$\begin{aligned} \&(\text{tab}[i]) &\equiv \text{tab} + i & \text{tab}[i] &\equiv *(\text{tab} + i) \\ \&\text{tab}[0] &\equiv \text{tab} & \text{tab}[0] &\equiv *\text{tab} \end{aligned}$$

Ces équations font le lien entre les tableaux et les pointeurs.

Langage C	Langage Pascal
<pre>float tab[10]; float *adr; adr=tab; while (*adr != 0) { printf("%f ", *adr); adr = adr + 1; }</pre>	<i>pas d'équivalent normalisé</i>

Ce programme affiche les premiers éléments du tableau tab, jusqu'à rencontrer un zéro. Il faut noter qu'aucun contrôle de sortie du tableau n'est fait (rien ne permet de le faire).

On peut également soustraire un entier d'un pointeur afin de revenir sur un élément précédent.

Noter le moyen mnémotechnique, en déplaçant mentalement les espaces :

int* p;	int *p;
peut se lire : p est un (int *) : un pointeur sur un int	peut se lire : (*p) est un int

F - INSTRUCTIONS ET STRUCTURES DE CONTROLE

1 - INSTRUCTIONS SIMPLES

L'instruction de base en C est composée d'une expression suivie d'un point virgule. Les affectations sont considérées comme des expressions par ce langage.

a) Affectations

Les affectations obéissent à la syntaxe suivante :

Syntaxe : `variable = expression;`

Exemple :

Langage C	Langage Pascal
<code>i = 3; deg = rad * 180.0 / PI;</code>	<code>i := 3; deg := rad * 180.0 / PI;</code>

A la différence du Pascal, dans lequel les ; sont des séparateurs d'instructions, le langage C utilise les ; comme des terminateurs. C'est pourquoi il faut en placer au bout de chaque instruction.

Langage C	Langage Pascal
<code>{ x = 15; }</code>	<code>begin x := 15 end</code>

Dans le cas des chaînes qui sont des tableaux normaux contenant des caractères, il faut employer les fonctions suivantes pour réaliser une affectation.

Syntaxe : `strcpy(chaine_affectée, chaine_valeur);`

Cette fonction place la chaîne valeur dans la chaîne affectée. La valeur est soit une chaîne encadrée par deux doubles-quotes (voir paragraphe D - 4 - d) page 21), soit un autre tableau contenant une chaîne.

Langage C	Langage Pascal
<code>strcpy(nom, "nemo"); strcpy(prenom, nom);</code>	<code>nom := 'nemo'; prenom := nom;</code>

Syntaxe : `strcat(chaine_affectée, chaine_valeur);`

Cette fonction concatène la chaîne valeur à la chaîne affectée.

Langage C	Langage Pascal
<code>strcpy(nom, "magic"); strcat(nom, ".wav");</code>	<code>nom := 'magic'; nom := nom + '.wav';</code>

b) Autres instructions

Parmi les autres instructions, on trouve les appels de procédures et les structures de contrôles qui seront étudiées dans le paragraphe F - 2 - page 28.

c) Comment rendre une instruction illisible

Certaines syntaxes peu lisibles ne sont pas recommandées pour deux raisons. Premièrement, les personnes qui seront amenées à travailler sur un programme doivent pouvoir le comprendre sans difficulté. Ensuite, les compilateurs sont capables d'optimiser n'importe quel programme écrit sans artifices. Parmi ces syntaxes absconces :

Syntaxe : `variable opérateur= expression;`

Elle est équivalente à écrire : `variable = variable opérateur expression;`

Exemples :

```
i += 1; code <=< 1; angle *= PI/180.0;
```

Syntaxe :

syntaxe	équivalence
<code>valeur = ++variable</code>	<code>variable = variable + 1 ; valeur = variable;</code>
<code>valeur = variable++</code>	<code>valeur = variable ; variable = variable + 1;</code>
<code>valeur = --variable</code>	<code>variable = variable - 1 ; valeur = variable;</code>
<code>valeur = variable--</code>	<code>valeur = variable ; variable = variable - 1;</code>

Exemples :

Langage C	Langage Pascal
<code>f = facto(n++);</code>	<code>f := facto(n); n := n + 1;</code>
<code>f = facto(++n);</code>	<code>n := n + 1; f := facto(n);</code>
<code>f = facto(n--);</code>	<code>f := facto(n); n := n - 1;</code>
<code>f = facto(--n);</code>	<code>n := n - 1; f := facto(n);</code>

L'emploi de ces instructions n'est pas recommandé dans les paramètres de fonctions car le résultat à l'exécution est souvent imprévisible (l'ordre d'évaluation des paramètres n'est pas normalisé).

On rencontre aussi la syntaxe suivante pour des expressions conditionnelles :

Syntaxe : `(condition ? valeurV : valeurF)`

Cette expression vaut valeurV si la condition est vraie et valeurF sinon.

Langage C	Langage Pascal
<code>hiii = (nb>1) ? "chevaux" : "cheval";</code>	<code>if nb>1 then hiii := 'chevaux' else hiii := 'cheval';</code>

2 - BLOCS ET SOUS-BLOCS

Un bloc permet de regrouper plusieurs instructions afin qu'elles n'en forment plus qu'une seule. Un bloc contient des définitions de variables locales et des instructions. Un bloc est délimité par deux accolades {} et peut contenir des sous-blocs (corps des boucles, alternatives...).

Langage C	Langage Pascal
<code>{ int i = 15; printf("i = %d\n", i); }</code>	<code>begin var i: integer; { interdit !! } value i = 15; { interdit !! } writeln('i =', i); end</code>

Il est d'usage d'indenter les éléments d'un bloc à raison d'une tabulation ou de quatre espaces supplémentaires par rapport à la position des accolades.

a) Définitions de variables locales

Les variables locales d'un bloc sont définies exactement comme les variables globales. Elles ne sont visibles (utilisables) que dans le bloc.

Langage C	Langage Pascal
<pre>{ int i, j; long f; i = 2; j = 3; f = fact(i+j); }</pre>	<pre>var i, j: integer; var f: integer; begin i := 2; j := 3; f := fact(i+j); end</pre>

b) Instructions

Toutes les instructions d'un bloc sauf les sous-blocs et les structures de contrôle doivent se terminer par un ;. Le ; joue le rôle d'un terminateur et non pas celui d'un séparateur, il en faut donc un à chaque instruction.

3 - INSTRUCTIONS CONDITIONNELLES

a) Conditionnelles simples

Syntaxe1: `if (expression) instruction;`

L'instruction n'est effectuée que si l'expression est différente de zéro. L'instruction est soit une instruction simple (suivie de son ;), soit un bloc.

Langage C	Langage Pascal
<pre>if (x < 0) x = -x; if (x == 3 fin) { printf("x = 3 ou c'est fini\n"); }</pre>	<pre>if (x < 0) then x := -x; if ((x = 3) or fin) then begin writeln('x = 3 ou c'est fini'); end</pre>

Syntaxe2: `if (expression) instructionV
else instructionF;`

L'expression est d'abord évaluée. Si elle n'est pas nulle, l'instructionV est faite sinon c'est l'instructionF.

Langage C	Langage Pascal
<pre>if (x == 2) x = 3; else { printf("x != 2\n"); }</pre>	<pre>if (x = 2) then x := 3 else begin writeln('x <> 2'); end</pre>

Il est important d'indenter correctement afin de faire ressortir la correspondance entre le début d'une structure et sa fin.

b) Conditionnelles multiples

Syntaxe: `switch (expression) {
 case valeur1:
 instructions1...;`

```

        break;
    case valeur21:
    case valeur22:
        instructions2...;
        break;
    default:
        instructionsX...;
}

```

La valeur de l'expression détermine quelles sont les instructions qui sont effectuées. Dans le cas où cette valeur vaut valeur1, les instructions1... sont exécutées. Si la valeur vaut valeur21 ou valeur22, les instructions 2... sont exécutées. La ligne default indique les instructions qui sont à faire lorsque l'expression ne correspond à aucun des cas énumérés.

Langage C	Langage Pascal
<pre> switch (reponse) { case 'o': case 'O': ok = 1; unlink("Fichier"); break; case 'n': case 'N': ok = 0; break; default: printf("(O)ui ou (N)on ?\n"); } </pre>	<pre> case reponse of 'o','O': begin ok := true; unlink('Fichier') end; 'n','N': ok := false; else begin writeln('O)ui ou (N)on ?') end end </pre>

4 - BOUCLES

a) Boucles 'Tant que'

Syntaxe1: `while (expression) instruction;`

Syntaxe2: `while (expression) {
 instructions...;
}`

La ou les instructions sont effectuées tant que l'expression est vraie (non-nulle). Si l'expression est fausse dès le début, aucune boucle n'aura lieu. Il est possible de définir des variables locales dans le bloc du corps de la boucle.

Langage C	Langage Pascal
<pre> while (c != 'Q') c = getchar(); while (i < max && t[i] != 0) { printf("%d ",t[i]); i = i + 1; } </pre>	<pre> while (c <> 'Q') read(c); while ((i<max) and (t[i]<>0)) begin write(t[i], ' '); i := i + 1 end </pre>

b) Boucles 'Jusqu'à'

Syntaxe1: `do instruction;
while (expression);`

Syntaxe2: do {
 instructions...;
 } while (*expression*);

L'instruction est effectuée au moins une fois puis l'expression est évaluée. L'instruction est refaite si l'expression est vraie et ainsi de suite jusqu'à ce que l'expression devienne fausse.

Langage C	Langage Pascal
<pre>do c = getchar(); while (c == ' '); do { printf("%d ", t[i]); i = i + 1; } while (i < max && t[i] != 0);</pre>	<pre>repeat read(c) until (c <> ' '); repeat write(t[i], ' '); i := i + 1 until (i>=max) or (t[i]=0);</pre>

c) Boucles 'Pour'

La boucle 'pour' du langage C est extrêmement différente de celle du Pascal. Elle se rapproche d'une boucle 'tant que'. Voici sa syntaxe :

Syntaxe1: for (*instr1*; *expression*; *instr2*)
 instruction;

Syntaxe2: for (*instr1*; *expression*; *instr2*) {
 instructions...;
 }

Cette syntaxe est totalement équivalente à la suivante :

Syntaxe: *instr1*;
 while (*expression*) {
 instructions...;
 instr2;
 }

Au début, l'instruction 1 est exécutée ; elle sert à initialiser les itérations. Ensuite tant que l'expression est vraie, les deux autres instructions sont faites. Cette structure compliquée permet de réaliser toutes sortes de boucles :

Langage C	Langage Pascal
<pre>for (i=0; i<=9; i=i+1) printf("%d ", i);</pre>	<pre>for i:=0 to 9 do write(i, ' ');</pre>
<pre>for (i=1; i<=100; i=i+5) printf("%d ", i);</pre>	<pre>for i:=1 to 100 step 5 do write(i, ' ');</pre>
<pre>for (i=20; i>=10; i=i-1) printf("%d ", i);</pre>	<pre>for i:=20 downto 10 do write(i, ' ');</pre>

NB: step n'existe généralement pas en Pascal.

d) Contrôle des boucles

Le langage C offre la possibilité d'arrêter à tout moment une itération dans une boucle :

Syntaxe: break;

Arrête les itérations de la boucle en cours, continue l'exécution après le bloc courant.

Syntaxe : `continue;`

Saute directement juste avant la fin du bloc courant afin de recommencer immédiatement une nouvelle itération. Ces deux instructions permettent d'éliminer des conditions supplémentaires.

5 - INDENTATION

Un programme C peut être illisible si le programmeur ne respecte pas une norme de présentation. En particulier, il est important de faire ressortir nettement la structure des blocs d'une fonction. L'indentation proposée dans ce cours est celle de Kernighan & Ritchie.

G - FONCTIONS ET PROCEDURES

1 - DEFINITION D'UNE FONCTION

a) Définition

Une fonction est définie selon la syntaxe suivante :

Syntaxe : `typerésultat nomfonction (déclarations_paramètres)`
 {
 définitions_locales;
 instructions...;
 return expression;
 }

La déclaration des paramètres est identique à la définition de variables : elle consiste à indiquer le type puis le nom du paramètre et à séparer ces déclarations par des virgules. Il n'est pas possible de déclarer en même temps des paramètres de même type (`int a, b, c` doit être écrit `int a, int b, int c`).

Lorsque la fonction n'a pas de paramètres, il est nécessaire de mettre le mot clé `void` entre les parenthèses.

L'instruction `return` peut être mentionnée plusieurs fois dans le corps. Son exécution provoque la sortie immédiate de la fonction.

Langage C	Langage Pascal
<pre>char amaj(void) { return 'A'; }</pre>	<pre>function amaj: char; begin amaj = 'A'; end;</pre>
<pre>int facto(int n) { int i, f; f = 1; for (i=1; i<=n; i=i+1) f = f * i; return f; }</pre>	<pre>function facto(n:integer):integer; var i, f: integer; begin f := 1; for i:= 1 to n do f := f * i; fact := f end;</pre>

Langage C	Langage Pascal
<pre>int max(int a, int b) { if (a > b) return a; else return b; }</pre>	<pre>function max(a,b: integer):integer; begin if (a > b) max := a else max := b end;</pre>

b) Appel

L'appel d'une fonction en C est identique à celui du Pascal :

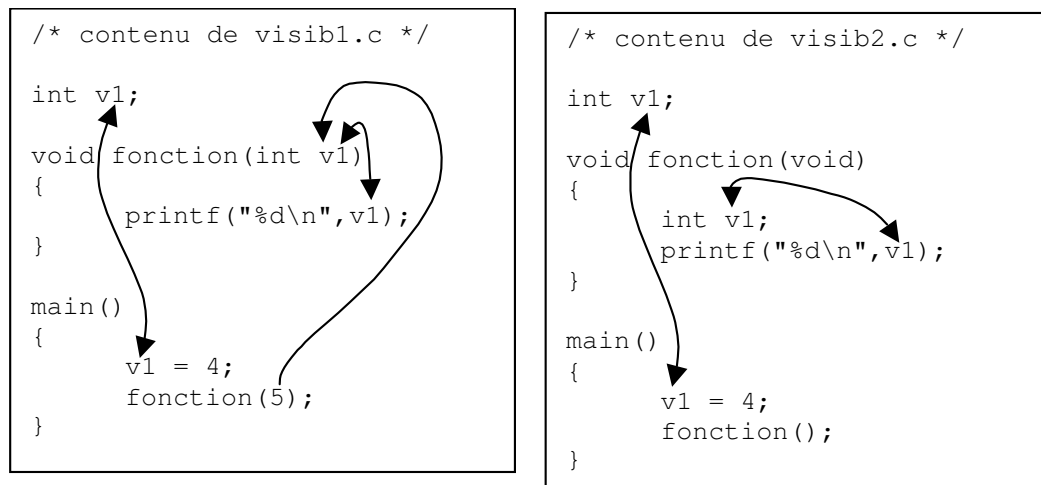
Syntaxe : `nomfonction(arguments)`

Les types des arguments doivent correspondre (à une conversion près) à ceux des paramètres de la fonction. Certains compilateurs ne sont pas vigilants et ne contrôlent pas les types, ni des arguments, ni du résultat de la fonction.

Langage C	Langage Pascal
<pre>{ int x; for (x=1; x<=7; x=x+1) printf("%d! = %d\n", x, fact(x)); }</pre>	<pre>var x:integer; begin for x:=1 to 7 do writeln(x, '! = ', fact(x)) end.</pre>

c) Rappels sur la visibilité d'un symbole

Pour compléter ce chapitre, on rappelle quelques notions sur la visibilité des identificateurs. Dans l'exemple suivant on crée deux variables du même nom, l'une globale et l'autre locale à une fonction :



Dans les deux cas, lorsqu'on est entre les `{}` de la fonction, le symbole `v1` désigne la variable locale et lorsqu'on est hors des `{}` de la fonction, `v1` désigne la variable globale. Si on veut manipuler la variable globale à l'intérieur de la fonction, il faut donner un autre nom à la variable locale.

d) Prototypes et prédéclarations

La prédéclaration permet au compilateur C de compiler correctement les appels situés avant la définition de la fonction. Cette prédéclaration est nécessaire dans le cas d'une définition externe, c'est à dire située dans un autre fichier source (compilation modulaire). La prédéclaration consiste à définir un prototype de la fonction, il s'agit de la fonction privée de son corps.

Syntaxe : `typerésultat nomfonction (paramètres);`

Dans ce cas, il n'est pas nécessaire de fournir le nom des paramètres, seuls leurs types sont pris en considération par le compilateur. Lorsque la fonction n'a pas de paramètres, il est nécessaire de placer `void` entre les parenthèses.

Langage C	Langage Pascal
<pre>char amaj(void); int facto(int); int max(int, int); void echange(int *i, int *j);</pre>	<pre>function amaj: char; forward; function fact(n: integer):integer; forward; function max(a,b: integer):integer; external; procedure echange(var i,j:integer); external;</pre>

En Pascal, `forward` signifie que la fonction complète suit dans le fichier. On peut utiliser `external` pour signaler que la définition est présente dans un autre fichier.

e) Passage par valeur

Par défaut, les paramètres d'une fonction C sont passés par valeur : une fonction ne peut pas modifier la valeur d'une variable passée en paramètre.

Langage C	Langage Pascal
<pre>void SansEffet(int n) { n = n + 10; printf("dedans: n=%d\n", n); } main() { int i = 3; printf("avant: i=%d\n", i); SansEffet(i); printf("apres: i=%d\n", i); }</pre>	<pre>procedure SansEffet(n:integer); begin n := n + 10; writeln('dedans: ', n); end; var i: integer; begin i := 3; writeln('avant: ', i); SansEffet(i); writeln('apres: ', i); end.</pre>

Ces deux programmes qui affichent tous deux : « avant i=3, dedans n=13, apres i=3 » montrent que le paramètre n'a pas été modifié en dehors de la procédure.

f) Passage par variable

Lorsqu'on désire qu'un paramètre soit modifié en sortie, il faut le faire précéder d'une `*` à chacune de ses mentions (déclaration du paramètre et utilisations dans le corps de la fonction). D'autre part, il est nécessaire de placer un `&` devant les arguments d'appel passés par variable.

Langage C	Langage Pascal
<pre>void PlusDix(int *n) { *n = *n + 10; printf("dedans: n=%d\n", *n); } main() { int i = 3; printf("avant: i=%d\n", i); PlusDix(&i); printf("apres: i=%d\n", i); }</pre>	<pre>procedure PlusDix(var n:integer); begin n := n + 10; writeln('dedans: ', n); end; var i: integer; begin i := 3; writeln('avant: ', i); PlusDix(i); writeln('apres: ', i); end.</pre>

Ces deux programmes qui affichent tous deux : « avant i=3, dedans n=13, apres i=13 » montrent que le paramètre a pas été modifié par la procédure.

Cette étoile concerne les variables de type scalaire. Les tableaux sont toujours passés par variable.

g) Cas des paramètres de type tableau

Les paramètres de type tableau doivent être déclarés sous la forme :

Syntaxe : *typecase nomparamètre* []

Syntaxe : *typecase nomparamètre* [] [*nbc colonnes*]

et ainsi de suite pour des tableaux de plus grandes dimensions : il ne faut pas indiquer la taille de la dimension la plus à gauche.

Exemple :

Langage C	Langage Pascal
<code>int NbreMots(char page[][24][81])</code>	<code>function NbreMots(page:array[1..100,1..24,1..80] of char):integer;</code>

Les tableaux et les structs sont passés obligatoirement par variable. C'est à dire que si la fonction *NbreMots* change un caractère dans son paramètre *page*, ce caractère sera également changé dans le tableau passé lors de l'appel de la fonction :

NB : Dans une fonction, les paramètres de type tableau sont considérés comme des pointeurs. On ne peut pas trouver facile leur manipulation par rapport à ce qu'offre le langage Pascal.

2 - DEFINITION D'UNE PROCEDURE

Le langage C ne propose pas de syntaxe particulière pour définir une procédure. Il s'agit simplement d'une fonction dont on n'utilise pas le résultat (déclaré en général de type `void`).

Syntaxe : *void nomprocédure(déclarations_paramètres)*
 {
 définitions_locales;
 instructions...
 }

Les instructions `return` ne doivent pas mentionner de valeur.

Langage C	Langage Pascal
<pre>char Messages[2][21] = { "plus de mémoire", "trop fatigué" }; void Erreur(int n) { printf("Erreur %d: %s\n", n, Messages[n-1]); exit(n); /* arrêt du programme */ } main() { Erreur(2); }</pre>	<pre>var Messages: array[1..2,1..20] of char; (* initialisation ??? *) procedure Erreur(n: integer); begin writeln('Erreur ', n, ': ', Messages[n]); halt { arrêt du programme } end; begin Erreur(2) end.</pre>

3 - PROCEDURES ET FONCTIONS STANDARD

Le langage C ne possède aucune fonction ni procédure intégrée comme celles du langage Pascal (`write`, `read`,...). En revanche, toutes les bibliothèques standard, mathématique et système sont facilement accessibles sans aucune contrainte.

- La bibliothèque standard (`stdio`) contient toutes les fonctions d'entrées/sorties (terminal et fichiers) et de gestion des chaînes de caractères ;
- La bibliothèque mathématique (`math`) contient les fonctions mathématiques (racines, logarithmes, trigonométrie) ;
- Les bibliothèques système contiennent toutes les fonctions de gestion de la machine — voir le cours de système de deuxième année ;
- Il existe de très nombreuses autres bibliothèques pour gérer les fenêtres (`Curses`, `X11`, `OpenGL`...).

a) Lecture du clavier

Syntaxe : `char getchar(void);`

Renvoie le prochain caractère tapé au clavier. `void` signifie qu'elle n'accepte pas de paramètre.

Exemple :

Langage C	Langage Pascal
<pre>main() { char c; do c = getchar(); while (c != 'Q'); }</pre>	<pre>var c: char; begin repeat read(c) until (c = 'Q'); end.</pre>

Syntaxe : `void gets(char s[]);`



Place dans le paramètre tableau (passé en variable) une ligne lue au clavier.

Langage C	Langage Pascal
<pre>char ligne[80]; main() { do gets(ligne); while (ligne[0] != 'Q'); }</pre>	<pre>var ligne: string(80); begin repeat readln(ligne) until (ligne[1] = 'Q'); end.</pre>

Cette fonction est dangereuse et le compilateur le signale, si on ne fournit pas une chaîne assez grande. La lecture au clavier débordera de l'espace alloué et écrasera les variables suivantes. Cela parce que le langage C ne gère pas la taille des tableaux. Dans l'exemple précédent, si on tape plus de 80 caractères, on fait planter le programme. Beaucoup de virus d'internet exploitent ces faiblesses du C.

Syntaxe : `int scanf(char format[], &variables...);`

Place dans les variables (toutes passées par variable, donc toutes précédées d'un &²) les données correspondantes au format. Ce format définit le type des variables à lire. C'est une chaîne contenant des jokers indiquant ce qu'il faut extraire :

joker	pour extraire un...
%c	char
%s	un seul mot (char [])
%d	int
%ld	long
%f	float
%lf	double

La fonction `scanf` renvoie le nombre de variables qui ont pu être correctement affectées.

Langage C	Langage Pascal
<pre>int numero; char nom[20]; printf("Taper un nom et un numéro:"); scanf("%s %d", nom, &numero) ;</pre>	<pre>var numero: integer; nom: string(20); write('Taper un nom et un numéro:'); read(nom, numero) ;</pre>
<pre>int x, y; double d; main() { printf("Entrer x,y:dist "); scanf("%d,%d:%lf", &x, &y, &d);</pre>	<pre>var x,y: integer; var d: real; begin write('Entrer x,y:dist '); read(x, y, d);</pre>

NB : faire attention au mélange de %c et %s. Le joker %s lit un mot (suite de char suivie d'un séparateur) et non pas une ligne de texte. Faire également attention au mélange entre la fonction `scanf` et les fonctions `getchar` et `gets` : en principe, on n'utilise pas les fonctions `get...` dans un programme contenant des `scanf`.

b) Affichage sur écran

Syntaxe : `void putchar(char c);`

Affiche le caractère passé en paramètre.

²ce prototype n'est pas correct mais la définition de fonctions à nombre d'arguments variable sort du cadre de ce poly.

Langage C	Langage Pascal
<pre>main() { char c; for (c='A'; c<='Z'; c=c+1) putchar(c); putchar('\n'); }</pre>	<pre>var c: char; begin for c:='A' to 'Z' do write(c); writeln; end.</pre>

Syntaxe : `void puts(char s[]);`

Affiche la ligne située dans le tableau passé en paramètre en ajoutant un retour à la ligne.

Langage C	Langage Pascal
<pre>char ligne[80]; main() { do { gets(ligne); puts(ligne); while (ligne[0] != 'Q'); } }</pre>	<pre>var ligne: string(80); begin repeat readln(ligne); writeln(ligne); until (ligne[1] = 'Q'); end.</pre>

Syntaxe : `void printf(char format[], valeurs...);`

Affiche les valeurs fournies en fonction du format. Le format est composé d'une chaîne de caractères contenant les mêmes jokers que ceux de `scanf`. Le joker `%s` affiche la totalité de la chaîne indiquée.

Langage C	Langage Pascal
<pre>char c, mot[80]; int i, j; double a; main() { printf("c=%c, m=%s, i=%d, j=%d, a=%lf\n", c, m, i, j, a); }</pre>	<pre>var c: char; m: string(80); var i, j: integer; var a: real; begin writeln('c=', c, ' m=', m, ' i=', i, ' j=', j, ' a=', a); end.</pre>

c) Fichiers

Ce polycopié ne présente que les fonctions de base. Voici les équivalences Pascal/C pour les fichiers texte. En C, il faut commencer par inclure le fichier `stdio.h` afin de définir le type `FILE` (les majuscules sont obligatoires).

Syntaxe : `FILE *f;`

définit la variable fichier `f` : il s'agit d'une structure de données décrivant l'état d'un fichier. Il est inutile de savoir ce qu'elle recouvre.

Syntaxe : `FILE *fopen(char nomfichier[], char mode[]);`

Le paramètre chaîne `mode` indique si on ouvre en lecture "r" ou en écriture "w". En retour, la fonction renvoie une valeur de type `FILE *` ou la constante `NULL` dans le cas d'une erreur (fichier absent ou interdit...). Le langage Pascal standard ne permet pas de récupérer les cas d'erreur.

Syntaxe : `int fclose(FILE *fichier);`

Cette fonction referme le fichier ouvert. Elle renvoie une valeur qu'on peut ignorer, -1 en cas d'erreur, 0 si la fermeture s'est bien déroulée.

Langage C	Langage Pascal
<pre>#include <stdio.h> FILE *f1, *f2; { f1 = fopen("donnees","r"); f2 = fopen("copie", "w"); if (f1==NULL f2==NULL) exit(1); fclose(f1); }</pre>	<pre>var f1,f2: text; begin reset(f1,'NAME=donnees'); rewrite(f2,'NAME=copie'); (* pas de controle possible *) close(f1); end</pre>

NB: la version Pascal est à la norme XL Pascal sur AIX (IBM).

Syntaxe: `void fprintf(FILE *fichier, char format[], valeurs...);`

Cette fonction s'applique aux fichiers ouverts en écriture, elle fonctionne comme la fonction printf.

Syntaxe: `int fscanf(FILE *fichier, char format[], &variables..);`

Cette fonction s'applique aux fichiers ouverts en lecture, elle fonctionne comme la fonction scanf. On rappelle que le joker %s ne lit qu'un mot à la fois. La fonction fscanf renvoie le nombre de variables qu'elle a pu lire, -1 s'il la fin de fichier est atteinte.

Langage C	Langage Pascal
<pre>int v; while (fscanf(f1,"%d",&v) == 1) { fprintf(f2,"%d\n",v); }</pre>	<pre>var v : integer; while not eof(f1) do begin read(f1,v); writeln(f2,v); end;</pre>

Il existe de nombreuses fonctions qu'il serait trop long de documenter ici.

4 - LIBRAIRIE MATHÉMATIQUE

Les fonctions mathématiques (racines, logarithmes, trigonométrie...) ne sont pas disponibles directement en tant qu'éléments du langage. Ce sont au contraire du Pascal des fonctions externes qu'il faut déclarer au moment de la compilation puis lier avec l'éditeur de liens. Voici une description de ces deux étapes qui convient également à toute autre librairie (fichiers, système, graphique).

a) Déclaration des fonctions

Normalement, une fonction externe doit être prédéclarée par un prototyp. Ainsi, le compilateur ne commet pas d'erreur sur les types des paramètres qui lui sont passés et le type du résultat que fournit la fonction.

Exemples :

```
extern double sqrt(double);
extern double pow(double, double);
```

Pour éviter de respécifier à chaque programme les types des paramètres et du résultat de chaque fonction appelée dans la librairie ainsi que différentes constantes telles que PI, E..., on peut inclure un fichier indiquant au compilateur les déclarations des fonctions des librairies et les types des données. Cette inclusion est provoquée par une directive `#include`, voir le §H - 1 - page 41.

```
#include <math.h>
```

Le fichier `math.h` est recherché dans le répertoire `/usr/include`.

NB1: Il faut veiller à ne pas inclure systématiquement trop de fichiers car ça ralentit la compilation.

NB2: ne pas confondre les fichiers `.h` qui ne contiennent que les prédéclarations des fonctions et constantes avec les fichier objet librairie `.a` contenant le code machine des fonctions — voir plus bas.

b) Utilisation des fonctions

Une fois prédéclarées, toutes les fonctions de la librairie sont utilisables, voici quelques exemples :

racine carrée(a) :	<code>sqrt(a)</code>
a^b :	<code>pow(a,b)</code>
logarithme népérien(a) :	<code>log(a)</code>
logarithme de base 10 :	<code>log10(a)</code>
e^a :	<code>exp(a)</code>
fonctions trigonométriques et hyperboliques :	<code>sin(a), acos(a), tanh(a) ...</code>
fonctions de conversion :	<code>atof(chaine)</code> convertit une chaîne en float
constantes :	<code>M_PI, M_E, MAXFLOAT</code>

Exemple :

Langage C	Langage Pascal
<code>#include <math.h></code>	<code>const pi = 3.1415926535897;</code>
<code>main()</code>	<code>begin</code>
<code>{</code>	<code> writeln(sqrt(log(pi)))</code>
<code> printf("%f\n", sqrt(log(M_PI)));</code>	<code>end.</code>
<code>}</code>	

c) Compilation

Le seul changement concerne l'édition des liens, puisqu'il est nécessaire d'indiquer le nom de la librairie à lier. L'ensemble de ces librairies se trouve dans `/usr/lib`, ce sont les fichiers `lib*.a`, exemple: `libm.a`.

Commande : `cc -Wall prog.c -o prog -lm`

compile le source `prog.c`, le lie à la librairie mathématique (m) et crée l'exécutable `prog`.

5 - ARGUMENTS D'UN PROGRAMME

Le langage C-Shell propose des variables spéciales `$1...$9` et `$argv` pour passer des chaînes en paramètre d'un programme. Un tel dispositif existe en C : il s'agit des paramètres de la procédure `main()`.

Syntaxe : `void main(int nombre, char *dollar[])`

`nombre` est un paramètre indiquant le nombre de paramètres qui ont été passés au programme + 1, `dollar` est un tableau de chaînes, `dollar[0]` contient le nom du programme, `dollar[1]` est le premier paramètre, `dollar[2]` est le deuxième et ainsi de suite jusqu'à `dollar[nombre-1]`.

NB: ces deux paramètres sont souvent appelés `argc` et `argv` mais ces noms ne sont pas imposés.

Langage C – <code>params.c</code>	Langage Pascal
<pre>main(int nombre, char *dollar[]) { int i; printf("programme %s appelé avec %d paramètres\n", dollar[0], nombre-1); for (i=1; i<nombre; i=i+1) printf("paramètre %d = %s\n",i,dollar[i]); }</pre>	<p>aucune équivalence normalisée</p>

H - PREPROCESSEUR C

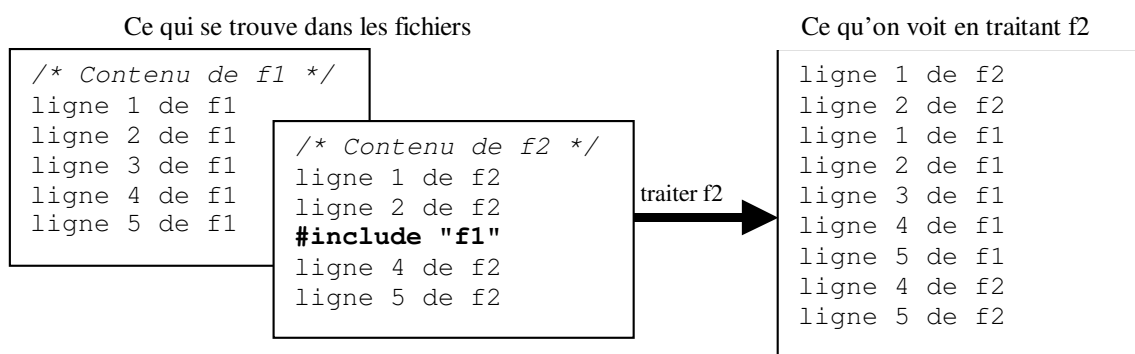
Le préprocesseur C est une sorte d'éditeur de texte automatique qui transforme le programme source C juste avant la compilation : il insère les fichiers inclus, il remplace les constantes et il effectue certains choix de compilation. Ce prétraitement était nécessaire à l'origine car les compilateurs n'étaient pas assez puissants. Une syntaxe vieillie et parfois trompeuse est restée de cette époque.

1 - INCLUSION DE FICHIERS

a) Présentation

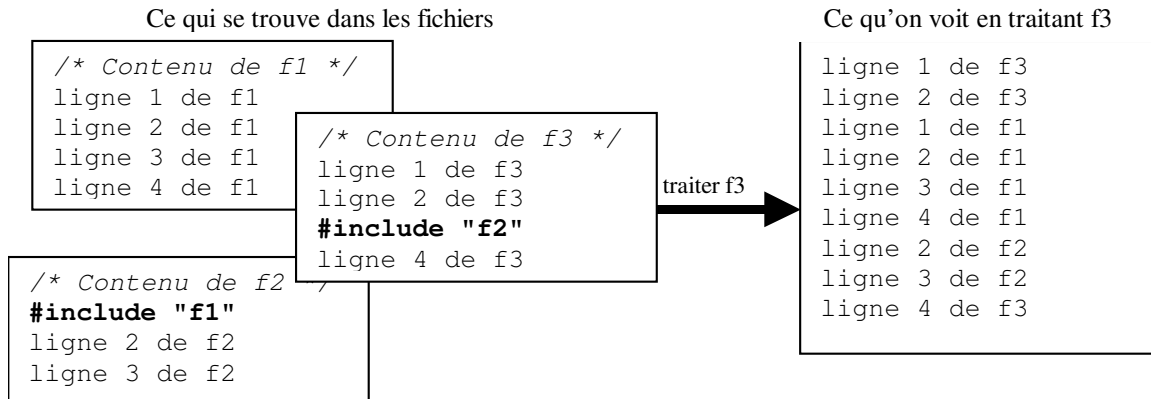
Inclure un fichier `f1` dans un autre fichier `f2` consiste à ce que lorsqu'on parcourt `f2`, on parcourt également `f1`. En compilation C, on se sert de cette technique pour partager des déclarations et définitions entre différents programme.

Il existe deux manières de faire cela : insérer le premier fichier dans le second de manière permanente avec l'éditeur de texte ou demander au compilateur de se dérouter temporairement par une « directive » spéciale :



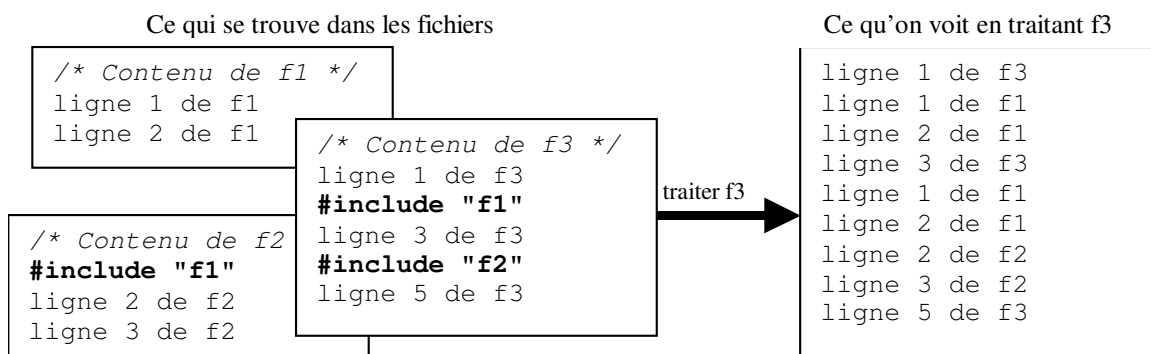
Remarquer que l'ordre et le contenu des lignes est exactement préservé.

Ce mécanisme fonctionne également en cascade :



Remarquer l'ordre exact des lignes, selon la position des directives d'inclusion.

Un même fichier peut se retrouver inclus plusieurs fois au final :



On voit que dans cet exemple, certaines lignes sont présentes en multiples exemplaires. Cela posera un problème au compilateur C si ces lignes sont des définitions.

b) Directives #include

Il y a deux syntaxe pour inclure un fichier selon qu'il est dans le répertoire courant ou dans un répertoire connu du compilateur. La différence se fait sur les délimiteurs du nom de fichier "" ou <> :

- **#include "fichier"**

Cette directive cherche le fichier dans le répertoire courant. Il est possible de donner un nom complet afin d'inclure un fichier situé ailleurs :

```
#include "...chemin/fichier"
```

- **#include <fichier>**

Cette directive cherche le fichier dans l'un des répertoires privilégié du compilateur : /usr/include, /usr/local/include... Ces répertoires contiennent tous les fichiers inclus correspondant aux bibliothèques du système. Par exemple :

```
#include <stdio.h>
#include <math.h>
```

Il est possible de donner un autre chemin au compilateur C grâce à l'option -I, par exemple si les fichiers à inclure se trouvent dans ../inclus :

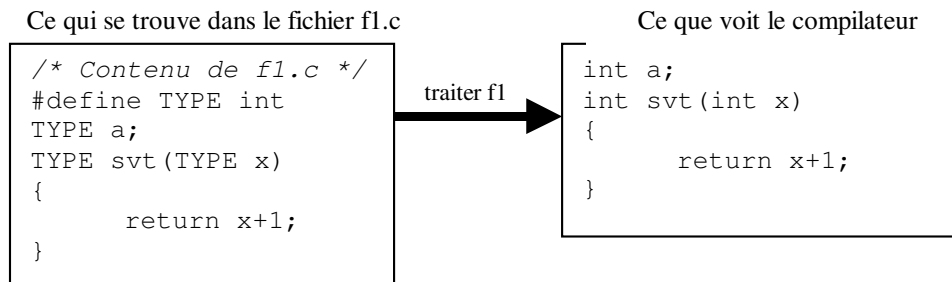
```
Cc -Wall -I../inclus prog.c ...
```

2 - DEFINITION DE MACROS

a) Constantes

#define SYMBOLE CHOSE

La directive #define permet d'associer une chaîne ou une valeur à un symbole (identificateur). Toutes les mentions du symbole seront automatiquement remplacées par la valeur, par exemple :

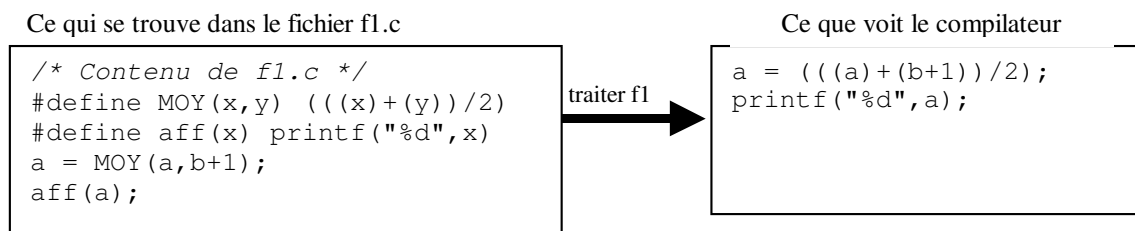


Remarquer que les remplacements n'ont lieu qu'à partir du moment où la définition est faite et d'autre part, il faut que le symbole soit clairement visible selon les règles de syntaxe du langage.

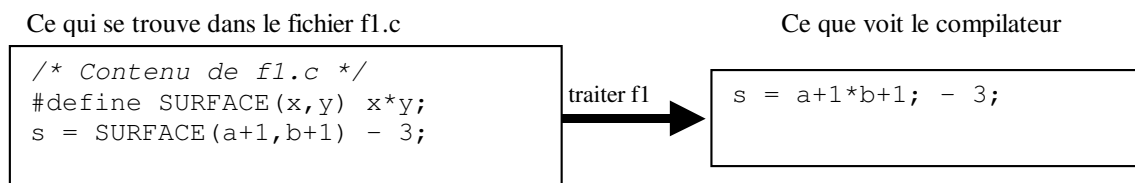
b) Macrofonctions (macros)

#define SYMBOLE(paramètres) CHOSEPARAMETRÉE

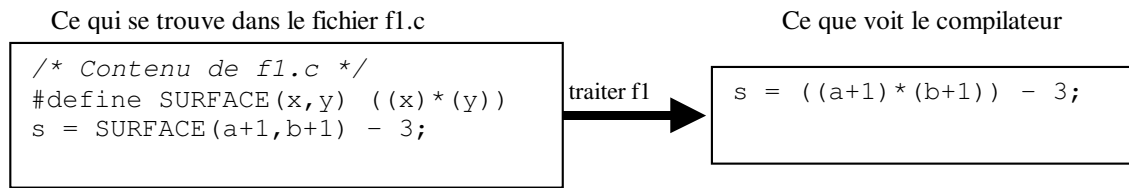
La directive #define permet d'associer une chaîne variable en fonction de paramètres à un symbole ressemblant à une fonction. Cela signifie que toutes les mentions du symbole accompagné de paramètres seront automatiquement remplacées par la chaîne.



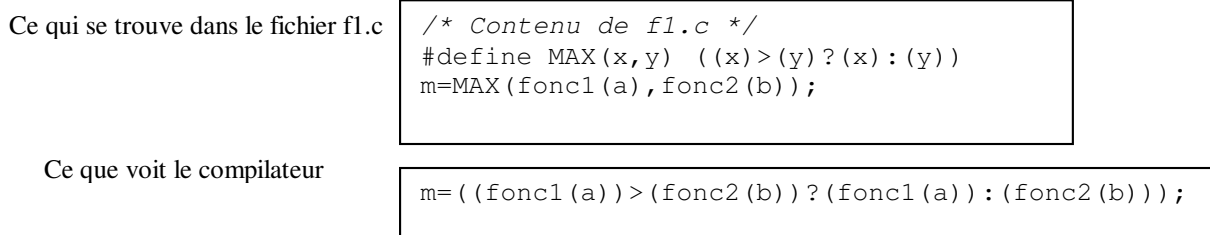
Remarquer que le remplacement est uniquement syntaxique, c'est à dire qu'on ne vérifie pas la syntaxe de ce qui en résulte. Cela peut faire des erreurs de syntaxe et de sémantique comme dans l'exemple suivant :



L'erreur provient du fait que le remplacement de x par a+1 et y par b+1 n'est pas protégé, le compilateur calculera a+b+1 au lieu de (a+1)*(b+1). D'autre part, le ; est de trop. Il aurait fallu prévoir cela :



Attention, ces macros ne sont pas des fonctions. Les employer à mauvais escient peut rendre le programme inefficace. Dans l'exemple suivant, on rappelle une deuxième fois la fonction qui rend la valeur la plus élevée.



3 - COMPILATION CONDITIONNELLE

Ce mécanisme permet de ne pas compiler certaines parties du programme selon des conditions purement syntaxiques (symbole défini ou pas par un #define).

#ifdef SYMBOLE

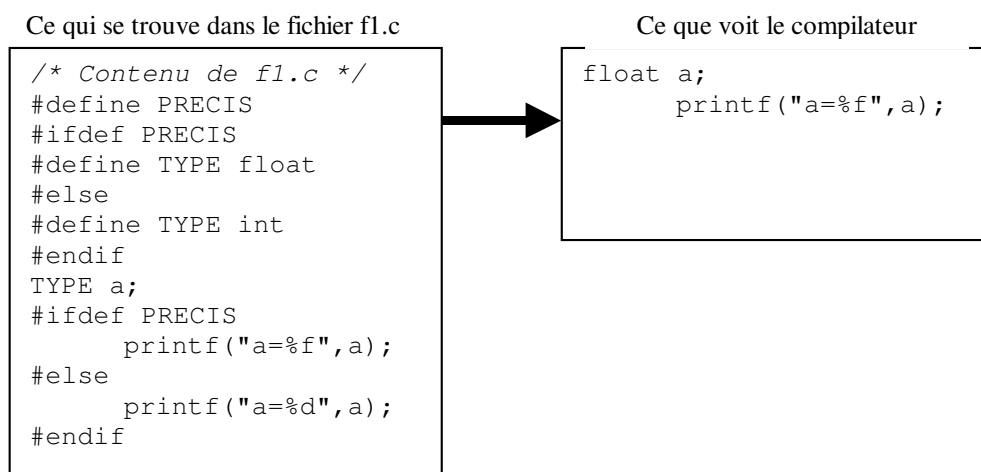
ces lignes sont prises en compte ssi le symbole est défini

#else

ces lignes sont prises en compte ssi le symbole n'est pas défini

#endif

On rappelle qu'on se situe dans la phase précédant la compilation, dans laquelle le préprocesseur transforme le fichier source. Avec la directive #ifdef, on peut choisir les transformations à effectuer.



Comme le symbole PRECIS a été défini au début du fichier, le préprocesseur laisse passer les lignes situées entre #ifdef et #else. Dans l'exemple suivant, on met en commentaire la définition du symbole PRECIS, les lignes situées entre le #else et le #endif sont maintenant les seules à être considérées.

Ce qui se trouve dans le fichier f1.c

```
/* Contenu de f1.c */  
/* #define PRECIS */  
#ifndef PRECIS  
#define TYPE float  
#else  
#define TYPE int  
#endif  
TYPE a;  
#ifndef PRECIS  
    printf("a=%f", a);  
#else  
    printf("a=%d", a);  
#endif
```

Ce que voit le compilateur

```
int a;  
    printf("a=%d", a);
```

L'option `-Dnom` permet de définir un symbole lors de la compilation, comme si on avait mis une ligne `#define nom` dans le source, ça permet de définir le comportement du compilateur lors de la compilation seulement :

```
cc -DPRECIS f1.c
```

Remarque finale : le langage C offre de nombreuses autres subtilités qu'il n'est pas possible de développer dans ce polycopié.

* *
*

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com